University of Twente

Faculty of electrical engineering, mathematics and computer science

UNIVERSITEIT TWENTE.

Nedap Securiy Management

Using an NFC-equipped mobile phone as a token in physical access control

Author:
Martijn Bolhuis

Committee:
Dr. Ir. G.J. Heijenk
Ir. Drs. T. Garthoff
Dr. Ir. P.T. de Boer
Dr. J. Petit

July 2, 2014

Contents

Chapter 1

Introduction

This thesis investigates the use of a mobile phone with Near-field Communication (*NFC*) in physical access control. A typical physical access control scenario is a company office with several departments that are protected by electronically controlled doors. An employee can open a door by presenting his/her phone to an NFC reader that is located near the door. The reader and the phone communicate in order to verify that the user has access. Subsequently, the electronic door is opened and the employee can enter the building.

This introduction chapter first briefly discusses background information about NFC and physical access control (Section 1.1). Subsequently, this research is motivated in Section 1.2. In Section 1.3, a high level view of the problem statement is given which is used in Section 1.5 to define the research questions. Section 1.6 discusses the organisation for the remainder of this thesis.

1.1 Background information

NFC is a technology that enables near-range (up to 10 cm) radio communication between two parties. NFC operates at 13.56 MHz and can achieve transfers rates up to 424 kbit/s. An increasing number of smart phones today are equipped with NFC-capabilities [1]. Applications include mobile payment, ticketing and access control. Moreover, NFC can be used to bootstrap other types of connection such as Bluetooth or Wifi. Because the NFC connection is so easy to set up this can be used to exchange parameters for another connection type ultimately making the latter one easier to set up.

A *physical access control system* (PACS) can be defined as a system that restricts access to a physical resource, e.g. building, room, parking garage or a locker to a selected number of users. A PACS supports two main activities: *authentication* and *authorization*. When a user requests access to a physical resource it claims an identity (e.g. "my name is Martijn") and the authentication process will verify the validity of this claim. Authorization is determining what a user is allowed to do.

Authentication in physical access control typically involves the authentication of a person. This can be accomplished by verifying one of the following properties.

1. **Something that the person knows** - such as a password or a personal identification number (PIN).
2. **Something that the person has** - such as a key or a token. A token is a physical entity or device such as a smart card. In this thesis, the possibility of using a mobile phone as a token is investigated.
3. **Something that the person is** - biometric properties such as fingerprints, retina or iris data

1.2 Motivation

This work is motivated by looking at the use of tokens in daily life and not limiting the scope to physical access control. In today's information society many services that we use in daily life require some form of physical token. Examples of tokens include contactless or contact based smart cards, other plastic cards (with bar code or magnetic stripe) and keys. These tokens are used in many services / applications such as:

- **Payment** - Cards used for making payments
- **Ticketing** - Cards are commonly used in public transport.
- **Bonus cards** - Some supermarkets, fuel stations and other type of shops supply so-called bonus cards to their customers which can be used to save points and get discounts.
- **Identification** - Passport, ID-card, car registration and other governmental documents.
- **Physical access control** - Cards are commonly used in physical access control. This is used in hotels, offices, parking garages and locker systems.
- **Physical keys** - Most homes still use manual locks.

The use of many tokens can cause usability problems. *Usability* can be defined as "the *effectiveness*, *efficiency* and *satisfaction* with which specified users achieve specified goals in particular environments" by [2].

- **Effectiveness** - "*the accuracy and completeness with which specified users can achieve specified goals in particular environments*" by [2].
- **Efficiency** - "*the resources expended in relation to the accuracy and completeness of goals achieved*" by [2].
- **Satisfaction** - "*the comfort and acceptability of the work system to its users and other people affected by its use*" by [2].

Two problems with tokens related to the satisfaction aspect of usability can be identified:

1. As the number of used services increase, the users have to keep more cards and keys in their wallet or pocket which can be uncomfortable.

2. Obtaining or updating a physical token (e.g. the set up process, load the necessary credentials on to the card) requires physical presence at a point of service (POS).

These problems can be solved by integrating all tokens into one physical entity. The mobile phone is a suitable candidate as shown by various pilots that were held [3] [4]. The reason for this is that the phone has become an important device in daily life and many people do not leave home without it.

This solves problem 1 because all tokens are digitally stored on the phone meaning that no extra physical space is required. The mobile phone has the potential to address problem 2. The internet connection available on most phones could be leveraged to install or update the required components onto the phone. Therefore, it is not necessary for a user to go to a POS in order to do this.

Although the integration of physical tokens for various services is an important motivation, this thesis focusses on the use of NFC in one particular service (physical access control). The reason for this is that many research has already been done into how such integration can be accomplished as we will discuss in Chapter 3.

The current state of the NFC technology opens the door for various types of solutions each with their pros and cons. Each type of service may have additional requirements with regards to usability and security. Therefore, it is useful to identify and compare the possible solutions within the context of one service.

1.3 Problem Statement

Figure 1.1 shows a simplified PACS. A more detailed view of a PACSs will be discussed in Chapter 8. The PACS consists of the following components:

1. **Access Point (AP)** - The point where the user obtains access to the resource. In this case it consists of an electronic door and a card / NFC reader. In Section 2.3 it will be explained that NFC re-uses standards from contactless smart cards and therefore it is possible that an NFC capable mobile phone communicates with existing card readers.

2. **Controller** - An on-site computer that controls the behaviour of one or more access points. It receives authentication, authorization and configuration data from the ACS.

3. **Access Control Server (ACS)** - A central server that stores authentication, authorization and configuration data and pushes this data and updates to the controllers. The ACS may reside off-site.

4. **Phone** - The mobile phone with NFC capabilities. It contains a credential which is the (claimed) identity of the phone and the data to prove this identity in the authentication process.

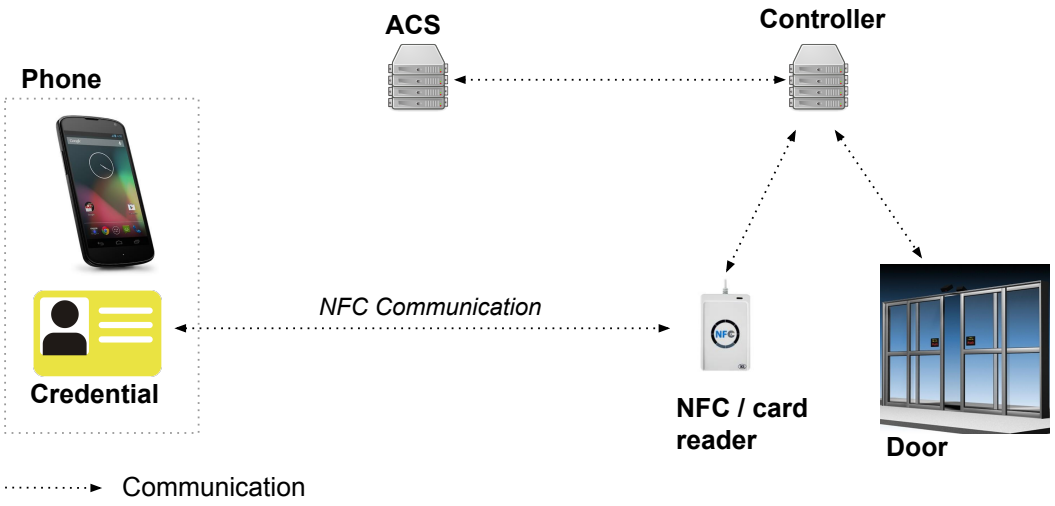

Figure 1.1: An example of a PACS that uses the mobile phone as a token

The following problems can be identified that are addressed in this work.

1. **NFC** - How can the phone communicate with the reader using NFC?

2. **Authentication** - How can authentication be accomplished in a secure way?

3. **Secure storage** - How can the credential be stored on the phone in a secure way such that it can not be stolen?

It is important to emphasize that in this work, we focus on an *on-line* PACS. This means that the controller is able to communicate on a frequent basis with the ACS through Ethernet, VLAN or other type of connection. In an *off-line* PACS, this communication can not be done frequently or not at all. Some off-line PACS use *off-line locks*. These are door locks with an embedded card reader that operate autonomously meaning that no controller is needed.

Focusing on an on-line system has two implications. The first is that, the behaviour of the controller and card reader can be modified more easily because the software on the controller / reader can be updated. For off-line locks this is typically more difficult.

The second implication of an on-line system is that the authorization data resides within the PACS (on the controller and ACS) and the authorization decision is made by the PACS. This is fundamentally different from off-line systems.

Off-line locks are typically devices with limited capabilities: they can not communicate with the ACS and can not receive any authorization data updates. Therefore, the authorization data resides on the smart card and thus the smart card makes the authorization decision.

This introduces additional security requirements. More specifically, the integrity of authorization data is more important compared to authentication data (credential). For example, if the authorization of a user is downgraded, i.e. the user has less rights than before, it is important that there is some mechanism in place that makes sure that the smart card / phone does not simply disregard the authorization update and continues to use its old authorization data.

The conclusion is that working with an on-line system means that there is more control over the behaviour of the reader and that the role of the mobile phone is limited to authentication (authorization is implemented by the PACS).

1.4 Scope

The scope of this research is limited by making the following assumptions.

1. As the time for doing this thesis is limited, it is assumed that a phone belongs to a particular person. This implies that when the authentication of the phone is successful, the systems assumes that the registered owner of the phone is authenticated. Of course, in real life, phones (and other types of tokens) can be stolen in which case this assumptions does not hold. How to deal with this is beyond the scope of this work.

2. This research will focus on NFC in Android-based phones. Android is an operating system for mobile phones that is currently the most popular [1]. Furthermore, most Android-based phones have NFC capabilities.

[1] The market share of Android was 81.3% in Q3 2013. [5]

1.5 Research Questions

The main research question of this work is:

- How can NFC-equipped mobile phones be used for secure identity authentication in a physical access control system?

The following sub questions are defined that aim to answer the main research questions:

1. What is NFC and how can it be used in mobile phones? The NFC technology and standards need to be studied.
2. Which security vulnerabilities exist in NFC-based mobile phone systems? Access control is a security related topic. Therefore, it is useful to investigate security in NFC.
3. Which security mechanisms can be used to protect against the security vulnerabilities? General knowledge about security related concept is required in order to deal with the vulnerabilities.
4. How can credentials on the phone be stored securely? Corresponds to problem 3 from Section 1.3.
5. Which authentication protocols can be used to securely verify the identity of a user over NFC? An authentication protocol performs authentication and should protect against the described security vulnerabilities. Corresponds to problem 2 from Section 1.3.
6. How can a mobile phone / NFC-based security token be integrated with an existing PACS? What is the best relation between the phone, the access point and access control server?
7. How secure and usable is this solution? It is important to reflect on the usability of the designed solution as it is the main motivation of using NFC in a mobile phone. Furthermore, it is important to reflect on the security since physical access control is a security related topic.

1.6 Organisation

This thesis is structured in parts. In Part 1, the background information and related work is discussed. This includes a chapter about the NFC technology (Chapter 2). Chapter 3 discusses other work that has been done on the development of NFC-based services.

Part 2 focusses on the security related challenges and solutions. The security vulnerabilities and threats are identified in Chapter 4. The next three chapters will focus on addressing those. Chapter 5 discusses general techniques about providing security such as cryptography and secure authentication. The next two chapters will address two important threats by discussing secure storage (Chapter 5) and authentication protocol (Chapter 6).

In Part 3, the studied technologies for secure storage and authentications are translated to a solution. In Chapter 9 two design choices are given and compared. In Chapter 10, one of the design choices is selected and the implementation of this is discussed. In Chapter 11, the implementation is evaluated by discussing functional tests, security aspects and usability aspects.

Finally, in Part 4 (Chapter 12 and 13) the conclusion and future work are discussed respectively.

Part I

Background and related work

Chapter 2

Near-field Communication

The NFC technology is based on radio technology: electromagnetic radiation of a frequency is used to transfer a signal with information through free space. NFC operates at a frequency of 13.56 MHz and can achieve transfers rates up to 424 Kbit/s. Communication is initiated by holding two NFC capable devices close (about 10 cm) together.

NFC devices include the following form factors:

1. **Tag** - A simple and cheap memory object (see Figure 2.1c).
2. **Contactless smart cards** - Smart cards have a microprocessor that can do more complex operations such as cryptography (see Figure 2.1a).
3. **Mobile phone** - A mobile phone with NFC can be used to communicate with contactless smart cards and tags. Furthermore, the phone can also behave as a contactless smart card or as a tag itself. In this case, the phone is passive and does not require battery power (see Figure 2.1b).

(a) A smart card (b) An NFC-equipped phone (c) An NFC-tag

Figure 2.1: NFC form factors

This chapter discusses the NFC technology. First, the modes of communication are discussed in Section 2.1. Subsequently, the NFC technology is compared to the Open

Systems Interconnection (OSI) model in Section 2.2. The standards and their place in the OSI model are discussed in Section 2.3. Section 2.4 discusses the architecture of a mobile phone with NFC. Section 2.5 takes a closer look at the NFC ecosystem.

2.1 Communication modes

NFC devices can be *active* or *passive*. Active devices have their own power supply whereas passive devices do not. The energy required by a passive NFC chip is received remotely from an active device by magnetic induction. Communication is only possible between one active and one passive device or between two active devices. In the first case, the active device will continuously generate a radio frequency (RF) signal that is used by the passive device as a source of energy. When two active devices communicate, only the sender will generate a RF signal and the receiver is 'quiet'. Note that the role of sender and receiver can swap throughout the communication process. Passive devices are typically contactless smartcards or tags.

The following communication modes are defined for an NFC device:

1. In **read/write mode** (R/W mode), the device is active and initiates communication with another passive device. The active device can send commands and the passive device replies to these instruction but does not initiate any commands itself [6] [7]. A device in this mode behaves as a smart card reader and thus it can be used to communicate with a smart card or with an NFC device in *card emulation mode*.

2. In **Card emulation mode** (CE mode) the NFC device behaves as a contactless smart card and is passive.

3. When using **peer-to-peer mode** (P2P mode), either one or both devices are active and able to initiate commands. This mode is always used between two devices that are both in peer-to-peer mode.

2.2 NFC layers

The Open Systems Interconnection (OSI) model [8] is a standard that describes the layers in a communication system. It partitions the functionality of this system into 7 abstract layers. Each layer has a specific responsibility and all layers together enable the communication. More information about the OSI model can be found in [9].

Figure 2.2 shows a comparison between the layers of the OSI model and the layers of NFC. This comparison is made by comparing the NFC standards with the OSI model description. This was based on an article found on the internet [10].

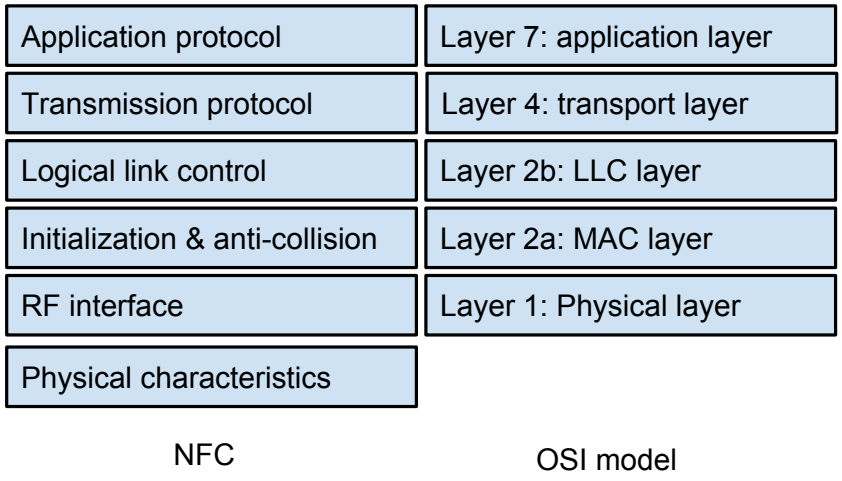

Figure 2.2: Comparison between OSI-model and NFC layers.

The NFC layers can be described as follows:

- **Physical characteristics** - Does not correspond to any of the layers of the OSI model. Physical characteristics should not be confused with layer 1 of the OSI model which is commonly known as the physical layer. Examples of physical characteristics include the size of a device, the temperature limits at which it should operate normally.
- **Radio frequency (RF) interface** - Corresponds to layer 1 (physical layer) of the OSI model. It specifies how bits are translated to a physical signal.
- **Initialization and anti-collision** - Layer 2 of the OSI model (Data link layer) is divided into two sub-layers: media access control (MAC) and logical link control (LLC). This NFC layer corresponds to the MAC layer.

  It deals with initialization process which includes the detection of NFC devices in the vicinity.

  It manages access to the medium. NFC communication always takes place between two devices but multiple devices can be in range which potentially interfere each others signals. Anti-collision deals with this problem.
- **Logical link control (LLC)** - This corresponds to the LLC sublayer (layer 2 from the OSI model). It provides reliable, in-order (data received in same order as sent) data transmission with flow control (a mechanism for controlling the speed of data transmission).
- **Transmission protocol** - Corresponds to layer 4 (transport layer) of the OSI model. This ensures reliable data transfer by doing error detection and recovery.

- **Application protocol** - Is marked by [10] as OSI layer 7 (application layer). It allows for transmission of application data. However, in my opinion it can also be marked as OSI layer 6 (presentation) as it also deals with the formatting of data.

2.3 Standards

A protocol suite can be seen as a set of protocols that each enables the behaviour described in one NFC layer. Figure 2.3 shows the protocol suite of NFC and the involved standards. The horizontal layers in this figure correspond to the NFC layers that were dicussed in Section 2.2.

This section discusses the standards related to NFC. The NFC standard is based on smart card standards. These standards are discussed in Section 2.3.1. Section 2.3.2 discusses the NFC standard. The remainder of the sections discuss other standards that are related to NFC.

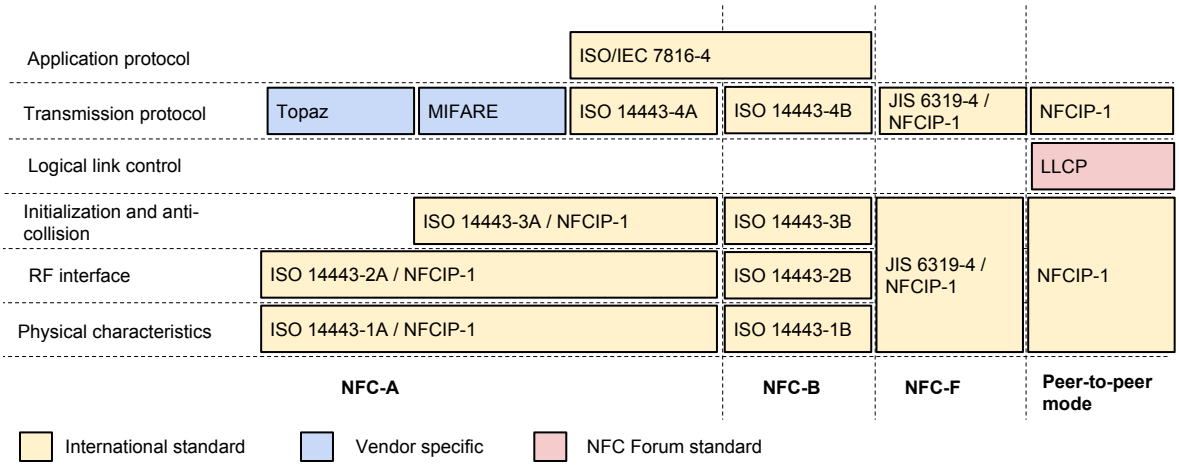

Figure 2.3: The NFC standards and protocol suite. Based on and adapted from [11]

2.3.1 ISO 14443 and JIS 6319-4

NFC was founded in the early 2000s by two major smart card companies, NXP Semiconductors (founded and previously owned by Philips) and Sony in an effort to improve the interoperability between various smart card and radio-frequency identification (RFID) technologies. It was designed to be compatible with the contactless smart card products of both companies. This includes NXP's MIFARE smart card which uses part of the international ISO 14443 [12] standard and Sony's FeliCa smart card that uses the JIS 6319-4 standard [13].

ISO 14443 is a standard for contactless smart cards that consists of 4 parts (ISO 14443-1 - ISO 14443-4). Refer to Figure 2.3 and Section 2.2 to see which part of this standard covers which aspect. Furthermore, the ISO 14443 standard is available in two variants (type A and B) that differ in the way they handle radio frequency power and signal interface. NXPs MIFARE smart card uses ISO 14443 part 1 - 3 type A and a proprietary transmission protocol instead of ISO 14443-4.

The JIS 6319-4 is comparable to ISO 14443 in the sense that it covers the same layers. In fact, at some point it was submitted to be standardized as ISO 14443 type C but it was rejected.

2.3.2 NFCIP

The actual NFC standard, referred to as *Near Field Communication Interface and Protocol* (NFCIP), consists of two parts: NFCIP-1 (standardized in ISO-18092 [14] and ECMA 340 [15]) and NFCIP-2 (standardized in ISO 21481 [16] and ECMA 352 [17]).

The NFCIP-1 standard was the initial NFC standard. It was based on ISO 14443 part 1 to 3 type A [12] and JIS 6319-4 [13] and is backwards compatible with both those standards. An NFCIP-1 device can communicate with smart cards in read/write mode and can behave as a smart card in card emulation mode (see Section 2.1). NFCIP-1 extends the existing standards by enabling the communication between two active devices which is called peer-to-peer mode.

The NFCIP-2 enables backwards compatibility with ISO 14443 type B and ISO 15963 [18]. The latter is another standard for smart cards. Furthermore, NFCIP-2 specifies a selection process that facilitiates this backwards compatibility. When an NFCIP-2 device detects another device it could be an ISO 14443, NFCIP-1 or ISO 15963 device since all these technologies use the 13.56MHz frequency. The selection process determines which technology it is dealing with.

When NFC is used in a mode that is compatible with one of the smart card standards, this is referred to as NFC A (ISO 14443 type A), NFC B (ISO 14443 type B) and NFC F (JIS 6319-4).

2.3.3 LLCP

LLCP is defined by the NFC forum. Besides the International Organization for Standardization (ISO) and European Computer Manufacturers Association (ECMA) standards that were mentioned previously, the NFC forum develops relevant standards. The NFC forum is a non-profit cooperation organisation consisting of more than 170 members including manufacturers, applications developers, financial services and more [19]. Its goal is to advance the use of NFC by developing specifications, promote the technology and ensure that NFC products comply with the standard.

LLCP can only be used with NFC peer-to-peer mode. It defines the upper part of layer 2 in the OSI model. LLCP offers both connection-oriented and connection-less service. Connection-oriented provides reliable, in-order (data received in same order as sent) data transmission with flow control (a mechanism for controlling the speed of data transmission). Connection-less does not provide these services.

2.3.4 ISO 7816-4

ISO/IEC 7816 [1] is a series of standards historically created for contact smart cards and later also used for contactless smart cards. It consists of 15 parts which includes the specification of the physical characteristics, the electrical interface (only for contact smart cards) and the smart card life cycle.

Part 4 of ISO 7816 specifies an application layer protocol. This protocol is used for contact and contactless smart cards and can also be used with NFC. It specifies application protocol data units (APDU) which is a message format that is used between a reader and smart card.

Moreover, ISO 7816-4 enables a multi-application environment where applications (which are called *applets*) can be installed onto one card. This is shown in Figure 2.4 where one card contains applets for a bank card, train ticket and student card. The card reader can select the appropriate applet when the card is presented to a reader by first communicating with the *card manager* applet. This applet is responsible for managing and selecting applets on the card. After that, the reader can communicate with a specific applet directly.

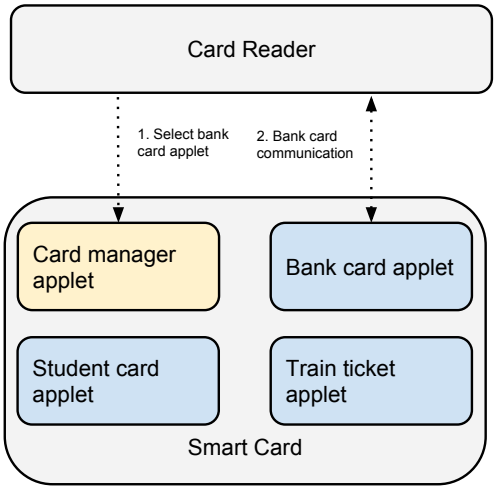

Figure 2.4: The organisation of a ISO 7816 smart card.

[1] This standard is maintained jointly by the International Standard Organisation (ISO) and International Electro-technical Commission (IEC)

2.4 NFC in mobile phones

Figure 2.5 shows the architecture of an NFC-equipped mobile phone.

1. The **application processor** is the phone's main processor which hosts the operating system and runs applications.

2. The **NFC Interface** is the radio interface where signals can be sent and received.

3. The **NFC controller** is the main component of all NFC related functionality. It controls the radio interface and does preprocessing on the data.

4. The **secure element** as separate integrated circuit (a smart card chip) in the phone that provides secure storage and a secure runtime environment. Secure storage means that information stored on the chip is physically protected and can not be extracted. A secure execution environment means that the execution of programs on the chip can not be tampered with. Secure elements are based on smart card technology and are evaluated and certified according to high security standards [20] and therefore they provide good security.

   The secure element also conforms with the organisation described in ISO 7816-4 (see Section 2.3.4).

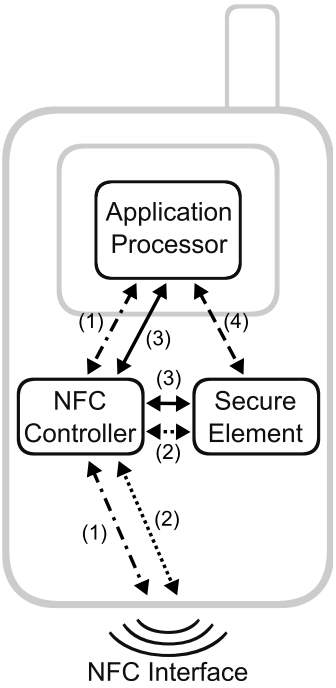

Figure 2.5: The possible paths of communication between NFC controller, secure element and phone processor, copied from [20]

Figure 2.5 illustrates the possible paths of communication between the components within the phone. These paths are all bidirectional and depend on the chosen NFC mode and on the internal design of the mobile phone. A summary of the paths is provided in tables 2.1 and 2.2. This shows which path of communication are used in each NFC mode. Furthermore, it shows how the phone can have access to the secure element.

Name	**NFC Mode** (Section 2.1)	
Peer-to-peer	P2P	Path 1
Reader/write	R/W	Path 1
Card emulation	CE	Path 2
Host card emulation (HCE)	CE	Path 1

Table 2.1: NFC modes and corresponding paths of communication as shown in Figure 2.5

SE access from phone's CPU	
Via NFC controller	Path 3
Direct	Path 4

Table 2.2: The paths of communication between the phone's CPU and the secure element as shown in Figure 2.5

When using NFC in peer-to-peer or reader/writer mode, the data is routed between the NFC radio interface, the NFC controller and the phone's CPU (path 1). A software application on the phone's main CPU can access the NFC interface through an application programming interface (API) provided by the operating system.

In card emulation mode, the data is routed between the NFC radio interface, the NFC controller and the secure element (path 2). The phone's CPU is not involved. Moreover, the phone is the passive device in this mode of communication meaning that it still works when it is switched off or when the battery is dead.

Host card emulation (HCE) is similar to card emulation (the same NFC mode is used). However, no secure element is used. The behaviour of the secure element is emulated in software on the main CPU (path 1).

The phone's application also has access to the secure element. This can be either a direct connection (path 4) or via the NFC controller (path 3) depending on the design of the phone. It can be used for secure application data storage. Furthermore, it can be used for OTA (over-the-air: WiFi, UMTS, 3G, etc.) management of applications on the secure element.

Various communication standards are defined for the communication between the components in the mobile phone. The *Single Wire Protocol* (SWP, standardized in ETSI TS 102 613 [21]) and the *NFC Wired Interface* (NFC-WI, standardized in ECMA 373 [22]) are two alternative standards that specify the communication between the NFC controller and the secure element. The main difference between the two is that SWP uses one physical line whereas NFC-WI uses two [23].

ETSI standardized the Host Controller Interface (HCI) in ETSI TS 102 622 [24] which defines the logical interface between a NFC controller and the application processor. It also enables the communication between the application processor and a secure element through the NFC controller (path 3). Furthermore, the NFC forum has standardized NCI (NFC controller interface) [25], an alternative for HCI.

2.5 The NFC ecosystem

Figure 2.6 illustrates the NFC ecosystem as presented in [26] [27]. The following paragraph discusses the components:

1. **Service Provider Component** (SP). This is the (legacy) component that implements the actuals service, i.e. business logic, without any NFC service. For example, for a banking application the service provider handles money transactions between accounts.

2. The **Service Provider Back Office** (SPBO) extends the functionality of the service provider with NFC related functionality. The implementation of the SPBO is up to the developer of the NFC service. It can communicate with both the NFC reader and the mobile phone. Which of the two paths of communication is used and the information that is sent, is a design decision that the developer will have to make. Thus, the SPBO provides an interface between the NFC equipment and the (legacy) business logic implementation enabling these two entities to integrate.

3. The NFC capable **phone** containing a **secure element** which architecture is described in Section 2.4

4. The **NFC reader** is an NFC device that communicates with the phone. It is located at the POS, i.e. the location where the user obtains the service. The NFC reader communicates with the SPBO, in order to provide the service to the customer.

5. The **Trusted Service Manager** (TSM) is a trusted third party entity providing a service that allows service providers to have remote access, i.e. remotely isnstall, manage and remove applets on the secure element. TSM services are typically provided and implemented by specialized companies which will charge a fee for this service.

6. The **Controlling Authority** (CA) verifies that applets installed via the TSM are harmless before they are installed on the secure element. No details are provided on how this is done and who is responsible for implementing this functionality.

Important stakeholders within the ecosystem are the *mobile network operators* (MNOs) and *service providers* that are interested in using NFC. The MNO supplies the subscriber identification module (SIM) in the phone which is used to authenticate the phone to the mobile network. The SIM is actually a smart card with a ISO 7816-4 organisation (see Section 2.3.4). It contains a SIM applet but other applets can be installed on it as well. Therefore, the SIM can also be used as a secure element.

However, the use of the SIM is restricted by the MNOs. Since there are many MNOs and phone manufacturers, a SP potentially has to make business agreements with many other parties which is impractical and expensive. The solution involves the introduction of a central entity called a trusted service manager (TSM). A TSM mediates between secure element owners and service providers (SP).

