Enhancing authentication in ebanking with NFC enabled mobile phones

Ortiz-Yepes, D.A.

Award date:
2008

Enhancing authentication in eBanking with NFC enabled mobile phones

Diego Alejandro Ortiz-Yepes

Master's Thesis
August 11, 2008

Supervisors
dr. Michael Baentsch (IBM Zurich Research Laboratory)
dr. ir. L.A.M. (Berry) Schoenmakers (TU/e)

Department of Mathematics and Computer Science
Eindhoven University of Technology

Enhancing authentication in eBanking with NFC enabled mobile phones

D.A. Ortiz-Yepes

Enhancing authentication in eBanking with NFC enabled mobile phones

D.A. Ortiz-Yepes

ACKNOWLEDGEMENTS

This Master's Project was conducted between February and July of 2008 at the IBM Zurich Research Laboratory. I would like to thank all the members of the *BlueZ Business Computing group* for their support and the interesting discussions during my stay at the Lab.

I particularly would like to thank Reto Hermann, Dr Michael Baentsch, Dr Tamas Visegrady and Daphne Bell for their helpful comments and feedback on the draft version of this document. Also, thanks to Michael Kuyper for providing the smart card CAP applet implementation. and to to Dr Berry Schoenmakers and Dr Boris Skoric at the Technische Universiteit Eindhoven for their participation on the assesment comitee for this project.

Finally, I would like to thank Dr. Peter Buhler and Dr Michael Baentsch for their guidance and advice prior and throughout the duration of this project.

CONTENTS

LIST OF FIGURES

ACRONYMS AND ABBREVIATIONS

AAC	Application authentication cryptogram
AC	Application cryptogram
AID	Application identifier
APDU	Application protocol data unit
API	Application programming interface
ARPC	Authorization response cryptogram
ARQC	Authorization request cryptogram
ARC	Authorization response code
ATC	Application transaction counter
ATM	Automatic teller machine
CA	Certification authority
CAP	Chip authentication program
CDA	Combined DDA/Application cryptogram generation
CIA	Confidentiality, integrity and availability
CPU	Central processing unit
CSU	Card status update
DDA	Dynamic data authentication
DNS	Domain name server
DPA	Dynamic passcode authentication
EDGE	Enhanced data rates for GSM evolution
EPROM	Erasable programmable read-only memory
EMV	Europay - Master Card - Visa

FINREAD Financial transactional IC card reader

GPRS General packet radio service

HCISec Human computer interaction security

HTML Hypertext markup language

HTTP Hypertext transfer protocol

HTTPS HTTP Secure

IAD Issuer authentication data

ICC Integrated circuit card

MAC Message authentication code

mTAN Mobile TAN

MITM Man in the middle

NFC Near field communication

OTP One time password

PC Personal computer

PCR Personal card reader

PDA Personal digital assistant

PKI Public key infrastructure

PIN Personal identification number

POS Point of sale

RAM Random access memory

RFID Radio frequency identification

ROM Read-only memory

RSA Rivest-Shamir-Adleman public key cryptosystem

SEPA Single Euro payments area

SDA Static data authentication

SIM Subscriber identity module

SMS Short message service

SSL/TLS Secure socket layer/Transport layer security

TAN Transaction authentication number

TC	Transaction certificate
TCP	Transmission control protocol
TLV	Tag - Length - Value
TPM	Trusted platform module
UI	User interface
UMTS	Universal mobile telecommunications service
UN	Unpredictable number
url	Universal resource locator
USB	Universal serial bus
ZTIC	Zone Trusted Information Channel

CHAPTER

ONE

Introduction

1.1 Problem Description

This Master's Project addresses the problem of *remote authentication of eBanking customers*. The notion of *eBanking* corresponds to the usage of the Internet site of a financial institution by its customers with the intention of checking their account balance or performing some kind of financial transaction, for instance, transferring money to other people or companies (e.g. paying a utility bill).

Authentication, on the other hand, is to be understood as the bank (hereinafter denoted as *the bank*) making sure that it is interacting with a legitimate customer, and vice-versa, i.e. the customer making sure that she is interacting with the bank. Additionally, authentication is not restricted to the *entities* as presented above, but also to the *information* exchanged between them. In other words, a party must be able to determine whether information seemingly coming from the other party does in fact come from it. For example, if the bank receives a message from a Customer C "*transfer \$X from my savings account #Y to account #W*", it must be able to make sure that such a message was effectively sent by C. Conversely, when a customer receives information related to her account(s), she must be able to determine that such information was in fact sent by the bank. Finally, authentication is considered *remote* because the end entities, i.e. the bank and the customer are geographically separated, bound only by their communication through the Internet.

The importance of remote authentication in eBanking lies in the fact that the bank must only allow legitimate customers to access *their* account information and perform financial transactions with *their* assets. Thus, in order to determine whether an entity accessing the eBanking portal is a legitimate customer or not, the bank needs to *authenticate* her.

The scope of this project consists of presenting a general survey of the state of the art in eBanking authentication, followed by the development and assessment of a Near Field Communication (NFC) based authentication mechanism. NFC is an emerging technology based on Radio frequency identification (RFID) that allows devices to communicate over very short distances, i.e. in the order of a few centimeters. This emerging technology has already been incorporated into some commercially available mobile phones, and when employed in tandem with authentication mechanisms such as Master Card's Chip authentication program (CAP) appears to enhance their level of usability as shown by

this project.

1.2 Motivation and Goals

The *raison d'être* of this project is that the resulting prototype will be used in a real world pilot in the second semester of 2008 by a financial institution working with the IBM BlueZ Business Computing Group, where this project has been conducted.

As a Master's thesis in Information Security Technology (IST) this project has been perceived by the author as an opportunity to adopt both a systematic and pragmatic approach to security, covering not only the security theoretical foundations of the problem, but also their relation to the adopted solution in terms of the implementation, touching also on other factors such as usability and convenience.

As a matter of fact, even though security is a prime aspect of the issue at hand, it is by no means considered the sole defining factor. It will be shown in Chapter 5 that from a security perspective there are solutions that are more effective, yet have not gained wide acceptance. Further, Chapter 2 will show that a perfect practical solution to the remote authentication problem is not possible, which should not be surprising considering that even in real life such perfect solutions rarely exist. In general, solutions to security problems are always bound to a set of assumptions within which the solution is guaranteed to work and which, if violated, can render the solution ineffective. With these considerations in mind, this project does not consider a perfect solution, but rather a *realizable* one *balancing security, functionality, and usability.*

Particularly, rather than dealing with a radically new solution to the authentication problem in eBanking, a solution leveraging existing standards and technology without requiring extensive infrastructure changes is considered and presented in Chapter 6. This approach is heavily motivated and inspired both by Gutmann et al. and Garfinkel, who propose that "*the primary goal for current security efforts shouldn't be to further refine how many key bits can fit on the head of a pin, but to figure out how to make the existing stuff usable*" [23], and that it should be "*ensur[ed] that systems offering some security features are deployed now, rather than leaving these systems sitting on the shelf while researchers try to develop "perfect" security systems for deployment later*" [20].

1.3 Document Outline

The remainder of this document is organized as follows: Chapter 2 introduces several concepts regarding information security, giving special attention to the subject of authentication. Chapter 3 scopes and describes eBanking, identifying its actors and associated goals. Further, it also presents the attacker model that shall be used to assess the effectiveness of the eBanking security systems presented in chapters 5 and 6.

Chapter 4 introduces the most relevant technologies and standards used in eBanking security, focusing on those used by the proposed solution presented in Chapter 6. Chapter 5 presents a survey of remote authentication techniques in eBanking outlining their strengths and limitations in the light of the goals presented in Chapter 3.

Chapter 6 is the core of this document as it describes and discusses the proposed authentication solution using NFC enabled mobile phones. Finally, Chapter 7 recapitulates the main lessons and conclusions obtained from this project.

CHAPTER

TWO

Overview of Information Security

This chapter presents a conceptual framework of information security that will be useful in the following chapters. By presenting this framework and its related concepts it is also hoped to avoid the *natural tendency among computer engineers to be loose with their choice of language* [20]. Naturally, though, only a brief introduction to the field of information security will be presented. For a more comprehensive and detailed coverage of the presented topics, the interested reader is referred to the main references of this chapter, i.e. [4, 5, 8, 9, 32, 51, 52, 54, 56].

Rather than presenting an encyclopedia-like description of information security in the form of a glossary, it has been decided to introduce the main concepts in Section 2.1 by depicting their relationships, as well as attempting some definitions and discussing their more important characteristics. Subsequently, the subjects of *Cryptography* and *Access Control*—particularly *Authentication*—shall be covered in more detail in sections 2.2 (page 8) and 2.3 (page 9), respectively.

2.1 Information Security: The Big Picture

Security is defined by Schneier in [52] as *preventing adverse consequences from the intentional and unwanted actions of others*. This is useful both as a definition and as a starting point for a discussion because it implicitly includes many questions that need to be answered: what is it meant by *unwanted* consequences? Who determines what these adverse consequences are? Who are the *others*? It can be easily seen from this definition that security is rather meaningless without a context within which to address these questions. Indeed, such a context can be defined by tackling four fundamental questions: *What* is being protected? *From whom* is it being protected? *Against what* is it being protected? and *How* is it being protected? In this fashion, before engaging in any discussion regarding security it must be absolutely clear what the answers to the questions are in order to ensure that there is a point in such a discussion. Note also that the answers to these questions are subjective as *someone*, not necessarily a person, but an organization, or a government must answer them in order to give security a meaning. Consequently, what security means for one person might be radically different to what it means for another. In this regard, Anderson offers in [5] the following example: "*To a corporation, it [security] might mean the ability to monitor all employees' email and web browsing; to the employees, it might mean being able to use email and the web without*

being monitored".

What it is being protected is called an *asset*. Assets are not necessarily *things*: they can be people, infrastructure, etc. They do not even need to be physical: they can be intangible, as information is. As a matter of fact, information is in many cases the most important asset that needs to be protected from being revealed, from being tampered with or from being made unavailable, just to cite some examples. Nevertheless, even though information may be a very important asset—if not the most important one—, it seldom is the *only* asset. This should come as no surprise due to the fact that information does not exists on its own: it requires someone/something creating it, someone/something transmitting it, someone/something reading it and or making sense of it, all of which often involve some type of computer device. These computer devices often deserve equal or even more attention than the information itself, and therefore the terms *Information Security* and *Computer Security* are often used interchangeably.

Assets such as computers, or software may be very complex and have associated *vulnerabilities*, i.e. weaknesses that can be exploited in order to produce *unexpected consequences*. As a matter of fact, these assets are usually very complex, which makes determining their vulnerabilities a very cumbersome task that rarely yields an exhaustive list, which partially explains why complexity is antonym to security.

Moreover, it must be noted that no *asset* exists independent of a context consisting of the other entities that it interacts with. For this reason, reasoning about security also requires a set of assumptions about these relationships, which may—and in fact, often do—change in time. It results naturally, then, for *perfect* (or absolute) security to be unrealizable unless such a set of assumptions is empty. In other words, perfect security can only be hoped for when there are no assumptions, and this is extremely unlikely. This is a very important factor because it is often claimed in the literature that perfect security is *impossible*. For instance—among many examples—Schneier eloquently states that "*There's no such thing as absolute security. It's human nature to wish there were, and it's human nature to give in to wishful thinking. But most of us know, at least intuitively, that perfect, impregnable, completely foolproof security is a pipe dream, the stuff of fairy tales, like living happily ever after*" [52]. In this guise, this document aims to present a notion of realistic *practical* security, or security *in the real world*, as opposed to ideal, abstract, unattainable security.

After having addressed the first question (*What* is being protected?), it is now time to engage in the second one: "*From whom* is the asset being protected?". This is achieved by defining the *attacker model* consisting of determining who the *attacker* is as well as its capabilities. Even though computers are often used to perpetrate attacks, they are either a *tool* or a *target*, whilst the source of the attack, i.e. the *attacker*, or *adversary* is always human. Furthermore, the term *attacker* does not necessarily represent a concrete person or collective of persons, but rather a class of people. For instance, a given system may be secure against *attacks* exerted by external agents (non-employees), but insecure when at least one employee is involved in the attack; or the system could be secure against casual attacks by unskilled people, but insecure against well-funded skilled attackers. In this fashion, the concept of security is not absolute, but bound to a set of assumptions not only related to the environment—as argued above—, but also to the attacker and its capabilities.

The attacks mentioned above are directly related to the question "*Against what* is the asset being protected?" and to the *actions* mentioned in the security definition presented at the beginning of this section. In general, an attack consists of any action (or lack thereof) leading to adverse consequences taking place. These consequences are often referred to as *threats*, and are normally defined in terms of violations of one or more *security goals* such as confidentiality, integrity and availability, popularly known as CIA. In fact, these goals had already been implicitly introduced above when it was stated that it might be required for information to be protected against being revealed (*confidentiality*), tampered with (*integrity*) or made unavailable (*availability*). Besides CIA, other goals may be considered depending on the case. For instance, even though it might be fine for information to be made publicly available—so confidentiality preservation is clearly not the issue—, the identity of the parties creating it and/or reading it may need to be protected, which is known as *anonymity*. *Non-repudiation*, on the other hand, can be defined as ensuring that an entity cannot deny that it generated a piece of information after it did so. Conversely, *denial-of-receipt* consists of ensuring that an entity cannot deny receiving some piece of information. Finally, *accountability* is closely related to non-repudiation, but rather than being related to the generation of a piece of information, it is related to the execution of an action. In fact, it is defined in [57] as being able to *trace the actions of an entity uniquely to that entity*. In general, it must be noted that depending on these definitions some overlap between the aforementioned goals may exist. For instance, anonymity could be seen as a special case of confidentiality where the information that must be kept confidential is not what is being exchanged, but the identity of the involved parties. Similarly, non-repudiation and denial-of-receipt could be seen as special case of accountability where the traced action consists of the creation and the reception of data, respectively.

An understanding of the attacker model and a succinct answer to the "*Against what* is the asset being protected?" question lead to the formulation of the *Security Policy*, which is a statement of what is and what is not allowed, usually stated in terms of the achievement of a set of security goals [8]. Security policies may be expressed at different levels of abstraction, ranging from succinct descriptions in natural language, e.g. in English, all the way down to pure mathematical abstract formulations [8]. Note that *how* the security policy is achieved is independent of the policy itself, which is—in fact—the fourth fundamental question driving the argument of this section.

The concrete way how assets are protected is known as the *security system*, which is *the set of things put in place, or done, to prevent adverse consequences* [52]. If the security system prevents all possible attacks, then it would follow that the security issue was completely solved to the extent of the assumptions about the attacker and the environment. It must be remarked, though, that such an effective security system is infeasible in practice not only because it is very hard ensuring that it does what it is expected to do (*assurance*), but also because it would normally be too expensive to build. For the latter reason—as in real life no budget is unlimited—instead of hoping for infeasible perfect security systems, a process widely known as *risk assessment* is followed. This process consists of determining the *risks*, categorizing them, and designing and deploying the security system that mitigates the biggest risks. The notion of *risk* is different but associated to the notion of *threat*. In fact, a *risk* is a quantitative measure of the financial damage that the occurrence of a given threat would generate, computed as the product of the financial damage (*impact*) and the probability of occurrence of the threat.

Once the risks have been determined, the threats are classified according to their associated risk (from high to low), making it easier to decide which risks will be mitigated and which ones will be accepted. Normally the bigger risks are mitigated, which does not necessarily mean that they are eliminated but rather that either their probability of occurrence and/or their impact is lowered to an acceptable level. Note that it is quite possible to mitigate a risk by transferring it, which is exactly what buying insurance does. Conversely, the risk can be so small that deploying functionality in the security system intended to mitigate it could exceed its impact, in which case the sensible choice would be accepting it, or in other words, not doing anything to prevent it.

The *security system* is indeed the concrete way how risks are mitigated, implementing this way the security policy. For this purpose, such a system may take three (not mutually exclusive) approaches: *prevention*, *detection* and *recovery*. In addition, it can be decomposed in parts, referred to as *countermeasures* [52], or *controls* [57]. These countermeasures may be of technical or non-technical nature, where the latter normally consist of procedures or regulations. On the other hand, the technical (counter)measures are chiefly related to enforcing the answer to the question *Who should have access to what?* which is known as *access control*, that is often realized using cryptography.

2.2 Cryptographic Primitives

Access control is very often implemented using cryptography, which is undoubtedly a very important tool of information security. Nevertheless, it seldom is *the* complete solution to a security problem, or as eloquently put in [50]: "*if you think cryptography can solve your problem, then you dont understand your problem and you dont understand cryptography*". In this fashion, cryptography will be used in the following chapters to achieve particular security goals at particular places of the security system. For this reason, it is necessary to introduce the main cryptographic primitives along with their notation, which will be used in the rest of this document.

Hash functions are denoted by $\mathcal{H}(\cdot)$. They map arbitrarily long bit strings to fixed length bit strings (*digests*).

Message authentication codes (MACs) are denoted by $c \leftarrow \mathcal{H}(k, m)$. They take a fixed length key k and an arbitrarily long message m, producing a fixed length digest c. From a practical point of view, MACs can be seen as *keyed hash functions*, although strictly speaking they are not equivalent.

Symmetric encryption is denoted by $c \leftarrow \{p\}k$. It consists of obtaining a ciphertext c from plaintext p using key k. Sometimes key subscripts, e.g. k_{AB}, are used to indicate that the key is shared by parties A and B.

Asymmetric encryption is denoted by $c \leftarrow \{|\ p\ |\}K_A$. It consists of obtaining a ciphertext c from plaintext p using A's public key K_A. The corresponding decryption can only be executed by A using her private key K_A^{-1}, which is denoted by $p \leftarrow \{|\ c\ |\}K_A^{-1}$. When asymmetric cryptography is used for *signing*, the private key is used to generate the signature and the public key to verify it.

PKI corresponds to the infrastructure for distributing, verifying and managing the life-cycle of public keys.

Public key certificate (or simply *certificate*) is a public key plus some additional information signed by a Certification authority (CA). The public key certificate of A issued by CA is denoted by $[\![A]\!]CA \leftarrow \{|\ K_A,\ \text{info}\ |\}K_{CA}^{-1}$, where "info" corresponds to some information regarding A (its name, identifier, location, etc.), the certificate itself (period of validity, intended usage of the associated key pair $[K_A^{-1}, K_A]$, etc), and/or the CA (name, identifier, *url* of the Certificate Revocation List, etc.).

The rest of this document will consider cryptographic primitives to be perfect as in the widely known Dolev-Yao model [12]. More precisely, the following statements will be assumed to hold unless explicitly noted otherwise:

Perfect hashes Hash functions satisfy the following properties: *One-wayness*: it is unfeasible to find m given $h \leftarrow \mathcal{H}(m)$, *Weak collision resistance*: given m and $h \leftarrow \mathcal{H}(m)$ it is unfeasible to find $m' \neq m$ such that $h = \mathcal{H}(m')$, and *Strong collision resistance*: it is unfeasible to find m and m' with $m \neq m$ such that $\mathcal{H}(m) = \mathcal{H}(m')$.

Perfect Message authentication codes MACs withstand *adaptive chosen message attacks*. This means that even after obtaining MACs for adaptively chosen messages $m_1, ..., m_t$, it is not possible to find a MAC for a new message m' without the key k.

Perfect encryption It is unfeasible to recover the plaintext from any ciphertext without the appropriate key.

No modification without key It is unfeasible to meaningfully alter the content of a ciphertext without the appropriate key. The ciphertext can be replaced or tampered with, but the decryption operation will fail, thus indicating that the ciphertext is invalid (i.e. malformed).

Perfect keys and random numbers Keys cannot be cracked or leaked, and random numbers are indeed so.

2.3 Access Control

Access control can be defined in terms of associating an entity with an identity (*identification*), proving such association (*authentication*), and determining what the entity is allowed to do (*authorization*). This model is tightly related to the concepts that have been presented throughout in Section 2.1 as it allows separating attackers from the legitimate users, and thus preventing the former from performing unwanted actions.

Before discussing the three components of access control presented above, it is important to note that up to this point references have been made to *entities* without going into much detail. For this reason, it is necessary to introduce a distinction between *subject* and *principal*. Based on [5], the former corresponds to a person or a group of people, while the latter may consist of any entity participating in a security system. A subject may be a principal, but a principal may also be a piece of equipment (such as a computer), a

process, a communication channel, etc. This is crucial in the context of eBanking—and in general, of remote authentication—because the customer (subject) is not usually authenticated by the bank itself, but rather her PC or a proxy such as a smart card acting on her behalf. In general, often some principals act on behalf of others as in the case of the customer's PC presented above. Clearly, this is a convenient and necessary abstraction, not to be overlooked, which is achieved by determining the level of *trustworthiness* of the principals. A *trustworthy* principal will not deviate from its expected behavior and will not collude with an attacker, while an *untrusted* principal might be compromised, deviating from its expected behavior and causing unwanted consequences—or using the terminology introduced above, breaking the security policy—. Naturally, the attacker model has to consider how trusted principals are, and to what extent the security system should be able to withstand compromised principals.

The rest of this Section shall be concerned with the three components of access control: *identification*, *authorization* and *authentication*. *Identification*, consists of making an identity claim/statement, or put another way, binding a principal to an identity. It must be noted that sometimes both identification and authentication are used to denote the conjunction of making an identity statement and proving it [8], [32], [55]. However, strictly speaking making the identity claim and generating a proof, i.e. *authenticating* are separate actions, reason for which they are addressed independently in this document.

Authorization, in turn, consists of determining which actions the principal is allowed to perform in the system (*privileges*). Note that authentication is a precondition to authorization, a relationship that motivates the concept of a *session*, during which a certain principal is allowed to do certain actions without asking it to re-authenticate. Naturally, if an impostor manages to impersonate the principal after it has authenticated, the security system would have been defeated regardless of having authenticated the principal successfully in the first place.

2.3.1 Authentication

Data authentication, also known as *data origin authentication*, consists of proving the binding between certain piece of information and a given entity. Similarly, *entity authentication* is concerned with ascertaining an association between a principal and an identity. Two participants are distinguished in entity authentication: The *prover*, and the *verifier*. The former is the entity proving the principal-identity association, while the latter is the entity responsible for verifying the correctness of the proof. Note that providing evidence that the entity being authenticated was active when authentication took place is a very important requirement of *entity authentication*.

Authentication is often considered as a security goal itself [37]. This follows from the fact that it is of crucial importance to security, which is adequately put by Garfinkel [20]: "*Without authentication, a computer system frequently has no basis for determining if access should be granted or not. Even capability-based systems that provide access without authentication need to have some kind of system for deciding who gets the capabilities. What's more, practically every modern computer user needs to authenticate and re-authenticate themselves multiple times throughout the day. As the current state of authentication systems is generally deplored and ridiculed, any advances in this field should have a huge social benefit*".

