Radboud University Nijmegen
Computing Science
Digital Security Faculty

Master of Science Thesis

Dutch EMV-cards and Internet Banking

by

Michael Schouwenaar

Supervisors: Erik Poll & Wojciech Mostowski

Nijmegen, 2010

Thesis number: 635

Dedicated to my parents, for giving me all the opportunities and space I needed to get where I am.

Contents

Acknowledgements

I would like to thank Erik Poll for supervising me during this research project, as well as giving me the ideas, directions and feedback I needed to successfully complete it.

I would also like to thank Wojciech Mostowski for his thoughts, input and feedback both during our meetings and outside of those, as well as providing most of the hardware and help I needed with getting it to work. In terms of hardware I would also like to thank Engelbert Hubbers for providing me with the passive traffic sniffer and helping me confirm it did not work properly with the devices I used during this research project.

My final thanks go out to Pieter Ceelen and Stan Hegt of KPMG, who have done related research of their own and helped me with both their findings and their ideas.

Chapter 1

Introduction

This chapter summarises the initial background and motivation that led to the research being performed for this master thesis. The goals of the research project are listed and explained, and the contributions of this research are summarised. An overview is also given for the potential use cases that are relevant for this research, such as how a customer of a bank might use an Internet banking application.

1.1 Background and motivation

Magnetic stripes have long been used by banks on bank cards to store and deliver information about the card owners so it can be easily used in transactions. However, due to the nature of the magnetic stripes, criminals could copy cards by simply reading the information from those magnetic stripes at modified terminals, also called *skimming*. The costs for this type of fraud in the UK alone are non-negligible [3, 12], and has caused countries in Europe (as well as a few other countries) to move from bank cards containing magnetic stripes to bank cards with integrated tamper-resistant chips. These cards are also known as *smart cards*.

Banks in the Netherlands have the intention to do that as well, and have opted for implementations that follow the EMV standard. Many European countries including all countries that are a member of the EU have done the same, and this decision should help reduce the costs associated with having to support multiple transaction systems being used in different countries.

The EMV standard is a standard that was initially developed by Europay, Mastercard and Visa (hence EMV), and has been released to the general public[*]

[*]Available at `http://www.emvco.com/`

[7, 8, 9, 10]. Several weaknesses [12, 4] have been discovered by researchers at the University of Cambridge, and even though the move in the UK from magnetic stripes to EMV chips (also called Chip and PIN) initially seemed to help with the reduction of bank card fraud, it seems to be on the rise again [12] due a shift to other types of fraud.

The EMV standard is known to be extremely lengthy, and it offers a wide variety of different options that can be chosen to be implemented or not. This means that results that have been found at the University of Cambridge might not apply to the banks in the Netherlands, as it is likely they have chosen to implement it in a different way.

Apart from the EMV standard there is another standard that deals specifically with EMV in the context of Internet banking called the Chip Authentication Program (CAP). Even though this standard is not publicly available, much of it has been reverse engineered [5].

Internet banking

Apart from this shift of magnetic stripes towards tamper-resistant chips, many banks also offer its client the possibility of performing transactions on the Internet through their computer. The PIN-protected chip offers many more possibilities for banks to properly authenticate the client, for example by having the chip perform certain mathematical operations to demonstrate the authenticity of the card and the intention of the user to perform a particular transaction. Banks often use special devices for this, that issue challenges based on the transaction that needs to be performed.

Several Dutch banks already provide these devices, and because there are not many limits to what a customer can do using Internet banking (compared to, for example, the daily withdrawal limit of a cash machine) it will be an interesting target for criminals.

Liability

A serious consequence of the move from magnetic strips to tamper-resistant chips with PIN protection is that banks can shift a big part of the liability from themselves to the customer. It will be much easier to claim for a bank that a customer was not careful with her PIN when fraud is suspected by a customer. This liability shift makes it extremely important that the implementations of the EMV chips are correct, since it is much harder for an individual to prove a transaction was fraudulent than it is for a bank.

Use cases

Two important use cases can be identified for this research project: paying for an item in a store using a smart card, and using a smart card log in to an Internet banking application and confirming that certain transactions need to be performed. Note that these use cases assume that a bank card is used, and in the case of Internet banking that a challenge/response device is used that allows for the generation of signatures by the bank card.

1) Paying in a store

A customer wants to be able to go to a store and purchase an item by inserting his or her bank card in a terminal, seeing which transaction is about to be performed and confirm this transaction by providing a PIN which is verified by the card. The customer does *not* want someone who stole the card to be able to make purchases with it, and also does not want a terminal to provide certain transaction information while in reality another transaction is confirmed.

A store owner wants to provide its customer with the ability to pay with his or her bank card. The store owner wants to be sure that the customer is the owner of the card by having him or her verify it by entering the PIN. The store owner does not want it to be possible that a fake card can be used to perform transactions, and also wants to make sure the correct transaction data is confirmed.

What usually happens in practice is that the first step taken by the terminal is to determine that the card is authentic. This is done by the card through certificates.

After it has been established that the card is authentic, the customer has to prove that he or she is the owner of the card. This is usually done by providing the PIN, which is either verified by the card or sent to the bank in an encrypted form for verification, but can also be done through other means such as by providing a signature. This is negotiated by the terminal and the card and is bound by several restrictions, such as only allowing *no verification required* for transactions up to a certain (low) amount.

When finalising a payment the card signs the transaction data, which can then be verified for its authenticity. By having the card do this, the owner of the card authorises the transaction to occur. The resulting signature can be used by the bank of the terminal to prove a certain transaction needs to be completed.

2) Internet banking

A customer of a bank wants to be able to perform financial transactions from his own home by using the Internet. The customer wants to be sure he or she is the only one able to both access the Internet banking website and authorise transactions.

The bank wants to provide a customer the possibility of performing financial transactions from home. The bank always wants to make sure that the customer really wishes to perform a certain transaction (it wants to capture intent), as well as make sure the customer is the one performing the transaction and not another (unauthorised) individual.

The steps that are needed here are slightly less complicated. A big difference with paying in a store is that there is one less party involved, namely the bank of the merchant. This relative simplicity is shown in the fact that the card *initially* does not need to prove that card is authentic. Instead this is done at the end of the transaction, by verifying whether the expected response is correct. Since the bank knows the keys that are used by the card, it can determine whether or not the data on the card is authentic.

What *is* still important (and possibly even more important) is that the person using the Internet banking application is actually the owner of the card. Because there are nearly no restrictions on what can be done on most Internet banking applications, PIN verification should essential when a bank card is used as a form of authentication, as there is often no limit to kinds of transactions that can be performed.

After the card has authorised a ”transaction”, the user provides (a part) of the signature to the Internet banking website, which can then be used to check whether or not it corresponds to what the bank has generated. This (partial) signature is shown on the challenge/response device. If it is correct, it will be used as a confirmation of the customer’s intent to perform a transaction.

1.2 Research goals

The goal of this research is to get an insight into the implementation of electronic chips on bank cards as they are being used by Dutch banks and their customers. The main focus point will be on how the card interacts with the devices that are used during Internet banking, and how this corresponds to the existing EMV electronic payment specifications. These results from the Dutch situation will be compared to some of the results as they have been published in the UK [1, 4, 12, 5], of which [5] is especially important as it relates to the devices used for Internet banking.

In order to reach these goals, the following main research question has been defined:

> *How do Dutch banks use the EMV specifications in their bank cards, in particular when using Internet banking, and what are the consequences and limitations on the security of the system as a whole?*

In order to be able to successfully answer this question, the following sub research questions have been defined:

Sub-question 1. *How are the EMV specifications used to design an implementation of bank cards usable for Internet banking in terms of protocols and capabilities?*

Sub-question 2. *Which options did the Dutch banks choose to implement of the EMV standard in terms of protocols and data authentication (SDA, DDA or even CDA) when using Internet banking?*

Sub-question 3. *Do the Dutch EMV bank cards suffer from the same problems that have been found [4, 5, 12] in the U.K.?*

At the third sub-question the most important paper to look at is [5], as it relates directly to the devices used when using Internet banking.

1.3 Contributions

The contributions made by this research project are mainly in the area of the security of the Internet banking devices (challenge/response devices as well as bank cards) used by a customer of a Dutch bank. It looks at what is needed for the user to use those devices, as well as how they work and if any issues that might affect the security of those devices exist. To do this, several applications have been developed that can help with both the process of reverse engineering, as well as interpret the information exposed by these devices.

Because comparable research has already been done in the U.K., the findings of those research projects have been compared to the results of this research.

1.4 Organisation of this thesis

This document is mostly structured in a manner that helps answer the three sub-research questions in the order of which they are listed. Chapter 2 looks at the EMV specifications in detail, and how this should lead to a model that can be used by banks to develop EMV applications applicable to this research projects.

Chapter 3 looks at how the Dutch banks (and in particular the ABN-AMRO) have actually implemented their Internet banking devices, which often use the EMV specifications. This allows a comparison to be made with the previous chapter that looked at the EMV specifications themselves.

The next Chapter, Chapter 4, looks at the weaknesses that have been reported in the U.K. [5, 12, 4] and reports on whether or not those are applicable to the situations as it was found in the Netherlands. It also looks at new vulnerabilities, and lists those as well.

Of course not everything that has been discovered during this research project could be investigated fully, which is why Chapter 5 lists the possibilities for future work based on this research.

Chapter 6 is the final chapter and deals with the conclusions that can be made based on this research. After this several Appendices are added, as well as a list of tables, figures and abbreviations.

Chapter 2

The EMV specifications

In this Chapter a thorough look is given at the EMV specifications. This is done by giving an overview of the content of the different books, but also by giving an indication of the target audiences per section, how information from these specifications were interpreted during this research project and by listing alternative literature that gives insight into the EMV specifications.