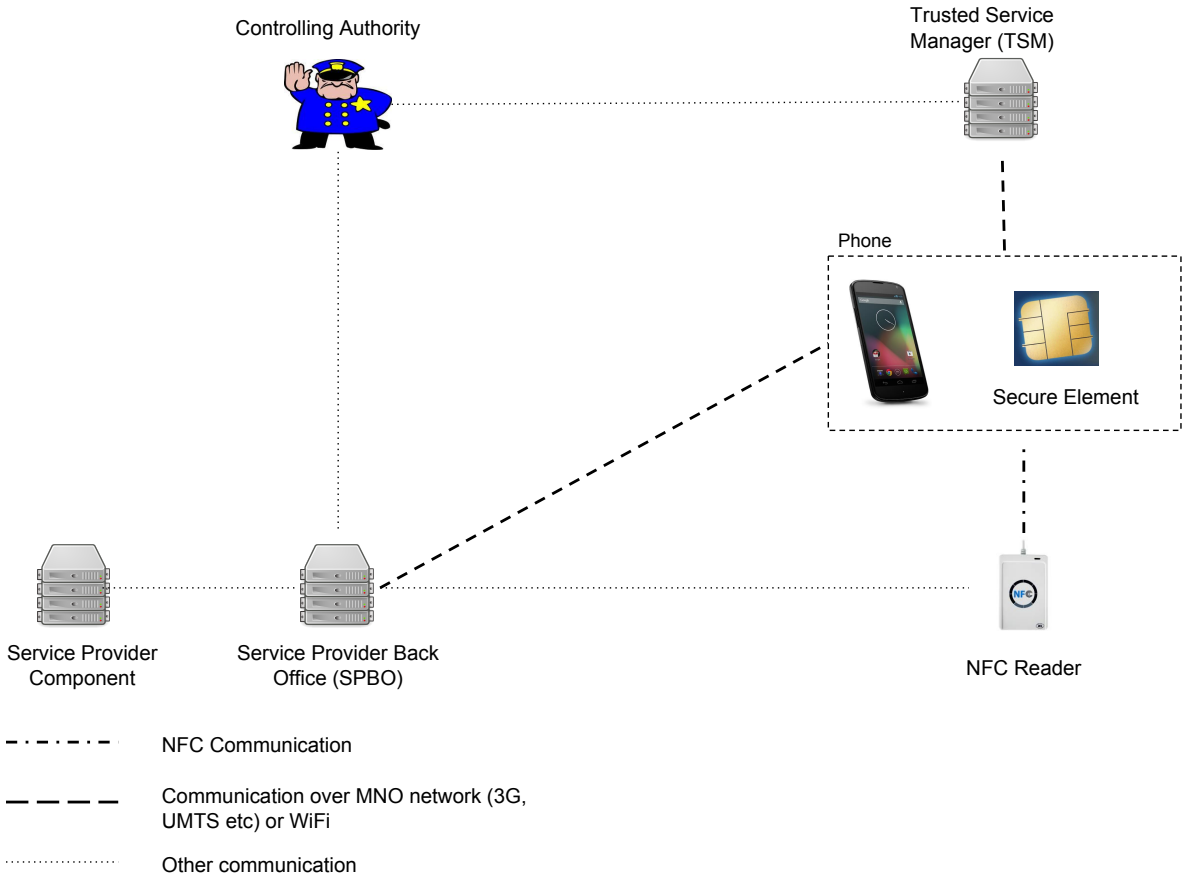

Figure 2.6: Overview of NFC ecosystem, Based on and adapted from [27]

Chapter 3

Related work

This chapter provides an overview of related work. Section 3.1 discusses a framework for the development of NFC services on mobile devices. Section 3.2 reflects on the current problems in NFC and discusses why the proposed framework is currently not widely deployed.

3.1 StolPaN/DIAD NFC framework

StolPaN (Store Logistics and Payment with NFC) is a pan-European consortium consisting of universities and companies. Their main goal is to *"develop commercial and technical frameworks for the remote management of NFC-enabled services on mobile devices"*, from [26]. These frameworks will enable NFC service implementation for multiple phone platforms. DIAD NFC is a Hungarian consortium which develops commercial NFC services by extending the StolPaN results. All the members of DIAD NFC are also a member of StolPaN [28].

This section provides an overview of the stolPaN/DIAD_NFC work. The architecture of the mobile phone is presented in 3.1.1. An analysis of the security features is discussed in 3.1.2.

3.1.1 Architecture of NFC based mobile

In [28] [27] Benyo et al. describe a virtual machine approach. The idea is illustrated in Figure 3.1. The numbers in this figure indicate an NFC or API connection. These numbers are used to refer to these connections from the text. Two software components are installed onto the mobile device. The **MidLet** is a software application that is installed and executed on the phone's main CPU. The universal **CardLet** is an optional applet that is installed on the secure element.

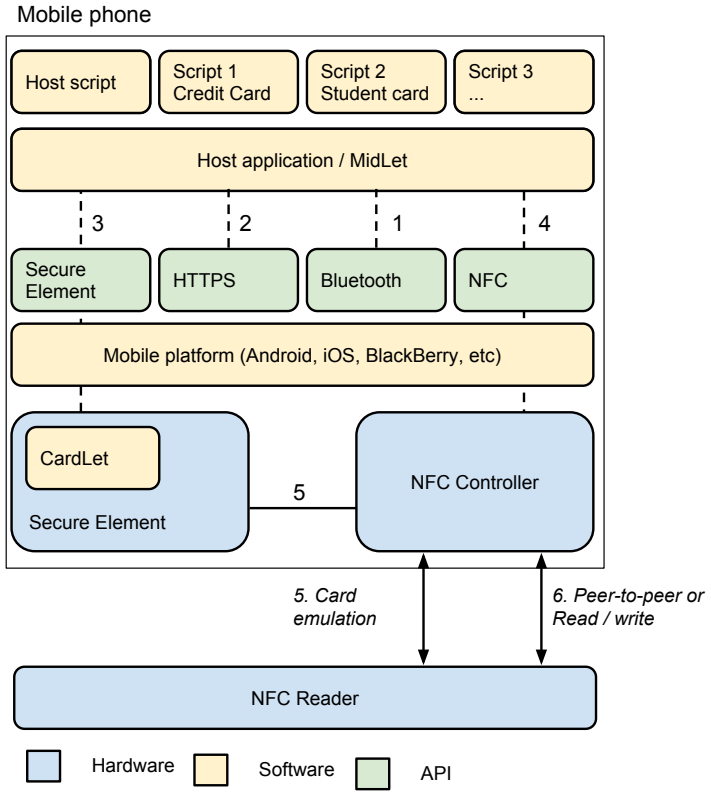

Figure 3.1: Architecture of the mobile device, figure taken and adapted from [27]

MidLet

The MidLet functions as a script interpreter which can host multiple customized *service scripts* that each correspond to an NFC service. For this reason, the MidLet is also referred to as the *host application*. The service script implements the logic for a specific NFC service that needs to take place on the phone.

The scripting framework supports all basic programming language functionality such mathematical operations, functions and control flow commands (if/else, switch/case etc). Moreover, it supports commands for constructing a basic graphical user interface (print text, create buttons, text input fields etc). An API for both NFC (connection 4 in Figure 3.1) and other link types (connection 1, 2 in Figure 3.1) is also provided by using the underlying operating system APIs for this.

The host application runs on multiple platforms, meaning that the scripts are platform-independent. The host application needs to be installed once on the mobile phone. Subsequently, multiple 'scripts' can be installed onto the phone by either holding the phone to an NFC tag or by going to a website. In both cases, the actual script is downloaded

from a back-office server. When using an NFC tag, the host application is responsible for reading the tag, containing a reference to the script itself, and downloading the script. Alternatively, a user can select a script on a website prompting the service provider to send an SMS to the mobile device. This message contains a link and in turn, it will prompt the host application to start and download the script.

The host application provides an application with a graphical interface (host script, in Figure 3.1) allowing the user to select a script. For example, when a user wants to pay he can select the credit card script. Furthermore, it allows the user to install and remove scripts.

CardLet

Apart from the MidLet, CardLet applications can be installed onto the secure element. Two options are available that can coexist within the same secure element: a universal and/or a dedicated cardlet.

A universal cardlet, is developed specifically for use in the framework. The functionality of the cardlet in this case is limited to encrypting and decrypting data. The universal cardlet can handle the encryption for multiple installed scripts/services, but guarantees the strict separation of data.

The universal CardLet can only be used with NFC peer-to-peer mode. In this case, the NFC communication is handled by the script component that runs on the host application (connection 6 in Figure 3.1 and path 1 in Figure 2.5). If required, the script can obtain credentials from the universal applet on the secure element by using API connection 3.

Alternatively, the cardlet can be a dedicated applet developed by the service provider and used for one particular service. In this case, NFC card emulation mode is used. The NFC communication is handled by the applet and not by the service script (connection 5 in Figure 3.1 and path 2 in Figure 2.5).

3.1.2 Security Analysis

The security issues are discussed by Benyo et al. in [29]. The following threads are identified:

- **Threat 1** All data that is stored on the main storage device in the phone is potentially insecure.
- **Threat 2** The separated runtime environment provided by the operating system on the phone's main CPU is potentially insecure [30].

- **Threat 3** The API that is used to access the secure element from the phone is potentially insecure.

The framework distinguishes three levels of security within the host application depending on if and how the secure element is used. The difference is the extent to which they deal with the security threats presented earlier. A summary of security levels and threats is provided in table 3.1. Based on the security requirements, the service provider can select a suitable level.

1. Low security level: the secure element is not used at all.
2. Medium security level: the universal cardlet on the secure element is used.
3. High security level: a dedicated cardlet is used on the secure element.

	Threat 1	Threat 2	Threat 3
Low			✓
Medium	✓	✓	
High	✓	✓	✓

Table 3.1: Security levels and resistance to the security threats

Low security level

Although the host application provides a strictly separated runtime and storage environment for each individual script, this cannot be considered secure. One reason for this is that the mechanisms that implement this separation rely on the mobile phone platform which might contain bugs and security vulnerabilities as is shown in [30] for the Android platform. Consequently, an adversary might be able to bypass these mechanisms and obtain access to sensitive information (threat 2).

Another reason is that the physical data storage hardware (flash drive or hard disk) can be accessible enabling one to gain access to all data including keys that are used for encrypting the data (threat 1). Hence, the usage of the phones internal memory for data storage as opposed to a secure element is considered to have a low security level.

Medium security level

When using a medium level of security, a universal cardLet is installed on the secure element which can be used for data storage of multiple services. It guarantees that the data for each of the services are strictly separated from each other. The data on this cardLet is managed through the host application which provides an API (API connection 3, Figure 3.1). As the host application runs on the phone itself, this API between the secure element and host application is a potential weak spot (threat 3).

High security level

Using a dedicated cardLet is considered as highly secure. The sensitive information is not communicated between the secure element and phone, meaning that the API in threat 3 is not used. Furthermore, the information does not reside on the phone's main CPU making it immune for threat 1 and 2. However, this option requires considerably more effort in the sense that the service provider is responsible for installing the cardLet on the secure element by itself which may require a TSM.

3.2 The problems in NFC

The first mobile phone with NFC capabilities was introduced 7 years ago. Today, many phones are equipped with NFC capabilities. However, few services have been deployed that use this technology. This is not solely due to technical limitations. A major problem is the NFC ecosystem as explained in Section 2.5 in which various stakeholders have different conflicting interests.

The secure storage of a credential in a mobile phone is a requirement for many types of NFC-based services including physical access control. A secure element is designed for this purpose but its use is restricted by its owners. Platforms have been proposed, developed and tested that regulate the use of the secure elements in mobile phones [28] (see Section 3.1) and [31] (see Section 2.5). This involves the introduction of a central entity called a trusted service manager (TSM) (see Section 2.5). In practice, however, these platforms have not been widely deployed. The reason for this is that the key stakeholders have not been able to work out a generally accepted business model [26].

This restriction does not only limit the ability of a phone to store credentials securely. It also limits the phone's ability to communicate with existing smart card infrastructures. The secure element allows the phone to use card emulation mode. This allows a phone to communicate with ISO 14443 [12] and JIS 6319-4 [13] smart card readers which are used in many existing payment systems, PACS and other services. If the secure element in the phone can not be used, this means that the existing smart card infrastructure will have to be replaced which will result in additional costs.

Two recent developments may provide the answer to the aforementioned secure element restrictions.

1. As of version 4.3, Android introduced "hardware-backed" credential storage for devices that support this [32]. This is a potential alternative for using a secure element.

2. As of version 4.4, Android supports host card emulation (HCE, see Section 2.4) which allows the phone to communicate with existing smart card infrastructures without using the secure element.

Currently, these two features are only available on Google's latest Android phone (the Nexus series) and not on other Android-based phones. The reason for this is that both hardware-backed credential storage and HCE requires certain hardware to be installed into the phone. However, the fact that a large company such as Google supports the HCE approach (as opposed to the secure element approach), may prompt other phone manufacturers to follow this approach as well. Some other companies in the payment and NFC industry already support this approach (for example Visa [33] and NXP [34]). A number of NFC payment services have been announced since the introduction of HCE [4] [35]. For these reasons, it is worthwhile to investigate the potential of this approach in physical access control.

Part II

Security related challenges and solutions

Chapter 4

Security Analysis

Physical access control is a security related topic. Therefore, it is useful to obtain insight into the potential security issues. In security terminology, a distinction is made between *vulnerabilities* and *threats*. A vulnerability potentially affects the security whereas a threat affects it directly.

This chapter analyzes the security. First, the definition of a secure system is given in Section 4.1. In Section 4.2, the system domain is discussed for which the security is considered. Subsequently, the vulnerabilities within this domain are discussed in Section 4.3. In Section 4.4, the threats are identified from the vulnerabilities.

4.1 Security Definitions

A secure system must provide a set of properties that are commonly known as the *CIA triad* (Confidentiality, Integrity, Availability) [36] [37].

1. **Confidentiality**
   (a) **Data confidentiality** - Ensures that confidential data is not accessible by unauthorized parties.
   (b) **Privacy** - Ensures that a person can control what information related to them is stored in the system and to whom it is disclosed.
2. **Integrity**
   (a) **Data integrity** - Ensures that data in the system is not modified by unauthorized persons or entities.
   (b) **System integrity** - Ensures that the system functions as intended.
3. **Availability** - Ensures that the system is available when required.

4.2 System domain

The system domain indicates for which parts of the system we consider the security. This is shown in Figure 4.1. The mobile phone (1) is part of system because it stores the credential and it handles the NFC communication with the NFC reader. Furthermore, the NFC communication channel (2) is considered in the security analysis because it used for transferring authentication information.

With regard to the security of the PACS, we limit our scope to the reader (3). The reader is typically located in a publicly accessible location meaning that it can be tampered with. The security of other components within the PACS is important for the security system as a whole but is not considered here. The reason for this is that the focus of this work is the usage of the mobile phone as a token in a PACS using NFC. Thus, we extend an existing PACS by adding the mobile phone and the NFC technology to it. Furthermore, a wide variety of PACS are available which makes analysing the security in all those systems infeasible.

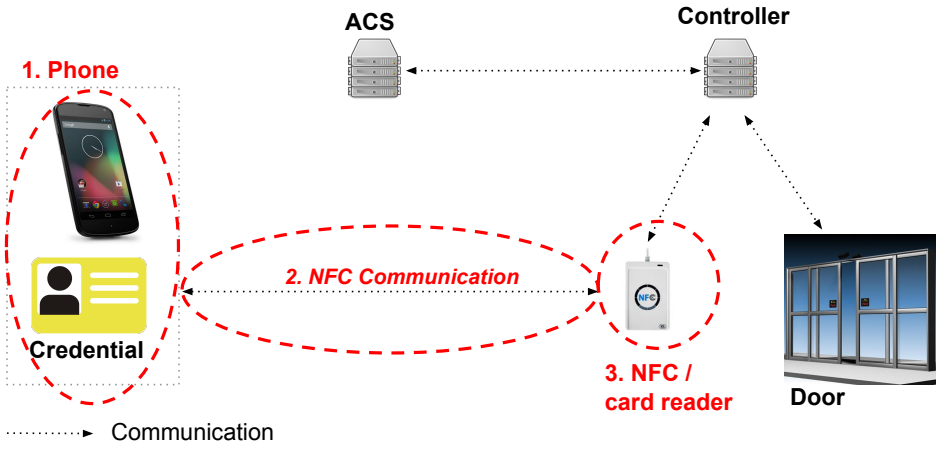

Figure 4.1: System domain

4.3 Vulnerabilities

This section discusses the vulnerabilities within the system domain.

4.3.1 Eavesdropping & replay

NFC utilizes the air as a medium for transferring data through free space. Potentially, these waves can be received or modified by an attacker. Figure 4.2 illustrates an eaves-

dropping and replay attack. In principle, it is possible to receive data for anyone that is close enough to a transmitter. This is called eavesdropping.

An important question with regards to eavesdropping NFC traffic is the maximum distance at which an adversary can still receive the data. Preferably this distance should be as small as possible. NFC is designed for communication up to 10cm.

The range of a transmitted signal depends on many factors. The most important one is whether the device is in active or passive mode. Active devices can have an range up to 10m whereas passive devices only have 1m of maximum range [38]. The reason for this is that passive devices obtain energy for generating a signal from an active device. The amount of energy that can be transferred to the passive device is limited by the technique used for this. Active devices use their own power supply and thus its signal is stronger.

Other factors that can affect the range are obstacles in the environment that can block the signals (e.g. walls, buildings). The quality of the attackers receiver equipment (e.g. antenna type, receiver quality) also determines the distance at which the signal can still be received.

A consequence of eavesdropping is that an unintended receiver can intercept sensitive information between a sender and receiver. Hence, the confidentiality of the information is violated. An extension of this attack is the replay attack.

Suppose that the mobile phone authenticates to the NFC reader by sending an unprotected 'secret' identifier. An attacker can eavesdrop the communication between phone and the reader, receive the identifier and resend or *replay* the identifier from his own phone when located at the door enabling him to illegally gain access to the building.

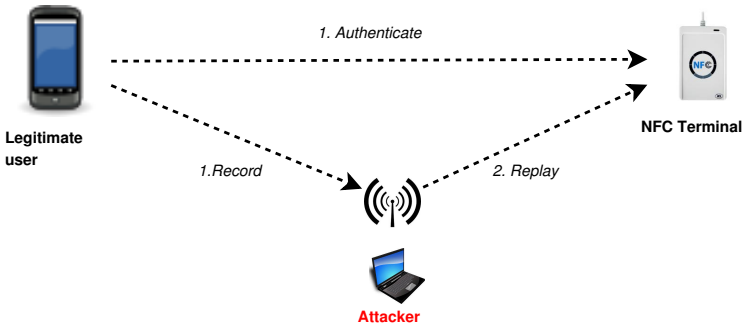

Figure 4.2: Eavesdropping & replay attack. The numbers indicate the order of the events.

4.3.2 Data corruption & modification & insertion

An attacker is not limited to receiving data between the phone and NFC terminal. It can send data as well with various purposes. The simplest possibility is **data corruption**, shown in Figure 4.3a, where the attacker corrupts the signal such that the intended receiver cannot understand it. This is can be qualified as a *Denial of Service* attack (DoS): the service, in this case access control, becomes unavailable to its intended users. Legitimate users that arrive at the door cannot gain access to the building. This affects the availability of the system.

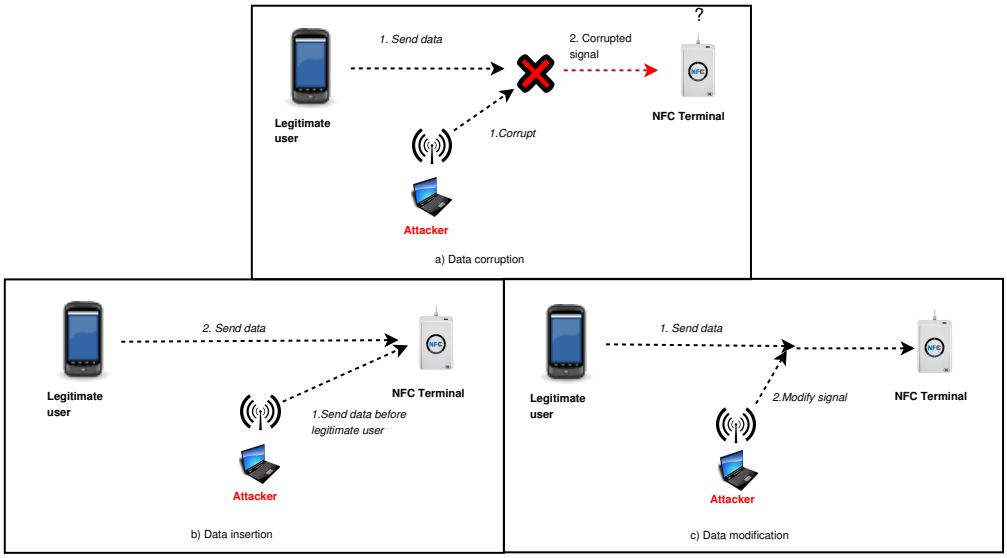

Figure 4.3: Data corruption, insertion and modification. The numbers indicate the order of the events.

Data insertion is illustrated in 4.3b. An attacker replies to a received message before the legitimate receiver can do so. This can only be done if the legitimate receiver takes a long time before answering, i.e. has a high response time. If the attacker's spoofed reply is not sent before the legitimate receiver starts sending, the two signals will overlap and interfere and thus the spoof fails. Data insertion affects the integrity property.

Data modification is illustrated in Figure 4.3c. An attacker alters the signals such that the legitimate receiver receives a valid but modified signal. In the case of NFC this attack is difficult to perform due to the short distance between the phone and the NFC terminal however it is feasible depending on the modulation and encoding used on the physical layer.

As shown in table 4.1, the modulation used in NFC is 10% or 100% Amplitude-shift keying (ASK) with either *Manchester* or a modified version of *Miller* encoding depending

on the data rate and whether the device is active or passive. Details about about these modulations scheme are not discussed here. When using 100% ASK with the modified version of Miller encoding it is feasible to modify a limited number of bits in the signal but when using 10% ASK with Manchester encoding, data modification is feasible on all bits in the signal [38].

The consequence of this is that the integrity of the information is violated. Every data that is received cannot be trusted as it is potentially changed by an attacker.

Data rate (KBPS)	**Active device**	**Passive device**
106	Modified Miller, 100% ASK	Manchester, 10% ASK
212	Manchester, 10% ASK	Manchester 10% ASK
424	Manchester, 10% ASK	Manchester 10% ASK

Table 4.1: The modulation used given the data rate and NFC mode, copied from [39]

4.3.3 Man-in-the-middle-attack

Figure 4.4 illustrates the idea of a man-in-the-middle-attack (MiTM). Two parties, *Alice* and *Bob*, believe they communicate with each other but in fact they communicate via *Eve*, an attacker. Eve can accomplish this by intercepting messages between by Alice and Bob, thus eavesdrop the connection and disturb the transmission such that the intended receiver does not receive it. Subsequently, Eve can modify the intercepted message and forward it to the receiver.

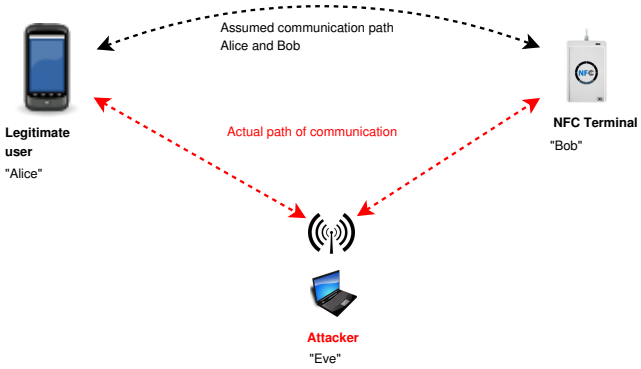

Figure 4.4: Man-in-the-middle-attack

The feasibility of this type of attack can be investigated by looking at communication between one active and passive device and between two active devices. Suppose that Alice is an active device and Bob is passive. In this mode, Alice will constantly generate a RF signal in order to supply energy to Bob. If Eve is close enough to Alice, she can

eavesdrop messages sent by Alice. Next, she will need to make sure that Bob doesn't receive the signal by disturbing the signal. However, this disturbance can be detected by Bob. Hence, it is possible for Bob to terminate communication with Alice when detecting disturbance.

Assuming that Bob does not detect the disturbance, the next step for Eve would be to forward a modified message to Bob. This, however, is infeasible as Alice is constantly generating a RF signal. The RF signals of Alice and Eve will collide and thus Bob is not able to understand it.

When both Alice and Bob are active devices, a RF signal will only be generated when one of the two is transmitting. Again, suppose that Eve successfully disturbs Alice's signal and that this is not detected by Bob. Now Eve is able to send a modified message to Bob, because Alice remains quiet after the transmission has finished. However, in this case Alice will also receive Eve's modified message meaning that she can detect the attack [38]. The conclusion is that a MiTM attack is possible in NFC but unlikely to succeed because it is easy to detect. It is unknown, however, to what extent such detection mechanisms are implemented in current NFC systems.

The consequence of this is that potentially, both *confidentiality* and *integrity* are violated.

4.3.4 Relay attack

In this type of attack a secondary communication channel, i.e. another communication channel than NFC, is used to tunnel communication between a reader and card (or phone). This attack became a major problem with the introduction of contactless smart cards. Figure 4.5 shows the normal communication between a contactless smart card and a smart card reader. In the case of a relay attack, two additional components are needed as shown in Figure 4.6.

1. The **mole** is a device acting as a reader and communicates with a legitimate smart card.

2. The **proxy** emulates a legitimate card by communicating with the mole.

This set up can be exploited in an access control system as follows. An attacker is located at a smart card reader near the door with a proxy. An accomplice of the attacker is located near an unaware victim carrying a legitimate card. The accomplice uses the mole to initiate contact with the card. The communication is relayed to the attacker at the door, who will present the proxy to the reader at the same time. The proxy is able to behave as a legitimate card by communicating with the real card through the mole and thus the attacker can get access to the room.

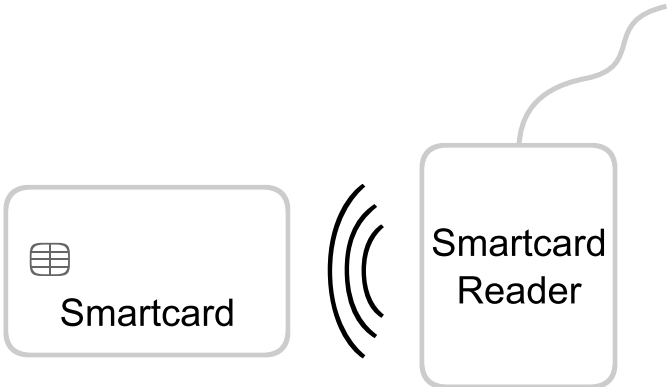

Figure 4.5: Normal communication between card and, copied from [40]

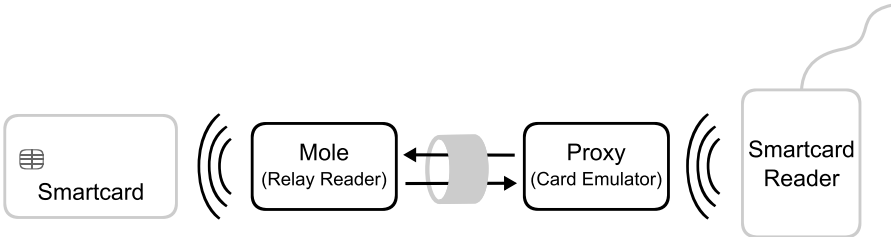

Figure 4.6: Relay attack, copied from [40]

Although, the described scenario illustrates the relay attack when using smart cards, it also works for NFC in mobile phones. When the phone operates in card emulation mode, the set up is the same except that the card is now a mobile phone. This is called an *external relay attack*.

The inclusion of NFC equipment inside a mobile phone introduces a new type of relay attack called an *internal relay attack*. Roland et al. [38] demonstrates how to perform a relay attack on Google Wallet; a mobile NFC based payment service. Figure 4.7 illustrates how this attack can be performed. Malicious relay software is installed on the victim's phone (on the left). This software forward messages between the proxy and the secure element in the phone of the victim that contains the Google Wallet applet. Thus, instead of an external hardware component, the mole is now malicious software that is installed on the victim's phone.

The proxy is another mobile phone that uses host card emulation to communicate with an NFC reader (point-of-sale terminal in 4.7) and its internet connection to communicate with the victim's phone. Figure 4.7 shows how APDUs (see 2.3.4) can be tunnelled between the victim's phone and the proxy.

The consequence is that an attacker can make use of a service on behalf of the victim without the knowledge and permission of the victim.

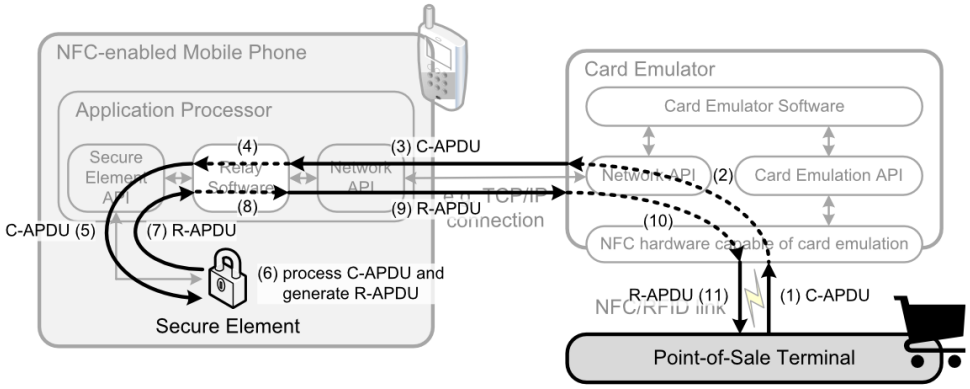

Figure 4.7: Relay attack on Google Wallet, copied from [40]

4.3.5 Vulnerabilities in mobile phones

In the last 15 years, the mobile phone has evolved from a device solely used for calling to a fully functional computer that can run custom applications for various other purposes. This introduced various security vulnerabilities to the mobile phone [41]:

- **Implementation error** - Legitimate software that is installed on the phone (including the operating system) may contain errors. These can be exploited by an attacker potentially allowing him to execute arbitrary code on the phone.
- **Incompatibility** - Incompatibility between applications or between applications and the operating system may disable those applications.
- **User unawareness** - Mobile phone users may be unaware of certain risks as their technological knowledge is limited. For example, a user can install malicious software or connect to an untrusted WiFi network. The phone can also be configured improperly, e.g. enabling a bluetooth without setting a password.
- **Vulnerabilities of wireless network** - information that is sent over the WiFi connection can be eavesdropped and modified.
- **Vulnerabilities of external objects** - External objects such as improperly configured WiFi access points, PC with malicious software may impose a risk to the phone.