When considering authentication systems, four essential issues must be taken into account. First, even though it is possible to have very accurate and effective authentication systems, it is extremely hard to achieve absolute effectiveness due to both technical and non-technical factors. Second, if these systems are going to be used by humans it is of the utmost importance to consider how *usable* they are. Third, as the authentication system is one part of the security system, it is constrained by financial considerations, consequently, even though very effective/efficient authentication systems exist, they are not always used. Lastly, *impersonation attacks* in which a (rogue) prover attempts to demonstrate a false identity claim, must be considered. In general, impersonation can be achieved by *guessing*, *replaying* or *relaying* the information used during authentication [55].

On a more practical level, there are three fundamental and non-exclusive bases or *factors* for authenticating a principal: using something that *it knows*, something that *it has*, or something that *it is*. The first consists of using a secret such as a password—in the case of a human principal—, or a key in the case of a device such as a smart card (cf. Section *Smart Cards*, page 21). Note that these secrets have to be as hard to guess as possible in order to resist guessing attacks. This is why people are encouraged to use passwords that are hard to guess and why cryptographic keys have to be picked at random. The second authentication factor is something that the principal has, which is usually known as a *token*. Smart cards and hardware code generators such as presented in Figure 5.1 (page 38) are prime examples of security tokens. Clearly, it is fundamental for these tokens to be hard to duplicate, i.e. clone, in order to be useful in authentication contexts. The third authentication factor only applies to people, and it is concerned with *biometrics*. It fundamentally consists of using some feature that varies greatly between people, and therefore can be used to authenticate individuals. Biometrics can be used both to identify, as well as to authenticate people. However, it is argued in [52] that whilst the latter usage often yields very good results, the former does not nearly perform as well. This follows from the fact that *authentication* reduces to the problem of one-to-one matching, while *identification* reduces to the problem of one-to-many matching, which is more difficult and yields higher error rates in general.

There are fundamentally two ways to achieve data authentication, both using cryptographic techniques. The first one consists of calculating a MAC over the information using a symmetric key k shared between the entity ascertaining the authenticity of the information and the entity verifying it. The second way consists of using digital signatures (asymmetric cryptography) in such a way that the data is signed using the private key K^{-1} of the entity ascertaining its authenticity.

In order to authenticate a principal based on something that *it knows*, four families of mechanisms can be distinguished: *Static passwords*, *OTPs*, *Challenge-response* and *Zero-knowledge* [9], [37], [55]. *Static passwords* consist of short shared secrets between the prover and the verifier, which the former must reveal to the latter in order to be authenticated. Static passwords are the simplest authentication mechanism but at the same time are very weak, principally because they are usually easy to guess and/or forget, and because using them for authentication requires them to be revealed. To make things worse, if the password is sent in *plain* to the verifier it may be eavesdropped by any entity with access to the channel across which it is sent. In this fashion, impersonation can be rather easily accomplished when static passwords are used, especially if authentication is

carried out over a public channel. Nevertheless, observe that if the channel between the prover and the verifier is secure, i.e. ensures confidentiality, and the verifier is honest, then using static passwords can be simple and effective as an authentication mechanism.

One time passwords (OTPs) are—as indicated by their name—passwords that are used only once. OTP based authentication schemes resist *passive attacks* in which an attacker is limited to eavesdropping on the communication channel. However, it might not resist *active attacks* in which the attacker is able not only to listen to, but also to remove and inject messages into the channel. Such attacks are illustrated in Figure 2.1 where the attacker A poses as the verifier V to the prover P obtaining the OTP from her, which then uses to (successfully) impersonate P to V. Such an attack can be prevented by authenticating V to P, i.e. reversing their roles, prior to P's authentication to V. In such a case, *mutual authentication*, as opposed to *unilateral authentication*), would be achieved.

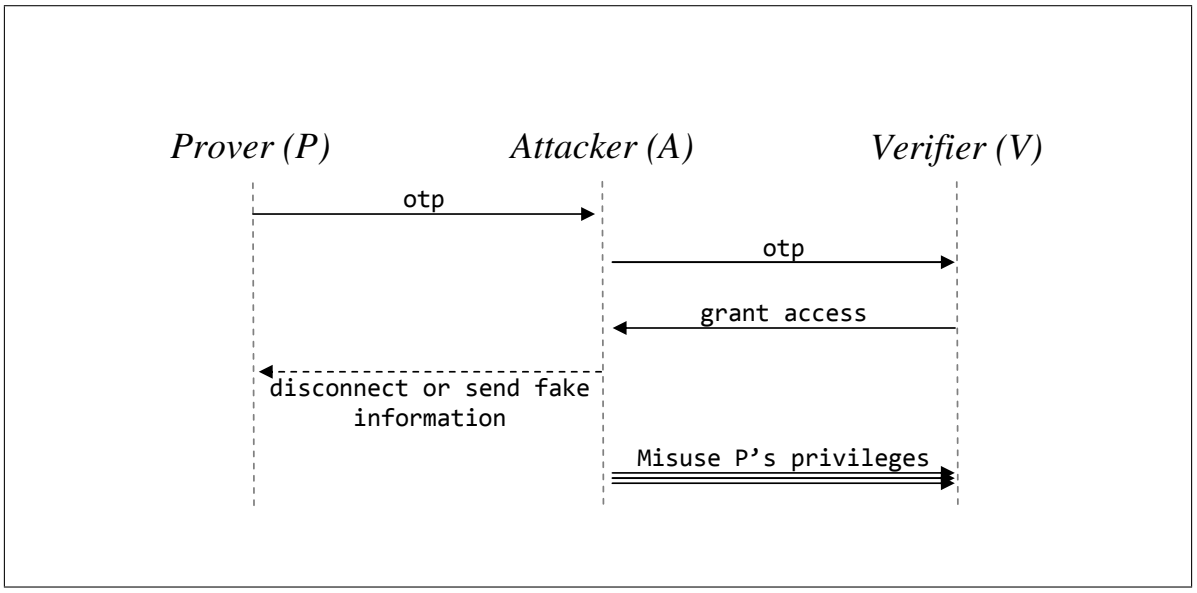


Figure 2.1: An active MITM attack against an OTP-based authentication scheme.

The third family of authentication mechanisms, *Challenge-response*, consists of the verifier presenting the prover with an unpredictable challenge every time that she attempts to authenticate. For every challenge there is an associated response that allows the prover to be authenticated if she is able to compute it and send it to the verifier. As a consequence of the unpredictability of the challenge, this family of mechanisms ensures that the entity being authenticated is active when authentication takes place, a requisite whose importance has already been remarked. Hence, passive attacks are not possible, and even though active attacks remain feasible, they need to be carried out in *real time*, which means that the time interval between the moment when the attacker captures the response and when it uses it needs to be very short for the prover to accept it.

Challenge-response mechanisms use cryptographic techniques in order to bind challenges to their corresponding responses, particularly, symmetric (resp. asymmetric) authentication and encryption, as presented in Figure 2.2. Part (a) of this figure shows a *Symmetric encryption* based protocol in which both parties, i.e. the prover and the verifier, share a symmetric key K_{PV} that the prover uses to generate the response r by encrypting an unpredictable challenge c sent by the verifier. In turn, the latter uses K_{PV}

to decrypt such a response in order to check that it matches c. Part (b) is based on *Symmetric authentication* wherein the shared key K_{PV} is used by the prover to compute the response r as a MAC of the challenge c sent by the verifier, who also computes such a MAC in order to check the response received from the verifier. Part (c) depicts a protocol based on *Asymmetric encryption* in which the verifier creates the challenge c by encrypting a random number n under the prover's private key K_P. Then, the prover is authenticated by being able to produce n as the response, which she obtains decrypting c using her private key K_P^{-1}. Finally, Part (d) illustrates an *Asymmetric authentication* based protocol in which the prover authenticates the verifier by requesting a signature of a random challenge c that the prover must produce using her private key K_P^{-1}.

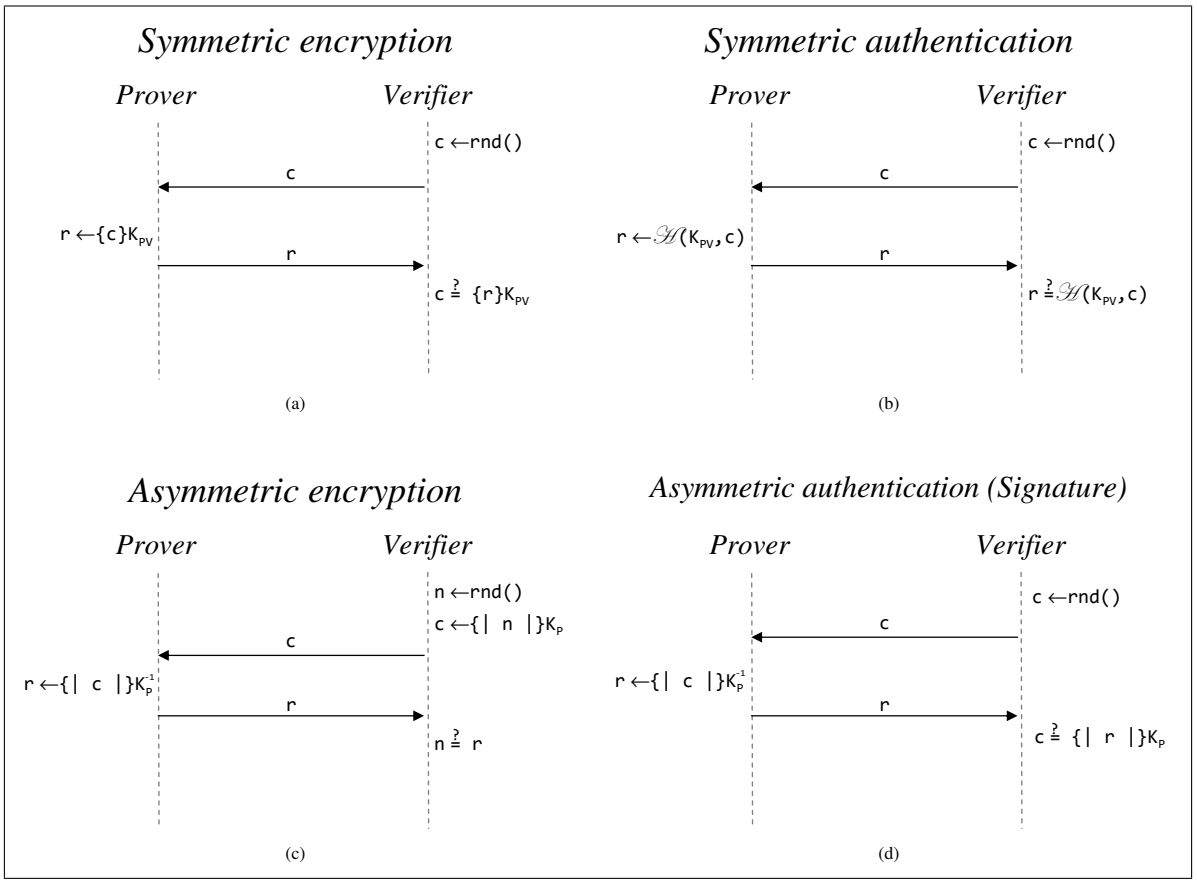


Figure 2.2: Challenge-response protocols [55].

Finally, *Zero-knowledge* is a family of mechanisms similar to *Challenge-response* where the prover is authenticated by producing evidence of the knowledge of a secret without revealing any information about the secret itself. *Zero knowledge* mechanisms differ from symmetric cryptography based challenge-response in the fact that there is not such a thing as a shared secret (key) between the prover and the verifier; and from asymmetric cryptography based *challenge-response* in the fact that they are more efficient. Regarding the attacks that have been considered so far, Zero-knowledge mechanisms yield the same results as *Challenge-response*, i.e. they withstand passive attacks but are vulnerable to on-line active attacks.

CHAPTER

THREE

Overview of eBanking

3.1 Scope

A typical eBanking scenario can be described as follows: *The customer accesses the eBanking portal over the Internet using a web browser. After she is successfully authenticated her account summary is presented on-screen along with several transaction options. Eventually, she might engage in a transaction by filling in and submitting the appropriate fields (generally a source account, a destination account and an amount).*

3.2 Actors

In the scenario presented above there are two main principals: the *user* (or *customer*) and the *bank*. The bank is represented by the bank *server*, which hosts the eBanking portal and services the requests send by the web *browsers* acting on behalf of the customers. In some cases, customers also need some additional device such as a hardware token and/or a personal device, for instance, a smart card or a mobile phone. All these principals are depicted in Figure 3.1 (page 17).

3.3 Goals

Banks and customers want to have access to financial services on the web, but at the same time they want to do so in a secure and cost-effective manner. As a result, both functionality and security are important considerations in eBanking. It can be said that the main goal of an eBanking system is *striking a balance between security, usability and cost-efficiency*. More precisely, the following goals will be considered for the purposes of this document:

Goal 3.1 (Server authentication) *The bank server must be authenticated by the customer. Achieving this goal is necessary because—as discussed in the previous chapter—before the customer releases sensitive information such as passwords, or in general, authentication credentials, she must made sure that she is* talking *to the bank server, and not to a third party* posing *as the bank server.*

Goal 3.2 (Customer authentication) *The customer must be authenticated by the bank server. In order to allow legitimate users to use the eBanking portal and prevent intruders from acting in their behalf, all principals attempting to access the eBanking portal must be successfully authenticated by the bank server.*

Goal 3.3 (Data authentication) *An attacker should not be able to perform a transaction if she is not a legitimate customer. This goal explicitly acknowledges that user authentication is necessary but not sufficient to ensure that transactions are only executed by legitimate customers. This follows from the fact that after the customer has been authenticated, an attacker may take over the session in certain scenarios. Naturally, then, it is not only necessary for the bank to authenticate the customer, but also the transactions that she requests.*

Goals 3.1, 3.2 and 3.3 will be jointly referred to as the '*Authentication goals*'.

Goal 3.4 (Privacy) *The information exchanged between a customer and the bank during the eBanking session should not be revealed to any third party. Note that in the case of transactions involving a destination account in some other bank it is unavoidable to reveal information to the latter bank or an intermediary financial network.*

Goal 3.5 (Usability) *The security system should be* usable. *The relationship between security and usability is not new, in fact, HCISec is an established research field addressing the impact of human factors in computer security*[1]*. Further, usability is identified as a crucial factor in the effectiveness of security systems [20, 22, 23, 62].*

Even though there is no standard "recipe" for evaluating the usability of security systems, three factors seem to be pervasive in the literature: *simplicity*, *convenience* and *(non-)obtrusiveness* [16]. *Simplicity* is related to the ease to use and the number of choices and rules that a user has to consider when using the security system. It is closely related to the sixth Kerckhoffs principle: "*The system must be easy to use, requiring neither mental strain nor the knowledge of a long series of rules to observe*" [28]. *Convenience* corresponds to the requirements that the system imposes on the user in terms information knowledge, e.g. secrets, possession of devices, location, etc. Finally, *(non-)obtrusiveness* consists of how much the security system interferes with the main task being performed by the user. This factor acknowledges that users have to go through the security system in order to achieve another more important functional goal [23, 62]. It should be remarked, though, that the popular belief that *users do not care about security* is rather misleading considering that most users are usually willing to engage in reasonably complex security procedures whenever they understand the rationale [2]. Particularly, in the case of eBanking, it has been argued that both security and functionality are important from the user's point of view, and hence, *(non-)obtrusiveness* can be regarded as less important than the two former factors.

Goal 3.6 (Cost) *The cost of the security system should be reasonable. As was mentioned in the previous chapter, security systems are constrained by economic factors.*

[1] See [2, 16, 20, 22, 23, 62]. Also, `http://www.gaudior.net/alma/biblio.html` contains a very comprehensive bibliography of HCISec publications.

3.4 Attacker Model

Attackers, also referred to as *intruders* or *adversaries*, attempt to subvert the security system, often by compromising one or more principals. Such compromises can be *passive* or *active*. The former occur when the attacker can only listen to, i.e. eavesdrop, the information handled by the principal (which may actually be a communication channel). On the other hand, the latter occur when the attacker can also can remove and inject messages. Naturally, an active attacker can also modify messages as follows: listen to message m, remove message m, and introduce the (modified) message m'.

Attackers can also be classified depending on their location. For the purposes of the problem at hand, two types of attackers will be considered: *local* and *remote*, as shown in Figure 3.1.

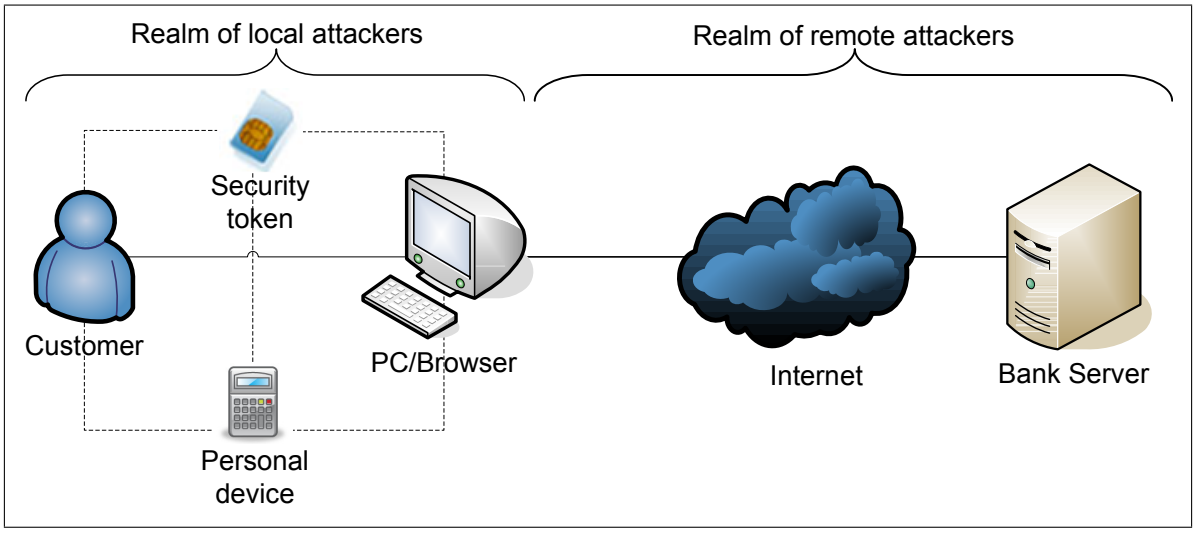


Figure 3.1: Actors and attackers in eBanking.

Local attackers operate in the physically accessible realm of the customer, and therefore might attempt to compromise any of the principals that the customer is directly interacting with, particularly its proxies. Very often local attackers (attempt to) compromise the browser by executing malicious software on the user's PC, or capture user input running key-loggers on the PC (software), or connecting them directly to the keyboard (hardware). On the other hand, *remote* attackers operate on principals that are outside the physical realm of the customer, such as the Internet. Note that the popularly known *Man in the middle (MITM) attack* in which the attacker mutually impersonates the customer and the server can be mounted either by a local or a remote attacker depending on where she operates.

CHAPTER

FOUR

Technology and Standards

This chapter presents the most relevant technologies and standards used in eBanking security, focusing on those used by the proposed solution described in Chapter 6 (page 53).

4.1 Secure Socket Layer and Transport Layer Security

SSL/TLS is the *de facto* standard for ensuring the confidentiality and integrity of communications over the Internet. It can also be used for authenticating the client and the server, but it is rarely used for the former purpose. SSL was developed by Netscape up to version 3.0, while TLS—which can be seen as its successor, i.e. v. 3.1—was developed by the Internet Engineering Task Force. The differences are rather technical but both protocol suites share the same underlying concepts/goals and are supported by most browsers, allowing them to be addressed generally as SSL/TLS.

SSL/TLS operates between the transport and application layers. It uses TCP (or some other reliable transport protocol) to transport messages and it is most often used by HTTP, a protocol combination known as HTTPS. HTTP is specified in RFC2616, HTTPS in RFC2818, and TLS in RFC4346. More details on the differences between SSL and TLS can be found in [47].

SSL/TLS is a protocol suite consisting of three protocols: *handshake*, used to establish a session; *record*, used to exchange data; and *alert*, used to signal errors and exceptional conditions. The end points are authenticated during the handshake protocol, and even though provisions for mutual authentication exist, the client is very seldom authenticated using SSL/TLS because it requires deploying extensive and complex PKIs.

The *handshake protocol* is particularly important for the purposes of this document due to the fact that it serves to authenticate the end entities to each other and to negotiate the keys that will be used to protect the integrity and confidentiality of the data exchanged between them. In this guise, the rest of this section shall describe the main messages of this protocol, which are also depicted in Figure 4.1.

1. `ClientHello`. The client sends the list of supported cipher suites along with a random number N_C.

2. `ServerHello`. The server chooses a cipher suite and sends a freshly generated random number N_S.

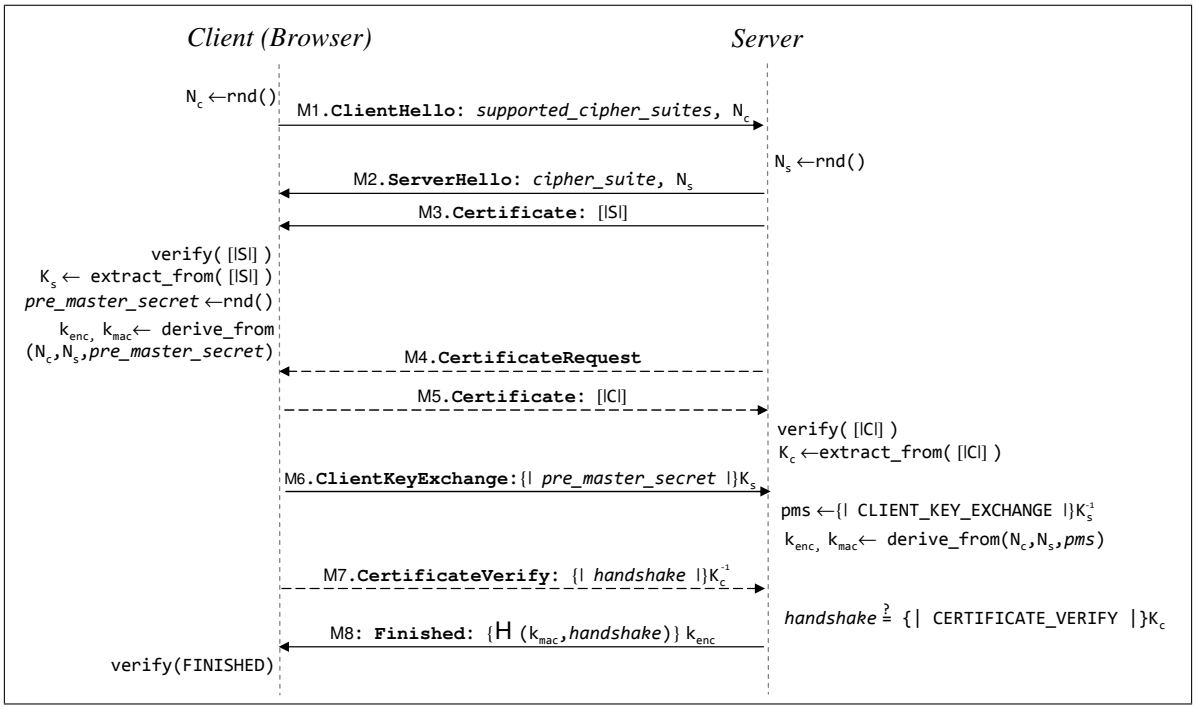


Figure 4.1: TLS handshake protocol.