The overview of the different books of the EMV specifications are given in Section 2.1.1 to Section 2.1.4, while Section 2.2 lists the different aspects of the EMV specifications that were relevant to this research project.

Because of the size and complexity of the EMV specifications it is recommended that readers who are interested in learning more about EMV look at the literature in Section 2.3 first, as that is academic literature *about* the EMV specifications. It can likely give a much better impression of the different aspects of the EMV specifications.

2.1 EMV unravelled

The EMV specifications is a standard for electronic tamper-resistant chip cards, and was developed by Europay, MasterCard and Visa before being released as an open standard. It consists of four different books [7, 8, 9, 10] and contains a lot of information on many aspects that are necessary to create electronic payment systems using an electronic chip.

Even though the size and complexity of the EMV specifications itself is already quite large (four books with a total of over 750 pages), it builds upon many different standards, such as the ISO 7816 standard that defines physical characteristics of an Integrated Circuit Card(ICC). It is even stated in section 1.3 of every book of the EMV specifications that *"it is based on the ISO 7816 series of* *standards and should be read in conjunction with those standards"*. This means that apart from the bulky and complex EMV specifications, those referenced standards also apply, which make it even more complex to analyse and understand. To top it off, the EMV specifications leave a lot of implementation details and options open to the implementer, which makes the analysis even harder.

This chapter will attempt to give a clear overview of the contents of the EMV standard. Apart from listing the contents of the chapters of each book and different sections of those chapters, an indication will be given on the intended audience of the different books and chapters. For example, manufacturers of the physical smart cards themselves will be more interested in book 1, while programmers interested in creating payment applications on those smart cards will generally use book 3 the most often. Each book has a general audience, while an additional audience might apply to individual chapters. The general audience is stated at the introduction of a chapter, while additional audiences (if applicable) will be mentioned at the chapters themselves.

The audiences are by no means to be used as a definitive lists, merely as an indication of who might benefit the most from the information contained within a specific book or chapter. These audiences have been defined as part of this research project, and have not been taken from the standards themselves.

Note that all four books start in the same manner; the first four chapters define the scope, the normative references, the definitions, abbreviations, notations, conventions and terminology. For this reason the first four chapters of each book are not listed in the overview of this document.

2.1.1 Book 1: Application independent ICC to terminal interface requirements

General audience: Manufacturers of smart cards, electrical engineers.

Book 1 [7] contains information on the physical and logical characteristics of the EMV specifications for both the ICC and the terminal. The physical characteristics are defined as things such as the position of the contacts of the ICC and the voltages used for communication, while the logical characteristics define the structure of the files on an ICC and the structure of the commands used to allow communication between an ICC and a terminal.

Section 5: Electromechanical interface

Additional audience: None.

In this section the electrical and mechanical characteristics of both the terminal and the ICC are listed. It starts off with the requirements for a move to lower voltage ICC, which implies that it builds on the standards that have been set in other (ISO/IEC) standards for ICCs.
It continues with mechanical characteristics, such as the maximum height and width of the ICC and the location of the contacts, before continuing with electrical characteristics. This section finishes with the same mechanical and electrical characteristics for terminals.

Section 6: Card session

Additional audience: None.

The different phases of a (normal) card session are listed here, from insertion of the ICC and activation of the contacts to execution of the transactions and the subsequent deactivation of the contacts. For each phase the states of the different contacts are listed, as well as the steps that need to be taken to allow a normal card session to be performed (which contact has to be activated using which signal at which point in time, how does a cold/warm reset need to be performed, et cetera).

Section 7: Physical transportation of characters

Additional audience: None.

Since it needs to be possible for ICCs and terminals to exchange data, a convention needs to be set on how this data is transmitted. In this section the conventions for this exchange are listed in the form of bit and character exchange. It contains information such as the bit duration and the character frame (such as which bit is transported first), allowing the exchange of information.

Section 8: Answer to reset

Additional audience: Low-level protocol implementers.

After the terminal has sent a reset signal, the ICC needs to respond with information on how to communicate from that point onwards. This information is defined in this section. It contains the characters that should be returned at an Answer to Reset for both the T=0 and T=1 protocol (explained in detail in Section 9), as well as their meaning and the behaviour of the terminal during the ATR. It concludes with a flow chart demonstrating an example flow at the terminal.

Apart from the general audience, the information in this section would be interesting for someone who would have to implement this, which is a very low-level protocol.

Section 9: Transmission protocols

Additional audience: Low-level protocol implementers.

In this section two transmission control protocols are defined: T=0 and T=1. One of these protocols is to be used for further communication, and the requirements for these protocols are listed in this section. The protocols are described in a layered manner:

- Physical layer; references section 7 as that section handles this layer.
- Data link layer; describes the timing requirements, the different options and the error handling needed for both protocols.
- Terminal transport layer; defines the layer that is responsible for the transport of the command and response APDUs described in the application layer.
- Application layer; defines the structure of command and response APDUs.

As with section 8, someone who has to implement T=0 or T=1 will be interested in this specific section, while more generic ICC application programmers might be interested in the details of T=0 and T=1 as well. The general structure of command and response APDUs are also mentioned and might be interesting for an ICC application programmer, but Section 11 of this Book and more specifically Book 3 (which is much more suitable in general for the ICC application programmer anyway) go into much more detail.

Section 10: Files

Additional audience: ICC operating system developers.

This section elaborates on how an application on an ICC should store information and how this information should be structured on the ICC itself. It references the ISO/IEC 7816-4 standard, which implies much more information can be gathered from that standard. Apart from the structure it also specifies how files can be referenced.

Apart from the general audience, an ICC operating system developer would most likely be interested in the information in this section, as well as the information referenced in ISO/IEC 7816-4.

Section 11: Commands

Additional audience: General ICC application and terminal programmers.

Messages that are sent from the terminal and the ICC are sent in a specific format called the Application Protocol Data Unit (APDU) format. However, this Book only lists the commands that are necessary to select an application on the ICC. Additional commands are defined in Book 3.

This section lists the structure of the *READ RECORD* and *SELECT* command and response APDUs.

The additional audience listed here might be interested in these commands, but will generally be much more interested in the contents of Book 3.

Section 12: Application selection

Additional audience: ICC operating system developers, general ICC application and terminal programmers.

This section describes how a terminal and an ICC can agree on which application should be used to perform transactions. This is the first step that needs to be taken after contact activation and the card reset. It describes the use and structure of Application Identifiers(AID) and how a list of candidate AIDs is used to allow selection of a particular application.

Annex A: Examples of exchanges using T=0

Additional audience: Low-level protocol implementers.

This section is exactly what its name implies: examples illustrating the exchanges of data between the terminal and the ICC.

Annex B: Data Elements Table

Additional audience: General ICC application and terminal programmers.

In this section the data elements that may be used for application selection are listed, as well as how they map to files and other data objects.

Annex C: Examples of directory structures

Additional audience: ICC operating system developers.

Examples of directory structures as they are specified in section 10 are listed in this annex.

2.1.2 Book 2: Security and key management

General audience: Cryptographers (analysts and implementers), certification authorities.

In book 2 [8] the policies, functionality and requirements for security and key management are described. It describes the minimum level of security required for both ICCs and terminals in order to guarantee proper interoperability and correct operation. Apart from the requirements between the ICC and the terminal, requirements and recommendations are also made with respect to on-line communication between ICCs, terminals and the payment system in place at the issuer.

Examples of information that can be found in this book are authentication of ICCs and how messages should be sent that allow for confidentiality and integrity of those messages.

Section 5: Static Data Authentication(SDA)

Additional audience: None.

The EMV standard allows for offline authentication of the data that is present on the ICC, and offers static and dynamic methods of verifying that unauthorised alterations to the data after personalisation have not occurred. Static Data Authentication(SDA) is described in this section, and as the name implies it only uses static means of verifying the authenticity of the data referenced by the Application File Locator (AFL) and the (optional) Static Data Authentication Tag List. What this means in practice is that after personalisation of the card, a digital signature is created for the static data on that card, so its authenticity can be verified.

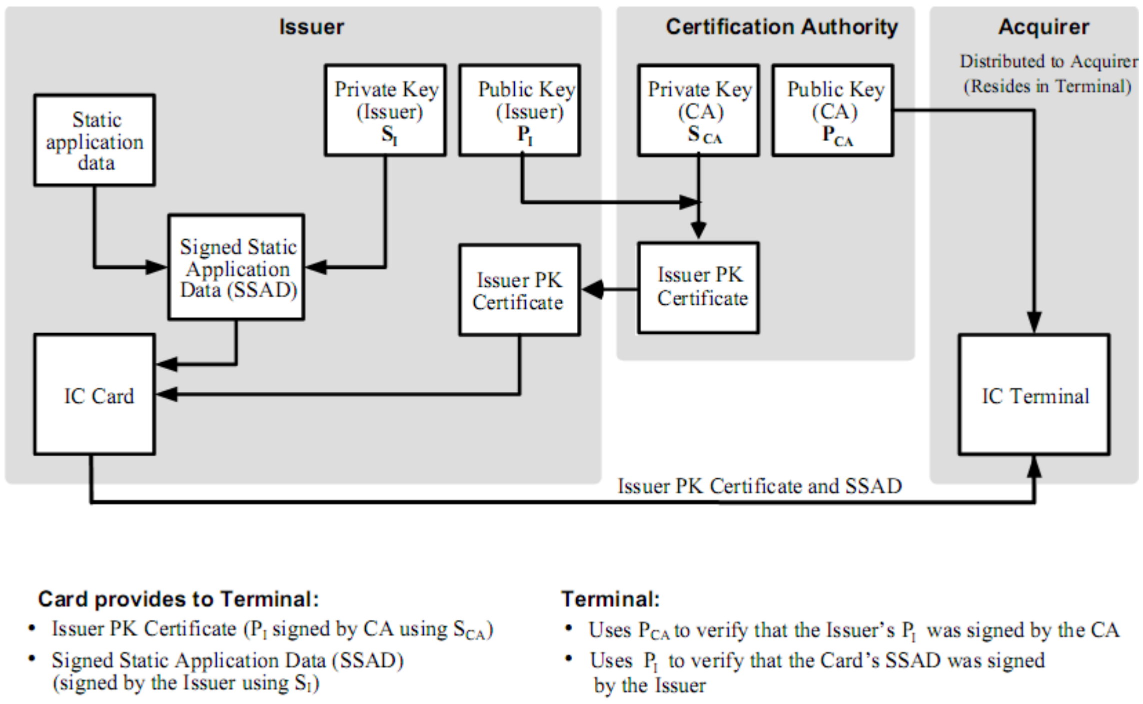
The section describes how SDA works in many different aspects, such as the data elements that are required to support SDA and how the different elements need to be signed and verified.