The Android operating system provides a number of security mechanisms that deal with some of the vulnerabilities. However, Android is also vulnerable to implementation errors. Attacks are known that circumvent these security mechanisms by exploiting implementation errors. For example, Davit et al. [30] describe a privilege escalation attack where an application obtains access to resources it is not supposed to.

Rooting is a technique to gain full access to the phone's operating system. It is sometimes done intentionally by the owner of the phone in order to overcome restrictions imposed by the phone's manufacturer. However, it could also be triggered from malicious software. A potential consequence of this is that all security mechanisms provided by the operating system can be circumvented.

The conclusion is that the mobile operating system does not provide security and can not be trusted. This does not only affect the applications and data on the mobile phone. It also affects the security of the secure element [42]. The secure element itself is secure as it is based on smart card technology but embedding it in a phone introduces the risk of a denial of service attack.

When the number of authentication attempts to the secure element exceeds a certain threshold it will enter an irreversible BLOCKED state meaning that applets can no longer be installed or removed. Applets that are already installed continue to operate normally. Applications on the phone can communicate with the secure element (path 3 or 4 in Figure 2.5 on page 18) through an API and thus these application can trigger the SE to go into the BLOCKED state on purpose. This API is protected but this protection requires that the underlying operating system can be trusted which is not the case.

4.3.6 Untrusted reader

The reader is located at a publicly accessible location. Potentially, it can be tampered with. For example, a reader can be replaced or modified without the user noticing. This can be used to enable the attacks that were mentioned earlier in this chapter (eavesdropping, data insertion/modification/corruption and MiTM). The difference is that in this case, the attack is not done on the NFC communication channel but inside the reader.

For example, in Section 4.3.3 it was shown that a MiTM on the NFC link is possible but difficult to perform. However, when the reader is modified it is possible that a legitimate looking reader is in fact the man-in-the-middle. This increases the feasibility of these types of attack.

4.4 Security threats

The identified vulnerabilities can be translated into threats. Table 4.2 shows how the threats are related to the vulnerabilities and security definitions and in which chapters / sections they are addressed.

- **Threat 1: insecure storage** - The storage in the phone is insecure allowing the credential in the phone to be stolen.
- **Threat 2: relay attack** - Internal or external relay attack.
- **Threat 3: insecure communication channel & untrusted reader** - Due to the insecure nature of the NFC communication channel and the fact that readers can not be trusted, an attacker can apply certain attacks such as eavesdropping and MiTM that ultimately allow it to authenticate successfully to the system on behalf of a victim.
- **Threat 4: denial of service attack** - A denial of service attack can be performed that render the phone unusable for authentication.
- **Threat 5: privacy** - An attacker can eavesdrop identity related information on the NFC interface without a user's knowledge and approval.

Threat 1 can be solved by using a component in the phone that provides secure storage (such as secure element). Threat 3 and 5 can be solved by using an authentication protocol. Threat 2 is currently difficult to prevent, however the possibility of relay attack can be minimized by taking a number of precautions.

To the best of my knowledge, no extensive work has been done in the field of DoS attacks on smart cards and/or NFC (threat 4). A possible reason for this is that there is not much to gain for an attacker. In the context of the internet, DoS attacks are an important topic because it is easy to perform, requires low cost but potentially it can disable services to many customers.

In the case of NFC, there are two forms of DoS attacks. The service can be disabled by disabling the reader or the NFC communication channel (data corruption see Section 4.3.2). Another possibility is that malicious software in the phone disables the credential in the phone (see Section 4.3.5). The first requires that the attacker is physically present at an access control location which makes it significantly harder. The second form involves the installation of malicious software on a user's phone.

Both types of attacks are feasible and may be desirable for an attacker when it targets a specific user or a small group of users. But compared to DoS attacks in the internet, it is not as easy to harm 'many' users. Furthermore, the implications of this threat are limited as it only affects the *availability*. Therefore, it is not addressed in more detail.

Threat	**Violated definitions**	**Vulnerabilities**	**Addressed in**
T1 insecure storage	1a, 1b, 2a	Section 4.3.5	Chapter 6
T2 relay attack	2b	Sections 4.3.5, 4.3.4	Section 5.5
T3 insecure communication	1a	Sections 4.3.1, 4.3.2, 4.3.3	Section 5.1, 5.2, 5.3, 7
T4 DoS	3	Sections 4.3.5, 4.3.2	Not addressed
T5 Privacy	1b	Sections 4.3.1, 4.3.2, 4.3.3	Chapter 7

Table 4.2: Security threats

Chapter 5

Security mechanisms

This chapter provides an overview of security mechanisms that can be used to address the security threats in NFC that were described in Section 4. Section 5.1 gives an introduction to cryptography. Section 5.2 and 5.3 explains how cryptography can be used to implement a secure channel and secure authentication. In Section 5.4, the security mechanisms in the NFC standard are discussed and assessed. The last section, addresses relay attacks.

5.1 Cryptography

Cryptography is a technique to secure information such that an unauthorized party cannot read it. The data that needs to be secured is called the plain text. Encryption is the process of encoding the plain text such that it becomes unreadable. The encrypted data is called the cipher text. Decrypting is encoding it back to the original information. The encryption and decryption function require a key as an input, e.g. a string of text, a password or mathematical structure. Two types of encryption are available that differ in the number of keys that are used [43].

1. **Symmetric encryption**: The same key is used for both encrypting and decrypting the information. This key is kept secret and should only be known by those that are allowed to read the plain text.

2. **Asymmetric encryption**: Two separate keys are used. One for encrypting the information which is called the *public key* and another for decrypting the information which is called the *private key*.

Several symmetric and asymmetric encryption algorithms are available. The quality of these algorithms are evaluated by a number of properties. The *cryptographic security* is a logarithmic measure that counts the expected number of attempts before the plain text is found when using the fastest known computational attack on the algorithm. This can not exceed the key size as the best attack in a strong algorithm will be a brute force

attack where decryption is attempted by trying all possible keys. This means that in such algorithms, the security depends on the key size that is used: a larger key means better security.

Other quality criteria are *diffusion* and *confusion*. Diffusion means that the statistical structure of the plain text is dissipated into long-range statistics of the cipher text [44]. This is accomplished by having an encryption algorithm such that a change of a single character in the plain text results in many changed characters in the cipher text. This makes *statistical analysis* (a method that uses the frequency statistics of characters in a text to obtain the plain text from cipher text) more difficult.

Confusion means that the relation between the value of the key and the statistics of cipher text is complex. More specifically, each character of the cipher text should depend on several parts of the key. This makes it difficult to deduce the key from the cipher text.

Another important property of an encryption algorithm is the computational complexity which influences the time it takes to perform the encryption/decryption given the available processing power and the length of the plain text. Asymmetric cryptography is significantly more complex compared to symmetric cryptography. For this reason, it may not be desirable to use asymmetric encryption algorithms in devices with limited processing power such as smart cards and secure elements.

A cryptographic *hash function* is a one way function, i.e. it is easy to calculate the output value from an input value but it is difficult to do the opposite. Ideally, a small change in the input, results in a large change in output. Hash functions can be used to verify the integrity of a message.

5.2 Secure channel

A secure channel provides secure transfer of data, referred to as *application data*, over an insecure communication channel such that the properties of a secure system, i.e. confidentiality, integrity and availability, are not violated. This is accomplished in the following way:

1. Authentication is optional. It is done before any application data is sent in order to establish that the communicating party is who it claims to be.

2. The application data is encrypted before it is sent. Typically, symmetric encryption is used because this is computionally less expensive compared to assymmetric encryption.

3. Optionally, a *key exchange protocol* establishes a secret key that is used for the symmetric encryption. This is referred to as the *session key*.

4. A *message authentication codes (MAC)* is added to the data. MACs are designed to detect any intentional or unintentional changes which guards the integrity of the application data. It uses cryptographic algorithms such as encryption (CMAC) or a hash function (HMAC).

Since secure channels and authentication protocols are closely related, most authentication protocols can help with setting up a secure channel. During the authentication, both parties negotiate a set of parameters for the secure channel. These parameters are referred to as the *security association*. This includes the session key, and encryption algorithms (also referred to as the *cipher suite*). These parameters are then used to secure all communication that takes place after the authentication.

Some secure channels support *forward secrecy* meaning that if the key(s) used for authentication are compromised in the future, the application data is not compromised.

5.3 Secure authentication

Authentication is *"the act of verifying a claimed identity, in the form of a pre-existing label from a mutually known name space, as the originator of a message (message authentication) or as the end-point of a channel (entity authentication)"*, by [45]. Two roles can be distinguished: the user that is being subject to identification and the authenticator which is the entity that performs authentication.

An authentication protocol is responsible for performing secure authentication over an insecure channel (threat 3 from Section 4.4). This can be accomplished by using cryptography and a **challenge/response** mechanism. The authenticator asks a question (challenge) to the user and if the response is a valid answer (response), the authentication is successful. A commonly used example of a challenge/response is a password. Using a password, however, will not prevent eavesdropping attacks because if the password is sent as plain text over the communication channel it can be intercepted by an attacker.

Sending the password encrypted does not solve this problem: although now an attacker can not read the password it can still *replay* (see Section 4.3.1) the conversation between a user and authenticator. The challenge and response that is used, i.e. password, is the same throughout multiple authentication processes which allows replay attacks.

Replay attacks can be avoided by using a random challenge that is different each time a user authenticates. This is illustrated in Figure 5.1. The authenticator first asks for the user's identity. Subsequently, it sends a randomly generated string to the user which is different for each authentication attempt. The user encrypts this string with the secret key and sends it back. The authenticator decrypts the string and verifies that it matches with the generated random string.

In some cases it is desirable that both communicating parties authenticate each other. This is called *mutual authentication*. In Section 4.3.6, it was shown that the card reader can be tampered with and thus it can not be trusted. Therefore, it is recommended that the phone and the reader mutually authenticate.

A secure authentication protocol performs authentication while protecting against eavesdropping, data corruption, insertion and modification by using cryptography. In addition to that it addresses replay attacks by using dynamic challenge/responses. Protection against MiTM-attacks is provided because of the authentication itself: it verifies that a communication partner is legitimate and not a man-in-the-middle.

In the next two sections, two approaches for authentication are discussed: a public key infrastructure (PKI) and pre-shared keys (PSK).

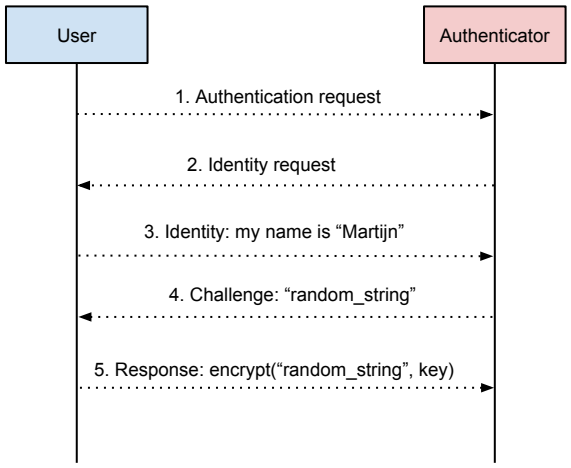

Figure 5.1: A simplified example of challenge/response authentication.

5.3.1 Pre-shared keys (PSK)

Pre-shared keys assumes that both communicating parties have one shared secret key which is used for symmetric encryption. The authentication is accomplished as illustrated in Figure 5.1. The `key` that is used for encrypting the challenge is known to both parties beforehand and does not need to be sent over the insecure channel.

The main trademark of PSK is simplicity and low set up cost. However, it does not scale well because a key is needed between every combination of peers. Consequently, a linear increase of peers in the network means an exponential increase in the number of required keys. Managing large amount of keys can be problematic.

One way to make PSK scale better is to use the same shared key for all users. This is often done in WiFi: all users of a network use the same passphrase to log on. This approach is not suitable when it is required that every user is uniquely identified.

Another technique to make PSK systems more scalable is to use a *key diversification* function. This function calculates a unique key for each user from a secret master key and other non-secret parameters.

This is used in contactless smart card systems such as MiFare. It would be difficult for each card reader in the system to store a separate shared secret for each individual card. The solution is that the reader can calculate the shared secret for each card from a master secret and other unique public information the can be read from the card. The card only knows the shared secret and not the master secret. If the shared key is compromised, only that card in the system is compromised. A disadvantage of this approach is that great care must be taken to secure the master key (on the reader). If this gets compromised, all cards in the system are compromised.

5.3.2 Public-key infrastructure (PKI)

In a public-key infrastructure (PKI), certificates are used for the purpose of authentication [43]. A certificate is a virtual document that allows an authenticator to verify the identity of a user without having any prior knowledge about the user (such as a pre-shared key used in PSK).

Several forms of public-key infrastructures exist. Figure 5.2 on page 45 shows the simplest form, in which one extra entity is required called the certificate authority (CA). The set up process for all peers is to get a certificate at the CA [46]. The CA is responsible for checking the identity one time 'in person'. In the case of a NFC mobile phone for example, the set up process could be done during the manufacturing of the phone/secure element.

After the CA has verified the identity of a requester, called peer 1 (P_1), it creates and signs a certificate C_{p1}. A well known standard for certificates is X.509 defined in RFC 2459 [47]. The most important information [1] on the certificate are the identity information of the peer I_{p1} and the CA that signed it (I_{ca}), a public key ($K_{pub,\ p1}$) and a signature (S_{p1}).

The signature is calculated by hashing and encrypting $K_{pub,\ p1}$ and I_{p1} using the CA's private key ($K_{priv,\ ca}$). After signing, the CA gives the requester its certificate C_{p1}. Furthermore, the CA provides its own certificate (C_{ca}) to the requester containing the public key ($K_{pub,\ ca}$). C_{ca} is a special type of certificate indicating that it can be used

[1] X.509 certificates contain more information than described here, but for simplicity these are omitted

to sign other certificates but C_{ca} is not signed itself. After this, the set up process is completed.

q

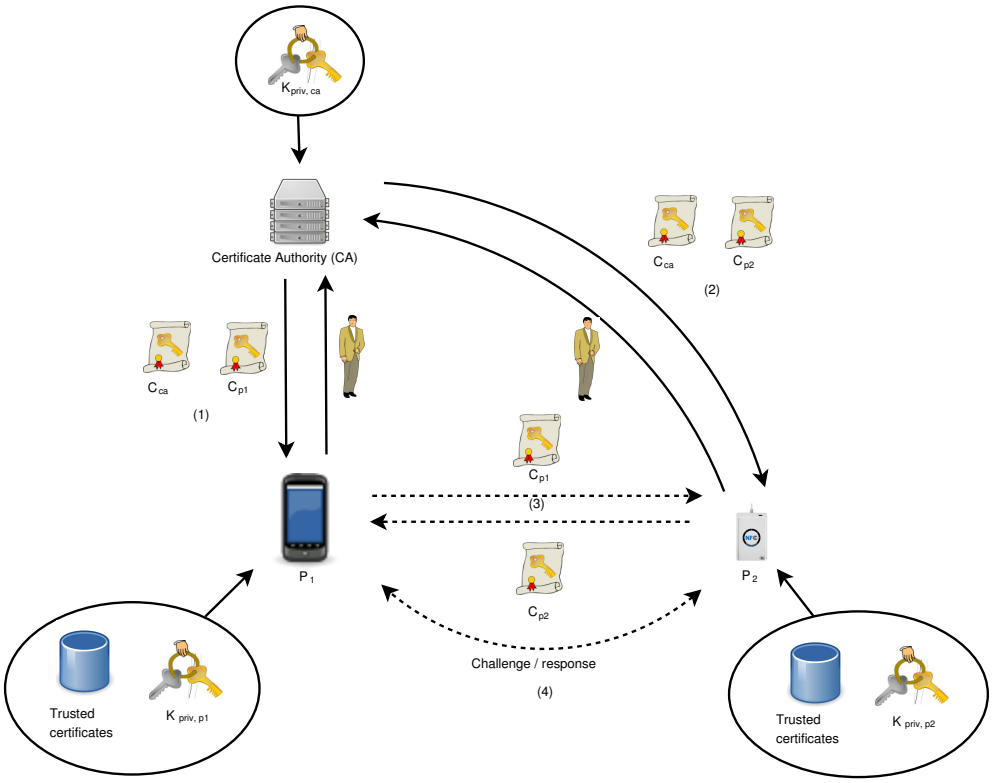

Figure 5.2: An example of a PKI authentication architecture. The numbers indicate the order of events

All peers keep a list of CA certificates that are trusted. This implies that every peer with a certificate signed by one of the CAs in its list is trusted. Therefore, this list must be chosen carefully and protected from unintended change.

When two peers, p_1 and p_2, want to authenticate each other, they first exchange certificates. They each verify that the certificate was signed by a CA that they trust. This is done by decrypting the signature on the peers certificate with the public key of the CA (which is on the certificate of the CA). Subsequently, the peers each verify that the other owns the private key corresponding to the public key on the certificate. This can be done by a challenge/response mechanisms similarly as done in PSK.

It is of great importance that all private keys remain confidential to their owners. This holds especially for the private key of the CA. If this is compromised, an attacker could generate valid certificates himself undermining the trust in the CA.

Though in the example discussed here, a PKI with one CA was considered, it is possible to have an hierarchical infrastructure of CAs. CAs in tier 2 have a certificate signed by CA in tier 1, which in turn are signed by the root CA. The root CA has a so-called self-signed certificate meaning that the signature is calculated from its own private key. The CA's below the root CA inherit the trustworthiness of the root CA. This architecture is useful in large systems for load balancing.

The use of certificate authorities in PKI overcomes the scalability issues in PSK. Each peer in the network only needs one private and public key pair. Furthermore, it can keep a list of trusted CA certificates meaning that certificates signed by such a CA, can be verified without contacting the CA. However, setting up certificate authorities and dealing with issuing certificates in PKI requires additional costs and complexity.

5.4 Security features in NFC

The international standard NFC-SEC [48] and NFC-SEC-01 [49] provide some security features in NFCIP-1. This can only be used in NFC peer-to-peer mode and not in read/write and card emulation mode [23].

NFC-SEC provides two services:

1. **Shared secret service**: establish a *shared secret key* and a symmetric encryption algorithm in a secure way.
2. **Secure Channel Service**: establish a secure channel by encrypting all data symmetrically using the previously defined shared secret key and chosen encryption algorithm.

The other standards of the form NFC-SEC-xx specify the implementation of these services using specific algorithms (thus cipher suites). Currently, only NFC-SEC-01 is available which uses elliptic curve Diffie-Hellman algorithm [2] for establishing the shared secret key and *Advanced Encryption Standard* (AES) [3] for establishing the secure channel.

Thus, NFC-SEC provides a secure channel which provides protection against eavesdropping and data modification [23]. However, it does not provide authentication and thus MiTM is not addressed. In this research, the main purpose of NFC is authentication. This means that an authentication protocol will have to be selected since NFC-SEC does not provide it. As most authentication protocols already provide protection against the mentioned attacks it makes the use of NFC-SEC superfluous.

[2] Standardized in NIST 800 56A [50]
[3] Standardized in FIPS 197 [51]

5.5 Protection against relay attacks

Relay attacks (threat 2) can not be prevented by using secure authentication or secure storage. An attacker simply forwards the communication between an NFC reader and a legitimate authentication token such as a card or a phone and it does not have to understand the meaning of the communication.

The following counter measures for relay attacks are mentioned in the literature [42] [40]. Solutions 1 and 3 can be implemented by the service provider. Solutions 2 and 4 are up to the phone manufacturers

1. **Time-out on the reader** - If a transaction takes longer than the time-out it must be cancelled by the reader. When a relay attack is performed, the communication channel between the proxy and the mole could introduce additional delay in the transaction depending on the speed of the used communication channel and the distance between the mole and the proxy. Hence, setting a time-out will narrow down the possibility of a relay attack but does guarantee that it can not happen. A disadvantage of this solution is that it prevents secure remote credential storage which is described in Section 6.6.

2. **Disable internal access to the secure element** - In most mobile phones, software applications can communicate with the secure element (path 3 or 4 in Figure 2.5 on page 18). This communication can be disabled which would stop internal relay attacks (see Section 4.3.4) where malicious software on the victims phone behaves as the mole. However, this comes at a price because it is no longer possible to manage the secure element remotely.

3. **Multi-factor authentication** - Combine more than one property for the purpose of authentication. For example, combine the mobile phone & NFC (something the person has) and a PIN code (something the user knows). This requires that the attacker also knows the PIN which can not be obtained in the relay attack assuming that the communication uses a secure channel and secure authentication. A disadvantage is that using multi-factor authentication has a negative effect on the usability because it requires more actions from the user.

4. **Physical activation and deactivation of NFC/SE** - Disable all secure element and NFC functionality with a physical buttton on the phone and let a user enable it when needed. This prevents that malicious 'mole' software on the phone or an attacker with a (physical) mole device to communicate with the phone of an unaware victim.

Chapter 6

Secure storage

This chapter directly addresses *threat 1* that was identified in Chapter 4. The credential in the phone must be stored in a secure way such that it can not be stolen.

A secure element can be used for this purpose. Various types of secure elements can be available in the mobile phone. Additionally, recent phones by Google provide an alternative hardware-based credential storage.

All possibilities for secure storage in the phone are considered in this chapter and assessed according to the following criteria.

1. **Secure storage** - Conforms to the security definition from Section 4.1 and provides protection against threat 1 from Section 4.4.
2. **Re-usability** - Is it possible to re-use the credentials if a user switches from one phone to another? Is it expensive to personalize the user's new device?
3. **Standardization** - To what extent is this form of secure storage standardized? In the case of secure element, this concerns the interface and communication between the secure element and components in the phone, e.g. CPU or NFC controller. This standardization enables direct communication between NFC controller and the SE meaning such a solution is independent of the phone's main CPU and battery.
4. **Availability** - To what extent is a form of secure storage available in phones currently on the market? It is in the interest of the service provider that their service is available to a large audience. The availability of the selected type of secure element in the mobile phone can potentially restrict the number of customers. Not to be confused with the availability definition in security (see Section 4.1).
5. **Accessibility** - Is it easy for a service provider to use a form of secure storage? Some types of secure storage hardware are controlled by an external party and can not be used without their permission (see Section 2.5).

Criteria 1-3 are investigated by Reveilhac et al. [52] for various types of secure element. The other criteria are defined in this work. Criteria 4 and 5 are added in this work in order to get better insight into the ease of use from the perspective of a service provider.

6.1 Embedded Hardware

Embedded hardware refers to a smart card chip soldered into the mobile device by the manufacturer (see Figure 6.1). This provides good *security* as it is based on smart card technology. However, *accessibility* is poor because the embedded secure element is controlled by the phone manufacturer.

The NFC-WI standard can be used for communication between the embedded hardware and the NFC controller (see Section 2.4). For example, this is used in the latest Google Nexus phones that are equipped with an embedded secure element [53]. Therefore, *standardization* is good.

A disadvantage is that the secure element cannot be re-used when the owner buys a new phone [54]. Furthermore, the chip will need to be personalized during the manufacturing of the phone [55] potentially increasing the production costs [52] due to the necessity of designing a new personalization process.

Most NFC capable phones today are already equipped with an embedded secure element [1] and this trend is expected to continue [56] meaning that the availability is good.

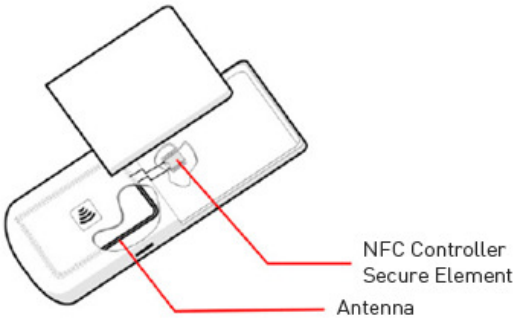

Figure 6.1: Embedded secure element, copied from [57]

[1] In 2011 it was estimated that 75% of the 70 million NFC phones sold that year would come with embedded secure elements [56]

6.2 Universal Integrated Circuit Card

The UICC is a removable smart card in the mobile device as shown in Figure 6.2. It contains the SIM application that is in mobile phones to provide access to MNO networks. For this reason, it is usually called a SIM card. But since the SIM is using smart card as an underlying technology, this means that other applets can be hosted on the card as well.

The UICC is based on smart card technology and thus the provided *security* is good. The SIM is generally available in every mobile device and thus *availability* is also good. SIM cards can be re-used when the user buys a new phone as it is removable. However, it can not be re-used when the user changes from one mobile network operator to another. *Accessibility* is poor because the UICC is tightly controlled by the MNOs. The interface between the NFC controller and the UICC is well *standardized* in the single wire protocol (see Section 2.4).

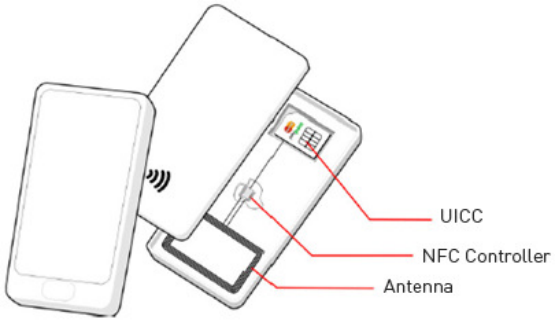

Figure 6.2: UICC secure element, copied from [57]

6.3 Secure Memory Card

A secure memory card is an SD card with a smart card chip installed on it. This card can be inserted into the micro SD slot of the phone. This solution is *re-usable* because it can be easily installed in another phone.

In addition to the smart card chip, some solutions also contain an NFC controller [58]. This is beneficial in the first place because this works on phones that do not have an internal NFC controller. For example, all current models of the popular iPhone are not equipped with NFC. Moreover, it guarantees that the communication between this 'external' NFC controller and the secure element is implemented.

According to Reveilhac et al., the communication between the secure element (on the memory card) and an internal NFC controller (in the phone) is not standardized. As

for the SMC solution with both secure element and (external) NFC controller, it is not generally known how the communication between these two components is standardized because this solutions are proprietary.

The security that is offered by this solution is just as good as the other solutions that rely on smart card technology [52]. Another advantage is that a memory card can be issued by the service provider itself which provides good *accessibility* to this type of secure element.

Many phones on the market have an SD slot but not all. In fact, it seems that a decreasing number of phones include an SD slot meaning that this solution cannot be used for all customers.

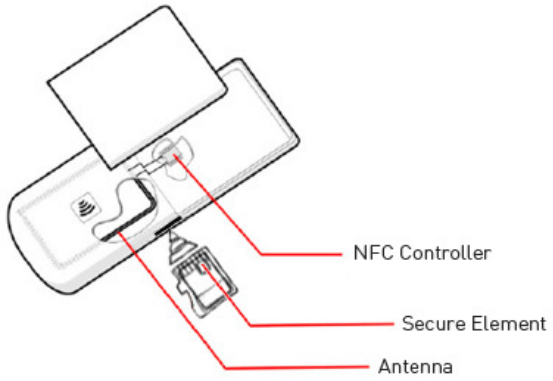

Figure 6.3: Secure memory card secure element, copied from [57]

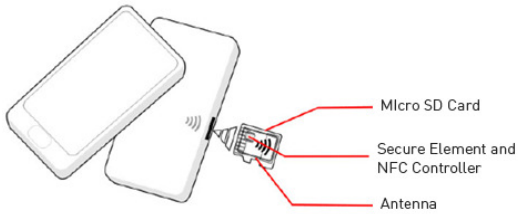

Figure 6.4: Secure memory card secure element with integrated NFC controller, copied from [57]

6.4 Main storage

All phones provide a main storage device where users can store their files such as music and photos. This can also be used to store a credential.

Most mobile operating systems provide a mechanism to store the credential encrypted by using a *key encryption key (KEK)* . For example, the Android keystore stores private keys or passwords encrypted. A potential problem with this approach is that the KEK is also stored on the main storage device meaning that ultimately the security of this solution depends on how well the operating system protects this key.

Another possibility is to not store the KEK. In this case user input is required during the transaction. For example, the KEK can be a PIN code or it can be derived from a biometric feature as some recent phone have a biometric scanner. This, however, significantly increases the transaction time which may be undesirable depending on the application.

The advantage of using the main storage is that it is available and accessible in every mobile phone. The *standardization* criteria does not apply here because there is no direct connection between the main storage and the NFC controller. The main storage in the phone is typically not removable and can not be re-used. As opposed to a secure element, it is possible to extract and migrate the information from one phone to another. Therefore, the personalization process of the users new device should be easier compared to the embedded hardware solution.

6.5 Baseband processor

The baseband processor handles all radio (GSM, UMTS, 3G, 4G but not NFC) related functions and is completely separated from the phone's main processor. It provides a secure memory for the storage of data, but this is inferior to the solution that are based on smart card technology.

This type of secure element is not re-usable because the baseband processor cannot be moved from one phone to another. The secure element needs to be set up every time a user buys a new phone.

The protocol between the NFC controller and the baseband processor has not been standardized. Furthermore, it is available on every mobile device as radio communication is essential for mobile phones. To the best of my knowledge, no details are known about the accessibility.

6.6 Secure element in the cloud

Pourghomi et al. [59] propose an NFC payment solution where applets are stored in a cloud structure and downloaded on the phone when a transaction takes place. The applet is temporarily stored in a 'traditional' secure element in the phone, and removed

after the transaction has completed. This solution builds on existing solutions by Fujitsu Laboratories [60] and A1 (subsidiary of Telekom Austria group).

A motivation for doing this is to enable an NFC mobile phone to be used for multiple services, replacing traditional plastic smart cards. Secure elements can only contain a limited amount of applets which limits the number of services for which it can be used. This limitation can be overcome by installing the applet only when it is required, and remove it after use.

The idea is illustrated in Figure 6.5 for a payment service. When a user wants to pay, the phone is held near the NFC payment terminal which will prompt it to download an application. This application is downloaded from the cloud using a internet connection and is installed into a secure element (theoretically this could be any of the other smart card chips mentioned in this chapter). The applet contains the necessary credentials. The payment terminal communicates with the phone using NFC in order to authenticate it and subsequently with the bank in order to perform the transaction. After the transaction, the application and credentials are removed from the secure element.