3. `Certificate`. The server sends its X.509 certificate $[\![S]\!]$. The client checks it extracting the server's public key K_S.

4. `CertificateRequest`. Optional message sent by the server to indicate that the client must authenticate (using SSL/TLS).

5. `Certificate`. If the server requested client authentication by sending a `Certificate Request` message, the browser sends its X.509 certificate $[\![C]\!]$. Upon receiving this message, the server checks $[\![C]\!]$ extracting the client's public key K_C.

6. `ClientKeyExchange`. $\{|\ pre\_master\_secret\ |\}K_S$. The client generates a random string *pre_master_secret*, which is sent encrypted under the server's public key.

   At this point, both the client and the server calculate the session keys using N_C and N_S and *pre_master_secret*. Two keys are generated: k_{enc} for encryption and k_{mac} for integrity.

   Observe that only a legitimate server can retrieve *pre_master_secret* from `ClientKeyExchange` using its private key K_S^{-1}, thus effectively being authenticated by the client, as shown Figure 2.2c (page 13).

7. `CertificateVerify`. $\{|\ handshake\ |\}K_C^{-1}$. Corresponds to the signature generated by the client of all the messages that have been exchanged so far.

   The client is authenticated by the server whenever the signature is valid. This follows from the fact that generating a valid signature indicates the possession of the private key associated to $[\![C]\!]$, as illustrated in Figure 2.2d (page 13).

8. `Finished`. $\{\mathcal{H}(k_{mac}, handshake)\}k_{enc}$. This message consists of the MAC of all the messages previously exchanged between the browser and the server encrypted under k_{enc}.

Once the handshake protocol has been completed, the client and the server start exchanging information using the *record* protocol which ensures data confidentiality and integrity.

4.2 Smart Cards

A smart card, also known as *Integrated circuit card (ICC)*, is a plastic card with an embedded chip. These chips may range from simple, containing just a small amount of memory and some hard-coded mechanism for access control, to rather sophisticated, consisting of a very small computer with a CPU, a system bus, different types of memory: ROM for the operating system, RAM for the execution heap and stack, and EPROM for persistent information; and sometimes a cryptographic co-processor. The latter chips do actually run a dedicated operating system and are often capable of hosting and executing multiple applications [46].

Smart cards are particularly relevant for security applications because they offer tamper resistant storage of cryptographic keys, while allowing them to be used for cryptographic operations. Tamper resistance means that the card provides mechanisms that make it very difficult—but not impossible—for an attacker to retrieve the keys from the card. These mechanisms are not only logical, such as memory access control, or system bus scrambling, but also physical, e.g. protective mesh layers and fuses. A comprehensive survey of attacks on smart cards and their respective countermeasures can be found in [29] and Section 8.2 of [46].

The overwhelming majority of smart cards communicate with the outer world using an interface of 8 contacts (5 of which are actively used) defined in ISO/IEC 7816-2. However, in the past years it has been seen that many cards also incorporate a contactless interface as defined in the ISO/IEC 14443-2. Such a contactless interface not only permits faster communication speeds, but also prevents wear and tear as a consequence of not requiring physical contact between the reader and the card.

The biggest shortcoming of smart cards is that they lack direct input/output interfaces to the user [7], [22]. This is particularly relevant because an external device, i.e. a smart card reader, has to be trusted to display the information going into the card and to gather the user's PIN without disclosing it. There have been prototypes of smart cards with a small screen and a keypad [45], but due to their rather high power requirements and fragility they have never gone beyond the prototype stage[1].

[1]There is a recent news article at: `http://www.heise.de/newsticker/Visa-plant-Kreditkarte-mit-eingebauter-TAN-Generierung--/meldung/109529/from/rss09` (in German) where it is reported that a card with a chip, a battery, a display and a 12-button keypad is being considered by Visa. For more details about this card—which to the best knowledge of the author has not yet been deployed—see `http://www.emue.com/devices001.html`.

4.3 EMV Circuit Card Specifications for Payment Systems

The EMV Circuit Card Specifications for Payment Systems can be found in [14]. These specifications, hereinafter simply referred to as *"EMV"*, have been defined and are currently maintained by EMVCo, an association of the most important credit card franchises in the world: JCB International, MasterCard Worldwide and Visa, Inc.

EMV defines a framework for payment applications using smart cards that ultimately aims at replacing magnetic stripe cards. EMV does not define the payment applications themselves, which must be defined by each issuer bank instead. In fact, it just defines the basic building blocks in terms of data objects, functions, and security mechanisms. Then, it is up to each issuer bank to assemble these functions and deploy the security mechanisms within their payment applications.

The fact that EMV does not define concrete payment applications does not mean that it is a simple specification. On the contrary, it is a rather complex piece of documentation comprising more than 726 pages at its core. Such complexity and the lack of a high level architectural guide showing how to create secure EMV applications are its main criticism [49].

The next section will provide a high level overview of EMV in terms of its general goals and terminology. Section 4.3.2 (page 24) will describe the most relevant functions used in EMV transaction processing, and Section 4.3.3 (page 26) will cover the security mechanism defined by EMV.

4.3.1 General Goals and Terminology

With traditional magnetic stripe cards a payment transaction is performed when a *cardholder* produces his card to a merchant. In order for the merchant to accept the payment she has to communicate with the *Issuer Bank* in order to authenticate the customer and to verify that he has enough funds in his account. This is known as *on-line* transaction processing. Note that the *Issuer Bank* is often referred to simply as *the bank*, and the *cardholder* as the *customer*.

EMV compliant smart cards support *on-line* transaction processing, just as magnetic stripe cards. However, they go further by leveraging the trusted execution environment provided by the chip in order to allow some transactions to be processed *off-line*. This means that when a payment is performed using an EMV compliant smart card, it is possible for the card to *approve* or *decline* the transaction *without* requiring any communication with the bank. Off-line processing is appealing to merchants because it is faster than communicating with the bank, as well as cost effective, particularly, when communications from the merchant location are difficult and/or expensive.

Besides being capable of handling transactions off-line, the other goal of EMV consists of shifting the liability in case of fraud from the bank and the merchant to the customer. The rationale of such a shift is that an EMV card should be extremely difficult to clone, and there must be evidence of its participation in the transaction. Besides, the customer must be authenticated in order to accept the transaction (even if it is processed off-line), which means that in order to commit fraud an attacker must both impersonate

the customer, e.g. by stealing his PIN, and get hold of the card, which is supposed to be very hard. It is the opinion of the author (supported by [5, 49]) that even though EMV provides the basic security mechanisms to reduce fraud, such a liability shift lacks a solid justification in the general case considering that in some cases the customers' PINs can be obtained by an attacker using very little resources [49]. Besides, EMV is too lax regarding the *baseline* security mechanisms and it does not mandate a *complete secure transaction flow*, but just its building blocks so that they can be adapted to suit every bank. This not only leaves too many choices—thus creating unnecessary complexity—, but it results in the feasibility of insecure EMV compliant payment applications.

EMV related information stored in the smart card is organized in *files* as dictated by the ISO/IEC 7816-4 standard. These files contain *data objects* consisting of data structures in Tag - Length - Value (TLV) format. This is an important fact considering that *all* information exchanged between the card and the terminal is structured using such a format. The most important data objects for the purposes of this document are:

Application transaction counter (ATC) is a monotonically increasing counter incremented by the card on a per-transaction basis.

Application cryptograms (ACs) are the most important data objects used to convey information regarding the transaction processing result. As indicated by their name, these data objects are generated using a cryptographic transformation binding the transaction to the entity generating the cryptogram—usually the card—. In fact, there are four types of application cryptograms relevant for this document, three or which are generated by the card, and one by the bank. Particularly:

- **Transaction certificate (TC)** Indicates and provides evidence that the transaction was approved by the card.
- **Authorization request cryptogram (ARQC)** Used by the card to indicate to the terminal that the transaction should be processed on-line with the bank.
- **Application authentication cryptogram (AAC)** Indicates and provides evidence that the transaction was declined.
- **Authorization response cryptogram (ARPC)** It is generated by the Bank after receiving an ARQC generated by the card.

Data object lists are lists of data object identifiers. The main lists used in the rest of this section are:

- ***ddol*** *(Default Dynamic Data Authentication Data Object List)* is a list of data object identifiers provided by the terminal whose corresponding values must be signed by the card during *Dynamic data authentication* (page 29).
- ***pdol*** *(Processing Options Data Object List)* is a list of data objects stored in the card whose values must be provided by the terminal during the *Initiate application processing* function (page 24).
- ***cdol1*** **and** ***cdol2*** *(Card Risk Management Data Object Lists)*: are two lists of data objects stored in the card whose values must be provided by the terminal during the *Terminal action analysis* and *Completion* functions, respectively (page 25).

Issuer authentication data (IAD) corresponds to the information sent by the bank to the card for authentication purposes.

Unpredictable number (UN) is a pseudo-random number provided by the terminal to the card during the the *AC generation* (page 27) in order to prevent reply attacks.

4.3.2 Main EMV Functions

The functions presented in this section serve as building blocks for payment transactions. During the course of these transactions it must be determined whether these payments should be processed off-line by the card or on-line by the bank. Ultimately, it must be decided whether the transaction is approved or declined.

The most important functions for financial applications are defined in part 3 of Book III of [14]. A typical transaction is composed of the functions described below and shown as lightly shaded boxes in Figure 4.2. This figure also shows the relationships of these functions with the EMV security mechanism (darkly shadowed boxes) as described in the next section (page 26).

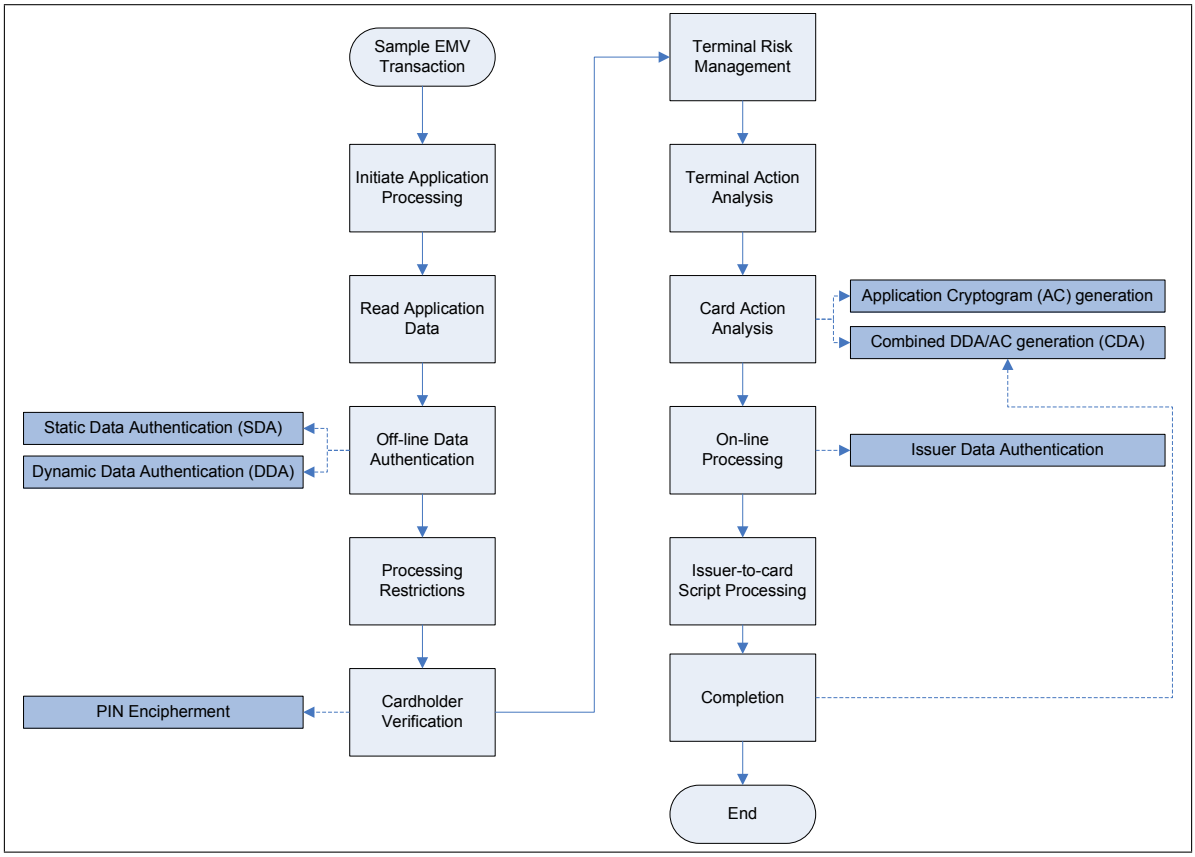


Figure 4.2: EMV: Main functions and security mechanisms.

Initiate application processing Besides informing the card that a new transaction will begin, it allows the card and the terminal to exchange information about the transaction and their particular capabilities. Ultimately, it determines whether the transaction can be handled by the card or not. It uses the `GET PROCESSING OPTIONS` command.

Read application data After the `GET PROCESSING OPTIONS` command has been sent to the card, the terminal receives a list of files that it must read form the card using the `READ RECORD` command.

Off-line data authentication The purpose of this (optional) step is authenticating the card to the terminal. Of the possible authentication mechanisms, one (or none) is selected based on the joint capabilities of the terminal and the card. Information about the card authentication capabilities is conveyed to the terminal during the *initiate application processing* step.

Processing restrictions This step is executed in order to make sure that the application in the terminal is compatible with the application in the card. In particular, the application version, usage restrictions and expiration date are checked.

Cardholder verification The purpose of this step is determining whether the person using the card is the legitimate cardholder. There are three cardholder verification methods: off-line PIN, on-line PIN and signature. Off-line PIN corresponds to letting the card authenticate the user using the `VERIFY` command. On-line PIN consists of sending the PIN to the bank for verification, and signature corresponds to obtaining a paper signature from the cardholder. The verification methods are not exclusive as any of the PIN verification methods may be combined with signature verification.

Terminal risk management This step is performed by the terminal in order to protect the system from fraud. There are several criteria used to determine whether it should be executed, particularly, whenever:

- The card indicates that it must be executed.
- The transaction amount is considered irregularly high.
- The transaction is randomly selected by the terminal.
- Several low value transactions are executed at the same terminal with the same card over a short period of time.
- A certain number of transactions have gone without executing it.

The outcome of this step is obtaining an assessment of how safe it is to process the transaction off-line as opposed to processing it on-line, i.e. communicating with the bank.

Terminal action analysis This step is always executed after *Terminal risk management* and all the off-line functions. The purpose is to allow the terminal to decide whether the transaction should be approved off-line, declined off-line, or processed on-line. In all three cases, the terminal issues a `GENERATE AC` command to the card. In the first case, it asks for a Transaction certificate (TC), in the second case for an Application authentication cryptogram (AAC), and in the third case for an Authorization request cryptogram (ARQC). Note that even if the terminal decides to deal with the transaction off-line, the card may overrule such a decision by responding with an ARQC or an AAC.

Card action analysis This step is triggered by the first `GENERATE AC` command received from the terminal. It complements the analysis performed in the previous step, and in a similar spirit, it is designed to prevent the system from fraud. The parameters and algorithms used to determine when a transaction should be approved off-line, declined off-line, or processed on-line are defined by each bank and not mandated by EMV. The results of this step are conveyed to the terminal by the cryptogram type returned by the card (c.f. *Application Cryptograms*, page 23).

On-line processing This step is similar to the traditional on-line processing executed with magnetic stripe cards. It is executed whenever the card returns an ARQC, which must be sent to the bank for approval. The card indicates in the response to the `GET PROCESSING OPTIONS` command whether the bank must be authenticated, in which case the information received in the response from the bank in the IAD must be sent to the card using the `EXTERNAL AUTHENTICATE` command.

Issuer-to-card Script processing When the transaction is processed on-line, it is possible for the bank to send a set of commands *(script)* to be issued to the card. The terminal is responsible for issuing all these commands to the card independent of their particular semantics.

Completion This step indicates to the card the transaction has finished. In case that the transaction was dealt off-line, the transaction is implicitly considered completed after the `GENERATE AC` command. In case that the transaction was processed on-line, completion is signaled by the terminal by issuing a second `GENERATE AC` command.

4.3.3 EMV Security Mechanisms

EMV defines several security mechanisms in Book II of [14] in order to achieve the following security goals:

Card authentication consists of making sure that the card being used in the transaction is genuine and legitimate. EMV defines three mechanisms used by the terminal to validate the authenticity of the card, namely: SDA, DDA, and CDA. *Static data authentication (SDA)* is the simplest one consisting of a static signature of some data objects which is generated by the bank and stored in the card. *Dynamic data authentication (DDA)* consist of a dynamic signature requested by the terminal and generated by the card over a dynamic set of data objects. Finally *Combined DDA/Application cryptogram generation (CDA)* is similar to DDA, but also includes a signature of all the data objects involved in the transaction. These mechanisms are described in more detail between pages 28 and 29.

Issuer authentication consists of the card making sure that a certain response comes from the bank. There is one mechanism for achieving this goal: *Issuer Data Authentication*, described on page 28.

Transaction authentication consists of providing evidence that the transaction was handled by the card. It is related to the generation of the Application cryptograms described on page 23.

PIN confidentiality consists of preventing the PIN from being eavesdropped when sent from the terminal to the card. It is realized by using a *PIN encipherment* mechanism explained in Section *Design and Architecture: PIN Encipherment* (page 56).

Session confidentiality consists of preventing the information exchanged between the card and the terminal from eavesdropping. It is realized by using *Secure Messaging* as described on page 28.

The cryptographic mechanisms intended to achieve these goals are described in sections 4.3.3.3 to 4.3.3.8, while sections 4.3.3.1 and 4.3.3.2 describe the symmetric and asymmetric, resp. key hierarchies required by these mechanisms.

4.3.3.1 Symmetric Key Hierarchy

There is a three-layer hierarchy of symmetric keys. At the top layer there is the Bank master key MK_B. This key is used to derive the ICC master keys (second layer) using the account number and the card identifier. There are 3 master keys per card: the application cryptogram master key K_{AC}, a encipherment master key K_{ENC}, and the MAC master key K_{MAC}. The third layer consist of the session keys, which are derived from the master keys using the ATC. Particularly, the application cryptogram session key k_{AC}, the encipherment session key k_{ENC}, and the MAC session key k_{MAC} are derived on a per-transaction basis from K_{AC}, K_{ENC}, and K_{MAC}, respectively.

4.3.3.2 PKI

There is a three-layer PKI consisting of a Certification authority (CA), the Bank and the ICC. The CA has a key pair $[K_{CA}^{-1}, K_{CA}]$, where the private key K_{CA}^{-1} is used to sign the Bank's public key certificate $[\![B]\!]CA$. In turn, the Bank's private key K_B^{-1} is used to sign the ICC public key certificates.

The ICC may have two certificates associated to two different key pairs: First, the ICC public key certificate $[\![C_{AUTH}]\!]B$ associated to a key pair $[K_{C_{AUTH}}^{-1}, K_{C_{AUTH}}]$. Second, an ICC PIN encipherment public key certificate $[\![C]\!]B$ associated to a key pair $[K_C^{-1}, K_C]$. The difference between these certificates and their associated key pairs lies on their purpose: The former are used by the card authentication mechanisms, while the latter are used by the PIN confidentiality mechanism.

4.3.3.3 AC Generation

An Application cryptogram (AC) consists of a MAC over certain data objects. The concrete data objects included in the MAC calculation are application-specific but the EMV specification suggests a minimum set including —among others— the ATC, the amount, the currency, the date and the UN.

In order to calculate the AC, the card uses k_{AC} (cf. *Symmetric Key Hierarchy*). Then, given the list l of n object identifiers to be included in the cryptogram computation, the AC is calculated as follows:

$AC \leftarrow \mathcal{H}(k_{AC}, e_{l_1}||e_{l_2}||...||e_{l_{n-1}}||e_{l_n}\}$,

Where e_{l_i} corresponds to the data *element* (i.e. object) identified by the object identifier in position i of list l.

4.3.3.4 Issuer Data Authentication

There are two mechanisms to authenticate data sent to the card by the bank when the transaction is processed on-line. Both mechanisms generate an ARPC data object, using k_{AC} and the ARQC generated by the card in response to the `GENERATE AC` command.

The first mechanism calculates the ARPC using encryption as follows:

$ARPC \leftarrow \{pad(ARC) \oplus ARQC\}k_{AC}$,

The second mechanism consists of calculating a MAC as follows:

$ARPC \leftarrow \mathcal{H}(k_{AC}, ARQC \;||\; CSU \;||\; prop\_auth\_data)\}$,

where:

Authorization response code (ARC) is used by the terminal or the bank to indicate the result of the transaction processing.

Card status update (CSU) is generated by the bank to indicate to the card whether it should accept or decline the transaction. Also, it may contain instructions as to how to update some of its internal data objects.

prop_auth_data conveys additional proprietary information regarding the transaction to the card.

4.3.3.5 Secure Messaging

Secure messaging is a mechanism providing confidentiality and/or integrity and origin authentication of the information exchanged between the bank and the card. The mechanism is used as described in the ISO/IEC 7816-4 standard. Particularly, in order to achieve integrity and origin authentication a MAC is computed over the exchanged data using k_{MAC}. Similarly, in order to achieve confidentiality, data —including the appended MAC, if present— is encrypted under k_{ENC}.