Figure 2.1 shows a diagram on how the relationship between cryptographic keys and data to be authenticated is established, and is taken directly from the EMV standard[8].

Section 6: Offline Dynamic Data Authentication

Additional audience: None.

This section of book 2 describes two different forms of dynamic data authentication in an offline environment: Dynamic Data Authentication(DDA) and Combined Dynamic Data Authentication/Application Cryptogram Generation


Issuer
Certification Authority
Acquirer
Distributed to Acquirer (Resides in Terminal)
Static application data
Private Key (Issuer) S_I
Public Key (Issuer) P_I
Private Key (CA) S_{CA}
Public Key (CA) P_{CA}
Signed Static Application Data (SSAD)
Issuer PK Certificate
Issuer PK Certificate
IC Card
IC Terminal
Issuer PK Certificate and SSAD

Card provides to Terminal:

- Issuer PK Certificate (P_I signed by CA using S_{CA})
- Signed Static Application Data (SSAD) (signed by the Issuer using S_I)

Terminal:

- Uses P_{CA} to verify that the Issuer's P_I was signed by the CA
- Uses P_I to verify that the Card's SSAD was signed by the Issuer

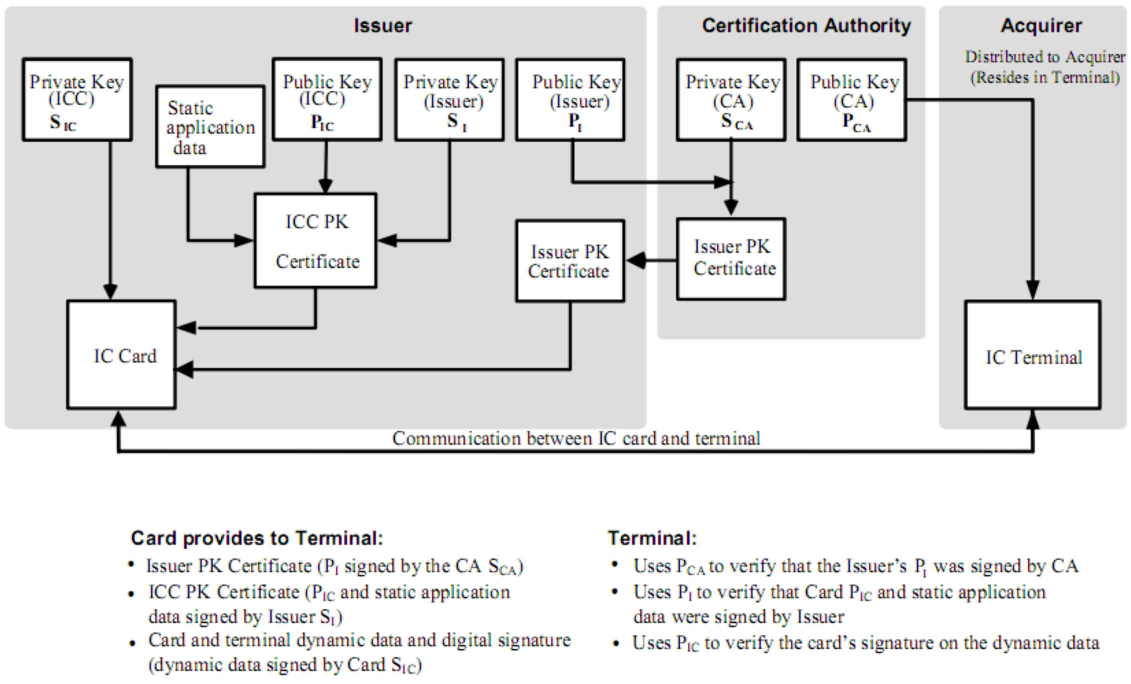
Figure 2.1: Diagram of SDA, taken directly from [8]

(CDA). Initially this led to some confusion during this research project, as Dynamic Data Authentication seemed to be the same as Offline Dynamic Data Authentication. However, it is good to distinguish between the two, as Dynamic Data Authentication is a subset (one of the two possibilities described in the EMV standard) of Offline Dynamic Data Authentication. From this point forward, when DDA is mentioned it refers to Dynamic Data Authentication and not Offline Dynamic Data Authentication.

The difference between SDA and DDA is that DDA allows for authentication of data that was generated during its lifetime, and not just data that was generated at its issuance. In practice this means that ICCs supporting DDA can create new authentication tokens when needed, while ICCs supporting SDA always use the same token to authenticate.

Both methods are described in this section in terms of requirements, the steps that are needed to create and verify such authentication tokens, et cetera. CDA can do everything that DDA can do, but can also create signatures on an *Application Cryptogram* (AC). This means transaction related data can also be signed, while normal DDA only provides the possibility of validating card data in a dynamic fashion. The section concludes with a flowchart depicting an overview of a sample flow of a CDA transaction.

Figure 2.2 shows how the cryptographic keys and card data relate to each other, and is directly taken from [8].


Issuer
Certification Authority
Acquirer
Distributed to Acquirer (Resides in Terminal)
Private Key (ICC) S_{IC}
Static application data
Public Key (ICC) P_{IC}
Private Key (Issuer) S_I
Public Key (Issuer) P_I
Private Key (CA) S_{CA}
Public Key (CA) P_{CA}
ICC PK Certificate
Issuer PK Certificate
Issuer PK Certificate
IC Card
IC Terminal
Communication between IC card and terminal

Card provides to Terminal:

- Issuer PK Certificate (P_I signed by the CA S_{CA})
- ICC PK Certificate (P_{IC} and static application data signed by Issuer S_I)
- Card and terminal dynamic data and digital signature (dynamic data signed by Card S_{IC})

Terminal:

- Uses P_{CA} to verify that the Issuer's P_I was signed by CA
- Uses P_I to verify that Card P_{IC} and static application data were signed by Issuer
- Uses P_{IC} to verify the card's signature on the dynamic data

Figure 2.2: Diagram of DDA, taken directly from [8]

Section 7: PIN encipherment

Additional audience: Terminal application developers

When a PIN is entered into a PIN pad, it is possible to insist on the secure transfer of this PIN to the ICC to prevent sniffing of this sensitive information. The EMV standard provides a solution for this in the form of PIN encipherment, and this section goes into the details of this solution.

Apart from referencing the appropriate book and section(section 6.5.12 of book 3) that describes the data structure in which a PIN should be stored, it states the necessary keys and certificates to allow for PIN encipherment. It is possible to use the same keys and certificates that are used in Offline Dynamic Data Authentication (as specified in section 6 of book 2), but the EMV standard prefers the use of a specific public and private key pair for PIN encipherment. The certificate that contains the public key is stored in exactly the same way as described in section 6: offline dynamic data authentication. This section also lists the steps necessary to retrieve the PIN encipherment public key, how the PIN entered into the PIN-pad should be enciphered using the PIN encipherment public key and how the ICC should extract and verify both the complete message and the PIN.

As a final note on this subject concerning the additional audience specified for this section it is important to realise that for the e-dentifier and the devices listed in [5] this section was not relevant, since the PIN it sent to the card in a plaintext format.

Section 8: Application cryptogram and issuer authentication

Additional audience: Application developers.

This section describes the different methods that can be used to generate Application Cryptograms. These cryptograms are used during a transaction to confirm that a transaction is legitimate, and are generated by the ICC. Three types of ACs are defined in this section:

1. Application Authentication Cryptogram (AAC) - Used when a transaction is declined.
2. Authorisation Request Cryptogram (ARQC) - Used when a terminal wishes to perform online authentication.
3. Transaction Certificate (TC) - Used when a transaction is accepted.

The structure of the application cryptograms is the same for every time; they are simply a Message Authentication Code (MAC) generated over certain data elements. The recommended minimum amount of data elements that should be included in this MAC (such as the amount, terminal country code, an unpredictable number, a transaction counter, et cetera) are described in this section, as well as how the key for the MAC algorithm should be determined. The sources for this data are also listed, such as that the transaction counter of the data elements to be signed is provided by the ICC while the terminal provides values such as the transaction amount, transaction date and an unpredictable number. Figure 2.3 gives an overview of the generation of an application cryptogram (note that not all possible fields have been included for brevity).

SKD(key) = Session Key Derivation function
MAC(data, key) = Message Authentication Code function
MK_{AC} = ICC Application Cryptogram Master Key
SK_{AC} = SKD(MK_{AC})
AC = MAC((amount, date, nonce, transaction counter, ...), SK_{AC})

Figure 2.3: Application Cryptogram generation

Note that this section in itself does not formally define the differences between the different ACs. Instead it refers to section 10.8 of book 3, but more relevant information is also available in section 6.5.5 and section 9 of book 3. More information on this shortcoming and the formal meanings of these cryptograms see section 2.2.2 of this document.