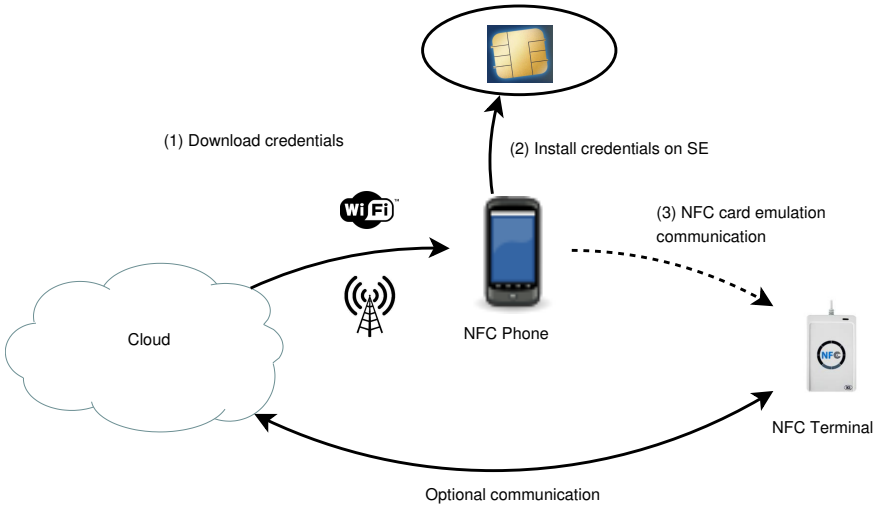

Figure 6.5: Secure element in the cloud using a traditional secure element, based on [59]. The numbers indicate the order of events. Communication between the terminal and cloud is optional

An advantage is that cloud-based solution can be easily re-used. When a user purchases a new phone, or changes subscription meaning a new SIM card is installed in the phone, this does not impose a problem because the secret data has to be re-downloaded for every transaction anyway.

A cloud based approach is easier to manage. There is no need for the service provider to push updates (software, permissions etc) to the secure element of each individual customer. The cloud is a central storage location from which the user will automatically pull the latest version whenever a transaction takes place.

An alternative but similar cloud-based solution is discussed in [61] and illustrated in Figure 6.6. This solution is proprietary meaning that not all details are known. A traditional secure element applet is stored in the cloud and transferred to the phone on-demand in a similar way as the solution by [59]. However, in this case the applet is executed on the phone's application processor by using host card emulation and transferred to a secure element.

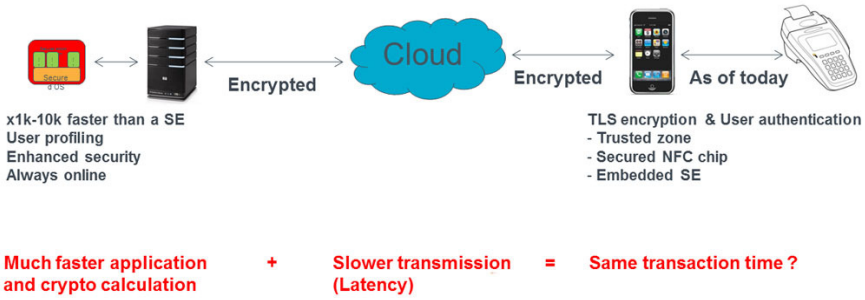

Figure 6.6: Secure element in the cloud without using a traditional secure element, copied from [59].

A number of problems can be identified with a cloud-based approach that are not discussed in [59].

1. **Authentication to cloud**: It is not clear if and how the phone authenticates to the cloud and where the credentials for this are stored. An encrypted channel is used for communication but in addition to that there should be some kind of authentication. For this reason, it is difficult to assess the security of this solution into great detail.

2. **Traditional secure element**: The choice for a 'traditional' secure element used for temporarily storing the applet containing the credentials is left open. Depending on this choice, accessibility and standardization may still be an issue.

3. **Internet availability**: Another potential problem is the internet connection which must be available on the mobile phone during the transaction. If this is not the case, the service cannot be used.

4. **Transaction time:** A slow connections between the phone and the cloud can potentiality increase the transaction time beyond acceptable limits.

6.7 TrustZone

Google introduced so-called 'hardware-backed' credential storage on their recent phone models. The implementation of this needs to be investigated in order to assess the security of this solution.

The hardware-backed security is implemented by the application processor in the phone which supports ARM's *TrustZone* technology. This is said to provide a secure execution environment and secure storage. The exact implementation details are unknown as this technology is proprietary but the following is known. The processor is divided into two 'virtual' processors: one providing a 'non-secure world' that runs 'normal' software such as the operating system and applications and another that provides a 'secure world' or a TrustZone which is capable of running low-level software in a secure way. Software in the non-secure world can only interact with the secure world through a predefined interface.

The secure and non-secure processors share the same physical processor, registers and memory but are isolated from each other. This is accomplished by the *Secure Configuration Register* (SCR) component. This controls access to registers and memory by tagging it with an extra bit that indicates whether a register or memory address belongs to the secure or non-secure world.

It is unknown to me where the secure low-level software resides or what its implementation exactly looks like. Therefore, the TrustZone is essentially a black box. However, its behaviour can be studied by studying the Android source code (which is open source) that interacts with the secure software in the TrustZone. This was done in [62].

Figure 6.7 illustrates the components in the non-secure world and in the TrustZone. The interface between those components are numbered. The component in Android that is responsible for credential storage is called *Keystore* which provides an API for applications (interface 2). It uses the TrustZone technology (when available in the phone) to secure the credential. The keystore communicates with low-level software in the TrustZone that is referred to as the "QCom Keymaster" (interface 4). The QCom Keymaster provides an interface for generating and importing public/private key pairs and signing/verifying data with those keys. At the time of writing, it only supports RSA 2048 bit and no other cryptographic algorithms.

The public/private keys are generated in the TrustZone. However, the source code references so-called *key binary large objects* (key BLOBs) [63] which suggests that the key pairs are **not** actually stored in the TrustZone itself. A key BLOB is a way to store a cryptographic key outside a secure container (in this case the TrustZone) [64]. In this case, the TrustZone only contains a *key encryption keys* (KEK) which is used to symmetrically encrypt the key pairs.

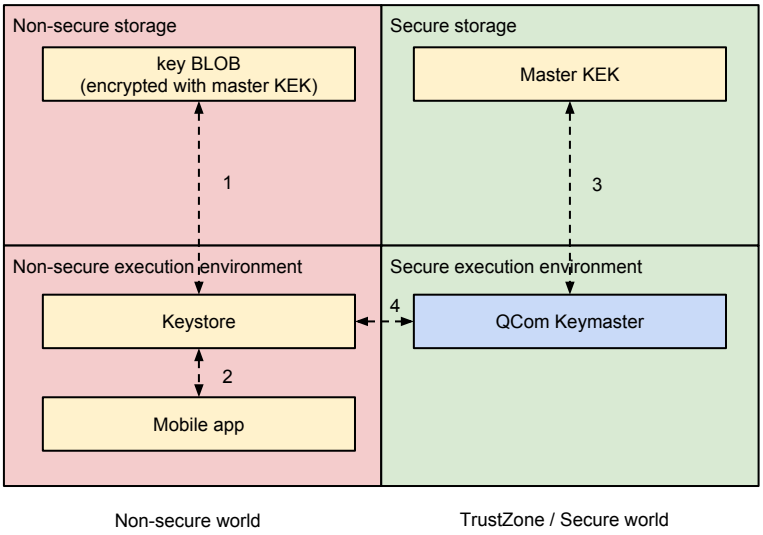

Figure 6.7: Explanation of the TrustZone. The interfaces are numbered for reference purpose.

A secure element is comparable to the TrustZone in the sense that they both provide an isolated storage and secure execution environment. However, it is difficult to compare the two based on protection against certain attacks that require physical access to the chip.

For example, attacks described on smart cards include *optical fault injection* which involves manipulating the execution of programs on the chip by using light such that the protection mechanisms (e.g. entering a PIN code) can be circumvented [65]. Another attack on smart cards is referred to as *Side-channel Analysis* (SCA) which is a method for extracting protected data by analysing the behaviour of the electronic circuit. For both type of attacks counter measures are described.

The literature review about attacks on smart cards left the impression that, although weaknesses have been found, security features are constantly being improved. For the TrustZone, on the other hand, this is not the case. The ARM website claims that the TrustZone offers some protection against SCA: " *The only mechanism to recover the secret key is to perform side-channel analysis of the cryptography, which the hardware accelerator design should prevent, or to dismantle the silicon to recover the key. This is likely to be too expensive*" by [66]. However, no independent research has been done to confirm this. Furthermore, in general little research has been done into the security of the TrustZone.

It must be noted that as opposed to a secure element there is no *standardization* between the TrustZone and the NFC controller. Hence, using a TrustZone can never be

completely isolated from the phone's CPU. This also affects the security of the solution as a whole. A potential problem is that, although the credential (private key) can not be extracted, the access for using the credential is still regulated by the operating system that is vulnerable to implementation errors. On the other hand, it is known that most mobile phone's and SMC manufacturers provide APIs for accessing SIMs, embedded hardware or SMC type of secure element. These APIs are also protected by the operating system and thus are subject to the same vulnerabilities.

I qualify the *security* for this solution as *good* because theoretically they offer a similar service. Furthermore, there is not direct evidence that the TrustZone is more or less sensitive to the described attacks on smart cards.

The main advantage of this solution is *accessibility*. As opposed to a secure element, it can be used without restrictions. Availability is currently fair. Many recent modern phones processor are equipped with the TrustZone technology. But it requires that the keymaster software is installed on it which is currently only the case for the Nexus phones designed by Google.

6.8 Conclusion

Table 6.1 summarizes the comparison between secure storage possibilities in a mobile phone. The following marks are used to evaluate the criteria.

1. **Good**: The criteria is fully supported/satisfied
2. **Fair**: The criteria is partially supported/satisfied
3. **Poor**: The criteria is not supported/satisfied

From this table it can be concluded that there is a trade-off between *secure storage* and *accessibility*. For a service provider, the secure memory card is currently the best option because it supports both criteria well.

A disadvantage of the secure memory card solution is that it conflicts with our motivation for using NFC which was the integration of various tokens into one. Technically, it would be possible to do this with an SMC but in practice the SMC is typically not shared among multiple service providers. Potentially, users that wish to use more than one NFC service will have to swap SD-cards in/out of their phone each time another service is used.

Given that the goal is token integration, it would be best that there is a generic secure storage solution available in every phone that is accessible to all service providers with good security properties. Such a solution was described in Section 3. Unfortunately, this

solution is currently not deployed but the recently introduced hardware-backed secure storage in Google phones can be seen as a step towards this direction. A disadvantage of Google's hardware-backed storage is that it currently only supports RSA-2048 and no other key formats.

	Secure storage (threat 1)	**Re-usability**	**Standardization**	**Availability**	**Accessibility**
Embedded hardware	Good [52]	Poor [52]	Good	Fair	Poor
UICC	Good [52]	Good [52]	Good [52]	Good	Poor
Secure Memory Card	Good [52]	Good [52]	Poor [52]	Fair	Good
Main storage	Poor	Fair	[4]	Good	Good
Baseband processor	Fair [52]	Poor	Poor [52]	Good	[1]
SE in cloud & traditional SE	Good	Good	[2]	Fair [3]	[2]
SE in cloud & HCE	[1]	Good	[4]	Fair [3]	Good
TrustZone	Good	Poor	[4]	Fair	Good

[1]No information available
[2]Depends on chosen traditional secure element
[3]An internet connection must be available
[4]Not applicable
[5]Based on protection provided by the operating system.

Table 6.1: Evaluation of secure storage possibilities in the mobile phone

Chapter 7

Authentication protocols

This chapter examines the available authentication protocols that could be used in physical access control using mobile NFC. The authentication protocol addresses threat 3 and 5 from Section 4.4.

Many authentication protocols are available and it is infeasible to study all of those. Therefore, a pre-selection is made and motivated in Section 7.1. Section 7.2 discusses a number of requirements for the authentication protocol. Subsequently, various authentication protocols are described in Section 7.3. Finally, the authentication protocols are assessed using the requirements in Section 7.4.

7.1 Pre-selection

Many type of authentication protocols are available. It is infeasible to study all of those. For this reason a pre-selection is made.

Based on the information about NFC that was obtained in Chapter 2 two approaches can be foreseen.

1. A smart card applet is developed that runs on a secure element in the phone. This applet is responsible for implementing the authentication protocol and performing cryptographic operations.
2. A mobile phone application is developed that runs on the phone's main CPU. This application is responsible for implementing the authentication protocol and perform cryptographic operations.

The first approach requires that a *smart card authentication protocol* is used. These protocols are specifically tailored to deal with the limited processing power in smart cards and secure elements.

Since the mobile processor is more powerful, the second approach allows the use of *computer security authentication protocols*. These protocols provide more flexibility. For example, most smart card protocols rely exclusively on PSK since symmetric encryption requires less processing power. This affects the scalability. Moreover, most smart card protocols use smaller key sizes due to the limited processing power. This affects the security.

For the computer security protocols, the focus will be on the Extensible Authentication Protocol (EAP) which is defined in RFC 3748 [67] and updated by RFC 5247 [68]. EAP is a general authentication framework which specifies the transport and message format for sending keys and other parameters. The framework does not define a specific authentication method. Instead, these are defined in various extensions called EAP-methods.

The motivation for focussing in EAP is that it is currently wide in use in other popular wireless technologies such as IEEE 802.11 (WiFi). Furthermore, an RFC draft [69] with title "EAP Support in Smartcard" enables to the use of EAP over the ISO 7816 application layer protocol meaning that any EAP method can be used over NFC.

7.2 Requirements

This section describes a set of requirements for the authentication protocol. These requirements are based on requirements available in the literature for similar technologies [70] [71] and others are derived from knowledge of NFC and related technologies.

7.2.1 Requirements from RFC 4017

RFC 4017 [71] defines a set of requirements for EAP-based authentication protocols for usage in wireless LANs. As NFC and wireless LANs are both wireless technologies the risks and vulnerabilities are similar. Therefore, these requirements are adopted. RFCs differentiate between mandatory requirements (indicated by the word "MUST"), recommended requirements ("SHOULD") and optional requirements ("MAY") [72]. Most of the requirements are explained in more detail in [67].

Mandatory requirements

1. Key derivation: generation of symmetric keying material. The authentication protocol must be able to generate the keys that are used for the secure channel.
2. The key derivation should generate keys with at least 128-bits of effective key strength. Effective key strength corresponds to the cryptographic security definition from Section 5.1.

3. The session key (referred to as Master Session Key (MSK) in the EAP-documentation) should be at least 64 bytes.

4. Mutual authentication support. This is important as it offers protection against MiTM attacks and untrusted readers.

5. Shared state equivalence: The shared EAP method state of the EAP peer and server must be equivalent when the EAP method is successfully completed on both sides.

6. Resistance to dictionary attack. A dictionary attack, is a technique to guess a encryption key or password by trying out many 'likely' attempts. An example is using a dictionary for guessing a password.

7. Protection against man-in-the-middle attacks (as explained in 4.3.3).

8. Protected cipher suite negotiation. Some authentication protocols support multiple cipher suites (see Section 5.3). A negotiation takes place during the authentication which determines what cipher suite is going to be used. It is important that integrity is guaranteed in this phase. Otherwise an attacker could trigger the use of a weak cipher suite potentially enabling him to compromise secret information.

Recommended requirements

9. Fragmentation - The Maximum Transfer Unit (MTU), i.e. the maximum amount of data that can be sent at once, in NFC is 256 bytes. If an authentication protocol requires to send more data it must implement fragmentation.

10. Identity privacy / End-user identity hiding. The authentication protocol must protect the identity information from being obtained by unauthorized parties. This is referred to as *identity protection* and it provides protection against threat 5.

7.2.2 Additional requirements

In addition to the requirements from RFC 4017, the following requirements are defined in this work.

11. The authentication protocol has been tested and proven in the field.

12. The authentication protocol can be used with NFCIP-1 and NFCIP-2.

13. The authentication protocol can be used in devices with limited computational power such as a secure elements.

7.3 Authentication Protocols

7.3.1 EAP-PSK

The EAP Pre-shared key is a mutual authentication protocol defined in RFC 4764 [73]. It relies on the PSK authentication scheme that was discussed in Section 5.3.1.

Figure 7.1 shows a representation of the message exchange in EAP-PSK between a server and a peer (note that these terms are just the names that are used in the standard). The `ID_S` and `ID_P` is the identity information of the server and peer respectively. The `RAND_S` and `RAND_P` fields are the two random fields for the dynamic challenges.

The responses that accomplish the authentication are named `MAC_S` (for the server) and `MAC_P` (for the peer). Figure 7.1 shows how these response values are calculated. The `CMAC-AES-128` function is a MAC based on the AES encryption algorithm with a 128 bit key size. The server and peer can verify the received `MAC_P` and `MAC_S` respectively by calculating the values themselves and compare it to the received one. If the received and calculated are values match, the authentication is successful.

Additional information (`PCHANNEL_S_0` and `PCHANNEL_S_1`) is exchanged in order to set up the secure channel that can be used after the authentication is completed.

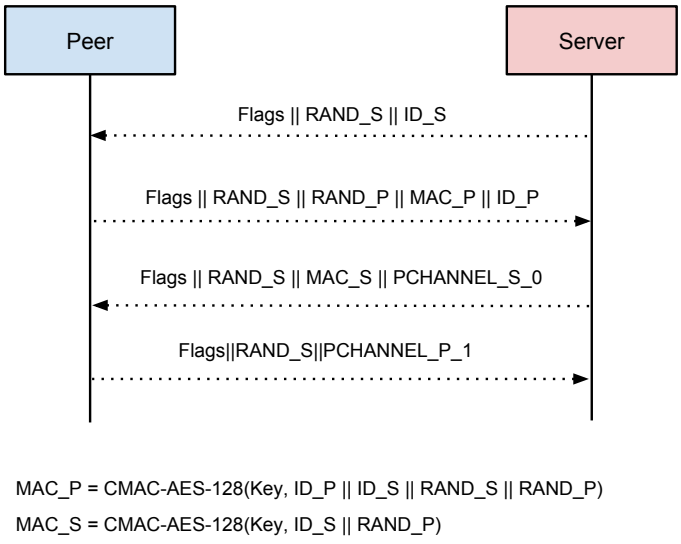

Figure 7.1: Representation of EAP-PSK message exchange. Figure based on and adapted from [73]

7.3.2 EAP-TLS

EAP Transport Layer Security (EAP-TLS) [74] is a PKI based authenticated scheme which uses X.509 certificates. EAP-TLS is based on TLS which is widely used in the internet to provide a secure channel in HTTP, FTP and e-mail connections.

EAP-TLS provides mutual authentication, protected cipher suite negotiation, privacy and automatic key management. Key management refers to the automatic establishment of a key(s) for the symmetric encryption algorithm.

Figure 7.2 illustrates the message exchange between the client and a server. The client first sends a `ClientHello` which includes a random value (which is later used as challenge) and a list of supported cipher suites (see Section 5.3). The server responds by sending five messages.

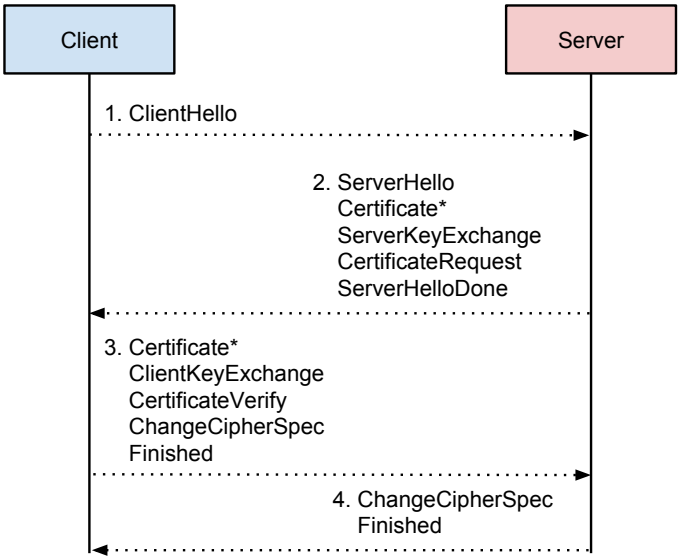

Figure 7.2: Representation of EAP-TLS message exchange. A '*' indicates an optional message. Figure based on and adapted from [74]

The `ServerHello` contains the server's challenge and a selection from the list of cipher suites that was sent by the client. The `Certificate` message contains the X.509 certificate of the server. The `ServerKeyExchange` is optionally sent when the certificate does not contain enough information for the client to send information confidentially. The `CertificateRequest` message is sent if the server wishes to authenticate the client (mutual authentication). The `ServerHelloDone` indicates that the server has sent all its messages.

The client processes the server's response. Most importantly, it will verify the trustworthiness of the received certificate as explained in Section 5.3.2. If the server requested it, the client will send its certificate. The `ClientKeyExchange` contains a pre-master secret. The pre-master secret consist of another random value generated by the client and of the TLS version number. The server and client both calculate the master secret from the pre-master secret and random values that were exchanged in the hello messages. The master key is used as a session key in the secure channel.

The `CertificateVerify` message authenticates the client to the server. This is done by hashing and encrypting all data that is sent until now (this includes the random value / challenge) with the client's private key. The `ChangeCipherSpec` message indicates that all subsequent traffic will be secured with the negotiated parameters (a secure channel with the selected cipher suite and calculated master key). The `Finish` message is the first message in the secure channel which allows to server to verify that it is working properly.

The server responds with a `ChangeCipherSpec` and a `Finish` message. In case of the server, the purpose of the `Finish` message is authentication to the client by showing that it was able to decrypt the pre-master secret.

Optionally, EAP-TLS provides privacy. This is accomplished by executing the TLS handshake twice. In the first handshake, the server only authenticates to the client. When this authentication is successful, the resulting secure channel is used to perform a second handshake in which both peers mutually authenticate. This makes it impossible for an unauthorized party to obtain the client's certificate. In [75] an alternative approach for privacy in EAP-TLS is suggested which leverages the TLS extension mechanism. This approach is more efficient because it does not require the handshake to be executed twice.

7.3.3 EAP-TTLS

The EAP Tunnelled Transport Layer Security (EAP-TTLS) [76] is similar to EAP-TLS and thus PKI-based but it provides backwards comparability with password based authentication.

This scheme describes mutual authentication between a user and authenticator. Only the authenticator has a certificate. The user verifies this certificate in order to make sure that the authenticator is legitimate after which a secure channel is set up as normally done in TLS. Subsequently, this secure channel is used to send the user's password to the authenticator as if it was application data after which the user is authenticated.

7.3.4 EAP-IKEv2

The EAP Internet Key Exchange Protocol is defined in [77] and based on IKEv2 which is defined in [78]. IKE is used in IPsec in order to perform mutual authentication and setting up a security association.

EAP-IKEv2 is similar to EAP-TLS as it provides similar functionality. It supports X.509 certificate for authentication which are either pre-shared or distributed using domain name systems (DNS). Furthermore, it supports cipher suite negotiation, automatic key management and identity confidentiality (depending on the mode of operation).

A number of extensions to IKEv2 are documented which provide additional functionality. Session resumption allows the continuation of a session between a peer and authenticator after a failure without going through to the complete set up process [79]. IKE redirect enables the redirection of incoming request to other servers in order to perform load balancing [80]. EAP-IKEv2 extends IKEv2 by including support for password based authentication.

7.3.5 PEAP

The Protected-EAP (PEAP) protocol, encapsulates EAP messages within a TLS authenticated secure channel. This is the opposite of the previously discussed EAP protocols where authentication protocols packets are wrapped in EAP-messages. The motivation for this design is that the initial EAP-message exchange, which includes an identity request and response, in plain text.

The functionalities provided by PEAP are similar to EAP-TTLS; the authenticator authenticates to the user by means of a certificate. The user authenticates using a password.

PEAPv0/EAP-MSCHAPv2

PEAPv0/EAP-MSCHAPv2 is the most common version of PEAP and is documented in [81]. It is said to be the second most supported EAP protocol behind EAP-TLS. TLS is used to encapsulate EAP messages for the authentication of authenticator to the user (certificate based) and MS-CHAPv2 is used inside EAP to authenticate the client to the user. The latter allows password based authentication using databases compatible with MS-CHAPv2 format such as Microsoft NT and Microsoft Active Directory.

PEAPv1/EAP-GTC

PEAPv1 is developed by Cisco and uses the Generic Token Card (GTC) protocol inside EAP as defined in [67]. A text challenge is sent from the authenticator to the user. The user responds by sending a valid password or a token. A token is an alternative way to

prove one's identity. In this case it is generated by a token card which is a smart card that the user owns. GTC does not protect the authentication data but since EAP-GTC uses TLS, the authentication data is protected.

7.3.6 EAP-FAST

EAP-Flexible Authentication via Secure Tunnelling (EAP-FAST) is also based on TLS, created by Cisco and standardized in [82]. It aims to be simple by allowing simple authentication methods such as passwords while not relying on certificate based authentication at all but still provide mutual authentication [1].

EAP-FAST authentication is in two stages. In the first stage a secure TLS tunnel is set up and mutual authentication is performed. The latter is accomplished by using Protected Authentication Credentials (PACs). PACs are essentially pre-shared keys but the major advantage compared to PSK is that it is not required for the authenticator to store a PAC for each individual user as is done with pre-shared keys. Instead, PAC's are derived from a secret master key K_{master} only known to the authenticator.

The PACs consist of three components:

1. **PAC-key** (PAC_k) - a random 256 bit key. This is used as a secret key to encrypt data symmetrically in the TLS tunnel (TLS pre-master secret)
2. **PAC-opaque** (PAC_{op}) - is PAC_k encrypted with the master key K_{master}. The PAC-opaque can only be interpreted by the authenticator
3. **PAC-info** - variable field for optional information such as PAC lifetime.

The user authenticates to the authenticator because PAC-opaque it submits to the authenticator, must match the PAC-key that is used to encrypt all communication. The authenticator authenticates to the user by proving that it has knowledge of K_{master} [83]. It uses K_{master} to decrypt PAC_{op}, send by the user, yielding PAC_k which is subsequently used to encrypt all communication. In the second stage the weaker user/password authentication takes place over the secure, mutual authenticated channel.

A challenge in EAP-FAST is distributing the PACs to the user. As with certificates in PKI based systems, PACs can be given to the user when the user is physically present at the authenticator. For example, when this solution would be used in mobile NFC access control system, the PAC can be installed to the when the user registers at the clerk.

[1] PEAP, EAP-IKEv2, EAP-TTLS allow password authentication but still rely on certificates for mutual authentication

Alternatively, it can be send to the user over a secure channel. An initial draft of EAP-FAST included automatic generation and distribution of PACs to authorized users (by using a password) over a secure channel but this was removed. The reason for this was that it was not secure since there is no prior relation between the authenticator and the user. The secure channel used for distributing the PACs relied solely on asymmetric encryption without providing (mutual) authentication meaning that it is was susceptible to MiTM attacks.

7.3.7 PLAID

PLAID is an authentication protocol targeted at smart cards. It was developed and standardized [84] by Centrelink which is part of the Australian Government. The protocol is open and supports both physical access control and logical access control. It was designed *"to bridge the gap between existing RFID-based technologies that offer speed but lack the necessary security features, and PKI-based authentication, which is cryptographically secure but lacks the speed necessary in many contactless smartcard scenarios"* [85].

Mutual authentication is supported and implemented using both symmetric and asymmetric encryption. The complete authentication process takes less than 0.3 seconds [85]. Furthermore, it supports multi factor authentication meaning that multiple authentication factors, i.e. something the user knows (such as a PIN) or something the user is (such as a fingerprint), are used in conjunction with the card itself.

At the time of writing, PLAID has not been adopted as ISO standard yet. However, the latest version of the PLAID specification supports part 3 [86] and 6 of ISO 24727, a standard that lists a number of requirements for smart cards used as identification cards. This means that it is on the right track for approval as an ISO standard.

The operation of the PLAID authentication protocol is as follows. First, the card is initialized meaning that a number of values are set on the card. This includes a list of asymmetric keys with corresponding identifiers referred to as `KeySetID`s. In addition to that, the card is preloaded with a `DivData` value which is used for diversification of the single shared symmetric key. The `OpModeID` is a two byte number on the card which indicates the supported mode of operation [84]. For example, such a mode could indicate that a PIN code is used for authentication. Finally, the card contains a `ACS Record` which can be used for authorization, i.e it indicates the role or level of permission that a legitimate user has.

Figure 7.3 illustrates the subsequent steps of authentication process. The term 'interface device' (IFD) is used in this figure which is the same as an NFC reader.

1. In the initial authentication request, the IFD send a list of `KeySetID`s that it supports in order of preference.

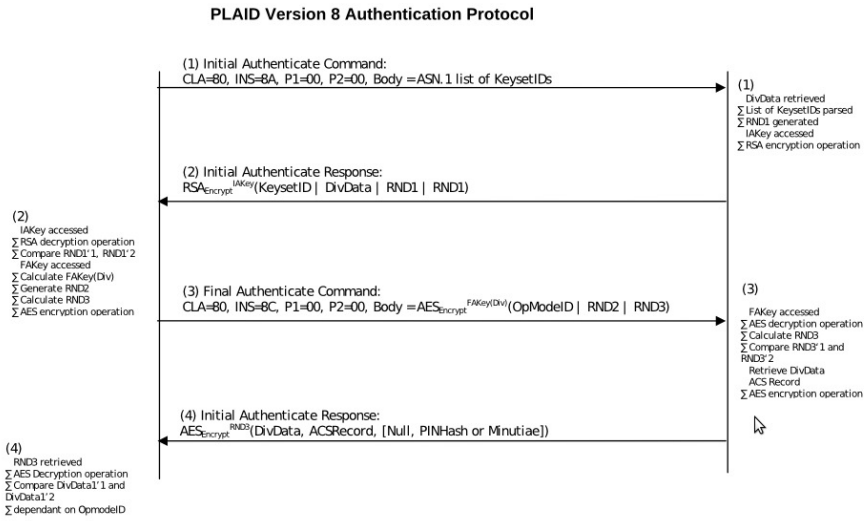

Figure 7.3: Operation of PLAID protocol, copied from [84]. The left side is the IFD and the right side is the ICC

2. The ICC will extract the list of received `KeySetID`s and select the first key, IAKey, that is supported.Subsequently, a string, `STR1`, is calculated from a randomly generated string, `RND1`, as follows:

   $$\texttt{STR1} = \texttt{KeySetID} \mid \texttt{DivData} \mid \texttt{RND1} \mid \texttt{RND1}$$

   The random string `RND1` is included twice such that it can be used as a checksum. Before being sent to the IFD, `STR1` is encrypted asymmetrically using the `IAKey`.