4.3.3.6 Static data authentication (SDA)

This mechanism is used to validate the integrity of some critical data objects stored in the card. If SDA is supported, the card stores a static list l of n data object identifiers, and a signature S corresponding to $\{|\ \mathcal{H}(e_{l_1}||e_{l_2}||...||e_{l_{n-1}}||e_{l_n})\ |\}K_B^{-1}$. Once the terminal reads $e_{l_1}, ..., e_{l_n}$ from the card, it can verify the signature by checking that the decryption of S under K_B yields $\mathcal{H}(e_{l_1}||e_{l_2}||...||e_{l_{n-1}}||e_{l_n})$ as computed by the terminal. K_B is obtained from $[\![B]\!]CA$, which is retrieved from the card and checked using K_{CA}, which resides in the terminal. This mechanism does not require any public key computation in the card, which avoids imposing computational requirements on the chip. On the other hand, it clearly does not preclude card skimming as the card can still be cloned.

4.3.3.7 Dynamic data authentication (DDA)

DDA is a mechanism consisting of having the card generating a signature over the data objects listed in the *ddol*. This signature S is requested by the terminal using the `INTERNAL AUTHENTICATE` command, and computed by the card as follows:

$\{|\ \mathcal{H}(e_{ddol_1}||e_{ddol_2}||...||e_{ddol_{n-1}}||e_{ddol_n})\ |\}K^{-1}_{C_{AUTH}}$, with

$n \leftarrow |ddol|$

Once the terminal receives S and $e_{ddol_1}, ..., e_{ddol_n}$, it can check that the decryption of S under $K_{C_{AUTH}}$ yields $\mathcal{H}(e_{ddol_1}||e_{ddol_2}||...||e_{ddol_{n-1}}||e_{ddol_n})$ computed at the terminal side. Both the bank and the card's certificates are retrieved from the card and checked using the CA's public key, i.e. K_{CA}, which is stored at the terminal.

Naturally, if the *ddol* refers only to static data objects, the value of S will always be the same. In practice, this list must include some dynamic data objects such as the ATC in order to have a different value for S being computed every time that DDA is executed.

4.3.3.8 Combined DDA/Application cryptogram generation (CDA)

CDA consists of combining DDA with the cryptogram generation executed at the end of the EMV transaction. In general, this mechanism consists of using the *pdol*, the *cdol*1, and the *cdol*2 (cf. *Data object lists*, page 23). Note that it is mandatory for the latter two lists to contain the identifier for the UN data object.

When the card receives the first `GENERATE AC` command, it computes H, the *Transaction data hash code* as follows:

$\mathcal{H}(e_{pdol_1}||e_{pdol_2}||...||e_{pdol_n}||e_{cdol1_1}||e_{cdol1_2}||...||e_{cdol1_m}||rdo\}$, with

$n \leftarrow |pdol|$, and $m \leftarrow |cdol1|$

Where rdo, i.e. the *returned data objects*, corresponds to the concatenation of the data objects included in the returned cryptogram. Similarly, in case of the second `GENERATE AC` command, H is calculated as follows:

$\mathcal{H}(e_{pdol_1}||e_{pdol_2}||...||e_{pdol_n}||e_{cdol1_1}||e_{cdol1_2}||...||e_{cdol1_m}||e_{cdol2_1}||e_{cdol2_2}||...||e_{cdol2_o}||rdo\}$, with

$o \leftarrow |cdol2|$

As in DDA, the signature S, which is stored in a data object called *Signed dynamic application data*, is computed as $S \leftarrow \{|\ H\ |\}K^{-1}_{C_{AUTH}}$. S is included in the cryptogram returned by the card, but it is naturally not included in rdo. When the terminal receives such a cryptogram, it extracts S and checks it along with the associated certificates in an analogous manner as in DDA.

Clearly, CDA is the most powerful authentication mechanism because it not only authenticates the card itself (entity authentication) but also the information sent by the card during the cryptogram generation (data authentication).

4.4 CAP and DPA

The Chip authentication program (CAP) is a specification developed by MasterCard described in [34] that heavily relies on the functions defined by EMV in order to provide mechanisms for customer authentication. Dynamic passcode authentication (DPA), on the other hand, is a similar specification developed by Visa. These specifications are both proprietary, and therefore it is not possible to discuss their details in this public document. Further, considering that they are rather similar, the rest of this section shall focus on CAP.

CAP requires the customer to have both an EMV compliant smart card and a Personal card reader (PCR), which is a pocket-sized card reader that provides a simple interface to the user consisting of a numeric keypad and a small text-only screen. Notably, while the smart card is personalized for every particular user, the PCR does not need any kind of personalization and can be used interchangeably with many cards. Figure 4.3 shows a few different types of PCRs.

Figure 4.3: Personal card readers (PCRs).

CAP can be used in several scenarios involving remote customer authentication such as eBanking and shopping on-line or over-the-phone. From the customer perspective it offers a coherent user experience as the procedure is always the same. Particularly, in order to use CAP she has to execute the following steps:

1. Insert the card into the PCR.
2. Select the mode (described in the next section).
3. Enter her PIN in the PCR.
4. Enter the additional information in the PCR (depending on the mode). The PCR sends the PIN and the additional information to the card obtaining an Application cryptogram (AC) in return. Then, the PCR uses the AC received from the card to generate the *CAP token*, which is the numeric code that is shown to the customer.
5. Provide the code shown by the PCR to the prover, usually the bank.

The order in which steps 3 and 4 are executed is irrelevant and not mandated by the CAP specification. In turn, step 4 consist of a simplified EMV transaction as described in the previous section. Particularly, the following functions are executed by the PCR in order to generate the CAP token:

Application selection Selecting the EMV application that shall be used for processing the transaction in the card.

Initiate application processing Allows the terminal and the card to exchange some information required for the transaction to take place.

Read application data Is used by the PCR to retrieve some required data objects from the card.

Cardholder verification Consists of authenticating the customer using her PIN.

Card action analysis Consists of requesting the Application cryptogram (AC) that will be used for the CAP token computation.

Transaction completion Indicates to the card that the transaction has been terminated. Particularly, it consists of sending a `GENERATE AC` command in some cases as indicated in the previous section.

The bank can calculate the CAP token following the same algorithm that is executed in the card. Particularly, the AC is generated using k_{AC}, which is a (derived) shared key between the card and the bank. Therefore, if the AC received from the card via the CAP token matches the same value calculated by the bank, then evidence of card participation in the transaction is provided. Further, due to the fact that in order to generate an AC the customer must have been authenticated by the card, some evidence of customer participation in the transaction is also provided.

4.4.1 CAP Modes

The CAP specification describes four modes of operation: *Challenge-response, Challenge-response + Additional data, Code* and *Sign*. In *Challenge-response* the card expects an unpredictable number that must be provided by the user and that is sent in the *Card action analysis* step[2]. The second mode, *Challenge-response + Additional data* , is a slight variation that besides the challenge also accepts a currency and an amount. The third mode, *Code*, generates one time passwords without requiring any input from the user, i.e. a challenge. Finally, the *Sign* mode uses the AC generated by the card to compute a MAC of some arbitrary data provided by the user into the PCR in step 4 as presented on page 31.

4.5 Mobile Phones

Mobile phones are truly pervasive devices that require little explanation. According to a survey by Ingenio Inc. conducted in 2007, 85% of individuals in the US own and use a mobile phone [26]. Similarly, according to a study by Research and Markets in the same year, mobile phones have already achieved market saturation in Europe [48]. Without a doubt, mobile phones have transcended from mere devices used solely to carry voice conversations to multi functional devices that enable their owners to interact with their surroundings. Particularly, they do not only offer long range voice and data capabilities, but also an assortment technologies such as cameras, Bluetooth and Near field communication (NFC), which will be succinctly described below.

Voice/data transfer technologies Mobile phones are no longer used to transmit only voice information. It is possible to download/upload information at broadband speeds using technologies such as GPRS, EDGE or UMTS. This enables people to produce and consume digital content from any place covered by the mobile phone network.

Cameras Virtually all contemporary mobile phones have at least one camera that can be used to take photographs or capture video. Putting aside these *traditional* uses, the camera can be used to capture text information which is then read using some optical character recognition software running on the phone. More practically, cameras can be used to capture information encoded into one or two-dimensional bar codes, which can be then decoded by the software running on the phone.

Bluetooth is a mid-range communication technology (about 10 meters) that allows exchanging information between the phone and another device. It was designed as a replacement for cables used to connect the phone to other devices such as computers, hand free headsets, other phones, etc. Using Bluetooth requires a configuration step in order to *pair* the devices, which is frequently touted as inconvenient.

[2]In the context of an EMV transaction, such an unpredictable number is generated by the terminal. As CAP builds on the same EMV primitive, it reuses the same data object, i.e. the *Unpredictable Number*, but *overloads* it so that it contains the challenge issued by the bank, and not a pseudo-random number generated by the PCR.

NFC is a short-range communication technology (about 10 centimeters) that allows exchanging information between the phone and another device without requiring any configuration. Due to the importance of this technology for the purposes of this document, it will be further explained in the following section.

4.5.1 Near field communication (NFC)

NFC is a very short range communication technology designed to work over a few centimeters. It allows contactless infrastructures based on ISO/IEC 14443, FeliCa and MIFARE to interoperate. It was invented by Sony and NXP Semiconductors in 2002, and standardized by ISO/IEC 18092 (ECMA 340). Nowadays it is being promoted by the NFC Forum, a non-profit association with more than 150 industry members[3].

One of the main goals of NFC is to be easy and intuitive to use by consumers. For this reason it does not require any pairing or configuration step. In fact, it is acknowledged that the simplicity and user experience makes NFC more appealing to users than other communication technologies such as Bluetooth [30]. Further, as communication only takes place over very short distances—shorter than 10 cm—users just have to make the devices touch in order to have them exchange information, which is easy and intuitive.

NFC is considered the second generation of Radio frequency identification (RFID). In fact, RFID clearly distinguishes between an active entity—the reader—, and a passive entity—the tag—. In NFC, entities are considered peers that can behave both actively or passively depending on the *mode*. There are basically three modes: *Peer-to-peer*, *Card emulation*, and *Reader mode*, which are described below along with their respective user cases:

Peer-to-peer Consists of information interchange between two NFC devices. The main use cases are exchanging contact cards, calendar items and photos. It can also be used to set up a Bluetooth connection by exchanging the pairing parameters. The Bluetooth connection can subsequently be used to exchange big volumes of data in a faster and more reliable manner.

Card emulation In this mode the NFC interface provides access to a secure chip (*secure element*) hosted by the phone. The prime use cases for this scenario are payment and ticketing [15], [42]. In the former, the chip hosts payment applications such as credit cards or electronic purses that are executed when the phone is touched to the Point of sale (POS). In the latter, the phone behaves as an access token, securely hosting the credentials (or tickets) which can be provisioned both using NFC, or some other mechanism such as text or multimedia messages sent to the phone using the mobile phone network.

Reader mode In this mode the phone behaves as a contactless reader capable of interacting with RFID tags or contactless smart cards.

[3]FeliCa (`http://www.sony.net/Products/felica/`) is a registered trademark of Sony Corporation, and MIFARE (`http://mifare.net/`) is a registered trademark of NXP Semiconductors. More information about the NFC Forum can be found at `www.nfc-forum.org/`.

The *secure element* mentioned in the *Card emulation* case corresponds to a chip similar to the ones embedded in smart cards. There are two fundamental approaches to the location of this chip: the SIM card or an independent chip [6]. In the first approach, the SIM card—which is a smart card itself—is leveraged to provide a secure storage and execution environment for the NFC enabled application. Naturally, in order to use this approach it is necessary to connect the SIM card to the NFC antenna. Such a connection can be achieved either by connecting the antenna directly to the SIM card using one of the unused contacts defined by ISO/IEC 7816-2, or by connecting the NFC antenna to a controller that is in turn connected to the SIM card using the *traditional* contact interface. The second approach consists of using an independent chip hooked directly to the NFC interface bypassing the SIM card altogether.

CHAPTER

FIVE

State of the Art

This chapter describes several eBanking security mechanisms. Some of these mechanisms are currently being used, while some others are rather innovative and not yet used in practice.

Section 5.1 provides some considerations regarding HTTPS, sections 5.2, 5.3 and 5.4 are concerned with the most prevalent remote customer authentication techniques used nowadays in the financial industry. In turn, sections 5.5 and 5.6 describe the most common transaction, i.e. data, authentication techniques. Sections 5.7 and 5.8 address some new and innovative authentication mechanisms originating both from academia and industry. Even though most of these mechanisms are promising and interesting, it must be remarked that to the best knowledge of the author, none of them is known to be currently being used at this time (August 11, 2008).

5.1 Bank Server Authentication

SSL/TLS as presented on page 19, Section 4.1 is the underlying mechanism used for authenticating bank servers and ensuring the integrity and confidentiality of the information exchanged between the browser and the server. These goals are achieved *provided that* the server certificate $[\![S]\!]$ is properly checked by the browser, and that the browser is not subverted. More specifically, checking a X.509 certificate in a SSL/TLS context consists of making sure that:

The certificate is issued by a trusted CA The certificate issuer, i.e. the one that has signed it, must be either trusted by the browser or have a certification path to a trusted CA.

The certificate's integrity is preserved The CA signature of the certificate must be valid.

The certificate is within its period of validity X.509 certificates are issued with a certain period of validity, i.e. not-before and not-after dates. A certificate should not be accepted if it is outside its period of validity.

The certificate has not been revoked When a private key compromise occurs, the corresponding certificate is revoked in order to preclude the compromised key being misused. A browser should not accept revoked certificates.

The certificate usage type is appropriate The browser should only accept certificates issued to be used for "*Web Server Authentication*"

The certificate matches the domain The certificate must be issued to the server's domain. For instance, if the *url* of the visited page is `https://login1.bank.com/login.jsp`, a certificate issued to "`bank.com`" should be accepted, but a certificate issued to "`mybank.com`" should not be accepted.

Usually, such checks are done automatically by the browser, but there are circumstances when the user is asked whether the connection should be established or not. For instance, when the browser is faced with a certificate issued by an unknown CA or when the domain does not match the certificate subject, the user is asked what should be done. The problem with this approach is that many users do not understand what they are actually being asked because the jargon used is too technical and most average users do not understand certificates [11], [18], [22]. Further, besides being hard to understand, these warnings are obtrusive, leading many users to dismiss them quickly without even reading them in order to get on with their primary task, a phenomenon called response bias in [60]. The result is that SSL/TLS sessions may be established with sites using questionable certificates.

Moreover, SSL/TLS does not offer any protection against establishing a connection with an invalid site as a result of using a misspelled address (*look-alike* attack). For instance, if the user misspells the bank's *url* (or follows a malicious link), she might establish a connection with "`www.mybanl.com`" instead of "`www.mybank.com`". The site may have a legitimate certificate for "`www.mybanl.com`" and unless the user carefully checks the address bar, a perfectly valid SSL session will be established with the impostor site. All the browser indicators of a secure connection such as the padlock icon will show up, and all succeeding messages will be revealed to the impostor site.

There have been some initiatives for using visual cues such as coloring the address bar in order to indicate the trustworthiness of Internet sites. Sadly, most of these indicators can be spoofed or overridden by carefully designed sites. What is more, research shows that these cues are not very effective, often being ignored because they are out of the user's periphery of focus, which is the webpage itself [11]. Shockingly, it has also been found that users rarely decide whether a site is legitimate or not using visual cues and/or technical information such as the server certificate. Rather, most users seem to base such a decision on how appealing and professional the site looks [11], [17]. This situation is even worse when a local active attacker is considered because in this scenario a compromised browser may take the user to a rogue site while at the same time showing a familiar *url* in the address bar. Consequently, when servers are authenticated using SSL/TLS, Goal 3.1 (*Server authentication*) is achieved under passive attackers, but it can be circumvented by active attackers, both local and remote.

5.2 Static Passwords

Despite being the weakest type of authentication, static passwords are still widely used for customer authentication in eBanking, particularly in countries outside Europe. Usually, after the browser has established a SSL/TLS connection to the server, the user is asked

for a *user-ID* and a *password* which must be provided using an HTML form. Once the server checks that the provided credentials are valid, the user is granted access to the eBanking portal and shown her account summary and transaction options. Afterwards, she can execute a transaction just by selecting the appropriate option and filling the necessary fields. Usually, no further authentication takes place unless a (server defined) time-out occurs.

Static passwords are touted as *convenient*, which is based on the fact that users just have to memorize their passwords and keep them secret. It is the opinion of the author that such a conception is a fallacy because it neglects the fact that for passwords to work as authentication tokens they must be kept secret *and* be hard to guess. As a matter of fact, using *password* as a password is quite simple, but prone to dictionary attacks and thus useless. On the other hand, passwords such as "y@kWa*${2t19d%" which are almost impossible to guess are also almost impossible to remember, thus making them less than convenient to use. Further, with a bit of effort people might learn a good quality password, but they will most likely reuse it for different services. The problem is that whenever the password is compromised at one of these services, it is rendered useless for all others as well.

Fundamentally, users need to be absolutely sure about whom they reveal their passwords to because they become useless as authentication tokens as soon as they are disclosed to a third party. Unfortunately, this is not trivial and it is in fact possible for passwords to be inadvertently disclosed to third parties, especially when faced with active attackers. A prime example of this vulnerability are the so called *phishing attacks* consisting of enticing customers to provide their authentication credentials to rogue websites posing as the bank. These attacks usually work by sending email messages to users inviting them to follow links to rogue Internet sites that *look like* their eBanking portal, or installing malicious software on the PC that redirects users to impostor sites whenever a known bank address is typed. Attack sophistication can be quite high, sometimes even involving the compromise of DNS servers, a practice known as *pharming*. The end result is that good phishing sites can fool even IT-literate users [11]. Naturally, using static passwords is highly susceptible to phishing attacks as they are valid for very long periods of time, e.g. months. Still, it must be remarked that phishing is just a technique to capture passwords, just as using *key-loggers*. The real problem is that the password loses its value as soon as it is revealed, and it remains *useful* for a long period of time, which is exactly the motivation for using other authentication techniques such as presented in the next sections.

By using static passwords the authentication and privacy goals are achieved when faced with a remote passive attacker, but they are not achieved when faced with an active attacker. When faced with a passive local attacker, only server authentication is achieved, but user and data authentication as well as privacy are not achieved.

The usability of static passwords is quite high. The mechanism is rather simple as the procedure to use them is straightforward. Similarly, it is rather convenient considering that there are not special requirements or conditions that must be satisfied in order to use it. Besides, the user only has to remember the password, albeit choosing and remembering good quality passwords is problematic for most people [5], [16].

5.3 One Time Passwords (OTPs)

Motivated by the long life of passwords and the ease with which they may be disclosed to third parties, many banks have enhanced their authentication systems by adding an extra authentication factor. Usually, this second factor consists of a hardware token that can generate passwords to be used a single time. Under this approach, users need to know a password (or a PIN) and be in possession of the token.

In a more practical level, many banks have made a move towards two-factor authentication as a consequence of phishing attacks and regulations such as the one issued by the US banking supervisory authority in October 2005 (and repeated in August 2006) suggesting that banks should employ two-factor authentication [36]. Besides, research has shown that hardware tokens increase the feeling of security perceived by users, which is rather important in eBanking [41]. Naturally, a security mechanism should not be used *just* because it arouses a *feeling of security* without actually effectively contributing in the security system, something that Schneier refers to as *security theater* in [52]. However, it must be acknowledged that if a certain device helps towards the achievement of certain security goal(s) and, at the same time is perceived by users as making things *more secure*, then it must be considered as advantageous.

There are fundamentally two approaches used by banks to generate one time passwords which will be addressed in the next two sections. The first employs a token that produces and shows a code that changes periodically, while the second consists of using a smart card and a PCR, as shown in Figure 4.3 (page 30).

5.3.1 Time Based OTPs

The foremost example of time based OTPs is the SecurID token. This token, as shown in Figure 5.1, generates a six digit numeric code every 30 to 60 seconds using a proprietary algorithm.[1]

Figure 5.1: A SecurID token.

User authentication with the SecurID works in a fashion very similar to static passwords, but it differs in the fact that besides the *user-ID* and a *password* the user must additionally fill in the code currently being displayed by the token, e.g. `032848` in Figure 5.1. Naturally, the clocks of the token and the bank server must be synchronized for the

[1] SecurID is a registered trademark of RSA Inc. For more information, see `http://www.rsa.com/node.aspx?id=1156`. The SecurID image has been borrowed from `http://www.pcbanker.com/images/sid700a.gif`.

scheme to work as the server needs to calculate the current code when it receives the information from the browser.

Time based OTPs are less usable and more expensive to implement than static passwords. The amount of information that the user must remember is the same, i.e. the password, but she has to type more information. The interaction with the token is straightforward as the user only has to read the code displayed on the screen. Nevertheless, as a consequence of the time dependency the user may either have to type a different code twice in case that it expires in transit to the server, or wait for a freshly generated code. Also, requiring a token slightly lowers the level of convenience because it is something that the user needs to have with her in order to engage in eBanking. Also, costs are bigger than using static passwords because tokens have additional costs associated with them.

In terms of security, the gain with respect to using static passwords consists of achieving the authentication goals in the presence of local passive attackers and making active attacks harder. If an intruder (either local or remote) manages to capture the password and the *current* code, she must use it quickly (within one minute, at most). Therefore, the attack must be carried almost in real-time. Even though these attacks seem theoretically harder, there have been cases of them seen *in the wild*[2].

5.3.2 Counter Based OTPs

Counter based OTPs are generated by means of a smart card issued by the bank. It is a two-factor mechanism because the user is authenticated using something that she knows—the card's PIN— and something that she has—the card—. Requiring the card does not impact convenience much because people usually carry their bank cards with them most of the time. Besides, in Western Europe smart cards are already pervasive in order to be compliant with SEPA. However, requiring a PCR in order to interact with the card does actually hinder convenience.

The way how counter based OTPs work is very similar to how static passwords do, but instead of typing in a static password the user must provide a OTP obtained after inserting the card into the PCR and typing in her PIN.

Not surprisingly, this technique is closely related to CAP, mapping to the CAP *Code* mode presented in page 32, Section 4.4.1. It must be noted that this type of mechanism, i.e. based on bank issued smart cards, is particularly appealing to banks because they control the complete security platform, particularly, the keys stored in the card.

In a similar way as time based OTP, the authentication goals are achieved in the presence of passive attackers. Regarding active attacks, the level of security is slightly lower compared to time based OTP because passwords do not expire promptly so they can be abused until the legitimate client logs in again.