Apart from these AC, this section also explains the Authorisation Response Cryptogram (ARPC), which can be used for issuer authentication. ARPC consists of two possible methods, and the steps needed for both methods (algorithms, data elements to be used in those algorithms and where the keys used in those algorithms should be taken from) are listed in this section.

This section concludes with some words on key management and involves a reference to Annex A1.4 of this book with respect to key derivation techniques based on the Primary Account Number (PAN) and the PAN sequence number.

Section 9: Secure messaging

Additional audience: Custom application developers.

Being able to send messages that are confidential, being able to guarantee the integrity of such messages and having the possibility of authenticating the sender of the message is obviously essential in an application such as an electronic bank card. This section describes how secure messages with these properties can be sent.

This section starts with explaining how secure messages can be formatted and lists two possibilities:

1. A format according to ISO 7816-4 which uses Basic Encoding Rules-Tag Length Value (BER-TLV) data field encoding, with strict enforcement of ASN.1/ISO 8825-1 encoding rules.
2. A format in which commands that use secure messaging do not use BER-TLV encoding, but may use it for other purposes. In this format the data objects and their lengths need to be known by the sender and currently selected application.

The rest of this section deals with how the secure message needs to be constructed for integrity and authentication (explained in a shared sub-section) and confidentiality (explained in a separate sub-section) per possible format. It deals with how the division of the response should be handled, as well as determining the required keys (such as the key for the MAC algorithm in case of integrity/authentication) and key management. For both sub-sections a reference is made to Annex D2, in which an example is provided.

Section 10: Certification authority public key management principles and policies

Additional audience: None.

Payment systems that follow the EMV standard and that want to use either SDA or offline DDA will inevitably run into the need to set up a public key infrastructure. This section defines the principles (*"concepts identified as the basis for implementing Certification Authority Public Key management"*) and policies derived from those principles that should apply to *all* payment systems.

A public/private key pair usually follows the following five consecutive phases:

1. Planning
2. Generation
3. Distribution
4. Key usage
5. (Scheduled) revocation

Each phase is described, along with steps that need to be taken to preserve the integrity of the keys. For example, in the distribution phase it is extremely important that keys that are sent from the certification authority to issuers (such as banks issuing EMV cards) and acquirers (such as suppliers of merchant terminals) are only sent over a channel that preserves the integrity of the messages. If this is not the case, corrupt keys could be distributed.

Apart from the normal life cycle of a public key it is also possible that a key is comprised. When this occurs, this section adds four new phases to the normal life cycle:

1. Detection
2. Assessment
3. Decision
4. (Accelerated) revocation

As with the normal phases, every phase is once again listed in this section. For example, in the detection phase it is not needed that a key has actually been compromised. If factorisation techniques have evolved in such a manner that it is possible to break keys, even if it has not happened yet, it might be necessary to take action.

The next part of this section goes into the actual principles and the policies that are derived from these principles, such as the principle that support for key revocation is mandatory for all payment systems. It lists the principles for each phase, and the policies that should apply for *all* payment systems. In some cases the EMV standard only provides the principles, as policies that apply to all payment systems cannot be established.

The final part of this section (section 10.3) gives sample overviews of the time lines based on the principles and policies that have been given.

Section 11: Terminal security and key management requirements

Additional audience: Terminal developers (physical terminals and applications).

Since terminals have to handle sensitive data such as cryptographic keys, and those terminals are often placed in untrusted locations, it is extremely important to set requirements on the handling of this data in terminals. This section elaborates on this aspect of the system, but only in terms of key management requirements for public keys. Security requirements for PIN pads are *not* listed here, but are instead delegated to be the responsibility of individual payment systems. However, they are referred to ISO 9564 for general PIN management and security principles.

As with the previous section, requirements for key management are divides into four phases:

1. Introduction of a certification authority public key in a terminal
2. Storage of a certification authority public key in a terminal
3. Usage of a certification authority public key in a terminal
4. Withdrawal of a certification authority public key in a terminal

For each phase the principles are listed, such as the need for a terminal to be able to verify that the key(s) it has received is error-free and originated from the proper authority. Apart from these principles, requirements are also listed. For example, terminals supporting SDA or offline DDA must support six public keys per application.

Annex A: Security mechanisms

Additional audience: None.

This annex gives an overview of the different cryptographic techniques and its settings that are used in the EMV standard. It explains how padding should be done if a message is not a correct multiple for encryption, which modes of operation are allowed (Electronic codebook (ECB) and cipher block chaining (CBC)) and how they work, and how essential aspects such as session key derivation and master key derivation should be implemented. It does this for both symmetric schemes (such as how to compute a MAC) and asymmetric schemes (such as signature generation and verification).

Annex B: Approved cryptographic algorithms

Additional audience: None.

The algorithms that are allowed to be used in EMV compliant applications are listed in this Annex. The algorithms that are approved as hashing algorithms are the following:

- Triple DES; approved for encipherment and MAC.
- RSA; approved for encipherment and digital signature generation.
- Secure hash algorithm (SHA-1); approved as a hashing algorithm. s

Annex C: Informative references

Additional audience: None.

Contains references to documents that can give more information on the subject of security and key management.

Annex D: Implementation considerations

Additional audience: None.

Considerations that need to be made when implementing a payment system according to the EMV standard are made in this annex, such as the notion that there is a maximum of 256 data bytes in a response. Public keys might be longer than this maximum response, meaning that this is something that should be considered.

A very interesting part of this annex is the part that illustrates how a format 1 (see section 9 of book 2) secure message can be constructed.

2.1.3 Book 3: Application specification

General audience: Application developers, operating system developers, reverse engineers.

This book specifies the necessary building blocks required to develop applications according to the EMV standard. It specifies the structures of the commands that can be used, the types of information sets that can be stored on a card and how the general flow of an application works. This book is the most important book for someone developing or reverse engineering EMV compliant applications, such as what has been done for this research project.

Section 5: Data elements and files

Additional audience: None.

Applications obviously need data to work with in almost all situations, and this section defines how applications can store and use data. This section heavily references both annex A and B, which contain a dictionary of data elements and how those data elements are encoded.

Data elements are raw pieces of data and can be considered the smallest data containers. It contains information such as the PAN or the Application Identifier.

Data objects are pieces of data that consist of a tag, length and value, and can contain other data objects or a data element. The tag is unique in the environment of the application and can be used by the application to reference a data object. It is possible for data objects to be present but not used. When this happens there is no data in the value field, and the length field of the data object is '00'. Terminals *must* be able to correctly interpret this situation!

It is possible to store one or more data objects as a template, and these templates are called records. How the different data objects are mapped into records is left to the issuer.

Physically storing those pieces of data on an ICC occurs in so called *Files*, and these files are available to the ICC as records. The files themselves can be referenced through a name (which somehow contains a reference to the application it belongs to through the Application Identifier) or through a Short File Identifier (SFI).

During an EMV transaction it is likely that the terminal will need to construct a dynamic list of data elements, for example to have the card create a signature for a transaction. Since the card only has limited processing power, those dynamic data lists are not encoded as normal data objects, but instead are constructed by concatenating different data elements together. Since normal encoding is not used, the ICC contains Data Object Lists (DOL) which specifies the format of the data. An example of when this is used is when the terminal sends a Generate Application Cryptogram command to the ICC. Card Risk Management Data Object Lists (CDOL) are used to indicate which elements are included in a signature.

Section 6: Commands for financial transaction

Additional audience: None.

This section contains the commands that can be sent from a terminal to an ICC during a financial transaction, and what an ICC can send back to the terminal as a response. Commands are sent in a so called Application Protocol Data Unit (APDU) format, and come in two flavours: command and response APDUs. This information is taken from the ISO 7816 standards, and should be known to anyone who has ever worked on developing an application that corresponds to this standard (which will most likely be anyone who has recently worked on a smart card application).

The only information that can be considered new in the first part of this section is the fact that commands that are proprietary to the EMV specifications are coded to have a class byte (CLA) starting with an '8'.

The final part of this section contains a very interesting list: an overview of the relevant command/response APDU pairs in terms of what it should do, which values should be set in the command header (such as the instruction (INS) byte), and what should be returned. Since this information was essential for this research project when attempting to get insight into the protocols that have been used, an overview has been made available in appendix A of this document with the different class and instruction bytes along with which command it corresponds to.

Section 7: Files for financial transaction interchange

Additional audience: Implementation verifiers.

Part III (Section 10 to 12) of Book 1 of the EMV standard [7] has already given specifications for files, but this has mainly been done for high level file types such as applications, directories and commands. This section defines files that *should* be present (Section 7.2) and how they should be mapped (Section 7.1). A prime example of two files that should be present for every financial application are the Card Risk Management Object Data List 1 and 2. These files should be readable through the *READ RECORD* command, and should be available in a linear file available within the SFI range of 1 and 10.

Apart from the data objects available through the *READ RECORD* command, some data objects are only available through the *GET DATA* or *GET*

PROCESSING OPTIONS command. Section 7.3 and 7.4 list these data objects, and whether or not they are mandatory.

The final section (7.5) specifies how a terminal should react if mandatory data (whether it's mandatory by default or made mandatory in order to support optional functions) is missing or incorrect. If a terminal discovers this, it is very important for it to set the *ICC data missing* indicator bit to 1 in the terminal verification results. Another important piece of information is that it is the duty of the issuer of the ICC to make sure the data is in the proper format. For example, data objects that do not parse correctly should lead to the terminal terminating the transaction, even if it can deduce the information it needs from it. This section concludes with a table of conditions in which the *ICC data missing* indicator bit should be set to 1.

Section 8: Transaction flow

Additional audience: None.

As the name implies, this section gives an overflow of the flow of a transaction. However, this section only looks at the transaction flow inside the application layer, as Book 1 has already specified the functionality outside the application layer (such as application selection). 8.1 states that if an exception is thrown this is signified through the SW1 and SW2 status words. Any other value than 9000, 63Cx, or 6283 shall cause the terminal to terminate the transaction.