3. The IFD will decrypt the received `STR1` by trying the supported keys. A successful decryption is indicated by the presence of two equal numbers (`RND1`) in the decrypted string. The IFD generates a new random number, `RND2` and calculates a new random number, `RND3` by using SHA as a hash function:

   $$\texttt{RND3} = \text{SHA}(\texttt{RND1} \mid \texttt{RND2})$$

   `RND3` is used as a session key to encrypt all traffic symmetrically

   $$\texttt{STR2} = \texttt{OpModeID} \mid \texttt{RND2} \mid \texttt{RND3}$$

   `STR2` is encrypted using a symmetric key, referred to as the final authentication key `FAKey`, which is calculated from a shared master key diversified by using `DivData`.

4. The ICC receives the encrypted `STR2` and decrypts it by calculating `FAKey`. After

this it verifies `RND3` by calculating this number itself. If it matches the received one, it indicates that IFD was able successfully decrypt `STR1`, containing `RND1` and thus the IFC is authenticated to the ICC. If the authentication is indeed successful, the ICC will respond with `STR3`, symmetrically encrypted using `RND3` as a key.

`STR3` = `DivData` | `ACSRecord` | (Null, PINHash and/or Biometric Minutiae)

The last object optionally contains information for the multi factor authentication such as a hashed PIN code or biometric information.

5. Upon reception, the IFD decrypts the message and obtains `STR3`. It verifies that the `DivData` is the correct value. After this the authentication is complete.

6. The authentication also results in a secure channel between the IFD and ICC: `RND3` is used as a session key to encrypt all data. After authentication, the card behaves as a secure memory: data can read and written from/to the card over the secure channel with a the `get data` and `set data` command. This allows the card to be used in a variety of other applications than physical access control. For example, in a payment application the secure memory could be used to store the saldo of a user and update it after a payment.

To the best of our knowledge, PLAID is currently not widely deployed and therefore its quality is not proven in the field. Hence, we depend on scientific publications that evaluate this protocol.

In [85], the PLAID protocol is evaluated. An advantage is the use of hybrid cryptography, i.e use both asymmetric and symmetric encryption. In terms of transaction time, this is better compared to using only asymmetric encryption [2].

In Chapter 5, it was concluded that PSK based systems are easy to set up but do not scale well whereas for PKI this is vice versa. The fact that PLAID uses an hybrid model poses the question about the set up cost and scalability of PLAID.

The hybrid model used in PLAID does not require the set up of certificate authorities since no certificates are used. This means lower set up cost. PLAID uses key sets (a set of multiple public and corresponding private keys). A key set can be used for multiple cards, meaning that no shared secret is required for each card as is done with PSK. This improves scalability compared to using PSK. However, an increase in card numbers in the system results in an increase in the number of keys in the key set. This means that the reader has to try more keys before finding the right one when decrypting `STR1` which may take too long. Thus, this limits the scalability. The conclusion is the hybrid model of PLAID can be placed between PSK and PKI in terms of scalability and set up cost.

[2] Asymmetric encryption requires considerably more processing power compared to symmetric encryption

7.3.8 OPACITY

Open Protocol for Access Control, Identification, and Ticketing with privacy (OPACITY) [87] is an open authentication protocol targeted at smart cards. Its application include access control and also ticketing in public transport systems. OPACITY is compliant with many smart card related standard/recommendation documents including ISO 24727. It offers mutual authentication (depending on the mode), support for multiple cipher suites, end-to-end confidentiality and a key-agreement protocol that supports forward secrecy (see Section 5.2).

As opposed to PLAID, support for multi factor authentication is not included. Furthermore, support for the exchange of authorization based information (PLAID enables this by using the ACS record as explained in Section 7.3.7) is not included.

OPACITY relies on PKI (see Section 5.3.2) in order to provide authentication. Every card and terminal are pre-loaded with a unique signed Card Verifiable Certificate (CVC), a corresponding private key and a number of trusted root certificates. The CVC contains the identity of the card or terminal and is defined according to the X.509 standard. The authentication process is similar as explained in Section 5.3.2: the certificates are exchanged and the signature is validated. Subsequently, a challenge/response mechanisms is used to verify that a node has the private key corresponding to the certificate that it presented.

OPACITY also supports *identity privacy* meaning that an attacker cannot read the identity (CVC) from the card. This is accomplished by encrypting the card's CVC symmetrically before it is sent to the terminal. The symmetric key is derived using a key-agreement protocol called Elliptic curve Diffie-Hellman (ECDH). The exact operation of this protocol is not discussed here but this protocol makes sure that only a valid terminal can read the CVC.

Persistent binding is an optimization feature. The idea is to save time in the key agreement phase. When a particular card is presented to a particular terminal for the first time, the session key is derived and stored in both the card and the terminal together with an one time card identifier (referred to as the PB record). This key and identifier can be used the next time the card is presented to the same terminal which will speed up the transaction time.

Two modes of operation are defined:

1. **OPACITY-FS** - This mode is designed for use in applications with high security requirements. Therefore, it supports mutual authentication, end-to-end confidentiality and forward secrecy.

2. **OPACITY-ZKM** - This mode is designed for applications where short transactions times are required. It does not offer mutual authentication: the terminal will only authenticate the card and not vice versa which decreases the transaction time but comes at the cost of security.

The architecture of a OPACITY based system is shown in Figure 7.4. The terminal consists of the Secure Authentication Module (SAM), which is responsible for authentication and cryptographic functions, and the client application which implements the service/business logic.

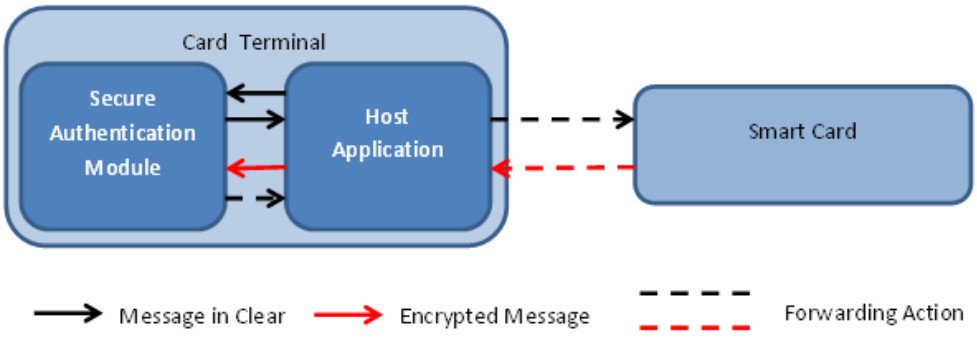

Figure 7.4: Architecture of a OPACITY based system, copied from [85].

This architecture allows for the easy integration of OPACITY in existing client applications. During the authentication process, the client application acts as an interface between the card and the SAM meaning that it simply relays the traffic between these two entities. After the authentication, the client application can communicate with the card over a secure channel by letting the SAM encrypt and decrypt the data.

To the best of our knowledge, OPACITY is currently not widely deployed and therefore its quality is not proven in the field. Hence, we depend on scientific publications that evaluate this protocol. A cryptographic analysis of OPACITY is given in [88]. In this work it is concluded that there are restrictions to the privacy guarantees (identity privacy). Furthermore, according to [88], it is not recommended to use OPACITY-ZKM mode for deployment due to weak authentication and identity privacy in this mode.

In [85], three limitations of OPACITY are described.

1. **High transaction times** - The asymmetric encryption used is slow compared to symmetric encryption due to limited processing capabilities in smart cards. This results in higher transaction times.
2. **No CVC revocation list** - Typically, in PKI-based systems a list of revoked certificates is maintained in a *certificate revocation list* (CRL). Certificates on this

list should not be trusted any more. OPACITY does not use a CRL or similar mechanism. Effectively, this means that when a smart card is stolen, there is no way to mark a card as untrusted. This will make it difficult to prevent that an attacker with a stolen card gains access illegally.

3. **No support for multi factor authentication** - In an environment where security requirements are strict, multi factor authentication is often desirable. The lack of support for this means that OPACITY can not be used in an environment where this feature is required.

7.4 Comparison of authentication protocols

Table 7.1 show which requirements are supported by the described authentication protocols. The following symbols are used:

1. ✓- feature is supported.
2. $\subseteq$ - featured is partially supported.
3. $\times$ - feature is not supported.
4. $\neg$ - feature is not applicable.
5. ? - Not enough information available.

	EAP-PSK	**EAP-TLS**	**EAP-TTLSv0**	**PEAPv0**	**EAP-FAST**	**EAP-IKEv2**	**PLAID**	**OPACITY**
1. Key generation	✓[73]	✓[70]	✓[70]	✓[70]	✓[70]	✓[70]	✓	✓
2.≥ 128 bits effective key strength	✓[73]	✓[70]	✓[70]	× [70]	✓[70]	✓[70]	✓[1 3]	✓[1 2]
3. ≥ 64 byte session key	✓[73]	× [70]	× [70]	× [70]	✓[70]	✓[70]	× [3]	× [2]
4. Mutual authentication	✓[73]	✓[70]	✓[70]	✓[70]	✓[70]	✓[70]	✓[84]	⊆ [6]
5. Shared state equivalence	?	?	?	?	?	?	?	?
6. Resistance to dictionary attack	✓[73] [14]	✓[70]	✓[70]	✓[70]	✓[70]	✓[70]	✓[7]	✓[7]
7. MiTM attack protection	✓	✓[70]	⊆ [70]	✓[70]	✓[70]	✓[70]	✓[17]	✓[17]
8. Protected ciphersuite negotiation	¬ [8]	✓[70]	✓[70]	✓[70]	✓[70]	✓[70]	¬ [8]	¬ [8]
9. Fragmentation	× [73]	✓[70]	✓[70]	?	✓[70]	✓	¬ [18]	¬ [18]
10. End-user identity hiding	× [73]	✓[70]	✓[70]	✓[70]	✓[70]	✓[70]	✓[10]	✓[9]
11. Tested in the field	✓	✓[70]	✓[70]	✓[70]	✓[70]	✓[70]	×	×
12. Compatible with NFCIP1 & 2	✓[19]	✓[19] [13]	✓[19]	× [4]	✓[19]	✓[19]	✓[12]	✓[12]
13. Limited processing power	?	× [14]	× [14]	× [14]	× [14]	× [14]	✓	✓

Table 7.1: Evaluation of authentication protocols

[1]The AES encryption algorithm is used for which the best known attack is a brute force attack meaning that key size equals the effective key strength.

[2]The minimum session key length in OPACITY is 112 bits or 14 bytes and the maximum is 256 bits or 32 bytes. [87]

[3]The session key in PLAID is 16, 24 or 32 bytes. [84]

[4]PEAP encapsulates EAP packets in a TLS secured channel which is not supported by [69]

[5]No EMSK or equivalent exists in this protocol.

[6]Mutual authentication is only supported in OPACITY-FS mode.

[7]RFC 4764 [73] concludes that EAP-PSK is protected against dictionary attacks because no passwords are used in this protocol. PLAID and OPACITY also do not use passwords. Hence, it can be concluded that these protocols are protected against dictionary attacks.

[8]EAP-PSK, PLAID and OPACITY do not support cipher suite negotiation.

[9]Identity information on the card (CVC) can only be accessed by valid terminals. Hence, it can be concluded that identity protection is supported.

[10]According to the standard, PLAID provides privacy protection.

[11]The Persistent binding feature in OPACITY, described in section 7.3.8, can be considered as fast reconnect.

[12]PLAID and OPACITY are designed for smart cards and uses APDUs as defined in [89] for encapsulating the data. Since the NFC standard also support APDUs, it means that PLAID can be used over NFC.

[13]An open source java card implementation of EAP-TLS can be found on github [90]. Since NFC can be used in conjunction with a java card based secure element this indicates that it should be possible to use EAP-TLS over NFC. But this would have to be tested in order to confirm this.

[14]The performance of the open source java card implementation of EAP-TLS [90] is discussed in a master thesis [91]. The TLS handshake takes 10.5 seconds to complete in this implementation (for 'typical' ciphersuite TLS_RSA_WITH_RC4_128_SHA). This is high compared to the authentication times that are common in smart card specific protocols such as PLAID (0.3ms). Therefore, it is concluded that TLS-based protocols do not satisfy this requirement.

[16]This depends on the chosen PACS.

[17]The standards of OPACITY [87] and PLAID [84] explicitly mention protection against MiTM.

[18]Support for fragmentation is not mentioned in the standards of OPACITY [87] and PLAID [84]. However, both protocols are designed for smart cards which uses the same communication standards as NFC and can cope with the MTU limitations in NFC.

[19]An RFC draft [69] with title "EAP Support in Smartcard" enables to the use of EAP over the ISO 7816 application layer protocol meaning that any EAP method can be used over NFC.

[20]These protocols support various cipher suites. Depending on the used cipher suite a combination of encryption algorithms is used. Therefore, the processing of an authentication protocol in a secure element depends on the selected cipher suite.

Part III

System design

Chapter 8

Physical Access Control System

Many physical access control systems are available. It is infeasible to describe all of those systems. For this work, three PACS were studied. Two of those are documented in scientific literature [92] [93]. The third PACS was developed by Nedap Security Management, the company at which this thesis was done. Section 8.1 describes a "typical" PACS that is based on the similarities between the studied systems. Section 8.2 describes media readers which is a particularly important component of a PACS when investigating NFC.

8.1 System Design

As mentioned in Section 1.3, we focus on an on-line system. Figure 8.1 shows an example of an on-line PACS with several access points. An access point is a point where an individual can enter and/or leave a room, building or other resource that is subject to access control. An access point consists of a door and a media reader. The door can be opened electronically. The media reader is the equipment that assists with the authentication. Examples are a card/NFC reader, biometric scanners and key pads. The controller is an on-site compact computer that orchestrates the behaviour between the media reader and electronic door.

Depending on the type, the media reader will read information from the media. This information is passed on to the controller. The controller performs authorization, i.e. determine whether this user is allowed to enter this door. This is done by means of the authorization data. If the the user is authorized, the controller will send a signal to the electronic door to open.

The access control server (ACS) is a centralized server that may reside on a different location then the controller. It stores the following data:

- **Authentication data** is data that can be matched with data obtained from the media reader. In the case of a biometric scanner it is a biometric feature that was

taken when the user was registered. In the case of smart card, it is a string that uniquely represents the identity of a user.

- **Authorization data** specifies what a user or a group of users can and/or cannot do.
- **Configuration data** specifies to which access point the doors and media readers belong. Furthermore, it may include information that maps access points in the system to a location in the real world.

The ACS pushes this data and any updates to the controllers using a computer networking infrastructure (Ethernet, VLAN etc). The controller will store this data in its local database. This happens in an intelligent way. For example, authorization data that is related to room A is only forwarded to the controllers that controls the doors/media readers of room A.

A number of applications are provided that enable the configuration and management of the system. A 'receptionist' application is provided for the receptionist that registers new users to the system. An administrator application is used for setting the configuration data.

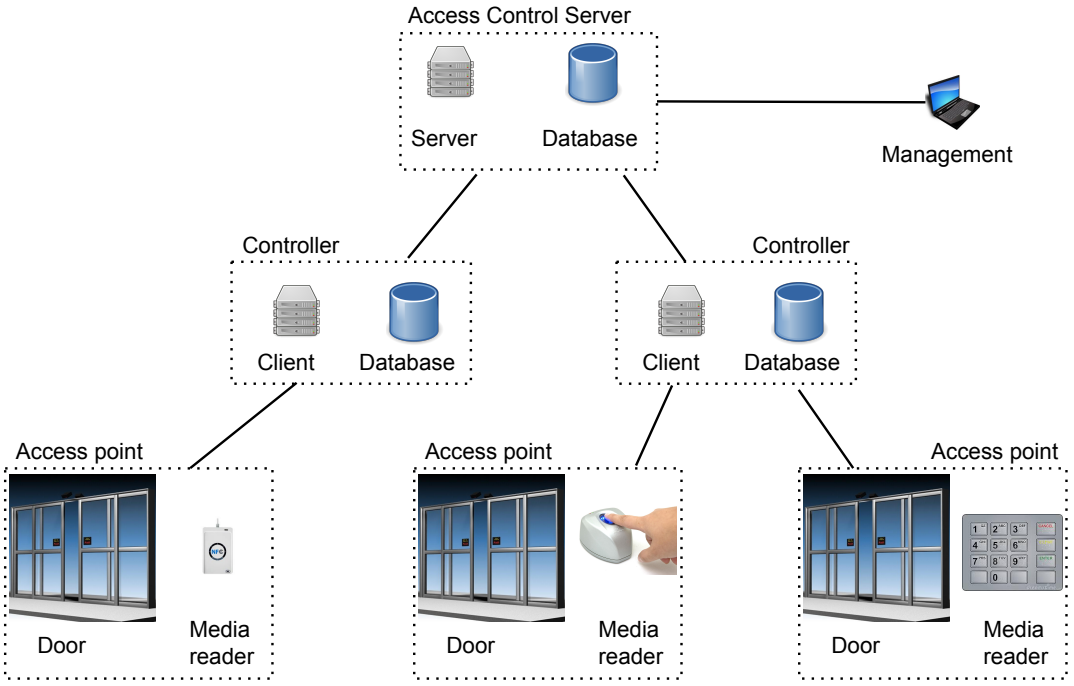

Figure 8.1: An example of a physical access control system.

8.2 Media readers

Media readers play an important role in physical access control systems as they enable the authentication between the user and the PACS. In this work, the focus lies on NFC compatible readers, e.g readers that support ISO 14443, because this enables the communication between the phone and the PACS using NFC. Therefore, this section briefly describes the functionality of an NFC readers and other type of media readers are not discussed.

A distinction can be made between two types of readers based on the interface type between the controller and the reader:

1. **Low-level interface reader** - The interface provided by the reader allows the controller to send and receive bytes to/from an NFC device. An example is the *Personal Computer Smart Card (PC/SC)* interface. In this case, the controller is responsible for implementing an authentication protocol.

2. **High-level interface reader** - In this case the reader performs the authentication. The controller receives the identity information from the reader and is not involved in the authentication process. To the best of my knowledge, there is no public documentation available about such an interface.

Some readers provide both types of interfaces. In a high-level interface reader, it may be difficult or impossible to change the authentication protocol because this is implemented in firmware, i.e. a program stored in a non-volatile memory in the reader, which may be difficult to modify.

Chapter 9

Design choices

This chapter identifies the design choices and discusses two approaches for using an NFC-equipped mobile phone in physical access control. Based on the studied technologies in the previous chapters many solutions can be devised. In Section 9.1 we eliminate solutions that we consider problematic or not interesting from a research point of view. Subsequently, we discuss two solutions in Section 9.2.

9.1 Pre-selection

9.1.1 Secure Element

In Chapter 6, several types of secure elements were discussed such as the UICC, embedded hardware and secure memory cards. The main advantage of these technologies is that they provide good security.

In most mobile phones with NFC capabilities, the NFC controller is connected to the UICC / embedded hardware type of secure element which allows the mobile phone to behave as a contactless smart card. However, up until now, few TSM services have been deployed and thus it is difficult for a service provider to make use of these types of secure element. Therefore, we focus on the secure memory card (a secure element and optionally NFC controller/antenna on an SD-card) which can be purchased and thus owned by the service provider itself.

It is worth noting that most type of secure elements are based on the Java Card Open Platform (*JCOP*, an operating system for smart cards / secure elements). An applet for a SMC secure element could also be used on the UICC. This means that if TSM services become more widely available, an SMC-based solution can be re-used.

9.1.2 Cloud-based solution

In section 6.6, a secure cloud storage solution is discussed. This solution could also be used in physical access control. One approach would be to download an 'authentication

applet' and install this on a conventional secure element similarly as described in Section 6.6. However, such a solution still requires a secure element which may be problematic.

An alternative cloud-based approach could be that the phone authenticates to a server using the network connection. This authentication could be done using the TLS handshake that is commonly used over TCP/IP. The phone contains a X.509 certificate with a public key. The corresponding private key is securely stored in the TrustZone (see Section 6.7).

If the authentication between the phone and the server is successful, the phone simply relays all communication between the NFC reader and the server from that moment on using HCE to communicate with the NFC reader. The server will perform the authentication with the NFC reader on behalf of the phone. The advantage of such an approach is that it does not require a secure element while it integrates more easily with an existing protocol on the NFC reader.

The latter may be desirable when it is difficult to change the software that controls the NFC reader. This is an important reason that a cloud-based approach is becoming popular in NFC payment services. In the last decade, much effort has been done to standardize payment protocols worldwide and many NFC payment terminals now use those protocols. Therefore, it is important for a service provider that this protocol is used in order to be relevant on the market. The scale of a PACS is typically smaller compared to payment systems and the behaviour of the reader can be modified more easily (see Section 8). If this indeed the case, it is devious to choose a cloud-based solution.

A major problem with a cloud-based NFC solution in physical access control is the high transaction time. The state machine of a 3G radio will typically enter a low power mode after several seconds of network inactivity. From this state, it will take between 1.5s and 2.0s before radio communication is possible [94]. The total transaction time will be significantly higher compared to many existing contactless smart card solutions. For example, in Section 7.3.7 we saw that PLAID provides a transaction time of 0.3s.

Internet availability may also impose a problem. Not all smart phone users have a subscription for data services. Most data subscriptions impose a monthly data limit. Furthermore, the mobile network may suffer from poor coverage on certain locations (remote area, in-door locations etc).

Using WiFi could provide a solution. To the best of our knowledge, it does not have a lower power mode and thus it should provide a better transaction time. However, setting up WiFi requires more configuration on the phone by the end-user. Furthermore, it may be difficult or costly to deploy a WiFi infrastructure.

Another disadvantage with a cloud-based approach is that it is not possible to limit the transaction time in order to protect against relay attacks (see Section 5.5). In fact, the cloud-based approach that was described in this section can be seen as a 'legitimate' relay (attack). Thus, a transaction time-out will disable relay attacks but also make it impossible to use a cloud based solution.

9.1.3 Storing symmetric keys

Most current physical access control systems rely on pre-shared key based authentication as described in Section 5.3.1. An example is the MiFare contactless smart card (Section 9.1.4) and PLAID (Section 7.3.7).

If it is important that the NFC solution integrates with such a solution, for example when the software on the readers can not be changed, this means that the phone will have to store the symmetric key. This can be done using a secure element (this approach will be discussed in Section 9.2.1).

Alternatively, the key can be stored on the main storage device. This, however, requires that the key is encrypted which in turn requires that the user provides input such as a PIN code that is used to decrypt the key (see Section 6.4). This is not desirable in all type of access control situations.

Another possibility is to secure the symmetric key ($k_{symmetric}$) using the TrustZone. The TrustZone can currently only store (asymmetric) RSA private keys ($k_{RSA,\ private}$). In order to secure $k_{symmetric}$, it can be encrypted with $k_{RSA,\ public}$ after which it can only be obtained if $k_{RSA,\ private}$ is known which is securely stored in TrustZone.

A problem with this approach is that cryptographic operations are performed by software on the main CPU and not in a trusted execution environment and secure memory. Potentially, the key can compromised when it resides in main memory (memory leakage).

9.1.4 MiFare and DESFire integration

MiFare is a proprietary contactless smart card product by NXP. These types of cards are commonly used in physical access control and public transport (for example, the OV-chipkaart in the Netherlands). MiFare uses the ISO 14443 standard for communication which is part of, and thus compatible with, NFC. The functionality and authentication protocol of MiFare is similar to PLAID: it provides a secure memory that is protected by one or more symmetric keys.

It may be desirable for a service provider that an NFC solution is compatible with the MiFare technology. This could be done by developing an applet that mimics the behaviour of such a card. Due the popularity of MiFare, some type of secure elements (such as secure memory cards) can optionally be pre-installed with such an applet by the secure element manufacturer. If MiFare integration is important, this option should definitely be considered. From a research point of view, however, it is not interesting to investigate such a ready-to-use technology which is why we do not consider it any further.

Alternatively, the MiFare behaviour could be implemented on the mobile phone using HCE. This may introduce the following problems:

1. **Limited compatibility with ISO 7816** - MiFare cards do not use the ISO 7816 application layer but a proprietary protocol. This is not compatible with the applet selection procedure that is used by the HCE implementation on Android phones. The newer DESFire card (the successor of MiFare) does support ISO 7816 by wrapping a proprietary command into an APDU. Furthermore, a 'native' ISO 7816 command set is defined for DESFire. In practice, however, many card readers only support the proprietary protocol. This means that the software on the reader will have to be modified in order to be compatible with HCE. However, an important motivation for retaining the MiFare technology is that the reader software does not have to be modified.

2. **Storing symmetric key** - See Section 9.1.3.

3. **Secure memory** - MiFare cards provide a secure memory, i.e. a memory which content can only be read/modified by an entity that knows the key of that card. Such a memory can not be implemented on the phone without a secure element. It is possible to protect data from being read by encrypting it in a similar way as done with "storing a symmetric key" (Section 9.1.3). However, protecting data from being modified will always rely on the security features of the mobile operating system which we consider not secure (see Section 4.3.5).

   For example, consider a secure memory that is used to store a money saldo. This saldo is decreased with every payment transaction. An attacker/user could potentially roll back the saldo to a previous recorded value given a non-secure memory which allows him to make payments for free. In access control, the secure memory is often used for storing authorization (not authentication) data for which this may also be problem.

4. **Variable UID** - The UID is a unique identifier that every ISO 14443 card emulation device has for use in the anti-collision protocol. The HCE implementation in Android uses a random UID on each transaction. However, some systems use the UID to identify a certain card. This is no longer possible when the Android HCE is used.

The functionality of PLAID is similar to MiFare. The difference is that PLAID always uses ISO 7816 for communication and does not rely on card UIDs meaning that problem 1 and 4 do not apply to PLAID. Problem 2 and 3 are still problematic in PLAID. For these reasons we do not consider the implementation of MiFare/DESFire and PLAID using HCE.

9.2 Solutions

This section discusses two solutions. A solution addresses the following:

- An selection of a secure storage technology.
- An selection of an authentication protocol.
- The interaction between the entities e.g. mobile phone, secure element, NFC reader and access control server.
- The personalization procedure for a user, i.e. how is required software and credential installed on the phone and/or secure element?

9.2.1 Approach 1: SMC with applet

Authentication & secure storage & interaction

Solution 1 combines a SMC Section 6.3 with the PLAID authentication protocol (Section 7.3.7). The SMC is a micro SD-card that is inserted into the mobile phone which architecture is shown in Figure 9.1. It contains an NFC chip and a secure element. The authentication occurs directly between the phone and the NFC reader. An applet is developed that can be installed onto the secure element. This applet is responsible for storing the credentials and implementing the PLAID authentication protocol. Furthermore, software is required in order to implement the PLAID authentication on the NFC reader side.

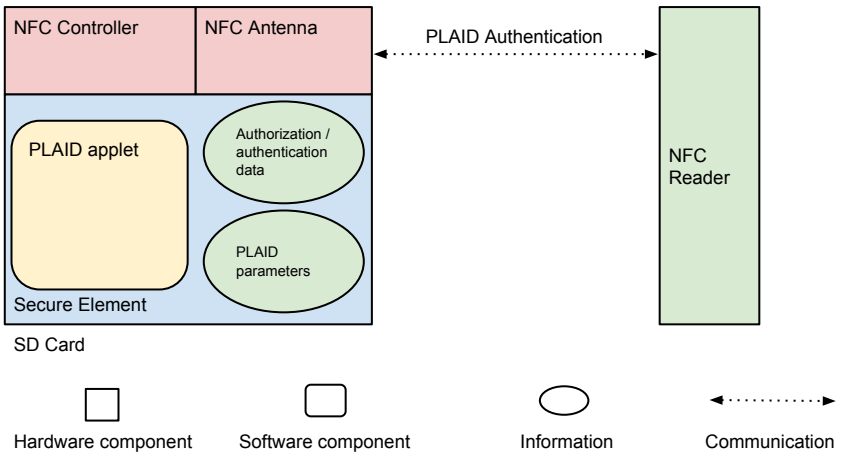

Figure 9.1: The architecture of an NFC secure memory card

The main motivation for using the SMC is that it is currently the only secure storage option that provides both good security and accessibility. The PLAID protocol is chosen because it is specifically designed for use in secure elements and smart cards. The hybrid encryption techniques provide good transaction times and low set up costs compared to OPACITY. Depending on scalability requirements, however, OPACITY may be favourable.

Personalization

The personalization process of the SMC occurs in two phases. The first phase is referred to as the pre-personalization process which involves the installation of the PLAID applet on the secure element and setting a number of variables that are required for the authentication protocol to function. These variables are specific for the security domain in which the card is going to be used, e.g. an organisation or a company. For the PLAID protocol this are `KeysetIDs`, `DivData` and `OpModeID`. This can be done when the card is manufactured or when the card is purchased by the service provider.

After pre-personalization, the PLAID applet is functional and the SMC card can be inserted into the mobile phone. The second phase, referred to as personalization, involves coupling the SMC card/phone to a specific user in the security domain. After pre-personalization, the PLAID applet provides a secure memory, i.e. a memory that has a security association with the PACS and is only accessible by the PACS. Therefore, the authentication information can be a simple string that uniquely identifies the user.

The authentication and optionally authorization information are stored on to the card. This can be done over the NFC interface. In this case, the phone is placed near

the NFC reader which writes the authorization data to the applet. Alternatively, it can be done over-the-air. This requires an application on the mobile phone that relays the communication between the PLAID applet on the secure element and a remote server. Personalisation over-the-air has the advantage that it does not require a user to come to a point-of-service.

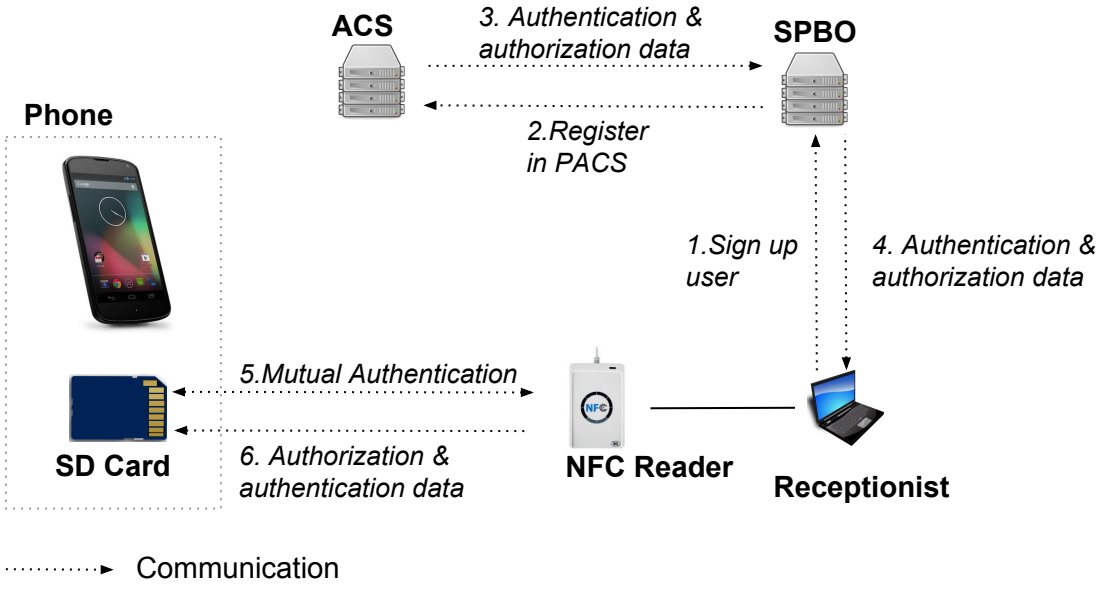

Figure 9.2: The personalization procedure over NFC.