From a usability perspective, the mechanism is fairly complex because the interaction with the PCR might be a bit cumbersome and annoying for some customers. On the other hand, there is no long password to remember and the generated code does not quickly

[2]For instance, as reported in `http://www.theregister.co.uk/2006/07/13/two-factor_phishing_attack/`.

expire, thus allowing plenty of time for the user to type it in the PC. As mentioned above, the level of convenience is constrained by the PCR requirement. It must be noted, though, that there are PCRs with form factors similar to that of the SecurID that can be put in a key ring as shown at the bottom of the bottom of Figure 4.3 (page 30). Unfortunately, user input in such small PCRs can be a frustrating experience because they lack a numeric keypad. In terms of cost, it is comparable—if not slightly lower—than that of time based OTPs. Besides, it must be noted that using the PCR approach still needs a smart card, which is probably more secure than using SecurID.

5.4 Challenge-response

This mechanism consists of employing a challenge-response protocol as presented in Section 2.3 in order to authenticate the customer. Fundamentally, after the browser has connected to the server and created a SSL/TLS connection, the customer is asked for her user-ID and presented with a random *challenge*. Then she has to provide the appropriate *response*, which is calculated using a token such as a smart card.

Often, this mechanism is implemented leveraging CAP mode *Challenge-response* presented in Section 4.4.1. It is indeed possible to have the OTP generation functionality running on a traditional debit or credit bank card, but some banks issue a separate smart card exclusive for eBanking purposes. Even though there is not a big difference from a technical point of view, this decision hinders the system convenience because the user needs *another* card if she wants to engage in eBanking. It can be argued that this approach is less secure than using a single card because if the eBanking card is not used often, it might be stored in a place where it can be easily accessed by an attacker when not in use.

Active attacks are harder when using challenge-response than when using time based OTPs because in the former case the challenge is tightly related to the session. In fact, in the context of challenge-response there is a one-to-one correspondence between a challenge and a session, and hence a one-to-one correspondence between a session and a response, i.e. the OTP. On the other hand, in the context of time based OTPs, there may be a one-to-many relationship between a OTP and a set of sessions, whose cardinality depends on the longevity of the OTP. Consequently, for an active attacker to successfully break a challenge-response scheme, she has to mount a *on-line real-time* MITM attack in order to be able to use the response generated by the user on the *same* session during which it was generated. Additionally, using a challenge improves security because it contributes an additional source of entropy to the calculation of the OTP which is not present when only time is used [25].

Using challenge-response authentication is slightly more complicated for the user than using counter based OTPs because of the additional input, i.e. the challenge, that needs to be typed into the PCR. The convenience and cost, on the other hand, are the same.

Both smart card based techniques, i.e. *Counter based OTPs* (page 39) and *Challenge-response* (page 40), assume the PCR to be tamper-resistant. If an attacker manages to replace it or tamper with it, then she can get hold of the customer's PIN, which may allow her to impersonate the user if she gets hold of the bank card as well. In any case,

it must be clear that neither of the two-factor authentication mechanisms presented so far completely solves the authentication problem in eBanking. In truth, real-time active attacks may still manage to subvert the authentication system, as argued in [53].

5.5 Message Authentication Codes

Unlike the techniques presented in the previous two sections—which are concerned only with customer authentication—the techniques presented in this and the next sections also incorporate authentication of the transaction information. This way, they strive to achieve Goal 3.3 (Data authentication) not by relying on customer authentication but by explicitly authenticating the transaction data.

This technique described in this Section uses a bank issued smart card to generate a MAC of the transaction information. It operates in conjunction with some of the previously presented smart card based customer authentication mechanisms, for instance, counter based OTPs. In such a case, it often leverages CAP modes *Code* and *Sign*, as follows:

1. The browser establishes a SSL connection to the server.
2. The server sends a form to the browser with an *user-ID* and a *code* fields.
3. The user puts her card in the PCR.
4. The user types her PIN in the PCR.
5. The PCR sends the PIN to the card, obtaining a cryptogram in return. Using that cryptogram, the PCR generates a code that is displayed to the user in the PCR's display.
6. The user fills and submit the web form with her *user-ID*, e.g. her account or contract number, and the *code* shown by the PCR.
7. The server checks that the given code corresponds to the next expected code for that username. If so, the user is granted access to the eBanking portal.
8. The user performs a transaction by selecting the appropriate option and filling the necessary fields in the eBanking portal.
9. Upon receiving the transaction information, the bank sends a set of numeric inputs to the user through the browser.
10. The user selects the *Sign* function (in the PCR) and types her PIN in the PCR.
11. The user enters the inputs sent by the bank into the PCR
12. The PCR sends the PIN and the inputs to the card obtaining a cryptogram in return. Using that cryptogram, the PCR calculates a MAC that is converted to a numeric code between 6 and 8 characters long. The result of the conversion is displayed to the user in the PCR's display.

13. The user enters the code shown by the PCR in the web form, submitting it to the server.

14. The server checks the validity of the code. If it is valid, the transaction is accepted and executed.

The bank and customer authentication goals are achieved to the same extent as the underlying customer authentication mechanism, in this case, counter based OTPs. However, due to the fact that the transaction information is authenticated, Goal 3.3 (Data authentication) is achieved depending on the inputs received from the bank in step 9. For instance, if the customer issues a transaction such as "*transfer Fr. 100 to account number 123*", it is crucial for her to understand the semantics of these inputs: if she receives a single random looking input from the bank, say "54634547", this number is meaningless to her, and even though it may possibly correspond to a digest of the transaction data, she has no way to validate it. Consequently, an active attacker can mount a MITM attack replacing the original transaction for "*transfer €1000 to account number 666*" which will pass unnoticed by the user, even as she generates the MAC. Clearly then, in order to properly authenticate the transaction it is necessary for the customer to generate the MAC over *all* critical transaction information, i.e. currency, amount and destination account.

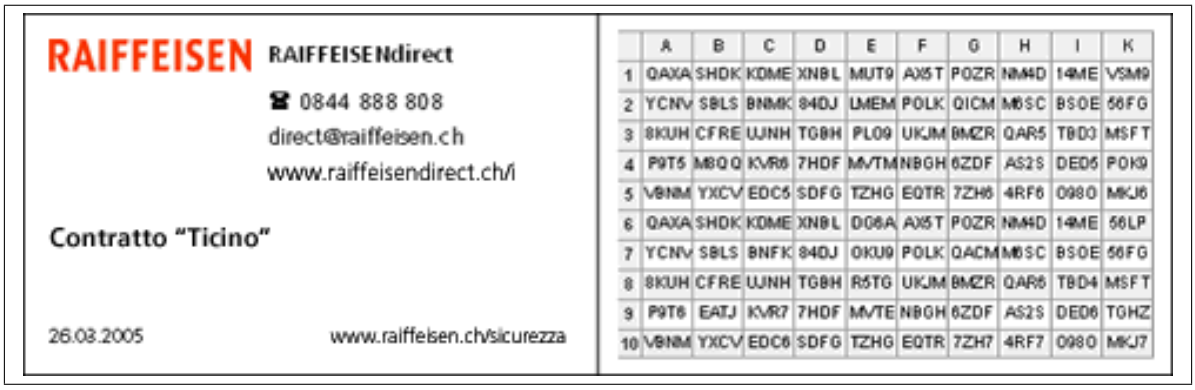
Inputting all the transaction information in the PCR imposes a heavy usability penalty. For this reason, when MACs are used usually just one input is provided by the server (the digest), and exceptionally—for high value transactions—the amount. Note that these simplifications are not worthless from a security point of view because they preclude active attackers from simply taking over the session after the customer has been authenticated. Nevertheless, they are by no means optimal because determined attackers might be able to compromise the system as outlined above. Additionally, note that customers have to interact with the PCR twice: once for logging in, and then again for authenticating the transaction, thus lowering the simplicity of the mechanism. In terms of convenience, it is the same as the previously presented smart card based customer authentication mechanisms.

5.6 Transaction Authentication Number (TAN) Lists

TAN lists, also known as "scratch lists", are used to authenticate transactions. They are a printed list of codes that the customer receives from the bank, as illustrated in Figure 5.2[3].In a similar way as the previous mechanism they must be employed in tandem with some customer authentication mechanism, which in practice is almost always static passwords.

Once the user has authenticated using her password and she has filled the appropriate transaction details, the server sends her a TAN identifier. For instance, if the user is using the TAN list depicted in Figure 5.2 (page 43) and the bank sends the identifier '`9-D`', she would have to send '`7HDF`' to the bank. Once the bank receives this value, it validates that the received TAN matches the expected one, accepting the transaction depending

[3]Figure from `http://www.raiffeisen.ch/`


RAIFFEISEN RAIFFEISENdirect

☎ 0844 888 808

direct@raiffeisen.ch

www.raiffeisendirect.ch/i

Contratto "Ticino"

26.08.2005 www.raiffeisen.ch/sicurezza

	A	B	C	D	E	F	G	H	I	K
1	QAXA	SHDK	KDME	XNBL	MUT9	AI5T	POZR	NM4D	14ME	V5M9
2	YCNV	SBLS	BNMK	84DJ	LMEM	POLK	QICM	M6SC	BSOE	56FG
3	8KUH	CFRE	UJNH	TGBH	PL09	UKJM	8MZR	QAR5	TBDJ	MSFT
4	P9T5	M8QQ	KVR6	7HDF	MVTM	N8GH	6ZDF	AS2S	DED6	POK9
5	VBNM	YXCV	EDC5	SDFG	TZHG	EQTR	7ZH6	4RF6	098O	M6J6
6	QAXA	SHDK	KDME	XNBL	DG8A	AI5T	POZR	NM4D	14ME	56LP
7	YCNV	SBLS	BNFK	84DJ	OKU9	POLK	QACM	M6SC	BSOE	56FG
8	8KUH	CFRE	UJNH	TGBH	R5TG	UKJM	8MZR	QAR6	TBD4	MSFT
9	P9T6	EATJ	KVR7	7HDF	MVTE	N8GH	6ZDF	AS2S	DED6	TGHZ
10	VBNM	YXCV	EDC6	SDFG	TZHG	EQTR	7ZH7	4RF7	098O	M6J7

Figure 5.2: Transaction authentication number (TAN) list.

on the result of such a validation. Naturally, once all the codes in the list are scratched, a new list has to be sent to the customer.

Arguably, using TAN lists can be considered a two factor authentication mechanism based on the list itself (what you have) and the log-in password (what you know). However, these lists can hardly be considered a token as they can be copied very easily.

From a security point of view, using TANs is weaker than using MACs. In the first place, customers are authenticated using passwords with the inherent limitations that have already been presented. Focusing on data authentication, TANs are not related to the transaction information, so the customer can be tricked by an attacker into revealing the appropriate TAN code, which can then be used for nefarious purposes. This means that active attacks might succeed even when TAN lists are used for data authentication. On the other hand, this mechanism is rather simple considering that users just need to remember their password and type an appropriate TAN when asked for it. It is also quite convenient because despite requiring the user to have the TAN list handy when engaging in eBanking, such a (paper) list is very easy to carry in a wallet or purse. Also, as no hardware tokens are required, the costs are smaller than with the previously presented authentication mechanisms.

5.6.1 Mobile TANs (mTANs)

mTANs, also known as SMS authentication codes, are a variation of the scheme presented above that replaces the paper TAN list with an authentication code (mTAN) that is sent to the customer's registered mobile phone number via SMS whenever a transaction is received by the bank. In fact, the bank does not only send the mTAN but also the transaction amount and sometimes the destination account. This way, the customer can verify that the transaction information that she entered in the PC is actually what the bank received and is being confirmed via SMS. If the information is correct, the customer inputs the mTAN in the browser, sending it to the bank for verification.

User authentication is achieved weakly because it relies on user passwords. Further, privacy is heavily impaired by the usage of SMSs, which in the best case reveal the existence of the transaction just to the network operator. In the worst case, i.e. a roaming user, not only the existence of the transaction but also its amount may be revealed to all the operators which the SMS is routed through. Still, this mechanism achieves data

authentication when faced with any attacker provided that an attacker is not able to operate in the mobile phone network, that the SMS contains the transaction information (destination account, currency and amount) and that the customer carefully checks the information received along with the mTAN.

Interestingly, using mTANs is quite convenient considering that users only require a mobile phone capable of receiving SMSs, which virtually all mobile phones are able to do. As it was mentioned in Section 4.5 these devices are truly pervasive in society and can hardly be considered to hinder convenience. Actually, only the verification of the information received in the SMS can be considered to be somewhat difficult in terms of usability. This follows from the fact that such information is presented as unformatted text that can be difficult to parse, read and verify by the customer. Additionally, sending SMSs carries pecuniary obligations that must be considered when assessing the cost of the system.

This mechanism is remarkably important for the purposes of this project because it introduces a secondary trusted channel of communication with the bank, namely, the mobile phone network. Even though such a network is not impossible to compromise, it is acknowledged that it is much harder for an attacker to simultaneously compromise both communication channels [52], [59]. Using a secondary channel for authentication with the mobile phone as an end point instead of the PC acknowledges that the latter cannot be trusted. This follows naturally from the sheer complexity of the software running on it which has too many interfaces and vulnerabilities, making it very hard to produce any well founded assurances regarding its *trustworthiness*. In terms of the attacker model presented in 3.4 mounting local attacks (both active and passive) can be achieved rather easily by targeting the PC [36]. It seems that in order to achieve sound authentication it is essential to have a trusted mechanism to interact with the user [58]. This is a concept of crucial importance that can be traced to the "splitting trust" paradigm proposed by Balfanz and Felten in [7] consisting of having a small critical part of the application running on a small, trusted device, and the other (less critical) part running on a bigger, more powerful, but untrusted device. This is exactly what motivates the next two sections, which consider that the trusted part might either reside on the PC itself (Section 5.7) or elsewhere (Section 5.8)

5.7 Trusting the PC

The main approach intended to make the PC trusted is based on TPMs, which are low cost passive hardware modules incorporated into computers to serve as a root of trust [31]. The specifications of these modules are maintained by the Trusted Computing Group, an industry alliance with more than 200 members[4].

TPMs have a few *platform configuration registers* used to store integrity *measurements* of the system. Such measurements consist of a hash chain of the software stack executed by the system. Initially, the BIOS code is hashed and stored. Then, the OS loader code concatenated with the previous hash result is hashed and stored. Then, the OS kernel modules are hashed—again, concatenated with the previous hash result—, and so on.

[4] `https://www.trustedcomputinggroup.org/home`.

This way, the software stack is *measured* yielding a hash result that represents the state of the system. The TPM can generate a *configuration certificate* of such a state that an external verifier can use to determine if the machine is in a trusted state, a procedure which is usually known as *remote attestation*.

Also, TPMs can be used for *sealed storage*, which means that a certain state measurement is used as a key for sealing, i.e. encrypting, some information. This way, the information can only be decrypted when the machine is in the same state, which is deemed to be *trusted*.

The prototypical usage case of a TPM in eBanking corresponds to the bank server requesting a configuration certificate from the client's PC, whose generation is delegated to the TPM. The aim of such a petition is ensuring that the customer's PC can be trusted, which is achieved by verifying that the certificate describes a well known, or *trusted* state corresponding to a *safe* configuration. The fundamental problem with this approach is that PC's software stacks are too complex and heterogeneous. This makes the determination of the trusted states a daunting nearly impossible endeavor considering that there are many types and versions of BIOS, operating system kernels, drivers, browsers, browser plug-ins, etc. that must be accommodated.

5.7.1 Compartmented Security for Browsers

This mechanism is proposed by Gajek et al. in [19]. It proposes using TPMs combined with virtualization in order to delegate the authentication tasks to a trusted "wallet proxy". It works by having a secure kernel hosting two isolated compartments connected via a virtual network connection. These compartments contain a legacy operating system containing the browser, and the wallet proxy, respectively.

The wallet proxy manages the user's credentials, i.e. passwords, keeping them using *sealed storage*, and is responsible for authenticating remote sites as well as the user. Naturally, all network traffic has to go through the wallet so it can authenticate the user when necessary. In order to accomplish this, the wallet intercepts all HTML code, blocking forms where the user could potentially disclose her credentials and prompting a profile creation page, where the credentials are typed so that the wallet can store them and use them afterwards.

Even though this is an interesting proposal from a theoretical point of view, its usability is extremely low because it requires TPMs, virtualization and secure kernels, neither of which are readily available to lay users.

5.7.2 Bump in the Ether

This mechanism is presented in [35]. It uses both a TPM enabled PC and a mobile device (phone or PDA) in order to create a "trusted tunnel" between the keyboard and a given application (for instance, the browser) in order to secure user input, particularly, passwords.

In this mechanism the keyboard is *not* directly connected to the PC, but rather, to the mobile device, e.g. using infrared, which is in turn connected to the PC, e.g.

using Bluetooth. The mobile device is capable of verifying the configuration certificate generated by the PC's TPM. If the configuration is accepted, then the mobile device offers a trusted display to show the list of registered target applications. Once the user selects the application in the mobile device, the latter creates a tunnel to that application in such a way that no other application running on the PC can receive the user's keystrokes.

Observe that requiring the mobile device follows from the practical impossibility of realizing a secure display in the PC. Such device is assumed to be trusted considering that it stores sensitive cryptographic material. However, in order to overcome this requirement the authors suggest using mutual attestation which would require the mobile phone to have a TPM as well.

From a security perspective, this mechanism is only concerned with local passive attackers, so a remote attacker could still compromise the password. In fact, the only risk that it mitigates is that of a software key-logger, leaving the password open to other types of attacks, such as phishing. Furthermore, an active local attacker able to hack into the kernel might be able to subvert the system, so it requires a secure kernel, which is not widely available.

The convenience of the system is very low because it requires a secure kernel and a special keyboard that can be connected to the mobile device. Further, the bridge between the keyboard and the PC through the mobile device is rather unnatural. It also appears that requiring both a TPM and a trusted mobile device is excessive. Further, some features such as the randomization of the application list order in the phone seem to have been introduced just to make the user's life more miserable.

5.8 Looking for Trust Outside the PC

5.8.1 Using Connected Smart Card Readers

A smart card can be used to authenticate the user, to secure the connection between the server and the PC, and/or to sign sensitive information. The user can be authenticated using challenge-response mechanisms such as presented in Section 5.4. The connection between the server and the PC can be protected by issuing the user with a public key certificate and having the corresponding private key stored in the smart card. Such a key can be used to establish a mutually authenticated SSL session between the browser and the bank server as presented in Section 4.1. Finally, signatures can be achieved using a private signing key stored in the card, which is very useful in order to achieve Goal 3.3 (Data Authentication) and attain non-repudiation.

Even though all this can be attained using smart cards, very few PCs incorporate smart card readers, making it necessary to resort to external readers. This requirement, in turn, generates criticism based on the inconvenience of carrying, connecting and configuring such a reader. All these criticism is addressed by state of the art smart card based solutions such as the Internet Smart Card and mIDentity, both of which have the same size as a USB memory and do not require any installation[5]. However, there still

[5]Information about Giesecke & Devrient's Internet Smart Card can be found at `http://www.gi-de.com/portal/page?_pageid=42,54860\&_dad=portal\&_schema=PORTAL`, and about Kobil's mIDentity

remains the fundamental problem that smart cards lack direct input/output interfaces to the user. This means that if the smart card is connected to an untrusted PC any information sent to the card to be signed may be different to what the PC shows to the user for confirmation. Moreover, whenever user authentication is required prior to using the smart card for a sensitive operations and the PIN is entered using the PC, it becomes vulnerable to compromise.

5.8.2 Financial Transactional IC Card Reader

The Financial transactional IC card reader (FINREAD) is an open specification for tamper-resistant smart card readers with trusted display and keypad[6]. There is an eBanking authentication solution proposed by Hiltgen et al. in [25] based on FINREAD compliant readers that achieves most of the security goals presented in Section 3.3. However this solution is not quite realizable due to the lack of mobility of the readers, their high costs, and the complex PKI that it requires.

5.8.3 AXSionics Internetpassport

This mechanism uses of a credit card size token with a fingerprint reader, an optical sensor and a small screen developed by AXSionics[7]). Such a token is used to read and decrypt the transaction information displayed in the customer's PC as a flickering code sent by the bank. The token reads the code from the computer screen using its optical sensor and decrypts it after the user has authenticated using the fingerprint reader. Then, the screen on the token shows the data to be authenticated, i.e. the transaction information, and the token generates a MAC that the user sends to the server thereby achieving data authentication.

Undoubtedly, this mechanism is innovative and very interesting because it manages to authenticate transaction data using the untrusted PC to show the (encrypted) information for the token to decode. However, it is unsettling that little is known about how the visual code is generated, i.e. could a local attacker mount a MITM attack replacing the transaction information included in the visual code? Also, the device seems rather thick and fragile, and due to hardware requirements it is certainly more costly that previously discussed tokens. In terms of usability the interaction using the visual sensor relieves the user from entering the transaction information manually, placing it on top of other approaches like CAP based MACs as presented in Section 5.5. It is not clear, however, whether the interaction between the token and the screen is as straightforward and seamless as advertised, or whether it involves a fair amount of effort to have the code being successfully read by the scanner, which would, undoubtedly, decrease its usability immensely.

---

at `http://www.kobil.de/index.php?id=49\&type=7\&L=1%22`

[6] Available at `http://www.cen.eu/cenorm/sectors/sectors/isss/cwa/finread.asp`.

[7] `http://www.axsportal.com/tce/frame/main/441.htm`.

5.8.4 The Zone Trusted Information Channel

The ZTIC is an interesting solution under active development at the IBM Zurich Research Laboratory [58]. It consists of a small tamper-resistant USB stick with a small display, two buttons and an embedded smart card in SIM form factor, or optionally, a smart card reader. Once the ZTIC is connected to the PC, it launches the user's default browser, runs a proxy service in the PC, and establishes a mutually authenticated SSL/TLS connection to the bank server. The user is authenticated using the private key stored in the smart card and her PIN, which she enters using the ZTIC's input buttons.

The proxy is a very simple component that relays all the traffic between the browser and the bank. The application running on the ZTIC handles the SSL/TLS connection scanning for sensitive operations such as log in, transactions, etc. When such operations are detected, the relevant information is displayed on the ZTIC's screen for explicit user approval using the *Ok* button before being sent to the server.

Regardless of the attacker type, goals 3.1 (Server Authentication) and 3.2 (Customer Authentication) are achieved using the ZTIC as a consequence of the mutually authenticated SSL/TLS connection between the server and the ZTIC. Data authentication (goal 3.3) is also achieved because the customer has to explicitly approve any transaction sent to the server, and such approval is associated to the calculation of a signature or a MAC over the transaction data, which is shown on the ZTIC's trusted display. On the other hand, privacy (Goal 3.4) is not achieved in the presence of a local attacker because sensitive information can be captured from the information displayed and/or entered into the browser.