Section 8.2 gives a useful flowchart, which has been used during this research project and has been copied in this document as figure 3.1. The final part of this section allows for additional functionality not specified by the EMV standard (or future additions made to the EMV standard), but states that may not interfere with terminals and ICCs not implementing these features.

Section 9: GENERATE AC command coding

Additional audience: None.

In this section focusses heavily on the *GENERATE AC* command, and gives an overview on what should be possible with this command. It is an important part of the EMV standard in terms of defining the semantics of the different types of cryptograms (see section 2.2.2 for its relevance for this research) and the flowchart it starts with gives a good impression of what is possible with the cryptograms.

Apart from the flowchart, it also defines that the CDOL 1 and CDOL 2 data object lists are extremely important (as they provide most of the transaction data to be used as input to the final cryptogram), and that the format of the data sent to an ICC is not in its usual TLV format, but instead in a concatenated format defined by the CDOLs. This section concludes with an overview of how the command should be used, as it can be used twice during a single transaction. For example, the ICC may only respond with an ARQC at the first request of the terminal. It may never respond with an ARQC to the second request.

Section 10: Functions used in transaction processing

Additional audience: Implementation verifiers.

This relatively large section starts off by referencing part II (Section 5 to 7) of Book 4 of the EMV standard, as it may define additional terminal-specific requirements. In this section a (chronological) list steps is given for a transaction in a textual format, and can be seen as a very informal description of the possible protocol used during an EMV transaction. It never actually gives a clear high-level overview of the protocol, which would have been useful as it is much easier to read than the wall of text that is used here. Note that the perspective of this section is entirely from the viewpoint of the terminal.

It defines the "phases" of a transaction, the conditions that apply to this phase having to be executed (for example reading application data is mandatory, while cardholder verification depends on whether the ICC supports it), when the particular phase may be performed and what the details are of the phase. This section defines the following phases:

- Initiate application processing
- Read application data
- Offline data authentication
- Processing restrictions
- Cardholder verification
- Terminal risk management
- Terminal action analysis
- Card action analysis
- Online processing
- Issuer-to-Card script processing
- Completion

Annex A: Data elements dictionary

Additional audience: None.

This annex defines all the data elements that may be used for financial transactions. It contains two tables with the same information, except that one table is sorted by the name of the data element, while the other is sorted by the tag. It is an extremely useful appendix, as it not only defines the data elements, but also describes them. This appendix was used often during this research project.

Annex B: Rules for BER-TLV data objects

Additional audience: None.

Defines how BER-TLV encoded objects should be created and interpreted. It (obviously) references ISO 8825, as it is taken from this standard. It describes how the tag, length and value parts of these objects should be encoded and gives an overview of the meaning of different parts of these objects at the bit level.

Annex C: Coding of data elements used in transaction processing

Additional audience: None.

The EMV standard defines a lot different data elements, such as the application interchange profile which tells a terminal the capabilities of an ICC. Since an ICC can have many different capabilities (such as supporting SDA, DDA or CDA), being able to interpreting these values is important, as most of these data elements are only a few bytes in size. This section lists how these values should be interpreted by listing what it means when a particular bit is flipped or when a byte has a particular value.

Annex D: Transaction log information

Additional audience: Log extraction device creators.

Defines a method for special devices to extract transaction log information from an ICC. It should be interpreted as an additional phase to those defined in Section 10 of this book. It describes the Log Entry and Log Format data elements, and additional information such as the range of the SFIs in which they can be found.

Annex E: TVR and TSI bit settings following script processing

Additional audience: None.

This annex lists which bits should be set n which values when processing a script. It gives four scenarios and describes what should be done in those scenarios. It also references Book 4 Annex A5 for the definitions of Issuer Script Results.

Annex F: Status words returned in EXTERNAL AUTHENTICATE

Additional audience: None.

The *EXTERNAL AUTHENTICATE* command is a command used to provide a cryptogram to the ICC which then has to verify this cryptogram. This annex defines the possible responses the ICC can give to this request, explains why and when a particular response should be given and what a terminal should do when such a response occurs.

Annex G: Account type

Additional audience: None.

Gives the possible values for the Account Type data element.

2.1.4 Book 4: Cardholder, attendant, and acquirer interface requirements

General audience: Terminal/point of sale/host devices developers (both physical and applications), software architects.

As the title of the book suggests, a lot of the information in this book relates to the requirements imposed on the devices used to communicate with EMV compliant ICCs. This includes Interface Devices (IFD), which are the devices that connect to an ICC and perform the physical communication, but also Point of Sale (POS) devices and merchant/acquirer host devices that manage the transactions.

Section 5: Terminal types and capabilities

Additional audience: None.

The different types of terminals, as well as their capabilities are briefly listed in this section. A division is made between the following aspects of terminals:

- **Environment:** Attended and unattended terminals (unattended terminals do not require participation from an attendant to provide transaction data, he or she may still be present).

- **Communication:** Online only, offline with online capability and offline only terminals.

- **Operational control:** Financial institution, merchant or cardholder; who bears the responsibility of the operation of a terminal.

As the terminal type needs to be made clear to the card somehow, an coding of this information is needed. This encoding is not listed in this section, but is instead listed in Annex A.

Apart from the types of terminals, the capabilities of such terminal are also listed in this section. These capabilities are elements such as which methods it has to verify the cardholder and which types of transactions it can do. The encoding of these capabilities are once again listed in Annex A, and not in this section.

This section concludes with three examples of possible configurations of the terminals, such as one being connected to a POS device without online capabilities.

Section 6: Functional requirements

Additional audience: None.

This section makes it very clear that it does not want to replicate the other books of the EMV standard, but still references all three of them in what an implementation should conform to. This section looks specifically at implementation issues and how the other books might impact the development of an EMV compliant terminal. This section does expand on some additional requirements to Book 3, and does so in terms of the phases as they have been defined in Section 10 of Book 3.

These additional requirements relating to Book 3 are mostly about setting which bit in which case, for example that you have to set the cardholder verification method results bit to 'unknown' if a paper signature was used, but also includes information such as what should be done if a card initially requests an amount field in the PDOL and it is not available (the terminal should display an *ENTER AMOUNT* message and forward the requested amount to the ICC).

After expanding on the additional requirements to Book 3, Section 6.4 gives the conditions for support of function. As an example, a terminal that supports DDA or CDA must also support SDA. Section 6.5 specifies other functional requirements for terminals, such as those relating to amount entry (what should be done if the amount is not known yet at the time of a transaction, like when at a night terminal of a petrol station) and the transaction counter present on the terminal.

Additional requirements for card reading have been specified in Section 6.6. During a transition phase, card owners may still use the old magnetic stripe of a card, and when the magnetic stripe indicates an ICC is present, terminals must prompt the card owner to insert the card into the IC reader. This is one of the requirements specified in Section 6.6.

The final part of this section, Section 6.7, gives terminal requirements for date management.

Section 7: Physical characteristics

Additional audience: None.

In this section physical characteristics for terminals are given, such as what types of keys a keypad can contain, how the (command) keys should be coloured and located on the keypad, and that the '5' key should have a tactile identifier. Displays are also mentioned, but more broad notions of physical characteristics are also mentioned, like how a terminal should provide memory protection (so data does not get lost) and the existence of an internal clock.

The two final sub-sections deal with the printer (mandatory for all non-cardholder controlled terminals) and the requirement for a terminal to have a magnetic stripe reader if payment system rules indicate this.

Section 8: Terminal software architecture

Additional audience: Terminal operating system developers.

Section 8 is the first section of Part III, Book 4 of the EMV standard, and this part is labelled as the Software Architecture part. What this particular section mainly deals with is the future, as it tries to provide insight into how the move from magnetic stripe cards to tamper-resistant chips also has consequences for terminals. With magnetic stripes the terminal has to do little more than interpret the limited amount of information, and send it on when needed. However, much more interaction is expected from a terminal that communicates with a chip, and making sure that the terminal can keep up with a changing environment over a long period of time is essential.

The need for terminals to accept new applications without having to modify and/or certify existing applications is made very clear in this section. In order to help with this need, this section suggests two approaches to the software architecture:

- Application Program Interface (API) approach
- Interpreter approach

The API approach is a software architecture in which all applications are made using a set of essential and frequently used functions. These functions are available through a standard interface called the API. This approach has the advantage of allowing code reuse through different applications, offering an abstraction layer for the hardware on which it runs, and by providing easier certification of new applications as they use an already certified API library.

The interpreter approach is an intermediate layer that defines a single kernel for multiple terminal types. The kernel exposes a virtual machine that can be implemented across different CPU types, and can expose drivers for I/O and low-level functions. This provides the advantages of having a single kernel that only has to be verified once for multiple platforms, that has generalised ICC support functions and only has to be installed once during the lifetime of a terminal (approx. 7-10 years). It also gives the advantage of allowing CPU-dependant plugs that can enhance a terminal's behaviour.

For both approaches the software architecture should have the ability to maintain an application library of modules or routines that may be invoked during the processing of a given transaction. These may be available to all applications, or may be restricted to particular applications or payment systems.

This section concludes with a suggestion of so called plugs and sockets, which allow a terminal to be enhanced by its acquirer/payment system owner. Necessary changes to the existing code or behaviour of a terminal can be added in this manner.

Section 9: Software management

Additional audience: Terminal operating system developers.

This extremely short section deals with the necessity of being able to upgrade software once it has been deployed in a terminal, for example to fix a bug. It does not specify much in terms of requirements, except for the need to verify the identity of the party loading the software and verifying the integrity of the software that is loaded.

Section 10: Data management

Additional audience: Terminal operating system developers.