After personalization the phone can be used in the access control system as shown in Figure 9.3. A user that wants to open a door at an access point presents its phone to the NFC reader. The PLAID protocol performs mutual authentication between the phone and the NFC reader after which the reader obtains the authentication and authorization data from the phone/PLAID applet. The physical access control system performs authorization using this information and makes decision whether to open or not open the door.

Evaluation

Table 9.1 summarizes the advantages and disadvantages of this solution.

9.2.2 Approach 2: Host card emulation

Table 9.2 summarizes the advantages and disadvantages of this solution.

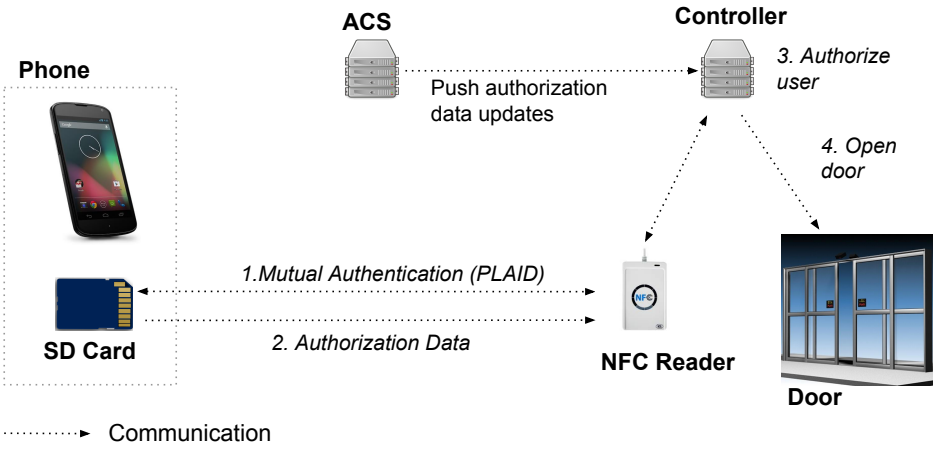

Figure 9.3: The authentication procedure between the phone with NFC and the PACS.

Pros	Cons
Good security (threat 1) due to secure element	Requires SD-slot in the phone
Integrates with ISO 14443 readers	Difficult to use for NFC services from multiple SPs.
Easy to re-use when user buys new phone	No protection against relay attacks

Table 9.1: Summary of pros and cons based on the selected secure storage and authentication technologies.

Authentication & secure storage & interaction

This solution utilizes the recently introduced host card emulation which enables integration with existing ISO 14443 reader infrastructures without using a secure element. The TrustZone technology is used for credential storage and the authentication is done between the NFC reader and the phone.

Because the current implementation of the TrustZone only allows for the storage of RSA keys, it is important to select an authentication protocol that supports this type of keys. Furthermore, using HCE implies that the authentication protocol and corresponding encryption is done on the main CPU. Therefore, it is not required to select an authentication protocol that supports the *limited processing power* criteria from Section 7.2.

EAP-TLS, EAP-TTLS, EAP-IKEv2 and PEAP support RSA keys and X.509 certificates. EAP-TTLS and PEAP provide backward compatibility with password-based authentication which is not a requirement. I chose EAP-TLS in favour of EAP-IKEv2 because Android comes with TLS [1] libraries which can be (partly) re-used making the

[1]EAP-TLS re-uses the authentication handshake part of TLS

realization of this solution easier.

Figure 9.4 explains the authentication procedure. The EAP-TLS authentication is done directly between the reader and the phone. EAP-TLS uses certificates for authentication which will have to be distributed. The systems contains three types of certificates.

- **Root certificate (C_{root})** - this certificate and corresponding private key is used to sign other certificates in the system. This certificate is self-signed meaning that the signature is generated by its own private key.
- **Phone certificate (C_{phone})** - the certificate that is unique for every phone
- **Reader certificate (C_{reader})** - a certificate that is unique for every reader

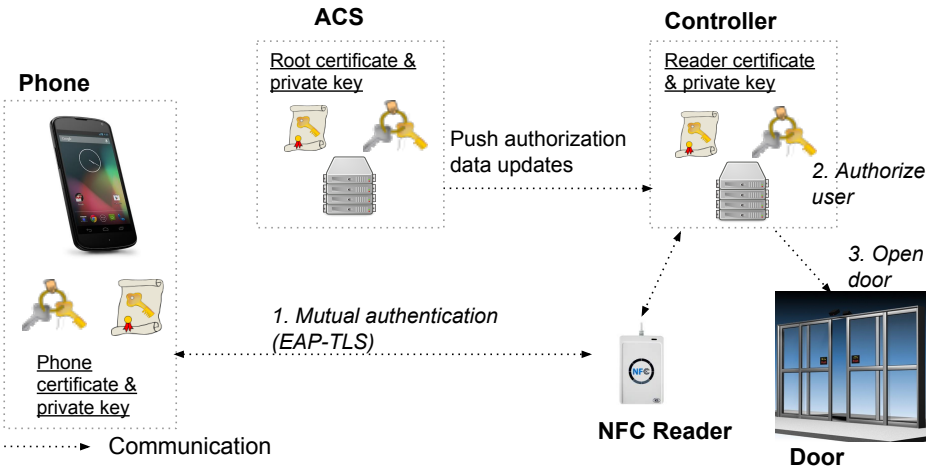

Figure 9.4: The authentication procedure between the phone with NFC and the PACS.

The phone and the reader each have their own type of certificate with the private key which together form the credential. Furthermore, both type of devices have a copy of the root certificate (not shown in Figure 9.4). This is used in the authentication process; when the reader and phone exchange their own certificate it is verified that this is signed with the root certificate.

The phone's certificate is used as a mean of authentication. If the system requires that authorization data is stored on the phone, this data can be made part of the certificate [95]. Since the certificate is signed, it can be detected if authorization data is modified. Alternately, a separate *authorization certificate* can be used that contains the authorization data which is standardized in [95].

Personalization

Personalization can be done over NFC or over the internet. In this design we focus on personalization over the internet. The advantage of this is that the personalization could be done remotely.

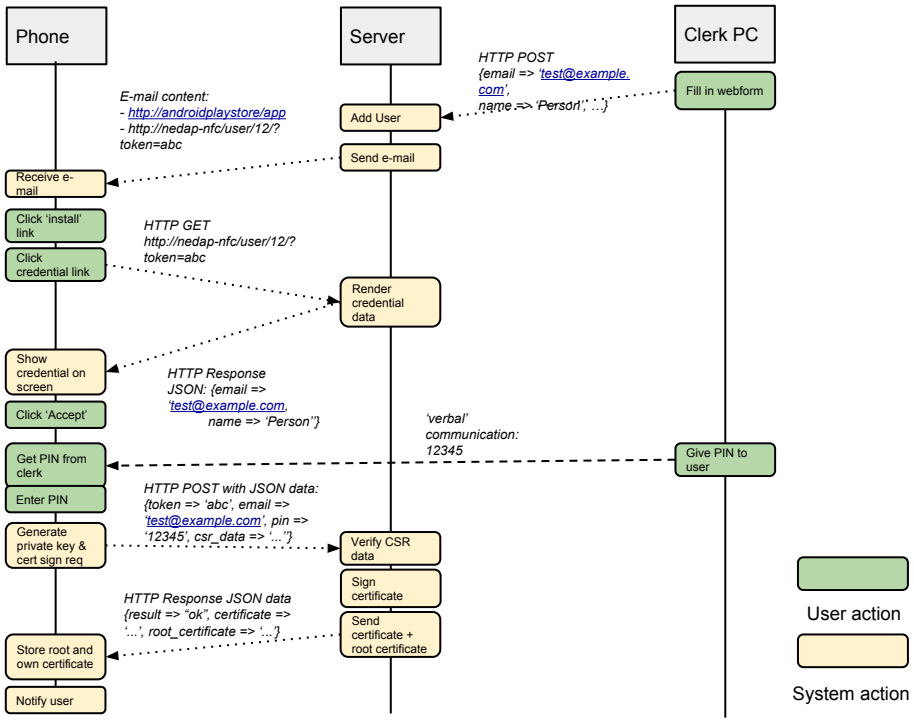

Figure 9.5: A sequence diagram of the personalization procedure.

Figure 9.5 shows a time sequence diagram of the personalization. First, the application has to be installed onto the phone. This step can be facilitated by using an NFC tag. The tag can be formatted such that it contains a link to the installation location. When the user holds the phone near the tag, the mobile phone will offer the user to open the link and the application can be installed. Alternatively, the link to the application can be sent in an e-mail (see the next step).

The second step is setting up a security association between the phone and the PACS and setting up the credential. Unlike the SMC solution, there is no prior security association between the phone and the PACS since no pre-personalization can be done. The phone is owned by the user and not by the service provider. Therefore, credentials can not simply be loaded to the phone as done in the SMC solution.

This issue can be solved by using a single-use token / one time password (OTP) that is sent to the user by e-mail and a confirmation code that is given to the user in person. A single-use token is a string that authenticates the phone to the server for maximum one time. The token is contained in an URL that is sent to the user's e-mail. The URL points to a server controlled by the PACS and uses a secure HTTP connection (HTTPS) which authenticates the server to the phone. The mobile application is set up such that it only trusts a server with a preconfigured certificate in order to prevent MiTM attacks.

E-mail communication is typically not secure meaning that potentially the token can be intercepted. For this reason a second authentication factor is required. This can be done by using a 5 digit confirmation code that is given to the user in person at the point of service. When the user clicks the link in the e-mail it is opened with the mobile application. The mobile phone asks the user to enter the confirmation code. Both the token send by e-mail and the confirmation code entered by the user are sent to the server which will verify both codes.

After this, the application on the phone will prompt the TrustZone to generate an RSA key pair and a so-called *certificate signing requests* (CSR). A CSR contains all the parameters of the certificate and a proof of the ownership of the private key. The CSR is sent to the server which verifies the request and subsequently generates a signed certificate. The signed certificate is sent back to the phone which together with the private key in the TrustZone form the credential.

In this solution, it is crucial that confirmation code is not given to the user electronically. There is not way around this since there is no prior security relation between the phone and the PACS.

Still there are a two important advantages to personalization as described here compared to personalization over the NFC interface. The first advantage is that after the initial personalization, any credential updates can be done fully remotely using the security association that was set up previously. The second is that the personalization process does not require any NFC equipment. For example, the personalization can be implemented as a web application meaning that registering new users can be done from any PC with an internet connection.

Pros	**Cons**
Does not require new hardware	Not easy to re-use when user buys new phone
Protection against relay attacks	Does not work when phone is off / dead battery

Table 9.2: Summary of pros and cons based on the selected secure storage and authentication technologies.

Chapter 10

Implementation

This chapter discusses the implementation. In Section 10.1, a selection is made from the design choices in Chapter 9. Section 10.2 provides an overview of the functionality and limitations of the prototype. The prototype consists of two logical layers that each implement a part of the involved standards. Section 10.3 discusses the implementation in more detail by looking at these two layers. Section 10.4 motivates the chosen ciphersuite of the EAP-TLS protocol. Section 10.5 discusses how the certificates are used. Section 10.6 discusses the usage of hardware and APIs on the mobile phone and reader side. Finally, Section 10.7 discusses the implementation of the personalization server.

10.1 Design selection

I chose to implement approach 2 as described in Section 9.2.2. Host card emulation and hardware-backed secure storage are relatively new and little research has been done into using these techniques. NFC capable phones have been on the market for 7 years but few services have been deployed due to restrictions imposed on the usage of secure elements and NFC in mobile phones. Using HCE and hardware-backed storage gets around these restrictions. It conforms with the StolPAN solution, described in Section 3.1 in the sense that a general secure storage solution is provided by the phone which can be used by any service provider that wishes to implement an NFC service. Moreover, this approach enables to use the mobile phone as an authentication token in multiple services which is an important motivation for using NFC.

10.2 Overview

Figure 10.1 shows the components of the implementation. A Google Nexus 4 phone was used with Android 4.4 which supports HCE and TrustZone hardware-backed credential storage. A smart card reader with a PC/SC interface was chosen which enables the communication with the mobile phone. The reader is connected to a laptop via an USB cable.

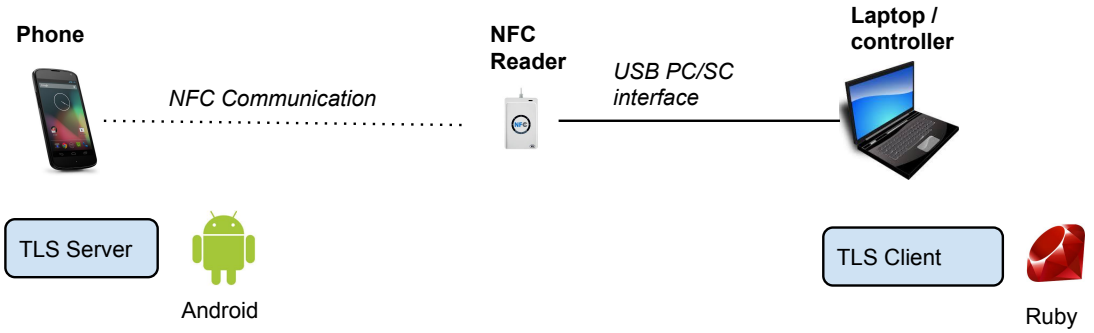

Figure 10.1: The components of the implementation.

In EAP-TLS, the client always initiates the communication. NFC follows the master/slave communication model where the reader is the master and thus initiates the communication. Therefore, it was decided that the reader is the EAP-TLS client and the phone is the server.

The role of the laptop corresponds to the controller component in the PACS that was described in Chapter 8. However, in the prototype only the authentication part is implemented. Thus, authorization and controlling the door is not implemented.

All the behaviour of the mobile phone is implemented in a software application which uses the standard Android API. The reader is controlled by a software application on the laptop that runs the Linux operating system (*Ubuntu 12.04*). This application uses the *PCSCLite* middleware which enables it to control the smart card reader. The reader software is implemented in the Ruby programming language.

No EAP-TLS implementation was found that could be re-used with NFC. However, there are galore TLS implementations which are generally available in most programming languages. The possibility of using such an implementation was researched but turned out to be difficult since all TLS APIs are specifically targeted at TCP/IP links.

Therefore, the decision was made to make a custom EAP-TLS implementation. This implementation is responsible for formatting outgoing and parsing ingoing messages. Furthermore, it keeps a state machine in accordance with the involved protocols. A well known TLS implementation, *OpenSSL*, was used for cryptographic functionalities.

In order to optimize the performance of the system, it was decided to compress the certificates before they are sent. X.509 certificates are around a 1000 bytes each and

compose the majority of the total data that is tranferred. Furthermore, we only transfer the subjects certificates and not the whole chain of certificates, i.e. include the certificates that were used for signing. This means that we have to transfer two certificates instead of 4.

Figure 10.2 shows a stack of the used technologies on the phone and reader side. These are explained in Section 10.6. Furthermore, it shows how the standards and technologies were translated into software which is explained in Section 10.3.

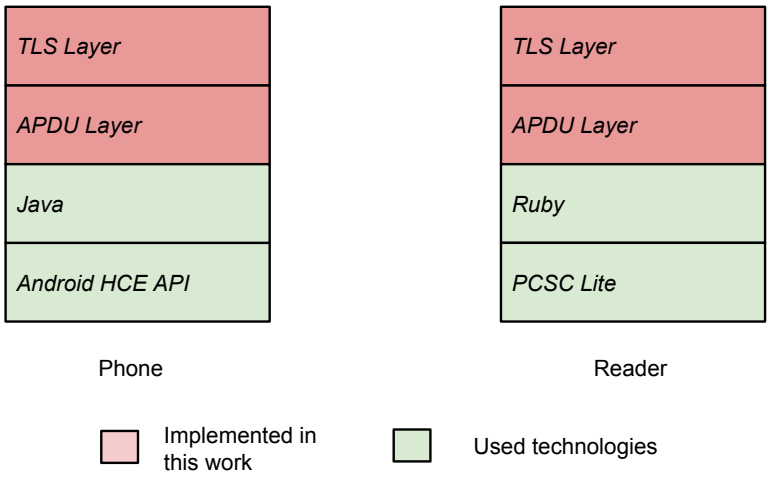

Figure 10.2: A stack of the used technologies and software layers.

Due to the limited time frame of this assignment some simplifications were made in the implementation:

- The EAP layer is not implemented. This is not required for making the implementation functional. EAP specifies an EAP method selection procedure which is not needed since the prototype only supports EAP-TLS. Furthermore, EAP specifies an initial identity exchange in case this is not implemented by the inner EAP method. EAP-TLS, however, can use certificates for the identity exchange. After this initial "EAP handshake" TLS messages are wrapped into EAP request/response messages. In my prototype, the TLS messages are directly wrapped into APDU messages.
- The various layers of the chosen protocols (ISO 7816-4, EAP and TLS) each specify their own error handling procedure. These were not implemented. If any error occurs, e.g. an unexpected or invalid message is received, the authentication is aborted. An error is only emitted on the APDU level.
- The EAP-TLS standard contains a list of mandatory cipher suites in order to guarantee operability. The prototype, however, does not implement one of these

ciphersuites. See Section 10.4 for more details about the ciphersuite.

- The EAP-TLS protocol supports privacy but this is not implemented.

10.3 Layers

Figure 10.3 shows the protocol stack and standards of this solution. The NFC functionalities of the mobile phone and the NFC reader already implement the RF interface up until the transmission layer. In my software solution I combined the APDU and "EAP support in smart cards" standards into one logical layer which is called the *APDU Layer*. This makes sense because the APDU is merely a message format whereas the latter standard describes how those messages are used. The TLS layer implements the EAP-TLS behaviour as described in Section 7.3.2.

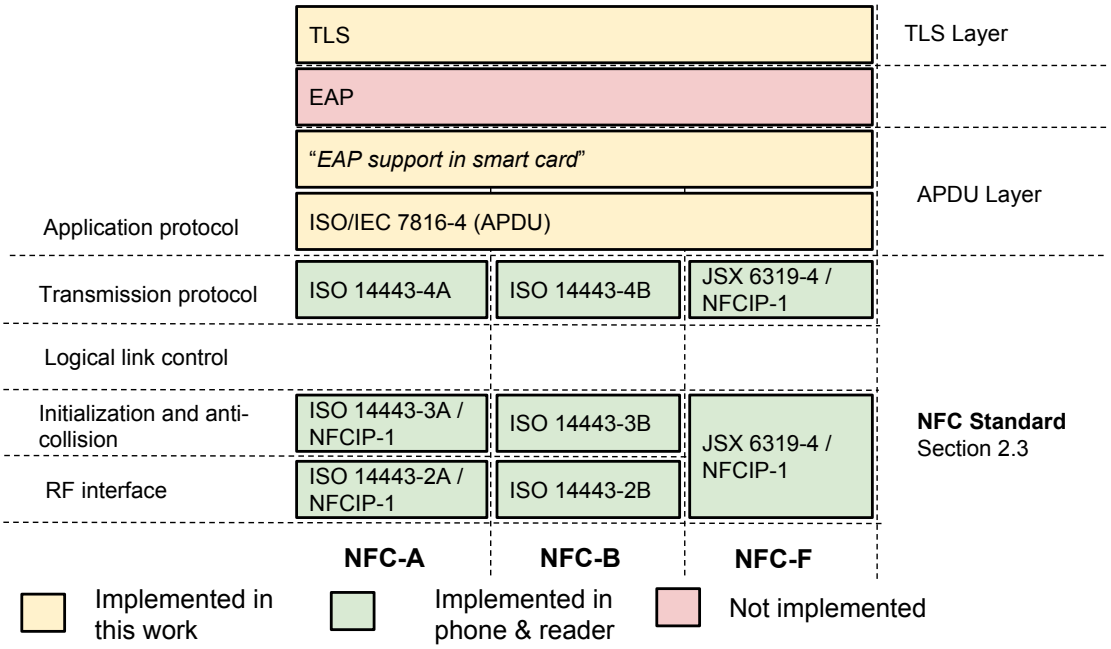

Figure 10.3: The protocol stack of the solution. The layers were combined in the software solution as indicated on the right.

10.3.1 APDU Layer

The APDU message format

ISO 7816-4 [89] provides an application layer standard for formatting data. An *Command Application Protocol Data Unit* (C-APDU) is sent from the reader to the phone. An APDU response (R-APDU) is sent back to the reader. An APDU message contains a mandatory header and optional body.

Figure 10.4 shows the header and body of a C-APDU message. It contains the following fields:

- **CLA** - The instruction class (CLA) is 1 byte. The CLA field is an indicator for a specific class of application specific instructions. This is a mechanism to categorize the instruction (INS) values. A number of classes have been predefined. For example, CLA values for file access, security and payment related operations. When the functionality of a device cannot be expressed by the predefined command set, a CLA value of `0x8n` can be used. In this case, one can define its own instruction set.
- **INS** - The instruction (1 byte).
- **P1** and **P2** - Some instructions may require parameter values. For example, an instruction to write data on the to the card may require a filename or file identifier as a parameter. The two parameter fields are 1 byte each. These are always included even if an instruction does not require parameters.
- $\mathbf{L_c}$ - One byte which indicates the length of the data field in bytes. This field is not included when there is no data.
- **Data** - A variable length data field that may be up to 255 bytes.
- $\mathbf{L_e}$ - One byte which indicates the maximum length of the expected response. A value of `00` indicates a request for all available data.

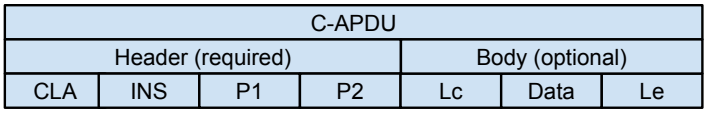

Figure 10.4: Structure of a C-APDU message which is sent from the reader to the phone.

The content of the R-APDU is shown in Figure 10.5. This contains the following fields:

- **Data field** - The variable length data field
- **Status word 1 (SW1)** - A status word which indicates whether the operation requested in the C-APDU was successful.
- **Status word 2 (SW2)** - Another status word. Typically, this provides more information if something went wrong such as an error code. Otherwise it is set to 0.

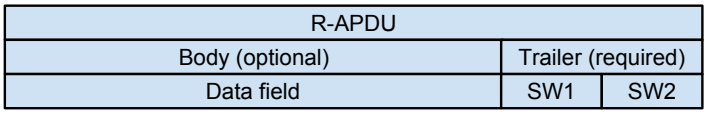

Figure 10.5: Structure of an R-APDU message which is sent from phone to the reader.

Functionality of the APDU layer

The "EAP in smart cards" is a draft standard that wraps EAP in APDU messages and thus enables the use of EAP authentication protocols in smart cards or NFC devices. Furthermore, the standard specifies identity management, e.g. add/remove/list identities, which allows for the personalization of the EAP device but is not implemented.

The most important message defined in this standard is the `EAP-Process` C-APDU. The structure of this message is shown in Figure 10.6. All numbers in this figure and in the text are hexadecimal.

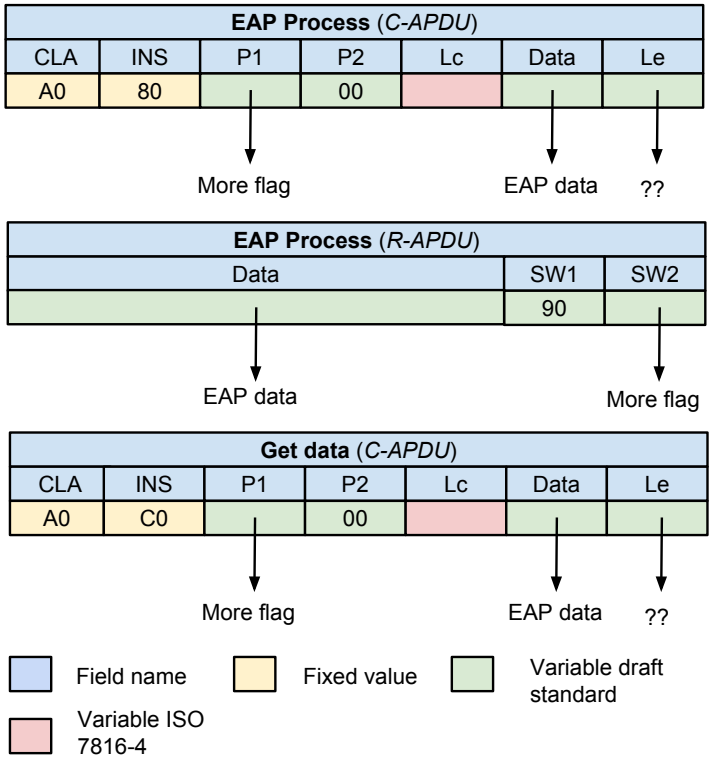

Figure 10.6: Three APDU messages that enable transferring EAP messages over the NFC link.

The `CLA` and `INS` field indicate that the body of the APDU contains an EAP message. When the content of an EAP message does not fit into the body of one C-APDU, it is spread over multiple `EAP-Process` C-APDUs. The `P1` field is used as a *more flag*: when the least significant bit of `P1` is `1` this indicates that more data is coming. Otherwise, it is set to `0`. The `P2` field can indicate the use of *Multiple EAP Identity selections*. The concept is explained in [69] but is not needed for our implementation. Therefore, the `P2` value is always set to `00`

Every C-APDU should be answered by the phone with an R-APDU. When the phone does not have any EAP data to send it replies with the status words trailer `SW1` and `SW2` set to `90` and `00` respectively. When the phone has EAP data to send, it may wrap this in the data field section of the R-APDU. The phone uses the `SW2` field as a more flag.

Since the read/write communication mode follows the master/slave principle, the phone can only reply to a C-APDU. When the phone has more data to send, the reader must continue to query the phone for the additional data by sending a `Get-APDU` until it receives a message with `SW1` and `SW2` set to `90 00`.

The standard also defines a number of status words that indicate certain types of errors. These apply to errors that occur on the APDU layer and not errors that occur on the EAP-TLS layer. The EAP-TLS layers has it own mechanism for propagating error messages.

The APDU layer on both the phone and the reader side work similarly. It receives an array of bytes from the TLS layer. This contains typically one or more EAP-TLS messages. Subsequently, these bytes are sent according to the protocol described above. The received data is buffered and passed on to the TLS layer when the sender indicates that it has no more data to send.

The meaning of the L_e in the context of the `Process-EAP` APDU is not clear to me. The standard says *"The EAP request or response packets lengths are represented by the unknown value XX and YY."* by [69] where YY is the value of L_e. However, this is not possible because the length of an EAP message can be more than 256 bytes which can not be represented by the L_e which is 1 byte. Therefore, I ignored this field in my implementation and set it to `00`.

10.3.2 TLS Layer

The TLS layer has the following responsibilities

1. The generation, formatting and checking of TLS messages.
2. Maintaining a state machine such that correct messages are send/received given the protocol state

3. Performing the cryptographic operations that enable the authentication

TLS Messages

The TLS layer implements the TLS handshake. TLS is divided into to several layers each of which defines one or more message types. The structure of this is represented by a UML diagram in Figure 10.7.

The `RecordLayerMessage` is on the top and contains one or more of the underlying messages. The header is always unencrypted. In TLS, the body (which contains the lower level message) would be encrypted and compressed after the authentication and negotiation is completed. Since EAP-TLS only uses the authentication part of TLS, this encryption is not done in EAP-TLS.

The `HandshakeMessage` type contains the messages for the authentication handshake as discussed in Section 7.3.2. The `ChangeCipherSuite` message is part of the authentication handshake but is not wrapped into a `HandshakeMessage`. Instead, it is directly wrapped into a `RecordLayerMessage`.

The UML classes in the diagram were translated into software classes. These classes are used for parsing (reading an incoming array of bytes) and generation (generate an array of bytes from a number of TLS variables) of messages. The inheritance relations in the UML diagram are also present in the software classes such that the parsing/generation functionality of the `RecordLayerMessage` and `HandShakeMessage` classes is inherited and re-used by the underlying messages (i.e. `ClientHello`, `ServerHello` etc).

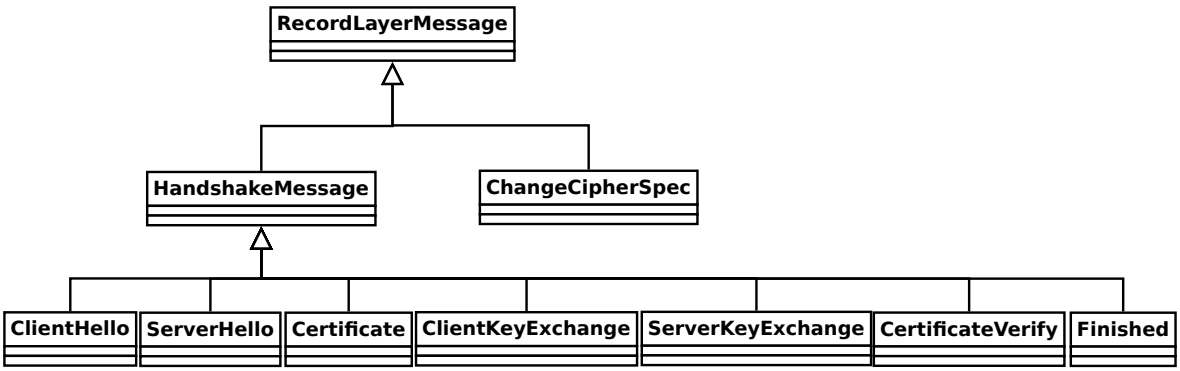

Figure 10.7: An UML diagram of the TLS protocol message types.

Protocol states

The TLS messages are exchanged in four steps as shown in Figure 7.2 on page 63. The message exchange is similar as shown in this figure, the only difference being that

the `ServerKeyExchange` message is not used. This is not necessary because the chosen cipher suite (see Section 10.4) does not require this message to be sent.

The software at the reader and the phone maintain the state as a variable. For each received message there is a state. When a message is received, it is verified that this type of message corresponds to the expected message in the given state. Subsequently, the content of the message is extracted and verified, a response is sent back and the state is increased to the next expected message. If an error occurs, the state is reset to the initial state.