Unlike the token presented in the previous Section, the ZTIC is rather inexpensive as a consequence of being a simple device with only a few components and minimal circuitry. Similarly, it achieves a good level of usability (Goal 3.5) as it works out-of-the-box with most consumer operating systems without requiring any installation or drivers and is reasonably small. On the other hand, it is *another* device that must be carried, and if the user credentials are stored in the embedded smart card, it cannot be used interchangeably unlike CAP compliant PCRs. Conversely, if credential storage is delegated to an external smart card, then ZTICs can be used interchangeably, albeit slightly increasing their form factor. Also, their small screen and the lack of a numeric key pad might be troublesome to some users as the screen may be hard to read and only two buttons are certainly inconvenient for PIN entry, although they are suitable for Ok/Cancel functionality.

In any case, due to the fact that the ZTIC has not been subject to usability tests with lay users, there is no hard evidence regarding its user acceptance [58]. In this regard, Chapter 12 of [16], a study about the usability of dedicated security devices, indicates that most users feel comfortable using advanced USB security tokens similar to the ZTIC while they would rather not use external smart card readers. Sadly, this study cannot be translated directly to the ZTIC because the USB tokens considered in the study lack a user interface, i.e. display and buttons, and are incapable of reading smart cards. Moreover, the card readers that it considers are rather bulky and without any user interface.

5.8.5 Mobile Phone Based Mechanisms

5.8.5.1 Phoolproof Phishing Prevention

This mechanism is described in [43]. It uses a mobile phone to aid in the establishment of the SSL/TLS connection between the user's PC and the bank server, requiring the browser to communicate with the phone using Bluetooth. It consists of two phases: *Account setup* and *Connection establishment*. Prior to executing account setup, a shared key k_{BM} between the bank and the mobile phone must have been established by some out-of-band means.

When executing account setup, the bank sends its MACed certificate $[\![B]\!], \mathcal{H}(k_{BM}, [\![B]\!])$ to the browser, which forwards it to the phone. The phone verifies the certificate and the MAC. Then, it creates a key pair (K_U, K_U^{-1}) and sends its authenticated public key to the bank $(K_U, \mathcal{H}(k_{BM}, K_U \mid\mid [\![B]\!]))$. Finally a bookmark to the *url* indicated by $[\![B]\!]$ is created in the phone.

A connection is established by using the secure bookmark in the mobile phone, which signals the browser in the PC to open the bank site. During the SSL/TLS handshake (cf. Section 4.1) the bank's certificate is forwarded to the phone and compared with the one received (and authenticated) during the account setup phase. The client is authenticated also by delegating the signature of the handshake messages to the mobile phone, which uses K_U^{-1} for this purpose. After the connection is established, the browser and the bank can communicate as usual, allowing the user to be authenticated by the bank using a static password.

Note that a passive attacker might be able to get hold of the user password, but would fail to establish a connection with the bank as it does not have the user's private key K_U^{-1}. On the other hand, the system fails to withstand active local attackers because it can rely on the user to establish the secure session and authenticate, and after that take over the session. This follows naturally from the fact that the phone does not take part in the communication after the SSL/TLS connection has been established. In other words, the PC is still an end-point of the SSL/TLS tunnel, which leaves the data stream open to manipulations by an attacker operating in a compromised PC. Additionally, the phone is assumed to keep the long term user's private key K_U^{-1} secure, which may be a rather big assumption. Consequently, Parno et al. propose to use a TPM in the mobile phone in order to prevent a private key compromise[8] [43].

In terms of usability, the convenience of the mechanism is restricted by the requirement for the Bluetooth connection. In fact, despite being pervasive in mobile phones, not all PCs are able to establish Bluetooth connections (especially desktop PCs). Also, it is somewhat complicated because Bluetooth does not excel by its user-friendliness. Further, as the phone itself *is* the authentication token, if the user changes it, she has to take care of key revocation and installation. This is a rather technical procedure that many users will most certainly try to avoid. Finally, this mechanism requires browsers to be modified in order to support the extensions required by the protocol. This carries a cost penalty in terms of development and testing effort compared to other mechanisms that do not

[8] There is a draft specification for these platforms available at `https://www.trustedcomputinggroup.org/specs/mobilephone/`, along with a security framework for credential management using that kind of platforms in [13].

involve modifications of the software running on the PC. Further, it might restrict the platforms, e.g. operating systems, browsers, etc. that can be used, consequently lowering its level of convenience.

5.8.5.2 Mobile Password Authentication

This protocol is presented in [33]. It proposes using a personal trusted device, i.e. a mobile phone, to enter the username and password which are then encrypted using the bank's public key. In this fashion, the password is protected from being disclosed to any third party even if the user's PC has been compromised. Additionally, it proposes a mechanism for ensuring transaction data authentication.

As the previous mechanism, the communication between the mobile device and the PC takes place using Bluetooth and assumes a trusted phone free of malicious software. It works as follows:

1. The customer's mobile phone is provisioned with the bank server's public key K_S.
2. The customer opens the browser which establishes a SSL/TLS connection with the bank server.
3. The bank server sends a random number N_S to the browser. The browser relays this number to the phone along with the bank's identifier.
4. The customer enters his user-ID u and his password p in the phone. The phone generates a session key k_{MS} using N_S and a freshly generated random number N_M. It then computes and sends the following message to the bank server (via the browser): $\{|\ N_M\ |\}K_S, \{\mathcal{H}(N_S), (u, p)\}k_{MS}$
5. Upon receiving this message, the server can derive k_{MS} by decrypting $\{|\ N_M\ |\}K_S$ using its private key. Then, using k_{MS} it can recover the user-ID and password in the second part of the message, using them to authenticate the user. Note that only the bank server and the mobile phone share k_{MS}.
6. Upon receiving some transaction information t that the user has sent via the browser, the bank sends a confirmation message to the phone via the browser: $\{t, N_{S_1}\}k_{MS}$, where N_{S_1} is a random number freshly generated by the bank server.
7. When the phone receives that message, it shows the transaction information to the user for explicit approval. After obtaining such approval, the phone sends a confirmation to the bank via the browser as follows: $\{\mathcal{H}((t, N_{S_1}))\}k_{MS}$.
8. When the bank receives the previous message, it checks the hash and accepts if it is valid.

This mechanism cannot be touted as two factor authentication considering that just by possessing the password it is possible to impersonate the user. Still, it is quite interesting because it accomplishes user authentication (Goal 3.2) under any type of attacker assuming a trusted phone, a properly provisioned server public key, and a properly selected and kept password. Furthermore, because of the transaction data authentication mechanism in the last three steps, it also achieves Goal 3.3 (Data authentication).

The usability of this mechanism is affected by the cumbersomeness of the Bluetooth connection establishment (pairing) and the limited availability of Bluetooth capable PCs. Further, simplicity is hindered by requiring the user to enter her user-ID and password in the mobile phone, which is not easy unless the phone has a QWERTY keyboard.

CHAPTER

SIX

EBanking Authentication with NFC Phones

This chapter presents the proposed authentication solution using NFC enabled mobile phones. Section 6.1 describes its design goals and architecture. Section 6.2 (page 57) comments on the prototype implementation. Section 6.3 (page 64) assesses the extent to which this implementation satisfies the goals previously presented. Lastly, Section 6.4 (page 69) points out some issues that could be improved in the implementation and propose further enhancements that would make the prototype both more usable and secure.

6.1 Design and Architecture

6.1.1 Goals

The goals *Server authentication, Customer authentication, Data authentication, Privacy, Usability* and *Cost* presented in Section *eBanking: Goals* (page 15) should be met at least to the same degree as with challenge-response solutions, cf. Section *State of the Art: Challenge-response* (page 40).

Additionally, given that the resulting prototype is being considered by a financial institution to be deployed in the short term, it is important for the solution to be realizable in at most 2 years. For the same reason, the solution must be compatible with CAP as currently being used by this financial institution. Also, the UI has to support several languages. Finally, it is desirable for customers to be able to download and install the application into the mobile phone themselves.

6.1.2 Principals

Dual interface smart *card* The smart card considered for this solution has both an ISO/IEC 7816 contact interface as well as an ISO/IEC 14443 contactless interface. It should support Java Card applets[1]. Further, it should have at least one selectable EMV 4.1 compliant (payment) applet to be used for CAP purposes. Such an applet will be referred to as the *CAP cardlet*, or simply, the *cardlet*.

[1] `http://java.sun.com/javacard/`.

The card itself might also be used for other (financially-related) purposes, e.g. as a debit or credit card. In fact, this situation is desirable in order not to require the customer to carry an additional card for eBanking purposes, as argued in Section *Challenge-response* (page 40).

The card's PIN is *known* by the card, the customer and the bank. Stringent procedures are assumed to be in place preventing the PIN from being disclosed or misused by bank personnel. The symmetric key hierarchy outlined in Section *EMV: Symmetric Key Hierarchy* (27) should be in place. Similarly, the card stores and is able to provide the bank's public key certificate, i.e. its Issuer's Public Key Certificate $[\![B]\!]$, and its own PIN encipherment certificate $[\![C]\!]$. Further, the private key associated to $[\![C]\!]$, i.e. K_C^{-1}, is securely stored in the card and cannot be used for signature generation. In fact, the sole purpose of this key is PIN decipherment.

NFC mobile *phone* Minimally, it has a numeric keypad, a small display with graphic capabilities and NFC interface, and supports executing Java Platform Micro Edition (ME) applications[2]. It runs a CAP application capable of interacting with the card, which will be referred to as the *CAP midlet*, or simply, the *midlet*. Such a midlet contains two static signed lists: the first with the public keys of the CAs allowed to issue the banks' as well as the cards' PIN encipherment certificates, i.e $[\![B]\!]$ and $[\![C]\!]$. In turn, the second list contains the supported CAP cardlet AIDs.

The phone does not store any long term secret, although it may temporarily store the customer's PIN at run time. Once the midlet has been installed in the phone, it is assumed that it is not possible for an attacker to compromise it. In fact, it is assumed that it is not possible for an attacker to compromise the phone, which means that any action leading to the disclosure of the PIN entered by the user, or a deviation from the expected behavior as presented in the protocol is not possible.

Customer is also referred to as the *account holder* or *user*. It is assumed that she exercises due care with her PIN in order to keep it secret.

Browser/PC is the device from which the user accesses the Internet eBanking site. No particular assumptions are made about this equipment.

Bank server is also referred to as *the server*. It corresponds to the computing equipment at the bank side that services the browsers' requests using SSL/TLS as presented in Section *Technologies and Standards: SSL/TLS* (page 19). It is assumed to be very unlikely to be compromised, both by outsiders and/or insiders. The mechanisms devised to ensure this condition fall outside the scope of this document. Finally, the challenges that it generates are assumed to be unpredictable.

Channel A (Card – Phone) It is a wireless (radio frequency) communication channel operating at 13.56 MHz as specified in ISO/IEC 18092. This channel should carry information reliably over a few centimeters, after which the signal's power should decrease making the information unintelligible.

Channel B (Customer – Phone) Consists of the interaction between the customer and the phone using the latter's screen and keypad.

[2] `http://java.sun.com/javame/index.jsp`.

Channel C (Customer – Browser/PC) Consists of the interaction between the customer and the PC using the latter's screen, keyboard, and mouse.

Channel D (Browser/PC – Bank Server) Corresponds to the Internet.

6.1.3 Customer Authentication

Customer authentication works in the same way as described in Section *Challenge-response* (page 40). More particularly, the customer authentication protocol is described below and illustrated in Figure 6.1.

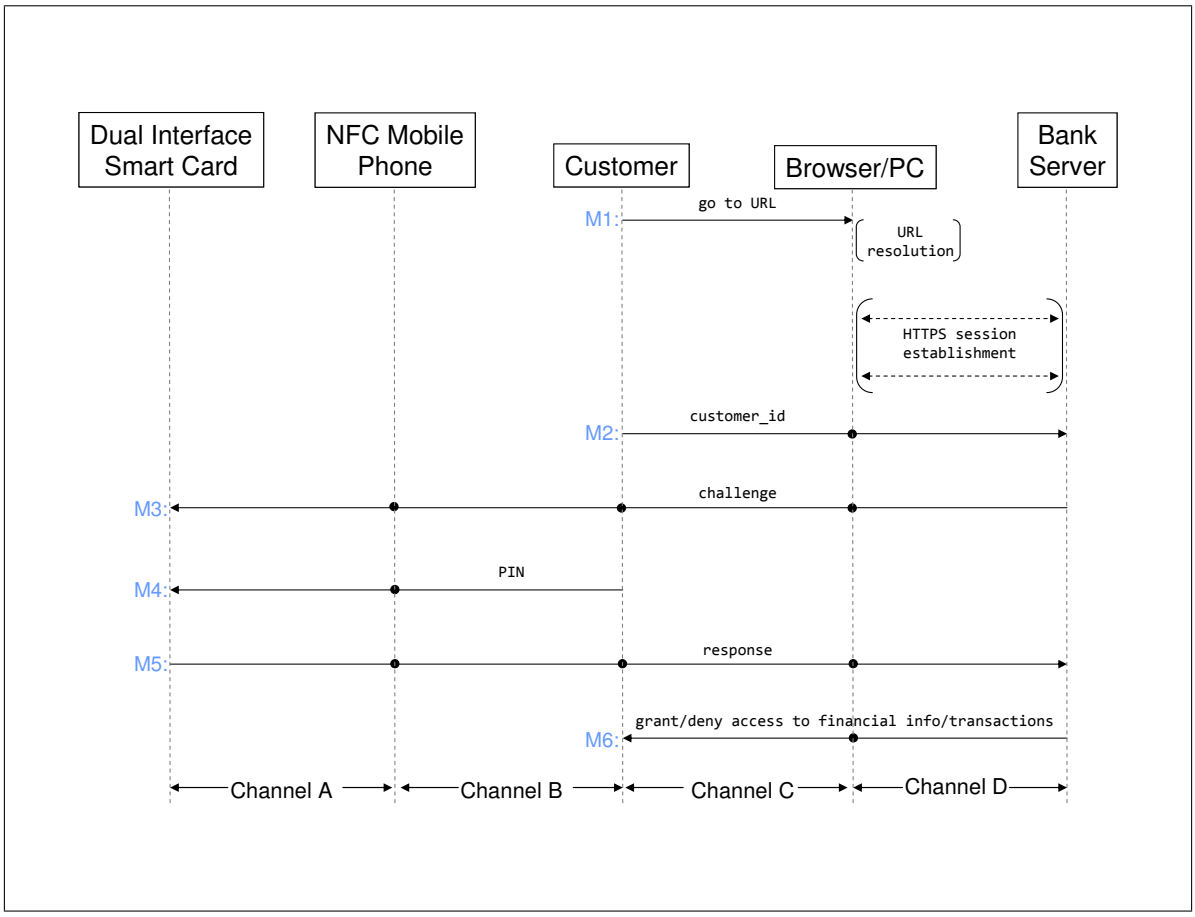


Figure 6.1: Customer authentication protocol.

1. The customer enters the *url* of the eBanking site in the PC (M1).
2. The PC resolves the *url* and opens up the bank's site. A HTTPS session is established between the bank and the browser over channel D.
3. The server sends a form to the browser with a *customer-ID* field.
4. The user types in her *customer-ID* in the PC, e.g. her account or contract number (M2).
5. The server replies a *challenge*, which is a random number between 6 and 8 digits associated to the SSL connection and the *customer-ID* provided in the previous step (M3).

6. The user starts the midlet in the phone.
7. The user selects the *Log-in* mode (in the midlet) and types the challenge in the phone. (M3)
8. The customer types her PIN into the phone (M4).
9. The phone sends the *challenge* and the PIN to the card obtaining a cryptogram in return. Using that cryptogram, the phone generates a code (*response*) that it displays to the user in its display. In order to generate this cryptogram, the card uses the EMV *AC generation* security mechanism, as presented in section 4.3.3.3 (page 27).
10. The user sends the *response* to the server by typing it in the appropriate field of the web form in the PC (M5).
11. The server checks that the received *response* corresponds to the issued *challenge* as presented in Figure 2.2b (page 13). If the *response* is valid, the bank presents the customer with her account(s) summary, as well as the appropriate transaction options (M6).
12. The user can perform a transaction by selecting the appropriate option and filling the necessary fields. No further authentication takes place unless a (server defined) time out occurs.

Clearly, the customer is authenticated by the bank by proving that she is able to produce an appropriate *response* to the random *challenge* sent by the bank. In order to do so, she must use her card and its respective PIN, which is in fact used to authenticate her to the card. Naturally then, the assets that need to be protected are both the PIN and the smart card.

6.1.4 PIN Encipherement

Data exchanged over Channel A (Card – Phone) is protected against eavesdropping by the intrinsic characteristics of the radio signal used to carry the information across this channel. More precisely, the low power of the electromagnetic field generated by the phone makes it very hard to recover any information from distances greater than a few centimeters. Nevertheless, due to the fact that the PIN is transmitted to the card using this channel, there exists a small risk that using specialized equipment a determined attacker might be able to compromise the PIN over this channel across a longer distance, e.g. in the order of a few meters. For this reason, we have decided that the PIN should not be sent in plain to the card, but instead, it should be sent encrypted. Rather than designing a custom made PIN encryption scheme, we have decided to reuse the *PIN encipherment* functionality defined by *EMV* (page 26). Particularly, the PIN should be sent to the card as follows:

1. The phone obtains the issuer certificate $[\![B]\!]$ and the card's PIN encipherment certificate $[\![C]\!]$ from the card and checks their validity.
2. A PIN block b is constructed as shown in Figure 6.2, where:

- Each square represents a nibble.
- 'T' is a static control field with value `0x2`.
- 'N' is the PIN length. It ranges from `0x4` to `0xC`.
- 'P_i' is the i^{th} PIN digit. Its value may range between `0x0` up to an including `0x9`.
- 'F' is a static filler with value `0xF`.

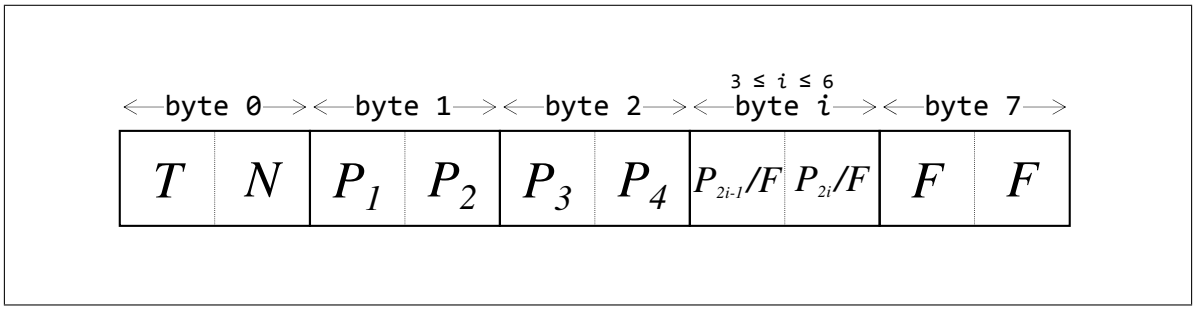


Figure 6.2: PIN block (b).

3. An 8 byte random number r_C is obtained from the smart card.
4. The phone generates a $N-17$ bytes long random bit string r_P, where N corresponds to the length in bytes of K_C, i.e. the public key included in $[\![C]\!]$.
5. Let h be a static one byte data header of value `0x7F`. Then, the enciphered PIN c is calculated as:
   $c \leftarrow \{|\ h\ ||\ b\ ||\ r_C\ ||\ r_P\ |\}K_C$.

Once the card receives c, it can recover the PIN by decrypting it using K_C^{-1}, checking h and extracting the PIN digits using the value of N in the PIN Block.

6.2 Implementation

6.2.1 Smart Card Application *(Cardlet)*

CAP functionality has been implemented as specified in [34] using Java Card 2.2.1. The cardlet has been tested mostly with dual interface JCOP41 cards, but it should run in any Java Card v2.2.1 compliant smart card as a consequence of the usage of standard Java Card APIs. This *cardlet* was developed by Michael Kuyper, an engineer at the IBM Zurich Research Laboratory.

The MIFARE portion of the card has been used to store the *url* from which the mobile phone application can be downloaded. This information is stored as a URI NFC tag following the conventions defined in [38], [39] and [40][3].

[3]The Smart cards used for the prototype are able to emulate a MIFARE Standard 4K card. This means that the EPROM is split in roughly two regions: one used by the *cardlets* and another used by

Figure 6.3: *Midlet* installation confirmation.

6.2.2 Bank Server

The financial institution for which the prototype has been developed has provided access to their test server. For this reason, the environment used for testing the prototype consists of a mirror of their currently deployed CAP server infrastructure.

the MIFARE data structure. MIFARE Standard is a specification for memory cards with provisions for fine grained memory access control. For more information see: `http://www.mifare.net/products/mifare_standard4k.asp`.

6.2.3 Mobile Phone Application *(Midlet)*

The phone application has been developed in Java ME using the Connected Limited Device Configuration (CLDC) version 1.1 and the Mobile Information Device Profile (MIDP) version 2.0. Remarkably, most of the midlet has been implemented using the intersection of the Java ME and SE APIs in such a way that the functionality offered by this application can be easily used both in the mobile phone as well as in the PC. This approach has proven extremely useful for testing and debugging the midlet.

Besides the main midlet, a small ancillary midlet allowing to write to the MIFARE area of the cards has also been developed in order to write the *url* from which the application can be downloaded by the user.

The prototype has been tested using several Nokia 6131 NFC phones[4]. Naturally, as the application has been written in Java, it can be easily ported and adapted to other Java ME compliant NFC enabled mobile devices by implementing the appropriate interfaces.

As required by the proposed design, the midlet contains the list of CA public keys allowed to issue public key certificates, as well as the list of supported CAP cardlet AIDs. The integrity and source authenticity of these lists as well as the application itself is ensured by signing the application resource bundle (JAR file) containing these lists and the application byte code.

Due to the fact that the download link has been written to the card, it suffices for the customer to touch it to the phone in order to initiate the midlet installation, relieving her from having to type the download *url* manually. Once the download link is detected by the phone, it prompts a dialogue asking whether the link should be visited. After the customer chooses *Start*, the phone checks the certificate and asks her whether she would like to install the application showing its details on screen, as illustrated in Figure 6.3 (page 58). When the user accepts, the application is downloaded and installed automatically. After the application has been installed, the user can start it both by selecting its icon in the application list, or simply by touching the card to the phone, which is intuitive and user friendly.

The midlet can be sub-divided in four main components: CAP Core, SC-Comm, UI and Crypto. These components, along with their main interfaces are illustrated in Figure 6.4 and described below.