This section deals with the management of data specifically during terminal initialisation or data that is needed during transactions. For example, when a data element is initialised or updated its integrity needs to be assured by the terminal. Different data elements should be controlled by a particular party, like how the IFD serial number should controlled by the terminal manufacturer and how the local date and time should be controlled by the merchant.

The section continues with listing the data elements that are application independent (such as the local date and time), as well as the data elements that are application dependant (such as the Application Identifier (AID)). The application dependant data elements are specified by individual payment systems.

Section 11: Cardholder and attendant interface

Additional audience: None.

Since attendants and cardholders will need to operate the devices specified in the previous sections, requirements and suggestions have to be made from that perspective as well. This section looks at these aspects, such as the language that needs to be used when operating a terminal. Not only does it list the requirements for support of the languages that need to be available for cardholders and attendants, it also gives a list of standard messages and the values of the message identifiers belonging to those standard messages. As an example, a message identifier of '09' means the ENTER PIN message needs to be displayed.

Another aspect of cardholder/attendant interfacing lies in the selection of an application. When inserting a card into a terminal, multiple applications might be usable for the terminal, and a terminal might be able to offer the cardholder the possibility of selecting one of those applications. How this should be done, as well as what should be done if a terminal can *not* offer this possibility is described in Section 11.3. The final section, Section 11.4 briefly mentions the need for the AID of the application to be printed on the receipt.

Section 12: Acquirer interface

Additional audience: Data exchange system developers.

So far the EMV standard has focussed on the transactions themselves, but when they are complete (or even during a transaction) the data resulting from that transaction needs to be communicated to the acquirer and issuer in order to process them. This section looks at messages sent from the terminal to the acquirer, mostly by establishing which data needs to be sent in those messages. For example, in an authorisation request message the application interchange profile, application transaction counter and the ARQC needs to be sent (among others, as well as some optional data elements). These data elements are both related to the ones defined specifically for ICCs as well as the data elements as they were used in existing systems (such as magnetic stripe systems).

Of course something can go wrong during an exchange of data (such as when the network is unavailable and going online is impossible, or a request/response might get lost or arrive too late), and Section 12.2 describes how the exception handling process should be implemented.

Annex A: Coding of terminal data elements

Additional audience: ICC application developers, reverse engineers.

This annex lists the encoding for the terminal types, (additional) terminal capabilities, cardholder verification method results, issuer script results and authorisation response code. It would most likely be useful to include this information in Book 3 as well, as it is extremely related to the information presented in that book.

The encoding is either given through direct values (a terminal type value of '34' means an unattended, online only terminal that has to be operated by the cardholder) or through a capabilities scheme in which each bit in a value represents some form of capability.

Annex B: Common character set

Additional audience: ICC application developers, reverse engineers.

In this annex a chart is given on how characters should be represented in an EMV compliant application. It uses a single byte to represent these characters, and the chart shows the values the bits need to have to represent a certain character.

Annex C: Example data element conversion

Additional audience: Data exchange system developers.

An example table of how the data elements mentioned in Section 12.1 relate to the ICC-related data elements listed in Book 3 of the EMV standard. This is done by listing the tag of the data element and its corresponding name, and relating them to the data elements from Section 12.1.

Annex D: Informative terminal guidelines

Additional audience: None.

Due to the wide variety of environments and location in which terminals can be installed, this annex provides guidelines on how this can best be approached. An outdoor terminal in the heart of Africa needs to be able to withstand different (environmental) influences than a terminal located in a northern part of Sweden.

This annex contains guidelines on the power supply, keypad (the lettering should be wear-resistant) and display. It also gives a useful list of informative references.

Annex E: Examples of terminals

Additional audience: None.

This annex does not provide requirements, but instead gives examples of different types of terminals. It gives a physical and functional description of those terminals, but also provides a list of how the coding of the terminal-related data would have to be. It gives an example of a POS terminal, an ATM and a vending machine.

2.2 Relevant aspects of the EMV specifications

Since the EMV specifications are extremely broad, and cover many different aspects that are relevant for the development of financial applications on an ICC it is obvious that some aspects were more important than others during this research project. For example, the physical characteristics of an ICC were not important compared to the structure of the commands that can be used for financial transactions. This section looks at which parts of the EMV standard have been the most important, and tries to collect and summarise the information that is relevant for comparable research projects, as some important aspects of the EMV specifications are scattered across different parts of the specifications and have to be pieced together.

2.2.1 Important EMV specification sections for this research

For this research project all four books have been studied, but book 2 [8] and book 3 [9] were the most relevant for this research. More specifically, the following list is an overview of the chapters that have been important during this research project and why they were relevant:

- Book 1, chapter 8: *Answer to Reset.* Used to help establish whether or not the baud rate was the problem when using the sniffing device (see 3.2.)
- Book 1, chapter 9: *Transmission protocols.* Used primarily to establish what went wrong when certain commands gave an incorrect response, and led to discovering a weakness in the EMV standard (see 3.3.)
- Book 2, chapters 5 and 6: *Static Data Authentication and Offline Dynamic Data Authentication.* Defines what is needed for both SDA and offline DDA.
- Book 2, chapter 8: *Application Cryptogram and Issuer Authentication.* Defines the different Application Cryptograms that can be generated and what they represent.
- Book 3, chapter 6: *Commands for Financial Transaction.* Defines the structure of the different commands that can be used according to the EMV standard. This is not a complete list of all possible commands, as some of those are defined in book 1 (such as application selection).
- Book 3, chapter 9: *GENERATE AC Command Coding.* Primarily used for its overview of the different steps that are needed to get an Application Cryptogram.
- Book 3, annex A: *Data Elements Dictionary.* Used for the overview of the possible TLV elements.

2.2.2 Information gathered from multiple parts of the specifications

Some aspects of the specification that were necessary for this research project were not fully described in a single section of the EMV standard. They had to be gathered from different sections of the different books, which makes it hard to figure out how certain parts of the specifications should be interpreted. In this subsection an overview will be given of how different parts relating to a particular subject of the EMV standard were interpreted during this research project.

Cryptogram types and their meaning

Cryptograms are first described in book 2 section 6.6, in which several requirements (including elements such as which bit needs to be set in which situation) for CDA and a description of transaction flow when using CDA are given. Several terms are introduced here such as ARQC, TC and AAC. These terms are not introduced properly and their meanings remain unclear until they are properly listed in book 3 section 6.5.5. This section lists the *GENERATE APPLICATION CRYPTOGRAM* definitions, and includes a simple overview on the meanings of the different cryptogram types. Combined with the sample CDA flow diagrams available in section 6.6.3 of book 2 and the flow chart in section 9 of book 3 (which gives an overview of the various options and use of data elements during transaction processing) it becomes clear how the different cryptograms relate to each other and what their semantics are.

What does not become clear is that cryptograms are *not* exclusive to cards supporting CDA, but that CDA makes it possible to create digital signatures over these cryptograms as well. It is possible to generate cryptograms on cards that support just SDA or DDA and not CDA as well. Cryptograms come in three different flavours: ARQC, TC and AAC.

It is interesting that the semantics of the cryptograms can quite easily be explained, as it is not extremely complicated, but this is not clearly done so in the EMV specifications. A terminal has to decide on how it wishes to verify that a transaction is authentic, and has several means at its disposal of doing this. It can do so offline (in which the data collected can be verified to be authentic immediately and will at a later point be sent to the issuer for processing) and online (in which it will send the data generated by a card in real-time to its issuer for verification).

Online request: When requesting online verification, the terminal will request an ARQC to let the card know it will verify the transaction online. The card either accepts this and generates the ARQC, or will reject it and generate an AAC. If the card accepts the online verification and the remote verification by the issuer succeeds, the terminal will request a TC from the ICC, which will then generate the TC. If the online verification fails, the terminal will not request a TC but will instead request an AAC.

Offline request: In case the terminal wishes to perform offline transaction verification, it will request a TC. If the card approves, it will generate the TC and the transaction will be complete. If the card rejects this, it can generate an ARQC to see if online processing can be performed, or reject the transaction completely by generating an AAC.

Termination request: Finally, the terminal can also reject a transaction (immediately or at a later stage as described earlier), after which it will request an AAC from the card.

The general format of the messages that are returned by an ICC are always the same:

- Cryptogram Information Data (CID) that indicates which type of cryptogram it is.
- An Application Transaction Counter (ATC) which indicates the current number of the transaction.
- An Application Cryptogram (AC), which is a MAC over data (referenced by the ICC's data objects lists and data available internally to the ICC) when it is an ARQC or a TC. If it is an AAC this field will contain information relating to the rejected transaction.
- Issuer Application Data (IAD), an optional field that contains information about the current transaction.

With this information it should be clear what the semantics are for the different cryptogram types, but also the importance of the optional IAD data. These semantics are listed in table 2.1.

AC type	**Semantics**
AAC	Used to provide proof that a transaction failed, why it has failed and the values relevant to this transaction.
TC	Used to provide proof that a transaction succeeded along with the values relevant to this transaction.
ARQC	Used to provide intermediate transaction information by having the card sign the transaction data to be authorised by the issuer, after which a TC or AAC will be requested by the terminal.

Table 2.1: Cryptogram semantics according to the EMV standard

Data that is signed in an ARQC or TC cryptogram

While it is clear from Book 3 Section 6.5.5 that the *GENERATE AC* command sent by a terminal should contain transaction-related data, it is not clear what this data actually is. Section 6.5.5.3 references Section 5.4 of Book 3 (Rules for using a Data Object List), but this section only elaborates on what a data object list is and how it should be used (it is not a TLV encoded data object, but instead a single field that consists of the concatenation of several data elements). This is used to make sure the ICC can minimise processing, and the ICC uses the data object lists to specify which data elements should be sent by the terminal to the card.