Cryptographic operations

The first cryptographic operation in the EAP-TLS protocol is the verification of the received certificate (C_{reader} on the phone's side and C_{phone} on the reader's side) against the trusted certificate (C_{root}). This is done by verifying that the certificate is valid. Certificates are valid from and until a certain date meaning that this should be checked. Furthermore, it is verified that the certificate is in fact signed by C_{root} and thus can be trusted. This is accomplished by checking the signature on the certificate as explained in Section 5.3.2.

The next phase involves a challenge/response mechanism which verifies that a peer owns the private key corresponding to the public key on the certificate that it sent.

The client proofs the ownership of its private key to the server by sending the `CertificateVerify` message. This message contains a signature that is calculated by hashing all transmitted handshake messages up until now (without the record layer header data) and signing it with the private key. The server can decrypt the signature using the public key from the client's certificate. Subsequently, it can verify the signature by calculating the hash itself and compare it with the decrypted one.

The client generates a 48 bits so-called `pre_master_secret` (2 bits indicate the TLS version number and 46 bits are generated randomly). This is sent to the server encrypted with the server's public key.

At this point, both the client and the server calculate the `master_secret` from the `pre_master_secret`. This is done by using the pseudo random function (PRF) which is defined in the TLS standard [96]. The purpose of the PSF is to expand keys. The exact functionality of the PRF function is not discussed here. The formula below shows the input variables of the PRF and how this function is used in order to calculate the `master_secret`. The client and server random values were exchanged in the

`ClientHello` and `ServerHello` messages. Note that the label of the PRF is set to the string "master_secret".

$$PRF(secret, label, seed) \tag{10.1}$$

$$seed = client\_random + server\_random \tag{10.2}$$

$$master\_secret = PRF(pre\_master\_secret, "master\_secret", seed) \tag{10.3}$$

The `finish` message contains the so-called verify data which is calculated from all handshake messages and the master secret. The server proofs the ownership of its private key to the client in this message: it demonstrates that it was able to decrypt the `pre_master_secret` and use this to calculate the `master_secret`.

$$verify\_data = PRF(master\_secret, "serverfinished", Hash(handshake\_messages)) \tag{10.4}$$

10.4 Ciphersuite selection

The cipher suite determines which encryption and hashing algorithms are used for the authentication and secure channel set up. EAP-TLS supports multiple cipher suites and at the start of each handshake the used cipher suite is negotiated. This prototype, however, supports only one cipher suite.

As shown below, the EAP-TLS standard contains one mandatory and a number of recommended cipher suites in order to guarantee operability. The ciphersuite's name contains the key exchange, asymmetric, symmetric encryption and hash algorithm. For example, ciphersuite 2 means RSA for asymmetric, RC4 for symmetric (with 128 bit keysize) and SHA as hashing algorithm.

Mandatory ciphersuite

1. `TLS_RSA_WITH_3DES_EDE_CBC_SHA`:

Recommended ciphersuite

2. `TLS_RSA_WITH_RC4_128_SHA`
3. `TLS_RSA_WITH_AES_128_CBC_SHA`

However, this list only contains cipher suite with weak or broken algorithms. The MD4, MD5 hashing algorithm should not be used according to [97]. The use of the SHA hashing algorithm should be phased out according to [37]. Finally, some attacks are known against the DES [98] and RC4 [99] symmetric encryption algorithms.

The following cipher suite was selected: `TLS_RSA_WITH_NULL_SHA256`

- **Asymmetric encryption algorithm: RSA**. This was chosen because the TrustZone supports the storage of RSA keys. The key size is 2048 bits which complies with the NIST recommendations [100]. The public key is part of the certificates.
- **Key exchange algorithm: none**. EAP-TLS allows for the selection of a key exchange algorithm such as Diffie-Hellman. Since we already use X.509 certificates with public keys these can be used for the key exchange. Using a key exchange algorithm in this situation may provide a secure channel with forward secrecy (see Section 5.2). Forward secrecy, however, is not required since we only use the authentication and not the secure channel part of TLS.
- **Symmetric encryption: none**. The symmetric encryption algorithm is used for encrypting data in the secure channel. Since we are only interested in the authentication, using a symmetric encryption algorithm is not required.
- **Hash algorithm: SHA256**. TLS uses the a hash function as a MAC and for expanding keying material.

10.5 Certificates

EAP-TLS uses X.509 certificates for authentication. The different classes of certificates used in this prototype were explained in Section 9.2.2. The phone's certificate is generated and signed during the personalization which will be explained in Section 10.7. The root and the reader certificate are created manually. This can be done with the OpenSSL command line utility.

The attribute on the certificate that provides information about the identity is called the *subject*. The subject is a string formatted as shown in code 1. It contains the country (C), state (ST), Location (L), Organisation (O), organisation unit (OU) and common name (CN) of the subject. The common name (CN) is the most important as it must be unique for every subject in the security domain.

In this implementation, the common name is simply a random string. For example, for the phone a common name of "phone-01020305080d152237" can be used. This string can be used by the PACS for authorization purposes but this is not implemented. Similarly, the reader's common name begins with "reader-".

```
C=NL, ST=Overijssel, L=Enschede, O=UT, OU=Dacs, CN=reader-1234
```

Code 1: An example of a certificate subject string

10.6 Technologies

10.6.1 Mobile phone: Android & HCE

Part of ISO 7816-4 is implemented by Android. This is only the *"card organization and structure"* part which enables a multi-application environment as described in 2.3.4. Each application on the phone specifies a unique AID (application ID). The reader that wishes to communicate with an application on the phone first sends a select APDU which contains the AID in the body. The Android implementation will verify that an application with this AID is installed on the phone and responds to the reader accordingly. If all went well, all subsequent communication from reader will be passed on to the application with the selected AID.

An Android application can use the HCE functionality through the Android API. The AID is registered in a configuration file of the application named `hceservice.xml` which is shown in code 3. Officially, AIDs are issued by ISO and should be registered. For this application, a random AID was chosen that, to the best of my knowledge, is not used in an existing service.

Code 2 is an example of an HCE application in Android. A class `MyHostApduService` is defined. This class must contain a method `processCommandApdu()` which is called by the Android HCE implementation when data for this application is received. The received data is in the parameters which is an array of bytes. The name of this parameter (`apdu`) suggests that the data must be formatted as an APDU but this is not the case. Only the selection of the application (AID) uses the APDU format and all subsequent communication may have any (custom) format.

```
public class MyHostApduService extends HostApduService {
  @Override
  public byte[] processCommandApdu(byte[] apdu, Bundle extras) {
     // Process the data
     ...
     // Send something back to the reader
     return response;
  }
}
```

Code 2: A simple example of using HCE in Android.

```
<host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/servicedesc"
    android:requireDeviceUnlock="false" >

    <aid-group
        android:category="other"
        android:description="@string/aiddescription" >
        <aid-filter android:name="F0030403040506" />
    </aid-group>

</host-apdu-service>
```

Code 3: The `hceservice.xml` file which registers the AID.

10.6.2 Reader: PCSCLite and Ruby

PCSCLite is a middleware which enables a PC to communicate with an NFC device through a smart card reader that supports the PC/SC interface. A SCM SDI010 reader was used for this prototype. The application that controls the reader was written in Ruby. This application uses the smart card gem (extensions in Ruby are referred to as a "gem") which provides an API for using the PC/SC middleware. Code 4 shows how the reader can select the correct applet on the phone by sending a select C-APDU that contains the AID.

```ruby
# Load the smart card gem
require 'smartcard'

context = Smartcard::PCSC::Context.new
# The name of the reader
reader = "SDI010 USB Smart Card Reader [Vendor Interface] (21120621200424) 00 01"

#A select applet C-APDU with the AID that was set in the hceservice.xml
# [CLA, INS, P1, P2, Lc, aid, le ]
aid = [0xF0, 0x03, 0x04, 0x03, 0x04, 0x05, 0x06]
select_apdu = [0x00, 0xA4, 0x04, 0x00, aid.length] + aid + [0x00]


# Query the reader's status.
queries = Smartcard::PCSC::ReaderStateQueries.new 1
queries[0].reader_name = reader
queries[0].current_state = :unknown
context.wait_for_status_change queries, 1000
queries.ack_changes

# Prompt for card insertion unless that already happened.
unless queries[0].current_state.include? :present
  puts "Please insert smart-card card in reader #{reader}\n"

  # Wait until a card is detected
  until queries[0].current_state.include? :present do
    context.wait_for_status_change queries
    queries.ack_changes
  end
  puts "Card detected\n"

  # Initialize the card object and start a transaction
  card = Smartcard::PCSC::Card.new(context, reader, :direct, :raw)
  begin
    card.begin_transaction
    # Transmit the select_apdu to the phone (400 is the the timeout in ms)
    response = card.transmit(select_apdu.map {|byte| byte.chr}.join(''), 400)

    # Format the response as a hexadecimal presentation and print it to screen
    response_str = (0..response.length).map { |i| ' %02x' % response[i].to_i }.join('')
    puts "Answer: #{response.unpack('H*')}\n"

    card.end_transaction(:unpower)

  rescue Exception => e
    puts e
  ensure
    card.disconnect
    context.release
  end
end
```


Code 4: A simple Ruby script that sends a select applet C-APDU to the phone.

10.7 Personalization

The personalization procedure explained in Section 9.2.2 was implemented as a web application in the Ruby on Rails framework. An administrator can access the web interface and add 'NFC users' to the system as shown in Figure 10.8. The entered information includes personal information about the user (name, photo) and the validity date of the certificate. In this prototype, no authorization settings can be done.

Furthermore, two attributes are randomly generated for this particular user.

- **Token**: This is a 25 byte token that is sent to the user via e-mail
- **Confirmation code**: This is a 5 digit code that is transferred verbally from the administrator to the user.

When the administrator clicks the 'Create NFC user' button, the information is stored in the system. The registered user receives an e-mail with instructions as shown in Figure 10.9. This e-mail includes a link for downloading the mobile application and a second link that points to the web application and contains the token. The mobile application detects when this link is clicked and will offer the user to open it.

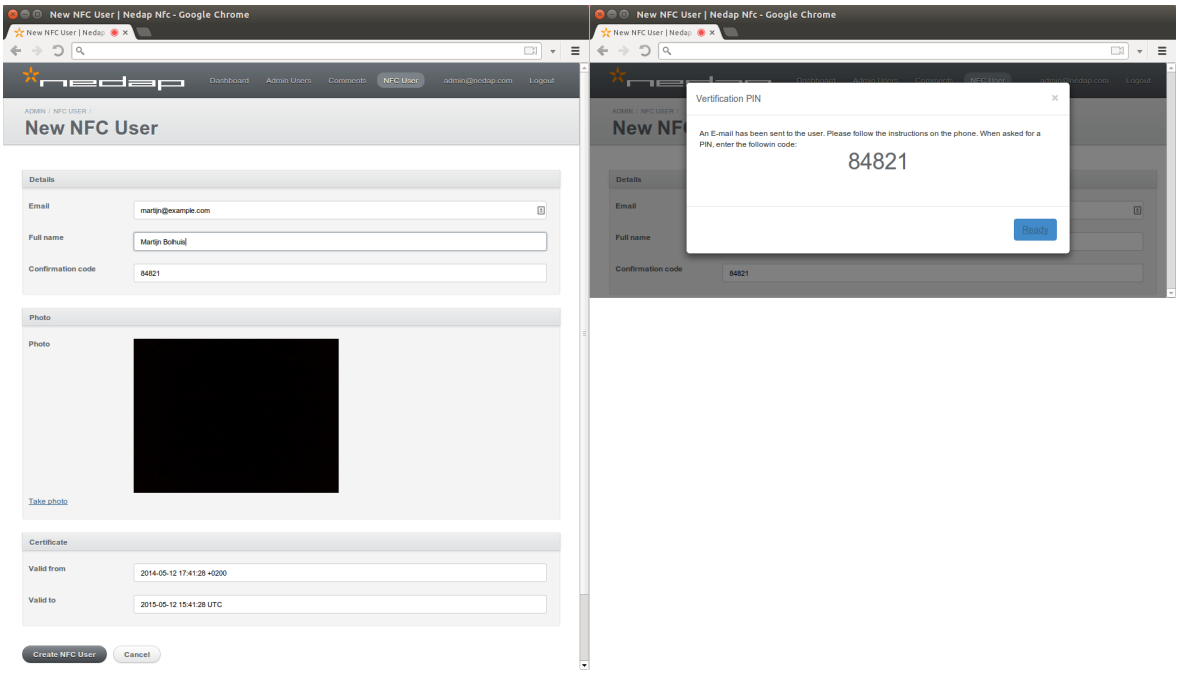

Figure 10.8: A screenshot of the web interface used for registering a new NFC user.

The mobile application will first open the link by using an HTTP GET request. This is done over a secure HTTPS connection. The mobile application comes with a pre-set

trusted certificate which it uses to authenticate this HTTPS connection. The server returns a number of attributes formatted as JSON. This includes the common name CN and validity period of the certificate. Furthermore, it contains the personal information about the user.

The personal attributes are presented on screen to the user such that these can be verified. Subsequently, the user can either *accept* or *reject* the credential. When the decision is accept, the phone asks for a confirmation code. This is given to the user by the administrator that added the user into the system.

Next, the phone prompts the TrustZone to generate a private key and certificate signing request (CSR). The parameters of the CSR are set in accordance with the parameters that were received previously from the server (CN and validity date). Then, the phone sends the CSR and confirmation code entered by the user to the server. This is done by doing an HTTP POST to the same URL. The server verifies that all parameters on the certificate are set correctly. Finally, the server will sign the CSR using the private key that corresponds to the root certificate. The phone's signed certificate and the root certificate are sent back to the phone. The phone stores both certificates after which the personalization is completed.

A potential threat on the server is that the credential is collected by an attacker instead of a legitimate user. For example, an attacker can guess possible tokens and confirmation code combinations. This is prevented by only allowing one registration attempt per token. This means that when a user enters a wrong confirmation code, it will need to be re-enrolled by the administrator.

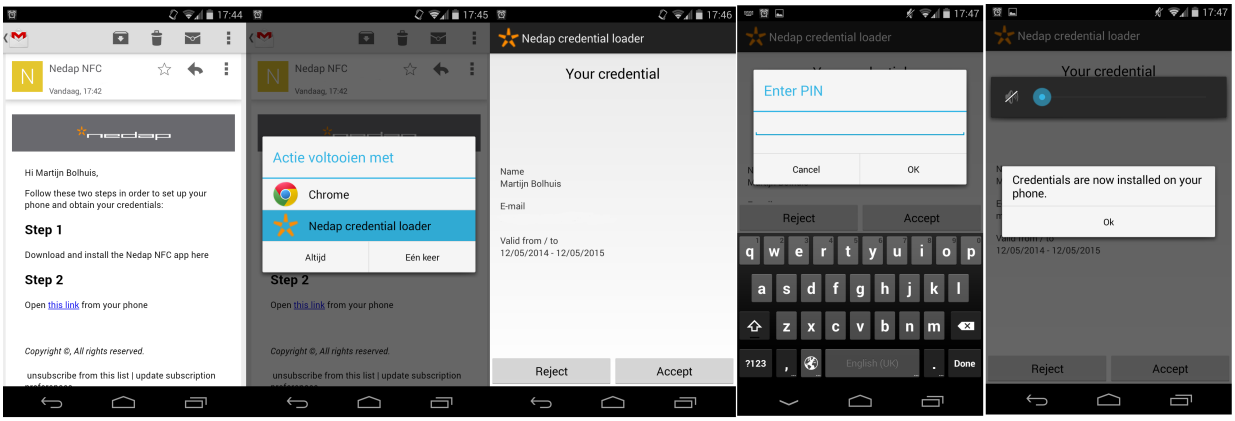

Figure 10.9: Screenshots from the mobile phone

Chapter 11

Evaluation

In order to evaluate the implementation, functional tests and performance tests were conducted. The functional tests in section Section 11.1 verify that the software developed in this work and APIs provided by Android work as expected. In Section 11.2 we reflect on certain usability related issues that were discovered during the implementation phase.

The performance tests reflect on the usability as defined in Section 1.2. It is undesirable that a user has to wait for a long time before the door opens. Therefore, it is important that the authentication latency is within reasonable time limits.

11.1 Functional testing

11.1.1 Faulty credentials

The following functional test were conducted that test the integrity of the software implemented in this work. These test use various forms of faulty credentials meaning that the test is successful if the authentication fails.

1. Expired certificate on the phone or on the reader
2. Certificate signed by an untrusted subject (untrusted C_{root}).
3. Authentication attempt with a private that does not match the public key on the certificate.
4. Client offers a cipher suite that is not supported by the server.
5. Use a certificate with a public key that does not correspond to the cipher suite. In this test I used a certificate and private key with DSA format instead of RSA.

11.1.2 Android APIs

The following tests were conducted that test integrity of the APIs provided by Android.

Extracting private key

A private key that is stored in the TrustZone is represented by the Java class `PrivateKey`. According to the documentation by Google, it is not possible to extract the private key from the TrustZone. However, the `PrivateKey` class has a number of methods that suggests otherwise. These are `toString()`, `getPrivateExponent()` and `getEncoded()`. A test program was written that output the returned value for each of these functions.

The private key consists of two parts: the *exponent* and *modulus*. The modulus is also contained in the public key and thus it does not have to be secret. The `toString()` method returns a string that represents the modulus of the private key. As shown in Figure 11.1, the `getPrivateExponent()` and `getEncoded()` return a `UnsupportedOperationException`. The conclusion is that indeed the Android API does not allow the secret private key exponent to be extracted.

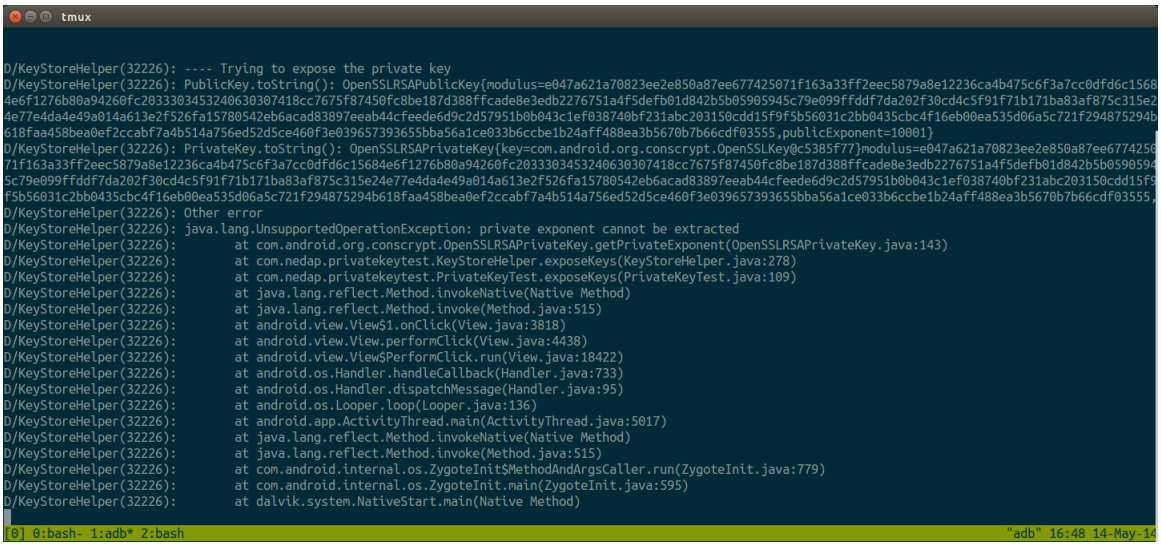

Figure 11.1: The log for an Android application that tests several methods that could potentially expose the private key

Third party application

Every Android application can store private keys in the key store. The keys are identified by their alias. Keys that are generated by application A under alias a, should not be accessible by another application B when using that same alias a. As shown in Figure 11.2, this was successfully verified by implementing a second test application that attempts to retrieve a private key with the alias that is used by the NFC authenticator application.

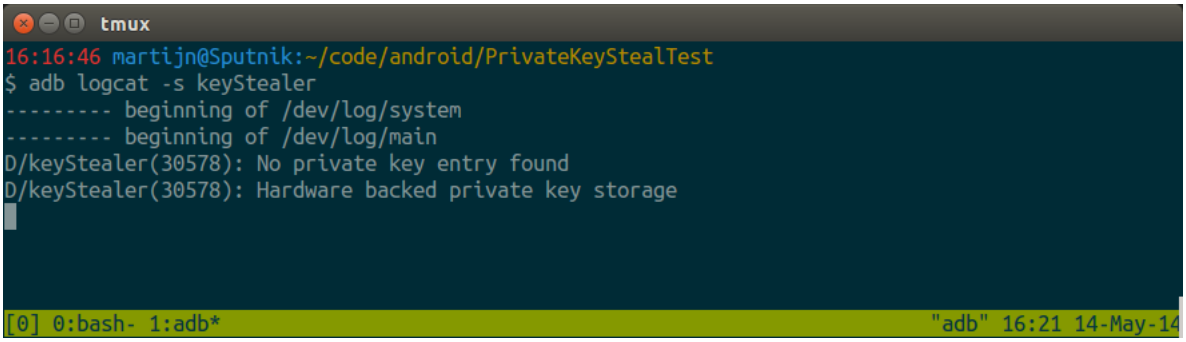

Figure 11.2: The log for an Android application that attempts to steal the private key from another application.

11.1.3 Privacy

In Section 4.4 *privacy* was identified as a threat. In Section 7.3.2 it was shown that the EAP-TLS protocol provides *Identity privacy* which offers protection against this threat by making sure that the certificate of the phone is sent after the reader authenticates to the phone. However, this is not sufficient to guarantee privacy.

The anti-collision mechanism in the NFC and ISO 14443 standards make use of a unique identifier (UID) which should be different for every card. The UID does not contain any identity information. As long as the PACS does not keep a database with the UID for each user it is not possible to correlate an UID to an identity.

However, potentially the UID can be used to track a user which affects the privacy. Therefore, some smart cards have an option to randomize the UID on each transaction. The documentation of the PLAID authentication protocol recommends service providers to enable this option [84] for privacy reasons.

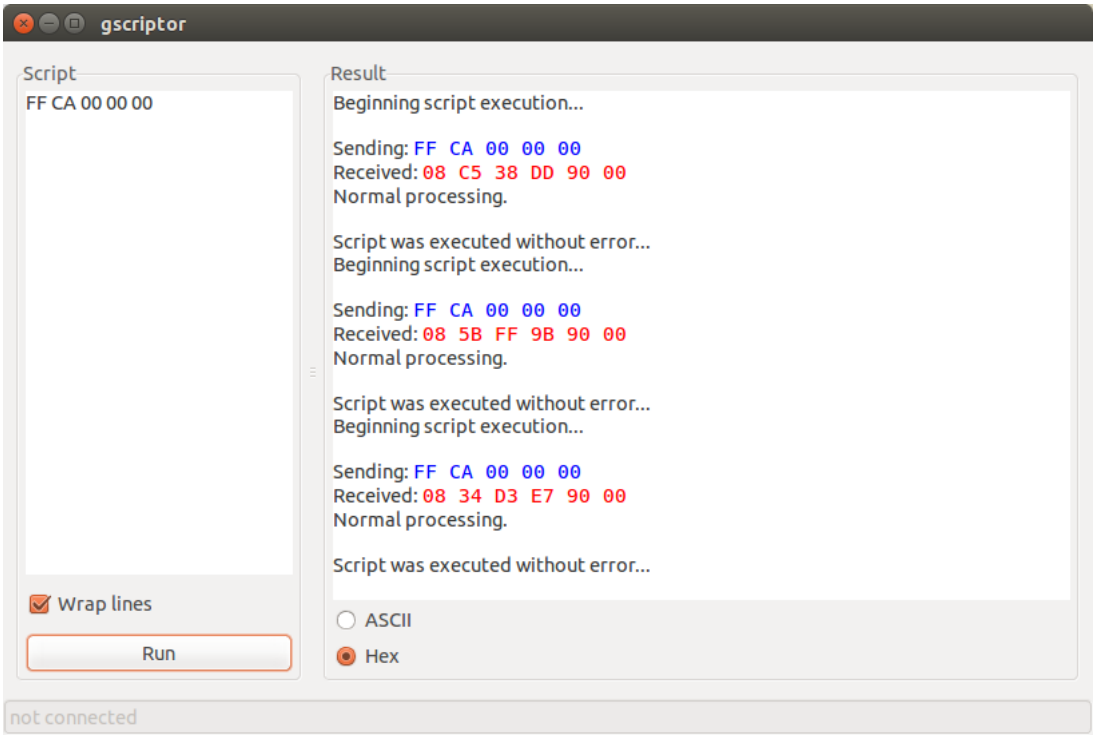

Figure 11.3: A test that obtains the UID value of the phone over multiple transactions

See Figure 11.3. In this test, the UID value of an Android phone was investigated in order to get more insight into the privacy guarantees. The UID can be requested from the device by sending a special APDU using a PC/SC version 2 reader (`CLA` = `0xFF` and `INS` = `0xCA`). Figure 11.3 shows the result of this test. The phone was held in front of the reader for three times and each time the UID was requested. The responses are printed in red. The first 4 bytes of this response represent the UID and the last two are the status words. The conclusion is that on the Android phone, the UID is randomized on each transaction which improves the privacy.

11.1.4 Personalization

In Section 4.3.5, user unawareness was identified as a vulnerability in mobile phones. As a result, a user may have malicious software installed on its phone. A potential threat is that a malicious application on the mobile phone pretends to be the authentication application. This is simple to accomplish because it does not require an attacker to circumvent restrictions imposed by the operating system.

The malicious application could register the same AID as the legitimate application. Potentially, the NFC communication is handled by this application. This can be seen as a DoS since it disables the user from using the phone as an access token.

For this reason it is useful to study the behaviour of the mobile phone when two applications (that use HCE) register the same AID. This was tested and it turns out that the mobile phone will show a window that allows the user to select which application should be used for the service when the phone is positioned in front of the reader. However, as shown in Figure 11.4 it is possible for an application to register the same name which makes it impossible for the user to differentiate between the legitimate and malicious application.

There is no way for an application to ensure that it is the default application for an AID. However, it is possible to detect whether it is the default application or not by using the `isDefaultServiceForAid()` function. A possibility would be to show the user a notification when the application is not the default. This can not prevent the problem but at least it would make the user aware of it.

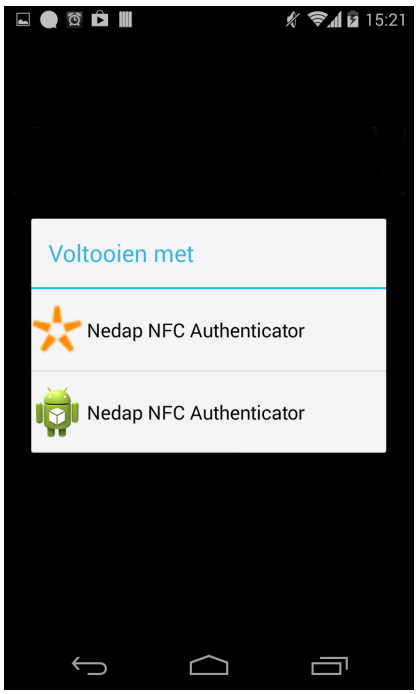

Figure 11.4: A pop-up that is shown to the user when two applications register the same AID

Another possibility is that a malicious application registers the domain in the link used for the personalization. Normally, the legitimate application will offer to open the link that is sent by e-mail as shown in Figure 10.9 on page 105. This is accomplished by registering a so-called *intent* on the domain name of the service. Similarly, as with registering double AIDs, a malicious application can also register the same intent.

A test was done to discover what happens in this case. As shown in Figure 11.5, both the legitimate and the malicious application show up in the list and there is no way for the user to differentiate between the two. Potentially, the malicious application can hijack the credential by imitating the legitimate application. It can ask an unaware user for the confirmation code and generate an own private key and CSR. The certificate and private key can then be forwarded to the attacker after the attacker as the credential instead of the user.

Similar as with the duplicate AID problem, it is possible to detect whether more than one application have registered an intent for a certain domain. This means that the user can be notified about this which should minimize the risk. However, more research is needed to improve the integrity of the personalization mechanism that establishes the initial security association with the mobile phone.

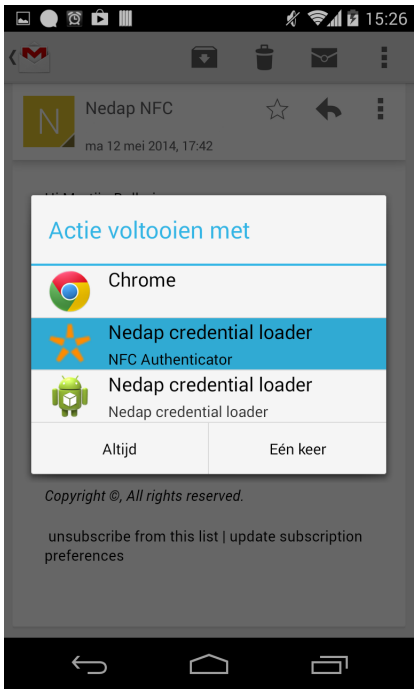

Figure 11.5: A pop-up that is shown to the user when two application register the same domain

11.2 Usability aspects

An important motivation for using NFC compared to traditional smart cards is usability (see Section 1.2). For this reason it is important to reflect on the usability of this solution.

A major disadvantage of using HCE is that it only works when the phone is on. It can not be used when the battery of the phone is empty which makes it unsuitable for applications where availability is critical. This behaviour was confirmed by switching the phone off and holding in front of a reader.

The NFC controller is completely switched off when the screen of the phone is off [101]. Consequently, a user first has to push the phone's power button before the NFC service can be used. From a security point-of-view this is good because it limits the possibility of a relay attack (see Section 5.5 point 4). It may affect the usability since it requires additional actions from the user. However, in my opinion the impact on the usability is limited and it is well worth the improvement in security.

Additionally, Android provides a per-service option that requires a user to unlock the screen before the NFC service can be used. This option can be enabled in an application by setting the `requireDeviceUnlock` attribute in the `hceservice.xml` file. Screen unlocking typically involves some type of authentication such as entering a PIN code. When enabled on an NFC service, it can be seen as multi-factor authentication which limits the possibility of a relay attack (Section 5.5 point 3). On the other hand, screen unlocking significantly increases the number of actions that a user has to do which is typically not desirable in a physical access control setting. Therefore, the screen unlocking option was set to off in this implementation.

A major problem with regards to usability was discovered during the implementation. It turns out that all stored private keys are removed from the device whenever the user changes the lock screen (for example, changing the PIN or disabling lock screen) even when screen locking was previously disabled. This means that a user has to obtain a new credential every time the lock screen screen is changed which is undesirable to say the least.