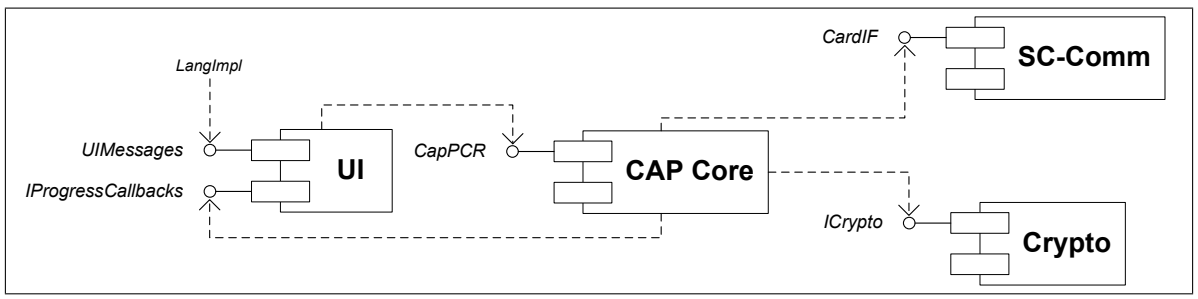


Figure 6.4: *Midlet* components.

[4] `http://europe.nokia.com/A4307094`.

6.2.3.1 CAP Core

This component is an implementation of CAP as specified in [34]. Further, it also supports the customizations designed by the financial institution for which the prototype has been prepared.

The interaction between CAP Core and the UI takes place via the `IProgressCallBacks` interface and the `CapPCR` class. The former is an interface that must be implemented by the UI in order to receive notifications about the progress of the CAP token computation. The latter is the main class of the CAP Core component which offers a CAP related API. It must be remarked that the CAP Core component takes great care in storing the user's PIN in memory for as little time as possible, wiping it as soon as the CAP token has been calculated.

The interaction between CAP Core and the `SC-Comm` takes place by using calls to `CardIF`, which is an abstract class that defines the basic methods to communicate with the smart card. The usage of this popular software pattern allows having some common platform independent logic included in the abstract class declaration, while at the same time allowing for platform dependent implementations to be written by extending the abstract class. This way, the communication with the card takes place using a suitable implementation depending on the environment, e.g. PC or mobile phone.

Finally, the CAP Core component relies on an implementation of the `ICrypto` interface whenever cryptographic operations need to be executed. By using this approach we have attained a good level of flexibility as it was possible to "plug in" different cryptographic implementations in order to determine which yielded the best results in terms of efficiency.

6.2.3.2 Smart Card Communications (SC-Comm)

In the PC, `CardIfPC` extends `CardIF` allowing the CAP application to talk to smart card readers connected to the computer using JPCSC[5]. In the mobile phone, `CardIfNFCNokia` extends `CardIF` by establishing a bridge to the implementation of the JSR-257, which is the standard API for using the near field contactless interface in mobile phones[6]. The particular phone used for the prototype supports this API almost completely. Due to the simplicity of the methods defined by `CardIF`, it is rather trivial to extend this class to suit any other mobile platform, should it not support the JSR-257 API. More precisely, `CardIfPC`, `CardIfNFCNokia`, and in general, any class extending `CardIF` implement the following four methods:

`public void connect()` Establishes a connection to the card. It blocks until a connection is made, or a timeout is reached. Each implementation can define its own timeout value. Should several readers be available, the implementation is free to choose which one to use for the connection. It throws a `CommunicationException` in case that a connection cannot be established, or if this method is invoked when there is already an established connection.

[5] JPCSC is a Java wrapper of the C library to interact with PC/SC compliant smart card readers. See `http://www.zurich.ibm.com/csc/infosec/jcop_tools/download/jpcsc-0.7.txt`.

[6] `http://jcp.org/en/jsr/detail?id=257`.

`public void disconnect()` Destroys the current connection. It has no effect if there is no established connection.

`public boolean isConnected()` Returns true if and only if there is an active connection.

`public byte[] sendApdu(byte[] apdu)` Sends a command APDU to the ICC. '`apdu`' is the command APDU to be sent to the ICC. It returns the response APDU received from the card, or throws a `CommunicationException` in case that there is a problem communicating with the card.

6.2.3.3 User Interface (UI)

A console interface was developed for testing the functionality offered by the CAP Core in the PC. This is a rather straightforward and simple part of the implementation developed just for testing purposes. A more complete graphic user interface was developed for the mobile phone using the generic user interface components offered by the Java ME API. Figure 6.5 (page 62) shows the main screen forms used throughout the login process.

The phone's UI has been developed using standard Java ME APIs, where the final appearance as well as the general on-screen layout of the user interface components is determined entirely by their particular implementation. At some point it was realized that in order to develop a more sophisticated and visually appealing user interface it would be necessary to rescind using standard UI components, doing it completely *by hand*. Even though a very appealing user interface could be created following this approach, it was not further pursued because it implied considerable effort and the gain was not significant for the prototype stage.

Additional to the CAP functionality, the UI offers a configuration screen where the language, the PIN transfer mechanism and the logging settings can be set up. Currently, the UI is implemented in two languages that can be selected and changed at run time using the aforementioned configuration screen. The initial language is selected automatically based on the current locale settings. If a matching language implementation is available, it is selected, otherwise, the default language, i.e. English, is used. Further, supporting new languages is rather easy as it suffices registering a language implementation at the Midlet main class `CAPPhoneUI` that defines suitable strings in the target language for all the messages declared in `UIMessages`.

The configuration screen offers two options regarding how the PIN is transferred to the card: plain and encrypted. This choice is not intended to be available when the prototype is used in a pilot, but rather, it has been created for the sole purpose of illustrating how much enciphered PIN transfer impacts the performance of the CAP token computation. In a similar fashion, a simple logging mechanism that can be turned on and off via the configuration screen has been used throughout the application. If the logging mechanism is enabled, a form containing the application log is displayed once the application is closed. This mechanism has proven quite useful for debugging and to gather performance data measured on the phone itself. In a pilot setting, a setup where only the language can be customized, logging is disabled and the PIN is always sent encrypted to the card would be desirable.

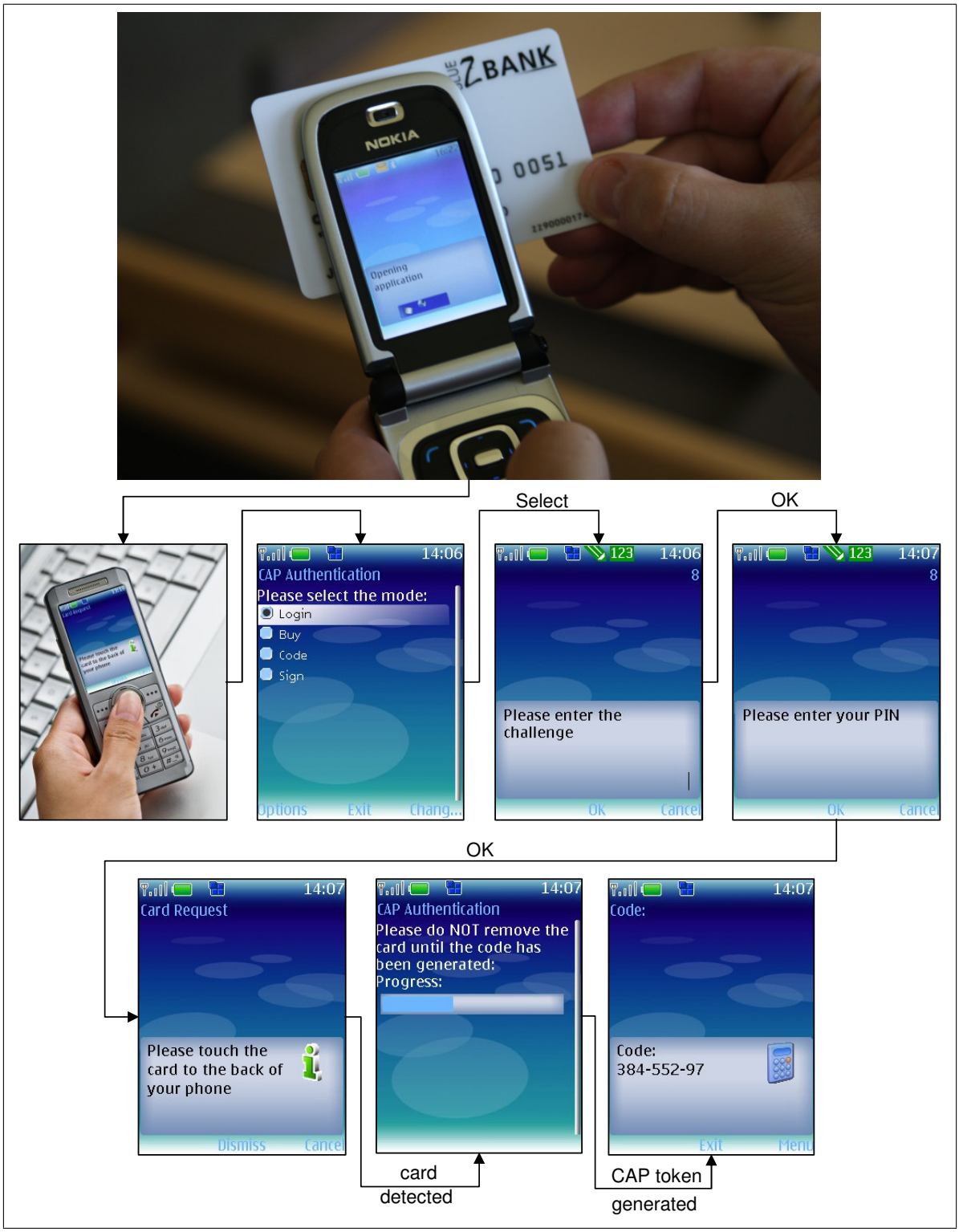


Figure 6.5: *Midlet's* UI: Login process dialogues.

6.2.3.4 Crypto

As it has been mentioned, all cryptographic operations, particularly, random number generation, asymmetric RSA encryption and hashing are declared in the `ICrypto` interface, whose methods are described below:

`public byte[] doRSA(src, srcOff, len, pubKmod, pubKexp)` Performs a raw RSA computation computing

$uint(src_{srcOff}...src_{srcOff+len-1})^{uint(pubKexp)}\ mod\ uint(pubKmod)$,

or throws a `CryptoException` in case that any of the parameters is invalid.

`public void sha1(src, srcOff, len, dst, dstOff)` Hashes a portion of an array using SHA1, writing the result in the specified array, as follows:

$dst_{dstOff}...dst_{dstOff+19} \leftarrow SHA1(src_{srcOff}...src_{srcOff+len-1})$.

It may throw a `CryptoException` in case that the hash cannot be calculated.

`public void macDesCbc(IV, src, srcOff, len, dst, dstOff, key, keyOff)` Calculates a MAC based on (single round) DES in CBC mode, writing the result in the specified array, as follows:

$dst_{dstOff}...dst_{dstOff+7} \leftarrow$

$MAC\_DES\_CBC[key_{keyOff}...key_{keyOff+7}](src_{srcOff}...src_{srcOff+len-1}, IV)$

It may throw a `CryptoException` in case that the MAC cannot be calculated.

`public void fillWithRandom(byte[] dst)` Fills the provided array with a pseudo-random bit string.

The `CL3` (CryptoLite in Java—CLiJ—v3) library has been used for the implementation of the aforementioned methods both in the PC as well as in the mobile phone. This high performance library has been developed by the BlueZ Business Computing group at the IBM Zurich Research Laboratory.

The main use of the cryptographic primitives offered by `ICrypto` consists of the processing of the enciphered PIN as presented in Section *Design and Architecture: PIN Encipherment* (page 56). This operation is interesting from a security point of view, not only because it protects the confidentiality of the PIN when sent over the wireless interface, but because it shows that even though the mechanism is secure under the *black box* paradigm, i.e. the Dolev-Yao model as presented in Section *Cryptographic Primitives* (page 8), it is crucial for the implementation to provide good quality pseudo-random information, as otherwise the mechanism can be easily defeated.

Indeed, considering the PIN encryption mechanism, a passive adversary with access to the communication channel between the phone and the card (Channel A in Figure 6.1, page 55) would easily gain access to c, h, r_C and K_C. In order to retrieve the PIN from that information, she would need to try several $X \leftarrow X_A||X_B$ until $c = \{|\ h\ ||\ X_A\ ||\ r_C\ ||\ X_B\ |\}K_C$ holds.

Once a suitable X has been found, then the PIN can be trivially retrieved from X_A. In practice, $N = |\ K_C\ | = 128\ bytes$, $|\ b\ | = |\ X_A\ | = 8\ bytes$ and $|\ r_P\ | = |\ X_B\ | = N-17 = 111\ bytes$. Consequently, X_A can take at most 2^{64} values, and X_B 2^{888}, resulting in 2^{952} possibilities for X. This means that an attacker would have to do roughly 2^{476} RSA computations in average in order to brute force the PIN from c. Even though these estimates appear to be good at first glance, they are overly optimistic. The estimate of 2^{64} possibilities for X_A assumes that all binary values can be taken, which is in fact, not true. As matter of course, the first nibble and the last byte of the PIN block have fixed

values, so there are 13, and not 16 nibbles whose values are variable. Further, the second nibble of the first byte can only take 8 values and the remaining 12 nibbles (bytes 1 to 6) can take just 11 values each. Consequently, there are not 2^{64} possible values for b, but rather $2^3 \cdot 2^{12 \cdot 3.41} = 2^{44.5}$. Note that this value is still an optimistic estimate because there are format restrictions on bytes 1 to 6 depending on the value of the second nibble. Particularly, the most significant bytes of the PIN cannot contain `0xF` nibbles. Besides, PINs are seldom longer than 4 digits in practice, and hence with very high probability X_A will be among the first 10.000 possible PIN blocks.

Consequently, it is of the utmost importance for r_P to be unpredictable. If a weak source of randomness is used, it would eventually be possible for the adversary to brute force the encrypted PIN block with relative ease. For this reason, some thought was given to the issue of randomness in the mobile phone. Four possibilities were considered to solve this issue: using some native operating system call, the Java API, i.e. the `java.security.Random` class; a Java cryptography provider, or `CL3`. It was decided for the latter approach because using a system call is not possible from the Java sandbox, using the Java API does not offer any guarantees regarding the *quality* of its output and using a provider would have increased unnecessarily the size of the phone application. On the other hand, `CL3` not only has an effective mechanism for generating pseudo random numbers based on [3], but it also had already been integrated into the prototype.

6.3 Discussion

6.3.1 Novelty

Involving mobile phones in the authentication mechanism can be traced to previous work such as presented in [7], published some 10 years ago. More recently mTANs (43) and the mechanisms presented in Section *Mobile Phone Based Mechanisms* (page 49) also involve using mobile phones.

The authentication mechanism presented in this chapter differs from these mechanisms in the fact that the NFC enabled phone assumes an active role replacing the PCR. Such a role is unconventional considering that it is far more usual for NFC phones to be used passively in *card emulation mode* rather than in *reader mode*. Moreover, allowing the phone to replace the PCR solves one of the biggest shortcomings of PCR-based authentication mechanisms in terms of convenience. Namely, the user does no longer need a PCR in order to engage in eBanking, while still being able to use a challenge-response protocol. Naturally, the customer needs the phone, but phones are undoubtedly much more ubiquitous than PCRs.

In terms of the protocols there is no novelty at all, which is an advantage rather than a disadvantage. This follows from the fact that by reusing existing and widely studied protocols the expertise and testing that these protocols have already undergone is leveraged. Besides, engineering our own protocols would have required additional time and effort devoted to their study and analysis, which would have prevented the achievement of the realizability goal.

6.3.2 Performance

The current version of the midlet occupies 160 KB, roughly distributed as follows: CAP Core 38% (60 KB), SC-Comm 3% (5 KB), UI 16% (25 KB), Crypto 31% (50 KB), and graphics, i.e. the application icon and the splash screen, 13% (20 KB).

It takes about 8 seconds to download the application over a standard data connection, i.e. using GPRS or EDGE. The CAP token computation time is on average 1.5 seconds. This time is closer to 2 seconds when the PIN is sent encrypted, and to 1 second when it is sent in plain. The RSA encryption of the PIN block takes approximately 75 msec, while its decryption and verification (by the card) takes about 500 msec. As expected, this is the most expensive operation of the CAP token computation. In contrast, PIN verification only takes about 85 msec when the PIN is sent in plain.

Naturally the application functionally works, as it has been confirmed after testing it against the test server described in Section *Bank Server* (58). It must be said, though, that the generation of the CAP token takes slightly longer than with the traditional PCR, although the difference is by no means abysmal. Certainly, using plain PIN authentication is very fast and the code is presented to the user very promptly and at a comparable speed to that of a PCR.

It was seen during experimentation that sometimes when the card is touched at certain positions of the phone, the link quality is not very good leading to dropped connections. It is presumed that this happens as a consequence of the low power of the phone's field and the small antenna that it houses. Nevertheless, in the overwhelming majority of cases the connection is maintained and the CAP token is generated successfully. It is very likely that future NFC enabled phones will permit more stable connections by using a bigger antenna or shaping the electromagnetic field more efficiently.

6.3.3 Security

The security analysis is equivalent to Challenge-response as presented in Section *Challenge-response* (page 40). Particularly, if active attacks are considered, neither Goal 3.2 (Customer authentication) or Goal 3.3 (Data authentication) are achieved. This follows from the fact that an adversary may be able to produce a valid response to the challenge generated by the bank by mounting a MITM attack, e.g. in the PC. Nevertheless, it must be noted that these attacks must be executed in real-time due to the unpredictability of the challenges issued by the server. Goal 3.4 (Privacy) is only achieved under a remote passive attacker, being defeated by a local or remote active attacker.

The PIN is a valuable asset and the implemented solution takes care of protecting it accordingly by not sending it in plain to the card. Still, it is important to remark that the phone must be trusted to keep the PIN secure from the moment that it is typed in by the user until the moment when it is wiped out from memory at run time. This assumption is not unreasonable considering that many authentication mechanisms involving mobile phones such as the ones presented in Section *Mobile Phone Based Mechanisms* (page 49) and in [61] make this assumption too. In fact, sometimes they make a stronger assumption because they expect the phone to securely store a long lived secret. As a matter of course, these solutions often work because they make use of the *splitting trust*

paradigm, where a sensitive part of the application runs on the mobile phone, which as of today can be considered a more trusted platform than the PC.

The midlet installation procedure is rather secure. Note that bank cards are often sent to their users by surface mail, so an attacker could in principle overwrite the download link by getting hold of the envelope containing the card at any point before it reaches its recipient. Interestingly, she would not even need to tamper with the envelope due to the fact that modifying the MIFARE area is done via the contactless interface, which can be done while the card is still inside the envelope. However, the memory area where the NFC tag is written can be protected straightforwardly using standard MIFARE access control mechanisms, effectively thwarting the abovementioned attack.

It could be argued that once the customer uses the card to start the download, she could be redirected to a rogue website using -for instance- a DNS poisoning attack. This attack is prevented by signing the application and asking the user to confirm the signer identity, as depicted by Figure 6.3 (page 58). Naturally, if the user fails to check this information, a trojaned application stealing the customer's PIN could be installed. Further, if the user leaves her phone unattended, an attacker could replace the midlet with such a trojaned version without the user's intervention.

6.3.4 Usability

Replacing the PCR with the phone increases substantially the level of convenience of the authentication mechanism. This follows from the fact that the user only needs her phone and her card in order to authenticate to the bank. On the one hand, phones are truly ubiquitous devices that can be hardly considered a burden to be carried; on the other, most people carry their bank cards with them in their wallets or purses at all times, therefore making the requirements of this authentication mechanism quite low.

Further, NFC phones do not only replace PCRs, but also make the security mechanism more user friendly as a consequence of their rich and familiar user interface capabilities, compared to the very basic displays offered by existing PCRs. Also, the feature of starting the midlet simply by touching the card to the phone helps making its usage rather easy and coherent from the user's point of view.

The installation procedure can be considered easy, albeit not trivial. The user only has to touch the card to the phone in order to launch the application download, which is easy and intuitive. Then she has to verify the certificate associated to the midlet, which is less trivial. Nevertheless, it suffices to instruct her to check the certificate's subject and issuer in order to verify the origin of the application.

6.3.5 Cost-efficiency

From a purely cost oriented perspective, developing and deploying the proposed mechanism is rather advantageous. This follows from the fact that already existing CAP server infrastructures as well as CAP compliant smart card applications can be reused without any modification. The hardware investment is lower than with other smart card based authentication solutions due to the fact that the bank does not have to provide smart

card readers to its customers, as these readers are actually provided by customers themselves in the form of their NFC enabled handsets. Eventually, however, the *fixed* costs related to supporting the mechanism could be slightly higher than those associated to authentication mechanisms based on more standardized hardware. This is related to the fact that for the proposed mechanism to be successful, as many NFC enabled handsets as possible would need to be supported, and such device heterogeneity may be harder to support than a closed environment where only a few hardware devices need to be supported. In the end, it seems that the most cost-effective and user-friendly approach would be realized by supporting good and stable NFC phones, while still allowing the rest of the customers, i.e. those without an NFC enabled phone, or with an unsupported one, to use the *traditional* PCR for eBanking purposes.

The proposed solution can be seen as an almost-only software solution where the significant part of the costs corresponds to the *fixed* cost related to the development, maintenance and support of the *midlet* and other software components, while the *marginal* costs are negligible due to the fact that they are mainly concerned with the smart card reader—provided by the customer—and the smart card itself, which in any case has to be issued by the bank for other banking purposes such as ATM usage.

6.3.6 Other Goals

Not only the goal of cost-efficiency, but also the goal of realizability is addressed by leveraging existing know-how and infrastructure, i.e. the server and the CAP application running on the smart cards. Further, these two goals are also addressed by using technology deployed or to be deployed in the short term, particularly, contactless bank cards capable of running CAP and NFC enabled mobile phones. Most smart cards currently deployed contain a payment application that can be used for CAP purposes, but they lack a contactless interface. Nevertheless, it is relatively easy for banks to start issuing dual interface cards if they deem it necessary. Further, these cards can also be leveraged for contactless payment applications such as MasterCard PayPass[7]. Consequently, dual interface smart cards are not a problem considering that they can be rolled out quite easily.

On the other hand, it is uncertain whether any particular certification must be attained for the software running on the phone, but this is certainly achievable. The availability of NFC enabled mobile phones is a more crucial issue considering that they are not yet widely available in the commercial market. The adoption of NFC technology ranges from pilots in Europe and North America to full deployment in Asia, where there are about 40 million users just in Japan [15]. In this fashion, excluding the Asiatic markets, particularly Korea and Japan where contactless payment technologies using mobile phones are pervasive, there are only 3 models commercially available: the Samsung SGH-X700NFC, the Nokia 6131NFC, and the Nokia 6212 (available by the third quarter of 2008).