So from Book 3 alone it is not possible to deduce which transaction information is used as input to the MAC algorithm when requesting a cryptogram. However, when looking at Book 2 Section 8.1 (Application Cryptogram Generation) it becomes clear which recommended data elements are specified in the EMV specifications. Table 2.2 lists these minimum recommended data elements.

Value	**Source**
Amount, authorised (Numeric)	Terminal
Amount, other (Numeric)	Terminal
Terminal country code	Terminal
Terminal verification results	Terminal
Transaction currency code	Terminal
Transaction date	Terminal
Transaction type	Terminal
Unpredictable number	Terminal
Application interchange profile	ICC
Application transaction counter	ICC

Table 2.2: Recommended minimum set of data elements for Application Cryptogram generation, taken from [8] Section 8.1

However, these data elements are recommended, and it is not mandatory to use these particular data elements. From this part and the previous part it is not even clear yet *which* data object list should be used to know which elements to send in the data field of the *GENERATE AC* command. For this information one has to look at Book 3 of the EMV specifications again, and look at table 33 in particular. This table is part of Annex A of Book 3, which is the data elements dictionary. When looking at the Card Risk Management Data Object List 1 and 2 (CDOL 1/2) entries, it states that this data element is used in the first and second *GENERATE AC* commands respectively.

To conclude, in order to determine which data elements are used in the generation of cryptograms in a particular payment application, it is a good idea to immediately look at the application itself, and to skip looking at the EMV specifications as they cannot give a definitive answer.

2.3 Alternative EMV literature

Apart from the EMV specifications themselves, there is also alternative literature that can give an overview of the capabilities of EMV. A good example of this is a book called *Implementing electronic card payment systems* by Christian Radu [13], which takes a deep look into implementing payment applications on smart cards using the EMV specifications, and also looks at aspects such as why one would migrate to smart cards from magnetic strips. For this research, Chapter 6 was the most useful, as it gives a less formal overview on how the EMV specifications define transactions and its possibilities. The biggest weakness of this book is that it is a bit outdated, and does not include newer aspects such as CDA.

In [2] a set of privacy and non-repudiation enhancing solutions are proposed, and whenever a real-world example is needed as a payment system it uses EMV as the example. It contains brief overviews of the capabilities, that can also be very useful. For example, Section 3.2.2.2 lists EMV capabilities in terms of data authentication, and manages to explain the difference between SDA, DDA and CDA in a single page.

SDA is the cheapest solution to data authentication, and the (signed) data and its certificates are statically available to a terminal.

DDA cards also have the possibility to create additional signatures on additional terminal provided data, further proving that the data on a card is legitimate and help with the prevention of making copies of a card and its statically signed data and its certificates. However, for a card it is only possible to do this with the validation of the card's resident data, and not (for example) for transaction data.

CDA is very similar to DDA, and the only difference is that CDA makes it possible for cards to also provide a digital signature for transaction-related data such as the cryptogram, instead of just the card's static data and some additional terminal provided data. This makes it possible to detect alterations made to this data as it is sent from the card to the terminal and/or issuer.

Since the EMV specifications don't give extremely formal definitions of the protocols that should be used, it is useful to look at literature that *does* have an overview of the protocols. There are several [5, 12, 14, 11] papers on this at varying levels of abstraction, and they all look at other aspects of the EMV specifications as well (such as potential weaknesses, consequences for card users and possible extensions to the current uses of EMV), making them an interesting read.

Chapter 3

Looking at Dutch Internet banking

Since one of the goals of this research project is to find out how Dutch banks use EMV for their Internet banking applications, both the bank card and the challenge/response device (called the *e-dentifier*) distributed by ABN AMRO have been researched during this research project. The main focus lied on the ABN-AMRO banks cards and the e-dentifier, but other cards and devices from some of the other bigger Dutch banks have also been looked at. The Rabobank and SNS bank also provide their own challenge/response devices, and these devices along with their bank cards have been looked at. The ING bank does not provide a challenge/response device to its customers, but what an ING bank card exposes in terms of applets and applet data *has* been investigated.

The basic idea of the e-dentifier is simple: by inserting the bank card into the device and verifying ownership of the card by entering the PIN, challenges posed by the Internet banking application can be signed by the bank card's chip. The response from the device can be used to authorise a transaction by transmitting a part of this signature back to the Internet banking application by converting it to a decimal representation, allowing it to be easily typed into the appropriate field.

Even though the EMV standard itself does not specify anything in terms of Internet banking transactions, Section 3.1 attempts to give an insight on how the EMV standard is being used for Internet banking. This is done by looking at research that has previously been done [5], and how those results relate to the EMV standard.

In order to understand the situation in the Netherlands, what protocol is used by the e-dentifier when communicating with a bank card, and how this protocol relates to the EMV standard, the following steps have been made in order to reverse engineer the e-dentifier:

1. Using a passive traffic sniffer to see which data is exchanged by the e-dentifier and a bank card. Section 3.2 goes into detail on the setup, and why this (presumably) failed.

2. Developing a relatively easy terminal application that extracts information from a bank card. This was done to become familiar with the different programming libraries, as well as discover how the applets and data on bank cards relate to the EMV specifications. Section 3.3 describes the problems that were encountered during development.

3. Using a programmable smart card to log the commands an e-dentifier sends, and developing a *home-brew* e-dentifier application that mimics these commands. This custom made e-dentifier can then be used to extract responses from an actual bank card that are used by a real e-dentifier. Section 3.4 elaborates on this process and its results.

The final part of this chapter (briefly) looks at how other Dutch banks use a challenge/response device and an (EMV) applet for Internet banking. However, the main object of study during this research project was the e-dentifier 2 provided by the ABN-AMRO bank, and section 3.5 looks at the differences between the e-dentifier 2 and the corresponding application on the bank card to see how other banks have implemented/configured this.

3.1 Using EMV for Internet banking

The EMV standard itself does not mention Internet banking as one of the (possible) goals of the specifications, but instead focuses on providing specifications for payment systems on ICCs. However, a specification *has* been developed for Internet banking under the name Chip Authentication Program (CAP). The CAP specifications are not freely available, but since they build upon the EMV specifications it should be possible reverse engineer many aspects of this specification.

This reverse engineering has already been done in [5] for Internet banking devices in the U.K., and part of this thesis project was to see if the situation for Dutch Internet banking devices is comparable.

What was found in [5] is that a CAP implementation follows a normal EMV session in which online transaction verification is done by entering a challenge that is sent in a *GENERATE AC* command that requests an *ARQC*, followed by the rejection of the transaction by requesting an *AAC* to make sure the card does not lock up due to too many (consecutive) failed attempts. A transaction flow diagram for a normal EMV transaction was included in Book 3 of the EMV standard in Section 8.2. This diagram has been copied as Figure 3.1 in this document.

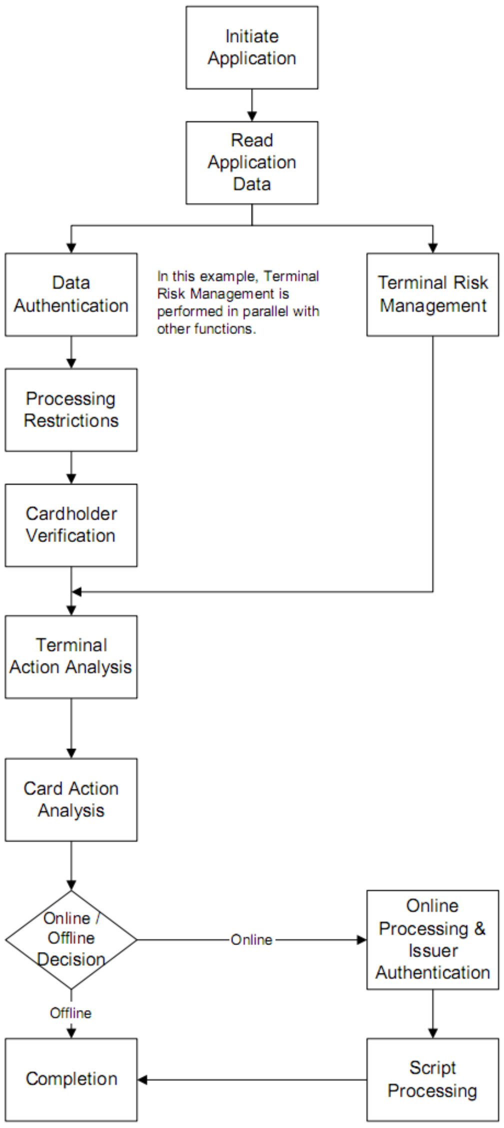


Figure 3.1: EMV program flow, taken directly from [9] section 8.2

The next chapter in the same book (Chapter 9 of Book 3) zooms in on the *online/offline decision* step, which happens to be very relevant for this research project as well as it focuses on the *GENERATE AC* command possibilities. In order to compare the possibilities with the actual implementation in the e-dentifier, the original diagram is included here in figure 3.2. This shows how a terminal and a card proceed with verifying a transaction and the steps that can be performed to do this. The end result is the acceptance or rejection of a transaction.