11.3 Performance

Another important aspect in physical access control is the transaction time, i.e. the time it takes to authenticate a user, which determines how long a user should wait before the door opens. It is desirable that this time is as low as possible.

The transaction time was investigated by doing measurements on the implementation using several hardware configurations. The set up of these measurements conforms to Figure 10.1 on page 91. The distance between the reader and phone could affect the data rate and thus the transaction time. In order to have a minimal and similar distance for all experiments, we placed the phone flat on the reader.

The following hardware was used. As a controller (the computer that runs the reader software) we used:

- **Laptop** - A Dell XPS 13 laptop with a 1.6 GHz Intel Core i7-3537U processor and 8GB of RAM.
- **Raspberry Pi** - A Raspberry Pi is a cheap single board computer. It has an ARM 11 processor with a clock speed of 700MHz and 512MB of RAM.

The following readers were used:

- **SCM SDI010** - This is a ISO 14443 contactless smart card reader that can be connected via USB. The driver for this reader is proprietary.
- **ACS ACR 122U** - This is a USB NFC reader (ISO 18092). Two drivers were available for this device: the proprietary driver and an open source driver. The open source driver is available from the libccid project which provides open source drivers for several smart card readers.

Unfortunately, the proprietary drivers are only available for a 'regular' 32 bit and 64 bit processor. The Raspberry Pi has an ARM processor which is incompatible with this architecture. Therefore, the ACR 122U could only be used on the Raspberry Pi using the open source driver. The SCM SDI010 device could not be used with the Raspberry Pi.

The following phone was used for all experiments:

- **Nexus 4** - This is a phone from Google manufactured by LG. It supports Host Card Emulation and has the TrustZone technology that is used for hardware-backed credential storage. It has a 1.5 GHz Qualcomm Snapdragon S4 Pro processor.

The transaction time was measured from the software developed for the reader. The reason for measuring on the reader and not on the phone is that the software on the phone phone is not activated until after the applet selection procedure (see Section 10.6.1). This means that if the transaction time was measured on the phone, the time of the applet selection procedure could not be included in the measurements. Therefore, measuring on the reader will yield a more accurate transaction time.

The developed reader software was extended such that it stores a time stamp when the phone is detected. Another time stamp is stored when the authentication handshake is successfully completed. The transaction time is calculated as the difference between the two time stamps. An extension for Ruby, the *timestamp* gem, was used to generate the time stamps. This extension provides nanoseconds precision. However, it is impossible that the measurement would also be accurate to this level. Furthermore, such a level of accuracy is not significant for the user. Therefore, all measured values were rounded to the millisecond level.

Table 11.1 shows the result of the experiments. The difference in transaction time between the two readers is remarkable. The smart card reader (SDI010) is about twice as fast compared to the NFC reader (ACR 122U).

Several attempts were made to explain this difference. For example, the SDI010 has an option for 848 kbps which is exactly twice as fast as the 424 kpbs that is supported by the ACR 122U. However, this options was disabled meaning that the two readers should have the same data rate. Furthermore, using a different driver for the ACR 122U does not seem to have a significant effect. The conclusion is that the type of reader may have a significant effect on the transaction time.

Reader	**Driver**	**Controller**	**Average**	**Standard deviation**
SDI010	Proprietary	Laptop	0.926s	0.068s
ACR 122U	Proprietary	Laptop	2.017s	0.050s
ACR 122U	Open source	Laptop	1.970s	0.053s
ACR 122U	Open source	Raspberry Pi	2.196s	0.105s

Table 11.1: Transaction time for various tested hardware/software configurations. For each configuration 30 tests were conducted.

Another result worth noticing is that the transaction time is significantly higher than we can explain. Our estimated transaction time was calculated as follows:

- **Transmission time** - This is the total time that is necessary to transfer data between the reader and the phone. In Table 11.2 the expected transmission time is calculated based on the various possible data rates in NFC and on the fact that our implementation transfers 2323 bytes in total. The actual transmission time may be higher because our calculations do not include any overhead that is added by the NFC protocol itself.
- **Processing time** - The software that was developed as a part of this thesis will process data. This includes buffering, parsing and generation of messages and cryptographic operations. The *total processing time* for the phone and the Raspberry Pi is shown in Table 11.3. This is the average total time of one transaction that the software developed by us is processing data.

  The actual processing time will be higher because we do not include the time that the underlying software, i.e. the PCSCLite middleware, the Android HCE implementation and software drivers are processing data. This would be possible as most of these software components are open source and can be modified. However, this would require considerable more effort.

The *expected transaction time* can be calculated by taking the sum of the *transmission time* and *processing times* (of the reader and the phone). This is shown in Table 11.4.

The expected transaction times are significantly lower than the times that were observed by us (see Table 11.1).

As indicated, both our expected transmission and processing time calculation are not a perfect representation of the actual times. However, still the gap between expected and measured time is large. We were unable to determine the exact reason for this.

Data rate	**Expected transmission time**
106 kbps	0.171s
212 kbps	0.086s
424 kbps	0.043s
848 kbps	0.021s

Table 11.2: The expected transmission time based on the possible data rates in NFC and on total number of octets that is transferred by our implementation (2323 bytes).

Device	**Average total processing time**	**Standard deviation**
Raspberry Pi	0.107s	0.012s
Nexus 4	0.258s	0.049s

Table 11.3: The measured total processing time on the Raspberry Pi and the Nexus 4 phone.

Data rate	**Transmission time**	**Processing (reader + phone)**	**Transaction time**
106 kbps	0.171s	0.107s + 0.258	0.536s
212 kbps	0.086s	0.107s + 0.258	0.451s
424 kbps	0.043s	0.107s + 0.258	0.408s
848 kbps	0.021s	0.107s + 0.258	0.386s

Table 11.4: The expected transaction time based on calculated transmission time and calculated processing times on a Nexus 4 phone and Raspberry Pi.

Part IV

Conclusions and Future Work

Chapter 12

Conclusion

In this thesis the use of a mobile phone with NFC capabilities in an on-line physical access control system was investigated. Before stating the final conclusions, the research questions from Section 1.5 are revisited.

- **Question 1**: What is NFC and how can it be used in mobile phones?

  In part 1, *"Background and related work"*, the NFC technology and the StolPAN / DIAD NFC framework were studied. The StolPAN / DIAD NFC framework enables the development and deployment of NFC-based services by providing a platform independent mechanism for controlling the NFC communication and secure storage. Unfortunately, such frameworks are currently not widely deployed. The reason for this is that the key stakeholders in the NFC ecosystem have not been able to come up with a generally accepted business model. This motivated our research to focus on potential alternatives.

- **Question 2**: Which security vulnerabilities exist in NFC-based mobile phone systems?

  Physical access control is a security related topic and therefore the security was assessed in part 2. This was done by identifying potential threats. The following threats were identified: insecure storage, relay attack, insecure communication, denial of service attack and privacy.

- **Question 3**: Which security mechanisms can be used to protect against the security vulnerabilities?

  The threats that were identied in Chapter 4 were addressed when possible. In Chapter 5, various well-known security mechanisms such as cryptography and authentication were discussed that deal with insecure communication and privacy. It is currently difficult to completely rule out the possibility of a relay attack. Nevertheless, we presented some guidelines that minimize the possibility of such an attack. Certain forms of denial of service attacks were identified but these were not directly addressed in this work. The reason for this is that little literature is

available about this subject in relation to NFC. Furthermore, the consequences are limited to the violation of the availability property.

- **Question 4**: How can credentials on the phone be stored securely?

  The insecure storage threat can be solved by using a secure storage technology. Various secure storage technologies were studied. Traditionally, a secure element is used for storing the credential. The advantage of this is that the secure element is directly connected to the NFC controller in the phone, making such a solution completely independent from the phone's application processor. Alternatives, for a secure element are storing the credential on the main storage. This, however, will require use input (such as a PIN code) in order to be secure which is not suitable for every access control situation. Recent phones by Google have the TrustZone technology that can be used to store RSA private keys.

- **Question 5**: Which authentication protocols can be used to securely verify the identity of a user over NFC?

  A number authentication protocols were studied that deal with privacy and well-known attacks that can be done on open / wireless mediums such as eavesdropping and MiTM. The main difference is that some protocols are specifically designed for use in devices with limited capabilities such as smart cards. Another observed difference is that some protocols are based on a pre-shared key (PSK) mechanism and others on a public key infrastructure (PKI).

- **Question 6**: How can a mobile phone / NFC-based security token be integrated with an existing PACS?

  NFC is compatible with many contactless smart card standards meaning that the existing reader infrastructure can be used. However, this requires that the phone is in card emulation mode.

  In part 3, the studied technologies were translated into design choices. First, a typical physical access control system was studied. Subsequently, two designs were presented.

  1. **SD Card** - An SD-card with NFC and secure storage capabilities can be purchased. Subsequently, an applet can be developed that is installed on the secure element. This solution provides additional security in the personalization procedure. Furthermore, it may provide better integration with existing authentication protocols and more flexibility. This is, however, more costly and not usable on all phones since some phone models lack an SD slot.

     Another disadvantage is that token integration is unlikely to happen with an SD card. Technically this is possible but it would require the development of a platform that shares the SD-card among service providers. Such a platform is currently developed for SIM cards but not widely deployed because

the involved parties can not work out a generally accepted business model. Therefore, it is likely that a similar platform for SD-cards will face similar issues.

However, using a mobile phone with NFC may offer additional advantages for the user compared to using a contactless smart card in specific type of systems. For example, an off-line physical access control system. As discussed in Section 1.3, these systems typically require that authorization data is on the card/phone. As authorization data tends to change more frequently than authentication data, this means that, normally, the user has to come to a POS in order to update this authorization data. When using a mobile phone, authorization updates can be done using the internet connection on the phone. Furthermore, some off-line locks generate data such as battery status. The internet connection on the phone could be used to transfer this information from the off-line lock to the on-line part of the system.

2. **Host Card Emulation** - A mobile application that uses Host Card Emulation, TrustZone hardware-backed credential storage and the EAP-TLS protocol can be used. This solution utilizes technologies that are available for free in modern Android-based smart phones allowing the service provider to get around current restrictions with secure elements while providing good security and acceptable transaction time. Furthermore, such a solution can easily be used for multiple services (token integration) which is an important advantage for the end-user. A disadvantage is that, although existing card readers can be used, the software on these readers require significant changes with this type of solution. MiFare smart cards are commonly used in physical access control. These cards use a proprietary authentication protocol. Unfortunately, this protocol is incompatible with the HCE implementation on Android. Furthermore, the TrustZone can currently only store RSA keys and no symmetric keys that are used by MiFare. Therefore, another authentication protocol was selected.

We chose to implement the HCE & TrustZone approach because of the advantages mentioned above and because of the fact that little research has been done into this. A mobile application and reader-side application were developed that communicate over NFC using the EAP-TLS authentication protocol.

- **Question 7:** How secure and usable is this solution?

Finally, the implementation was evaluated. This was done by doing functional testing on the created software. Furthermore, it was verified that the security related APIs provided by Google work as expected.

The transaction time can be within one second, depending on the reader hardware, which I think is acceptable for a typical access control situations. How-

ever, lower transaction times may be required in some scenarios. For example, in 'crowded' access control situations such as subways or stadiums lower transaction time can be desirable. Current smart card solutions provide transaction times as low as 0.3 seconds.

A weak spot is the initial personalization of the phone. It is possible that a seemingly legitimate application hijacks the personalization and steels the credential. However, the attack can be detected and the user can be warned. Furthermore, personalization is normally done only one time per phone which means that the risk is limited. The root cause is that there is no good way to establish an initial security association between the phone and the PACS. Secure elements currently suffer less from this problem because they are pre-personalized in a controlled environment. However, this problem will also arise in secure elements if they are going to be used more widely for multiple services, such as proposed by the StolPAN / DIAD NFC framework [31].

The main research question of this work was:

- How can NFC-equipped mobile phones be used for secure identity authentication in a physical access control system?

In this work we identified the security threats and techniques that solve these threats. Subsequently, we identified the two best solutions that use the studied techniques in order to enable the use of a mobile phone as a token in physical access control. The two solutions were compared allowing a service provider to make a choice based on its requirements. Finally, one of the solutions was implemented and evaluated successfully.

From the perspective of a *service provider*, there is a trade-off between costs, security and integration. Our HCE solution provides good security while the technologies used for this are available for free in the user's mobile phone. However, this approach may require significant changes on the reader which may be problematic or costly depending on the type of readers that are used. If this is the case, the SD card solution is better. Some SD-cards have an option to be pre-installed with MiFare which is commonly used in physical access control which makes the integration easy. Furthermore, SD cards offer better security in the personalization.

From the perspective of a *user*, the main advantage of NFC in a mobile phone is token integration. Therefore, it would be best if service providers choose an approach that enables this. Host card emulation is currently the best way to accomplish this. However, in some type of services or systems the use of NFC in a mobile phone may offer additional advantages to the user. For example, in an off-line physical access control system, the internet connection on the phone can be used to remotely send authorization data.

Chapter 13

Future Work

The personalization of a mobile phone needs additional research to improve the security. A quick solution could be that the operating system simply forbids multiple applications to use the same AID (in HCE) or to open an URL with the same domain. However, this does not prevent the problem if the user never installs the legitimate software.

A better solution could be that the personalization is done similarly as is done with secure elements using a TSM. A TSM is a trusted third party that facilitates the personalization. Applications in Android are already cryptographically signed. The trusted third party could maintain a database with application signatures that specifies which application can use which AID or domain. The operating system would then be responsible for enforcing the policies specified by this database.

More research is needed to investigate the security of the TrustZone technology. We found no direct evidence that the TrustZone is vulnerable to attacks such fault injection and side-channel attack. However, little research has been done into this. These attacks are described on secure elements but are difficult to perform. But the fact that secure elements have been thoroughly researched and the fact that they are widely used puts more confidence in this technology.

Bibliography

[1] S. Clark, "One in three mobile phones to come with NFC by 2017." http://www.nfcworld.com/2013/06/05/324448/one-in-three-mobile-phones-to-come-with-nfc-by-2017/, 5 June, 2013.

[2] ISO, "Ergonomics of human-system interaction," Tech. Rep. ISO 9241, International Organization for Standardization, Geneva, Switzerland, 2012.

[3] R. Boden, "Rabobank sets NFC mobile wallet launch date." http://www.nfcworld.com/2014/01/29/327674/rabobank-sets-nfc-mobile-wallet-launch-date/, 29 January, 2014.

[4] Rian Boden, "Video shows bankinter's hce payments service in action." http://www.nfcworld.com/2014/02/27/328124/video-shows-bankinters-hce-payments-service-action/, 27 February, 2014.

[5] Strategy Analytics, "Android captures record 81 percent share of global smartphone shipments in Q3 2013." http://www.engadget.com/2013/10/31/strategy-analytics-q3-2013-phone-share/, 31 October, 2013.

[6] R. Want, "Near field communication," *Pervasive Computing, IEEE*, vol. 10, no. 3, pp. 4–7, July-September.

[7] S. Ortiz, "Is near-field communication close to success?," *Computer*, vol. 39, no. 3, pp. 18–20, March.

[8] ISO/IEC, "Information technology - Open Systems Interconnection - Basic Reference Model," Norm ISO/IEC 7498-1, Geneva, Switzerland, 1994.

[9] H. Zimmermann, "OSI reference model–The ISO model of architecture for open systems interconnection," *Communications, IEEE Transactions on*, vol. C, no. 4, 1980.

[10] H. Funke, "Standards of smart cards in iso layer model." http://blog.protocolbench.org/tag/layer/, 2 November, 2013.

[11] Aditya Gupta, "Owning smart phones using NFC." http://www.chmag.in/article/jan2013/stand-close-me-youre-pwned-owning-smart-phones-using-nfc, January, 2013.

[12] ISO, "Identification cards - Contactless integrated circuit(s) cards - Proximity cards," Tech. Rep. ISO 14443, International Organization for Standardization, Geneva, Switzerland, 2000.

[13] JIS, "Specification of implementation for integrated circuit(s) cards - Part 4: High Speed proximity cards," Tech. Rep. JIS 6319-4, Japanese Industrial Standard, Tokyo, Japan, 2005.

[14] ISO, "Near field communication – interface and protocol (nfcip-1)," Tech. Rep. ISO 18092, International Organization for Standardization, Geneva, Switzerland, 2013.

[15] ECMA, "Near field communication – interface and protocol (nfcip-1)," Tech. Rep. ECMA 340, European Computer Manufacturers Association (ECMA), Geneva, Switzerland, 2004.

[16] ISO, "Near Field Communication – Interface and Protocol (NFCIP-2)," Tech. Rep. ISO 21481, International Organization for Standardization, Geneva, Switzerland, 2013.

[17] ECMA, "Near field communication – interface and protocol (nfcip-2)," Tech. Rep. ECMA 352, European Computer Manufacturers Association (ECMA), Geneva, Switzerland, 2010.

[18] ISO, "Identification cards – contactless integrated circuit cards – vicinity cards," Tech. Rep. ISO 15693, International Organization for Standardization, Geneva, Switzerland, 2013.

[19] NFC Forum, "About the nfc forum." http://www.nfc-forum.org/aboutus/.

[20] M. Roland, "Software Card Emulation in NFC-enabled Mobile Phones : Great Advantage or Security Nightmare ?," 2012.

[21] ETSI, "Single wire protocol," Tech. Rep. ETSI TS 102 613, European Telecommunications Standards Institute, Sophia Antipolis, France, 2009.

[22] ECMA, "Near Field Communication Wired Interface (NFC-WI)," Tech. Rep. ECMA 373, European Computer Manufacturers Association (ECMA), Geneva, Switzerland, 2012.

[23] B. O. Vedat Coskun, Kerem Ok, *Near Field Communication (NFC): From Theory to Practice*. Heidelberg; New York: John Wiley & Sons, 2011.

[24] ETSI, "Host controller interface (HCI)," Tech. Rep. ETSI TS 102 622, European Telecommunications Standards Institute, Sophia Antipolis, France, 2008.

[25] NFC Forum, "NFC Controller Interface (NCI) Specification," Norm TS-NCI-1.0, Wakefield, USA, 2012.

[26] B. Benyó, B. Sódor, G. Fördos, L. Kovács, and A. Vilmos, "The stolpan view of the nfc ecosystem," in *Wireless Telecommunications Symposium, 2009. WTS 2009*, pp. 1–5, April.

[27] B. Benyó, B. Sódor, G. Fördos, L. Kovács, and A. Vilmos, "A novel virtual machine based approach for hosting nfc services on mobile devices," in *Wireless and Mobile Networking Conference (WMNC), 2010 Third Joint IFIP*, pp. 1–7, 2010.

[28] B. Benyó, B. Sódor, G. Fördos, L. Kovács, and A. Vilmos, "A generalized approach for nfc application development," in *Near Field Communication (NFC), 2010 Second International Workshop on*, pp. 45–50, April.

[29] B. Benyó, B. Sódor, G. Fördos, L. Kovács, and A. Vilmos, "Security issues of service installation on a multi application NFC environment," in *2010 14th International Conference on Intelligent Engineering Systems (INES)*, pp. 145 –149, May 2010.

[30] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on android," in *Proceedings of the 13th international conference on Information security*, ISC'10, (Berlin, Heidelberg), pp. 346 – 360, Springer-Verlag, 2011.

[31] P. Pourghomi and G. Ghinea, "Challenges of managing secure elements within the nfc ecosystem," in *Internet Technology And Secured Transactions, 2012 International Conference for*, pp. 720–725, Dec 2012.

[32] Google, "Android 4.3 APIs." `https://developer.android.com/about/versions/android-4.3.html`, 24 March, 2014.

[33] Rian Boden, "Visa: Hce will drive innovation in payments." `http://www.nfcworld.com/2014/02/24/328013/visa-hce-will-drive-innovation-payments/`, 24 February, 2014.

[34] Sarah Clark, "Nxp updates nfc controller for hce." `http://www.nfcworld.com/2014/02/26/328088/nxp-updates-nfc-controller-hce/`, 26 February, 2014.

[35] Sarah Clark, "Banco sabadell picks carta for hce payments trial." `http://www.nfcworld.com/2014/03/04/328162/banco-sabadell-picks-carta-hce-payments-trial/`, 4 March, 2014.

[36] S. S. Rainer Baumann, Stephane Cavin, "Voice Over IP - Security and SPIT Swiss Army , FU Br 41 , KryptDet Report," *University of Berne*, pp. 1–34, 2006.

[37] W. Stallings, *Network Security Essentials: Applications and Standards.* Upper Saddle River, NJ, USA: Prentice Hall Press, 4th ed., 2010.

[38] E. Haselsteiner and K. Breitfuß, "Security in near field communication (nfc) strengths and weaknesses," *Computers, IEEE Transactions on.*

[39] Radio-electronics.com, "Nfc modulation & rf signal," May 23 April, 2013.

[40] M. Roland, J. Langer, and J. Scharinger, "Applying relay attacks to Google Wallet," *2013 5th International Workshop on Near Field Communication (NFC)*, pp. 1–6, Feb. 2013.

[41] W. Jeon, Y. L. J. Kim, and D.Won, "A practical analysis of smartphone security," *Springer Berlin Heidelberg*, pp. 311–320, 2011.

[42] M. Roland, J. Langer, and J. Scharinger, "Practical attack scenarios on secure element-enabled mobile devices," in *Near Field Communication (NFC), 2012 4th International Workshop on*, pp. 19–24, March 2012.

[43] R. Anderson, *Security engineering: a guide to building dependable distributed systems.* Indianapolis: Wiley, 2008.

[44] C. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal, Vol 28, pp. 656-715*, Oktober 1949.

[45] B. Aboba and J.Wood, "Authentication, Authorization and Accounting (AAA) Transport Profile," Tech. Rep. RFC 3539, June 2003.

[46] C. Paar, *Understanding cryptography: a textbook for students and practitioners.* Heidelberg; New York: Springer, 2010.

[47] R. Housley, W. Ford, W. Polk, and D. Solo, "Internet x.509 public key infrastructure certificate and crl profile." http://www.ietf.org/rfc/rfc2459.txt.

[48] ECMA, "NFC-SEC: NFCIP-1 Security Services and Protocol," Tech. Rep. ECMA 385, European Computer Manufacturers Association (ECMA), Geneva, Switzerland, 2010.

[49] ECMA, "NFC-SEC-01: NFC-SEC Cryptography Standard using ECDH and AES)," Tech. Rep. ECMA 386, European Computer Manufacturers Association (ECMA), Geneva, Switzerland, 2010.

[50] Elaine Barker, Don Johnson, and Miles Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography," Tech. Rep. NIST SP 800-56A, National Institute of standards and technology, Gaithersburg, Maryland, United States, 2007.

[51] NIST, "Advanced Encryption Standard (AES)," Tech. Rep. FIPS PUB 197, National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, United States, 2001.

[52] M. Reveilhac and M. Pasquet, "Promising secure element alternatives for NFC technology," Feb. 2009.

[53] N. Elenkov, "Accessing the embedded secure element in Android 4.x." http://nelenkov.blogspot.nl/2012/08/accessing-embedded-secure-element-in.html, 13 May, 2013.

[54] G. Madlmayr, "Management of Multiple Secure Elements in NFC-Devices NFC - Near Field Communication," *Royal Holloway University of London*, 2008.

[55] EMVCo, "Mobile Contactless Payment Technical Issues and Position Paper," no. October, 2007.

[56] N. Times, "Most NFC Phones to Pack Embedded Chips." http://nfctimes.com/news/most-nfc-phones-pack-embedded-chips, 29 April, 2011.

[57] Mastercard, "Secure elements." https://mobile.mastercard.com/Partner/MobilePayPass/SecureElements, 16 May, 2013.

[58] Device Fidelity, "microNFC." http://www.devifi.com/microNFC.html, 3 April, 2013.

[59] P. Pourghomi and G. Ghinea, "Managing nfc payment applications through cloud computing," in *Internet Technology And Secured Transactions, 2012 International Conferece For*, pp. 772–777, 2012.

[60] NFC World, "Fujitsu puts NFC into cloud-based data transfer service." http://www.nfcworld.com/2011/07/22/38759/fujitsu-puts-nfc-into-cloud-based-data-transfer-service/, 4 April, 2013.

[61] NFC World, "SimplyTapp proposes secure elements in the cloud - NFC World." http://www.nfcworld.com/2012/09/19/317966/simplytapp-proposes-secure-elements-in-the-cloud/, 28 Februari, 2013.

[62] N. Elenkov, "Credential storage enhancements in Android 4.3." http://nelenkov.blogspot.nl/2013/08/credential-storage-enhancements-android-43.html, April, 2014.

[63] Google, "Android source code: keymaster qcom ." https://android.googlesource.com/platform/hardware/qcom/keymaster/+/jb-mr1.1-dev/keymaster_qcom.cpp, April, 2014.

[64] Microsoft, "Key BLOBs." http://msdn.microsoft.com/en-us/library/ms884374.aspx, April, 2014.

[65] J. van Woudenberg, M. Witteman, and F. Menarini, "Practical optical fault injection on secure microcontrollers," in *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011 Workshop on*, pp. 91–99, Sept 2011.

[66] "Arm security technology building a secure system using trustzone technology," May 2014.

[67] J. R. Vollbrecht, B. Aboba, L. J. Blunk, H. Levkowetz, and J. Carlson, "Extensible authentication protocol (EAP)." `http://tools.ietf.org/html/rfc3748`.

[68] B. Aboba and P. Eronen, "Extensible authentication protocol (EAP) key management framework." `http://tools.ietf.org/html/rfc5247`.

[69] P. Urien, "EAP Support in Smartcard," Tech. Rep. Draft 26, 9 January, 2014.

[70] J. Beusink, "Secure access control to personal sensor information in federations of personal networks," 2012.

[71] S. Bradner, "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs," Tech. Rep. RFC 4017, March 1997.

[72] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," Tech. Rep. RFC 2119, March 1997.

[73] F. Bersani and H. Tschofenig, "The EAP-PSK protocol: A pre-shared key extensible authentication protocol (EAP) method." http://tools.ietf.org/html/rfc4764.

[74] R. H. D. Simon, B. Aboba, "The eap-tls authentication protocol." `http://www.ietf.org/rfc/rfc5216.txt`.

[75] M. Badra and P. Urien, "Adding identity protection to eap-tls smartcards," in *Wireless Communications and Networking Conference, 2007.WCNC 2007. IEEE*, pp. 2951–2956, March 2007.

[76] P. F. S. Blake-Wilson, "Extensible authentication protocol tunneled transport layer version 0 (eap-ttlsv0)." `http://tools.ietf.org/html/rfc5281`.

[77] P. E. Y. Sheffer, H. Tschofenig, "An extension for eap-only authentication in ikev2." `http://tools.ietf.org/html/rfc5998`.

[78] P. Eronen, C. Kaufman, Y. Nir, and P. Hoffman, "Internet key exchange protocol version 2 (IKEv2)." `http://tools.ietf.org/html/rfc5996`.

[79] Y. Sheffer and H. Tschofenig, "Internet key exchange protocol version 2 (ikev2) session resumption." `http://tools.ietf.org/html/rfc5723`.

[80] V. Devarapalli and K. Weniger, "Redirect mechanism for the internet key exchange protocol version 2 (ikev2)." `http://tools.ietf.org/html/rfc5685`.

[81] Microsoft, "Protected Extensible Authentication Protocol (PEAP)," 2013.

[82] N. Cam-Winget, D. McGrew, H. Zhou, and J. Salowey, "The flexible authentication via secure tunneling extensible authentication protocol method (EAP-FAST)." `http://tools.ietf.org/html/rfc4851`.

[83] InteropNet Labs, "What is EAP-FAST ?," no. April 2004, p. 2005, 2005.

[84] Centrelink, "Logical smart card application specification plaid version 8 final," tech. rep., Centrelink, Centrelink, Australia, 2009.

[85] L. Y. R. Koh Ho Kiat, *Analysis Of OPACITY And PLAID Protocols For Contactless Smart Cards*. PhD thesis, Naval Postgraduate School, September, 2012.

[86] ISO, "Identification cards – integrated circuit card programming interfaces – part 3: Application interface," Tech. Rep. ISO 24727-3, International Organization for Standardization, Geneva, Switzerland, 2008.

[87] ActivIdentity, "Open protocol for access control identification and ticketing with privacy specifications," tech. rep., ActivIdentity, Silicon Valley, California, 2011.

[88] M. Fischlin and C. Onete, "A Cryptographic Analysis of OPACITY," *Darmstadt University of Technology, Germany*, pp. 1–46.

[89] ISO, "Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange," Norm ISO/IEC 7816-4, Geneva, Switzerland, 2013.

[90] G. Bernabé, "Smart card TLS source code." `https://github.com/gilb/smart_card_TLS`.

[91] G. Bernabé, *TLS embedded in smart card*. PhD thesis, University of Plymouth, 2012.

[92] I. Daradimos, K. Papadopoulos, I. Stavrakas, M. Kaitsa, T. Kontogiannis, and D. Triantis, "A physical access control system that utilizes existing networking and computer infrastructure," in *EUROCON, 2007. The International Conference on 34;Computer as a Tool 34;*, pp. 501–504, 2007.

[93] F. Civico and A. Peinado, "Low complexity smart card-based physical access control system over ip networks," in *Electrotechnical Conference, 2004. MELECON 2004. Proceedings of the 12th IEEE Mediterranean*, vol. 2, pp. 799–802 Vol.2, 2004.

[94] A. developer documentation, "Optimizing downloads for efficient network access." `http://developer.android.com/training/efficient-downloads/efficient-network-access.html`, 2014.

[95] S. T. S. Farrell, R. Housley, "An internet attribute certificate profile for authorization." `http://tools.ietf.org/html/rfc5755`.

[96] E. R. T. Dierks, "The Transport Layer Security (TLS) Protocol version 1.2," Tech. Rep. RFC 5246, August 2008.

[97] J. Randall and M. Szydlo, "Collisions for sha0, md5, haval, md4, and ripemd, but sha1 still secure," 2004.

[98] "Data encryption standard," May 2014. Page Version ID: 602802969.

[99] I. Mantin and A. Shamir, "A practical attack on broadcast rc4," in *Fast Software Encryption* (M. Matsui, ed.), vol. 2355 of *Lecture Notes in Computer Science*, pp. 152–164, Springer Berlin Heidelberg, 2002.

[100] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid, "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," Tech. Rep. NIST SP 800-131A, National Institute of standards and technology, Gaithersburg, Maryland, United States, 2011.

[101] Android Developers Guide, "Host-based card emulation." `http://developer.android.com/guide/topics/connectivity/nfc/hce.html`, 20 May, 2014.