According to a report by ABI Research [1] published in 2006, there should be about 100 million NFC enabled handsets in the market by now, as presented in Figure 6.6, which even including the Asian markets, seems quite optimistic.

[7] `http://www.paypass.com/`

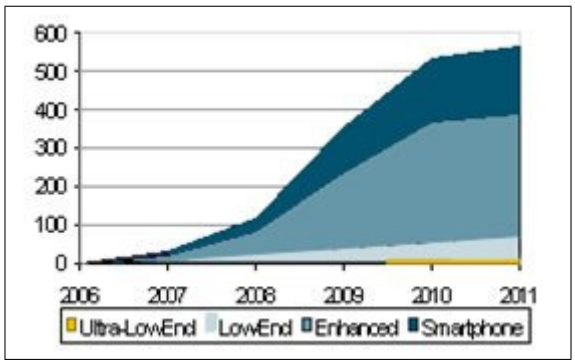


Figure 6.6: NFC enabled handset shipments of the world market, 2006–2011 [1].

Interestingly, the lack of penetration of NFC phones is a marketing problem rather than a technical one. As it was mentioned in Section *Near field communication (NFC)* (33), there are fundamentally two options for hosting the secure element: the SIM card or an independent chip. From a technical perspective, both options are equivalent, but from the mobile phone operators (*telcos*) perspective it is more convenient for the secure element to be under their control, as they can profit from the applications deployed into it. Further, they claim that by using the SIM card as the secure element the NFC applications are independent of the particular handset where they are executed. Consequently, they tend to favor the former option, i.e. having the SIM card acting as the secure element. In fact, this position is explicitly stated in [21]: "*The secure element recommended by the GSMA [GSM Association] for the payment application in the mobile phone is the Universal Integrated Circuit Card (UICC), commonly known as the SIM card*".

Conversely, having an independent chip hosting the secure element is more flexible as no collaboration from the mobile phone operator is required to deploy applications into it, making the business case for these applications is more attractive. In practice, though, telcos are ultimately the entities that decide which devices their customers get, so the handsets that do not adjust to their strategic requirements—such as NFC phones with secure elements independent from the SIM card—are simply not offered to their customers, despite having very exciting and interesting use cases as presented in Section *Near field communication (NFC)* (33).

Interestingly, the authentication solution presented in this chapter is oblivious to the secure element issue discussed above due to the fact that the secure element is only used when the phone acts in *card emulation mode* and not when it acts in *reader mode*, as in the proposed solution. Still, due to the problem outlined above there are not too many NFC handsets in the market despite the potential applications and use cases for NFC phones. It is the opinion of the author, however, that the outlook is not as bleak as it appears: once the secure element issue is solved, it is likely that the market penetration of NFC phones will increase dramatically as portrayed in Figure 6.6 (page 68), albeit, a little bit later in time. In any case, regardless of the particular solution to the aforementioned problem, NFC phones will be an interesting platform upon which authentication solutions can be built.

6.4 Future Work and Extensions

There are several improvements that could be done to the authentication mechanism presented so far. Particularly, it may be possible to further enhance convenience by getting rid of the smart card altogether. This would be attained by deploying the cardlet directly in the phone's secure element. Note that in this case, NFC as such would not be used anymore, but rather, the API to access the secure element from the midlet. Although this approach is technically feasible, the marketing issue regarding the location of the secure element in the phone that has just been explained must be considered. Ultimately, it may be necessary to reach some type of arrangement with the telcos in order to deploy the cardlet and its associated keys into the secure element.

In terms of usability, both the challenge and the response could be sent directly from the bank server to the phone and back via SMS, or some other suitable mechanism leveraging the mobile phone network (Channel E in Figure 6.7). This would not only simplify the authentication mechanism, but it would also increase the level of security. The mechanism would be simpler because the user would not have to transcribe any information between the phone and the PC, and it would be more secure because getting hold of the challenge/response when sent via a secondary channel such as the mobile phone network is harder than when sent over the Internet using the customer's PC [53]. Naturally, the downside of this improvement is that every SMS has an associated cost that would need to be assumed either by the bank or its customers[8].

The user interface in its current form is pleasant and simple to use. However, it is the opinion of the author that a much better result would be achieved by drawing it directly on the screen without using the `javax.microedition.lcdui` components, but rather a blank `Canvas`. Naturally, this would imply more effort and it would be necessary to design—or take into account—the screen resolution of the different phones supporting the application.

In terms of security, it is unavoidable to remark that the complexity of the software stack running on contemporary high end phones—particularly the operating system—is increasing dramatically. It remains a rule of thumb that complexity is the worst enemy of security and its increase means that it becomes less reasonable considering phones as *trusted*. As noted in [33] the solution to this issue consists of using a TPM (page 44) in the mobile device. Even though these platforms are not yet widely deployed, it seems that using them in mobile devices is more realizable than using them in PCs. This follows from the fact that mobile platforms are smaller and more homogeneous thus making them easier to *measure*. Besides, the restrictions that can be enforced with the TPM might be in line with the ownership model of mobile devices which are often not solely owned by the user but also by the service provider who has subsidized their cost [31].

6.4.1 Transaction Data Authentication

At a more fundamental level, the solution presented so far—even with the proposed improvements—remains vulnerable to on-line active attacks. By introducing an addi-

[8] In fact, this alternative was considered and discussed with the interested parties at the beginning of the project, but it was discarded because of the increase of the operational costs.

tional data authentication step in the spirit of [33] (cf. Section *Mobile Password Authentication*, page 50) much better result in terms of the achievement of the security goals can be obtained. The data authentication protocol would work as depicted in Figure 6.7 and described below:

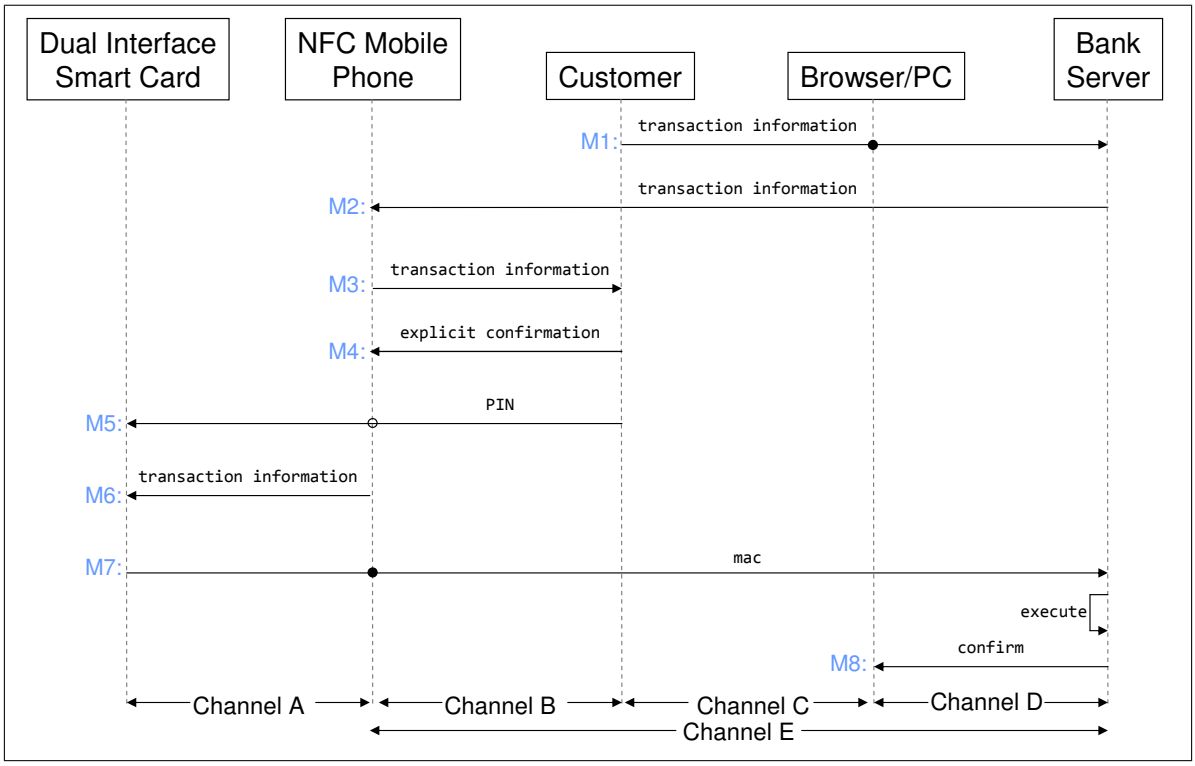


Figure 6.7: Transaction data authentication protocol.

1. After the customer has authenticated (steps 1 to 11 as presented in Section *Design and Architecture: Customer Authentication*, page 55), she fills in the transaction information in her computer, which is then sent to the bank (M1).

2. The bank sends the transaction information to the mobile phone using channel E, e.g. via SMS (M2).

3. The phone shows the transaction information to the user for confirmation (M3).

4. The customer checks the information presented by the phone and confirms it, e.g. by pressing an OK button (M4).

5. The customer types in her PIN into the phone, which then sends it (encrypted) to the card (M5).

6. The phone sends the transaction information to the card (M6), which in turn generates a message authentication code (MAC) over the transaction information.

7. The MAC is sent from the card to the phone, which then forwards it to the server using channel E.

8. Upon receiving the MAC, the server checks it and notifies the customer of the operation result (M8).

Ideally M2 should go encrypted, as it prevents passive attackers eavesdropping on channel E from having access to the transaction information. The encryption key does not necessarily have to be a static shared key between the bank and the phone. In fact, the information returned in the application cryptogram in step 9 (page 56) could be re-used for this purpose. Also, note that the protocol can be built by leveraging CAP mode *Sign* as presented on in Section *CAP Modes* (page 32).

Both M2 and M7 are exchanged between the phone and the server via channel E, e.g. the mobile phone network. In the case of M7, using channel E is a usability enhancement because it relieves the user from having to transcribe the MAC from the phone to the PC. However, in the case of M2, using channel E is a necessity because asking the user to type the transaction information twice is clearly unacceptable from a usability point of view. Moreover, usability could be further improved by discarding M5, which would be possible by letting the phone store the PIN temporarily until the midlet is closed, or until a certain time out.

Under an active attacker, Goal 3.3 (Data Authentication) is achieved regardless of the attacker locality, e.g. PC and Channels D and E. However, if the attacker manages to compromise the mobile phone itself, this approach is also rendered ineffective. If the phone is trusted, though, even in the worst scenario consisting of the attacker replacing the transaction information T in M1 with T' before it reaches the Bank server, and then replacing T' back with T in M2 before it reaches the phone (assuming that M2 is not being encrypted), data authentication would fail. This follows from the fact that the MAC calculated by the card and sent to the bank in M7 would correspond to $\mathcal{H}(k_{CB}, T)$, which would differ from the MAC computed by the bank, i.e. $\mathcal{H}(k_{CB}, T')$.

The proposed *transaction data authentication* protocol uses channel E, i.e. the mobile phone network, which seems rather natural considering that a mobile phone is involved in the mechanism. Nevertheless, the main drawback of relying on this channel corresponds to the costs associated to it. In fact, unlike the Internet, where the traffic exchanged between the customer and the bank is virtually for free, messages sent over the mobile phone network impose some additional costs that need to be borne either by the customer and/or the bank. Furthermore, using channel E—either for customer or data authentication—entails both the additional requirements of the phone being within the coverage area of the mobile phone network, and being able to send and/or receive messages. Naturally, in most scenarios this would be the case, but under certain circumstances it may not be so, thus hindering the convenience of the resulting mechanism.

What is more, using channel E involves at least one telco in the mechanism, clearly decreasing the users' privacy due to the fact that the telco is made aware that the customer is engaging in some kind of financial transaction. In fact, in the case of *roaming* users, both the cost per user/transaction authentication increases, and the level of privacy decreases as a consequence of the involvement of additional telcos in the communication path between the bank and the customer.

CHAPTER

SEVEN

Conclusions

Developing this project has been interesting, intensive and time consuming. The fact that many existing technologies were reused has been helpful in order to achieve an effective and efficient working prototype. Nevertheless, it was necessary to go over a rather large body of documentation in order to understand the intricacies of these technologies and use them properly. Moreover, the research regarding the state of the art in eBanking authentication took longer than expected and was more extensive than initially planned. Nevertheless, it is the hope of the author that this research is useful as a starting point for other people looking into the subject of remote authentication.

Covering the entirety of the considered solution, from the design to the implementation has been an enriching experience. After completing the courses of the Information Security Master's, it was only natural to have an academic security-oriented mentality. This has proven very useful, but it must be acknowledged that such a mentality has been complemented by other factors such as the importance of considering the user in the design process. In fact, even before starting the implementation of the prototype, it was known that the level of security of the considered authentication system was comparable to the level of security of other systems currently being used, not *better*. Nevertheless, the usability improvements provided by the former system, particularly, no longer requiring a *personal card reader* and improving the user experience by leveraging the user interface provided by the phone, justified considering it for a prototype. Indeed, this prototype –developed by the author in collaboration with other members of the IBM Zurich Research Laboratory– has already been delivered to the financial institution for which it was developed. The feedback that has been received so far has been very positive, and the project is moving ahead towards the pilot stage.

It is known that the current design does not withstand on-line active attacks. Nevertheless, an enhancement has been proposed to withstand such attacks, consisting of authenticating the user *and* the transaction data. Additionally, one of the main findings of this project is that using NFC phones in tandem with contactless smart cards for remote customer authentication is *indeed* technically feasible. In spite of this, the lack of market offer of NFC enabled mobile phones has been found to be the only relevant factor threatening the feasibility of the considered solution. The reason why NFC phones are not yet widespread seems to be the indecision as to whether the secure element connected to the NFC interface should be independent of the SIM, dependent on it, or the SIM itself.

Unfortunately, it is hard to predict how long it will take for the stakeholders to

reach a decision regarding the secure element location. In any case, it is the opinion of the author that a purely NFC phone based solution would hardly be realizable in the short term. Nevertheless, due to the fact that the considered solution builds on existing authentication mechanisms currently being used by the financial institution, chiefly, Chip authentication program, it is certainly feasible to consider a gradual transition starting with a few phones, which will grow as NFC market penetration increases. After all, it is transparent to the bank server whether customers are using a PCR and a contact-only card, or an NFC enabled mobile phone and a dual interface smart card. In fact, both systems may coexist seamlessly, which is undoubtedly one of the greatest advantages of the considered authentication solution.

BIBLIOGRAPHY

[1] ABI Research. *Near Field Communication (NFC). Leveraging Contactless for Mobile Payments, Content and Access*. ABI Research, 2006.

[2] A. Adams and M. A. Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):40–46, 1999.

[3] American National Standards Institute. *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA), ANSI X9.31-1988*, September 1998.

[4] R. J. Anderson. *Security engineering: a guide to building dependable distributed systems*. Wiley Computer Publishing, first edition, 2001.

[5] R. J. Anderson. *Security engineering: a guide to building dependable distributed systems*. Wiley, second edition, 2008.

[6] D. Balaban. The future of the contactless SIM. *Card Technology*, pages 16–22, January 2005.

[7] D. Balfanz and E. W. Felten. Hand-held computers can be better smart cards. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 2–2, Berkeley, CA, USA, 1999. USENIX Association.

[8] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, first edition, 2002.

[9] M. Bishop. *Introduction to Computer Security*. Addison-Wesley Professional, first edition, 2004.

[10] R. Clayton. Who'd Phish from the Summit of Kilimanjaro? In *Financial Cryptography and Data Security*, volume 3570 of *Lecture Notes in Computer Science*, pages 91–92. Springer, 2005.

[11] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590. ACM, 2006.

[12] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357. IEEE, 1981.

[13] J.-E. Ekberg, N. Asokan, K. Kostiainen, P. Eronen, A. Rantala, and A. Sharma. *OnBoard Credentials Platform: Design and Implementation*. Nokia Research Center Helsinki, Finland, January 2008.

[14] EMVCo LLC. *EMV Circuit Card Specifications for Payment Systems*, 2004.

[15] B. Ensor and A. Hesse. *NFC Technology Is Revitalizing Mobile Payments*. Forrester Research, April 2008.

[16] L. Faith-Cranor and S. Garfinkel, editors. *Security and Usability. Designing Secure Systems that People can use*. O'Reilly Media, Inc., August 2005.

[17] B. J. Fogg, C. Soohoo, D. R. Danielson, L. Marable, J. Stanford, and E. R. Tauber. How do users evaluate the credibility of web sites?: a study with over 2,500 participants. In *DUX '03: Proceedings of the 2003 conference on Designing for user experiences*, pages 1–15, New York, NY, USA, 2003. ACM.

[18] K. Fu, E. Sit, K. Smith, and N. Feamster. Dos and don'ts of client authentication on the web. In *Proceedings of the 10th USENIX Security Symposium*, Aug. 2001.

[19] S. Gajek, A.-R. Sadeghi, C. Stuble, and M. Winandy. Compartmented security for browsers - or how to thwart a phisher with trusted computing. In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, pages 120–127, Washington, DC, USA, 2007. IEEE Computer Society.

[20] S. Garfinkel. *Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable*. PhD thesis, Massachusetts Institute of Technology, May 2005.

[21] GSM Association. *Pay-Buy-Mobile. Business Opportunity Analysis*. GSM Association, November 2007.

[22] P. Gutmann. *Cryptlib. Security Usability Fundamentals*, 2007.

[23] P. Gutmann and I. Grigg. Security usability. *IEEE Security and Privacy*, 3(4):56–58, July 2005.

[24] A. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten. *Lest We Remember: Cold Boot Attacks on Encryption Keys*, February 2008.

[25] A. Hiltgen, T. Kramp, and T. Weigold. Secure Internet banking authentication. *IEEE Security and Privacy*, 4(2):21–29, 2006.

[26] Ingenio, Inc. *A Study about Cell Phone Usage*, June 2007.

[27] A. Jain, S. Pankanti, S. Prabhakar, L. Hong, and R. Anderson. Biometrics: a grand challenge. In *Proceedings of the 17th International Conference on Pattern Recognition.*, volume 2, pages 935 – 942. IEEE, august 2004.

[28] A. Kerckhoffs. La cryptographie militaire. In *Journal des sciences militaires*, pages 5–38, February 1883.

[29] F. Koeune and F.-X. Standaert. A tutorial on physical security and side-channel attacks. In *Foundations of Security Analysis and Design III*, pages 78–108. Springer, 2005.

[30] V. Kostakos and E. O'Neill. NFC on mobile phones: Issues, lessons and future research. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, 2007. PerCom Workshops '07.*, pages 367–370, 2007.

[31] K. Kursawe. Trusted platforms. In M. Petrovic and W. Jonker, editors, *Security, Privacy and Trust in modern data management*, Data Centric Systems and Applications, pages 119–132. Springer, 2007.

[32] B. W. Lampson. Computer security in the real world. *Computer*, 37(6):37–46, 2004.

[33] M. Mannan and P. C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *Financial Cryptography and Data Security*, volume 4886 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2008.

[34] Master Card Worldwide. *Chip Authentication Program. Functional Architecture*, February 2007.

[35] J. M. McCune, A. Perrig, and M. K. Reiter. Bump in the ether: a framework for securing sensitive user input. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, Berkeley, CA, USA, 2006. USENIX Association.

[36] MELANI. Swiss Federal Office of Police. Reporting and Analysis Centre for Information Assurance. *Information Assurance. Situation in Switzerland and internationally. Semi-annual report 2007/I (January – June)*, 2007.

[37] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, first edition, 1996.

[38] NFC Forum. *NFC Data Exchange Format (NDEF). Technical Specification*, July 2006.

[39] NFC Forum. *NFC Record Type Definition (RTD)*, July 2006.

[40] NFC Forum. *URI Record Type Definition*, July 2006.

[41] M. Nilsson, A. Adams, and S. Herd. Building security and trust in online banking. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1701–1704, New York, NY, USA, 2005. ACM.

[42] J. Ondrus and Y. Pigneur. An assessment of NFC for future mobile payment systems. In *International Conference on the Management of Mobile Business, 2007. ICMB 2007*, 2007.

[43] B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. In *Financial Cryptography and Data Security*, volume 4107 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2006.

[44] P. Phillips, A. Martin, C. Wilson, and M. Przybocki. An introduction to evaluating biometric systems. *Computer*, 33(2):56–63, 2000.

[45] D. Praca and C. Barral. From smart cards to smart objects: the road to new smart technologies. *Computer Networks*, 36(4):381–389, 2001.

[46] W. Rankl. *Smart Card Handbook*. Wiley, third edition, 2004.

[47] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley Professiona, 2000.

[48] Research and Markets. *Europe - Mobile Market - Overview & Statistics*, July 2007.

[49] R. A. Saar Drimer, Steven J. Murdoch. *Thinking inside the box: System-level failures of tamper proofing*. University of Cambridge. Computer Laboratory, February 2008.

[50] B. Schneier. Security in the Real World: How to Evaluate Security Technology. *Computer Security Journal*, 15(4), 1999.

[51] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, first edition, 2000.

[52] B. Schneier. *Beyond Fear. Thinking sensibly about security in an uncertain world*. Copernicus Books, first edition, 2003.

[53] B. Schneier. Two-factor authentication: Too little, Too late. *Communications of the ACM*, 48(4):136, 2005.

[54] B. Schneier. How security companies sucker us with Lemons. *Wired Magazine*, April 2007.

[55] B. Schoenmakers. *Lecture Notes Cryptographic Protocols*. Technical University of Eindhoven, July 2007.

[56] M. Stamp. *Information Security: Principles and Practice*. Wiley-Interscience, 2005.

[57] G. Stoneburner, A. Goguen, and A. Feringa. *Risk Management Guide for Information Technology Systems*. National Institute of Standards and Technology, July 2002.

[58] T. Weigold, T. Kramp, R. Hermann, F. Hoering, P. Buhler, and M. Baentsch. The Zurich Trusted Information Channel An efficient defence against Man-in-the-Middle and malicious software attacks. In *(To appear in) Proc. Trust 2008, Villach, Austria*, Lecture Notes in Computer Science. Springer, March 2008.

[59] C. West. "Phishing in the middle of the stream" - Todays threats to online banking. In *Proceedings of the AVAR 2005 conference*, 2005.

[60] R. West. The psychology of security. *Commun. ACM*, 51(4):34–40, 2008.

[61] M. Wu, S. Garfinkel, and R. Miller. Secure web authentication with mobile phones. In *DIMACS Workshop on Usable Privacy and Security Systems*, 2004.

[62] K.-P. Yee. Aligning security and usability. *IEEE Security and Privacy*, 2(5):48–55, September 2004.