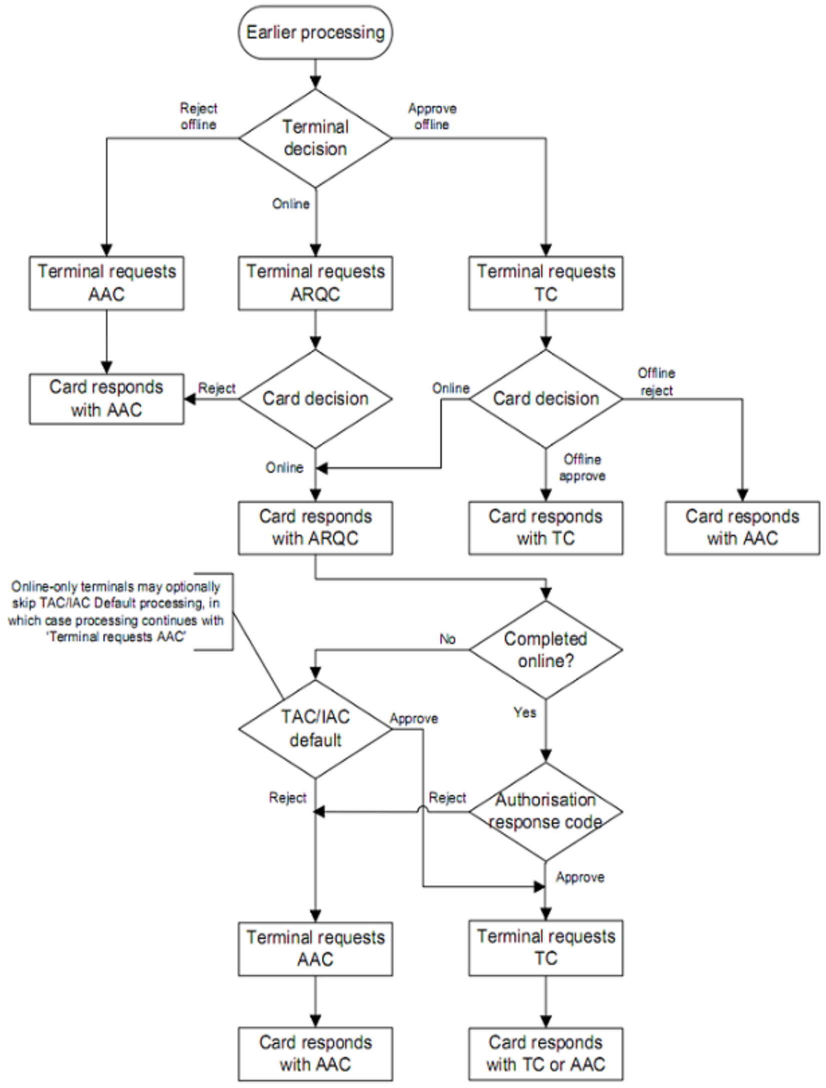


Figure 3.2: Online/offline decision - original, taken directly from [9] section 9

3.2 Sniffing reader and card traffic

The first attempt to get an insight into the protocols used by the e-dentifier was made through the use of a passive interface device that can sniff traffic passing between a card and a reader attached to the device. A *season 2 passive interface*[*]

[*]Very similar to the device available at `http://www.interesting-devices.com/asp/product.asp?product=160`

was used. This device has been successfully used at a previous research project conducted at the Radboud University, in which the Dutch ChipKnip application was reverse engineered.

3.2.1 Setup of the passive interface

The setup of the passive interface, the e-dentifier and the card was as follows:

- The passive interface was inserted into the e-dentifier.
- A bank card was inserted into the passive interface.
- The passive interface was connected to a PC on a COM port.
- The same custom software was used to collect data from the COM port as was used for the ChipKnip project.

With this setup, the passive interface will relay all the commands sent by the e-dentifier to the bank card, while simultaneously sending the data that was relayed to the COM port it is connected to. This data can then be read from the COM port by the custom software. This software uses the RXTX serial communication package[†] and is implemented in Java. This device had worked well for the ChipKnip project, and it was assumed that it could also be successfully used for the e-dentifier.

3.2.2 Results of using the passive interface

When trying to use this setup with the e-dentifier, it turned out to be not as easy as when it was used with the ChipKnip project. The traffic that was reported by the passive interface looked nothing like the traffic that should be used (ISO 7816 compliant traffic). Instead it looked like garbage, which indicated a problem existed in the setup.

In order to see where this problem originated from, a small test was set up to see whether the ChipKnip results could be reproduced. Instead of attaching the passive interface to the e-dentifier, it was attached to an OmniKey card reader to see if the passive interface would properly report traffic when used as a ChipKnip application sniffer. Note that the bank card used in both situations is the same ABN AMRO bank card.

The results from this test were only slightly better: the passive interface properly reported the ATR initially, but traffic after the ATR was once again not ISO 7816 compliant traffic. This raised some more questions, which led to a meeting with Engelbert Hubbers, who was involved in the ChipKnip project and knew how the passive interface was used during that project. In that meeting it was

[†]`http://www.rxtx.org/`

confirmed that both the e-dentifier and the Omnikey reader both mostly produced garbage. However, when attaching it to a slightly older Towitoko reader, the passive interface reported all (ChipKnip) traffic without any errors.

With these results, it is a likely conclusion that the newer readers (the Omnikey card reader and the e-dentifier) are capable of operating at higher speeds. According to the ISO 7816 standard cards initially operate at a baud rate of 9600, but in the ATR it can indicate it can operate at higher baud rates. Readers capable of operating at this higher speed can then communicate with the card at this speed.

Since the RXTX package allows a programmer to set the baud rate the serial port uses, it was possible to try and communicate at the higher baud rate that the reader and the card negotiate in order to get the results that were expected. However, this also did not seem to help. Values under 9600 baud (naturally) did not give the expected values, while rates above 9600 baud rarely changed its outcome. Whether this is because the passive interface does not support higher baud rates, and thus cannot sniff the traffic because it is too fast for the device to report is currently unknown. An alternative theory could be that a problem in the operating system or the RXTX package prevents the data from being collected at the proper baud rate.

In a comparable personal research project done by Stan Hegt and Pieter Ceelen at KPMG[‡], who used similar setup with newer versions of the same passive interface, the same results were found. It was not possible for them to use the passive interface to sniff the traffic between different bank cards and Internet banking devices and get meaningful results.

As a final remark: the baud rates that were tested were all multiples of 1200 baud up to 19200 for both the e-dentifier and the Omnikey card reader, and all multiples of 4800 up to the maximum that is supported by the RXTX package (115200) for the Omnikey reader.

3.3 Extracting application information

Before developing the bank card simulator and the software-based e-dentifier, another application was first developed that reads out the different (EMV) applets present on a bank card and the information these applets initially expose, such as the account numbers and public key certificates. This is a relatively straightforward process, which not only helped with familiarising oneself with different elements of the Java Card frameworks, but also helped shed light on how the information presented in the different EMV specifications should be interpreted by using real data present on bank cards. An example of data that EMV compliant

[‡]`http://www.kpmg.nl/`

applets expose are the two Card risk management Data Object Lists (CDOL) that should be used by terminals during a transaction.

Since this type of information is also needed in the process of reverse engineering the e-dentifier, it was deemed useful to start off with something relatively easy in order to get the feeling of how EMV applets expose their data. It also provided an opportunity to work with the different libraries that are needed when programming and get a feeling for how they work.

In retrospect this has been a good idea, since two major problems were encountered when developing this particular application. These problems could then be avoided when developing the applications that helped reverse engineer the e-dentifier.

3.3.1 TLV structures

A prime example of why it is good to start with such a relatively easy project came through the use of the javacardx.framework library provided by Sun. This library is an extension to the Java Card 2.2.2 API[§], and provides a library that allows support in creating Java Card applets. This library contains the javacardx.framework.tlv package, which is a package that provides help with interpreting raw data that is stored on a Java Card. This data is stored in a Tag-Length-Value (TLV) format, in which the tag part indicates what data can be found in that particular structure, and where the length part indicates the size of the value part. Both the tag and length part of this structure can vary in size, and how this should be interpreted is described in both the EMV specifications [9] as well as the standard it was taken from (ISO 8825). Apart from this complication, a TLV data element can contain other TLV elements, which means using an existing library is favoured over hand-writing a parser.

Sadly, the javacardx.framework.tlv package is meant to be used on a java card applet, and not a terminal application. It did not seem to work properly at all for a terminal application. Every time a TLV or Tag object was to be created (for both the Constructed and Primitive kinds), it would always return null, even if it was 100% sure that the data was in a proper TLV format. Even if it had worked, the library is extremely limited, as it does not look for TLV structures within a given TLV object and mandates the use of a find/findnext method in order to determine whether a given TLV exists within another TLV.

Luckily there are other libraries available for smart cards, such as the OpenCard Framework[¶]. This framework also provides a series of objects that help with parsing TLV structures, and it is much more powerful than the Java Card TLV support. For example, it does not provide two different objects for primitive TLVs (a TLV that does not contain other TLVs) and constructed TLVs, but

[§]`http://java.sun.com/javacard/`

[¶]`http://www.openscdp.org/ocf/`

instead has a collection of children and a flag that indicates whether or not a particular TLV is a constructed or primitive TLV.

3.3.2 Incompatible Java card T=0 library

Another example of why starting with the relatively easy project was a good idea came from a strange design decision in the EMV specifications, namely the definition of the *GET RESPONSE* command which can be found in book 1 section 9.3.1.3 of the EMV specifications [7]. As has previously been established, the EMV specifications build upon the ISO 7816 standard, which means a lot of elements of that standard have been used in the EMV specifications. Among others, EMV specifies two possible protocols for the data link layer: T=0 and T=1. The *GET RESPONSE* command is used in the T=0 protocol, and leads to the following sequence of events when the APDUs are correct:

1. The terminal sends the APDU to the card.
2. The card responds with a procedure byte and possibly a second byte.
3. The terminal sends another APDU to the card. The content of this APDU depends on the procedure byte.
4. The card returns the final response.

Since this is done on the data link layer, many libraries (including the one provided by Sun in the javax.smartcardio library) provide an abstraction layer for these exchanges, so someone using such a library does not have to worry whether the ICC uses the T=0 or T=1 protocol. This leads to programmers only seeing exchange 1 and 4: you tell the library to send an APDU and you get the response. A programmer using such a library will never see step 2 and 3 occurring unless looking for it in the implementation.

The strange design decision of EMV lies in the interpretation of the procedure byte that is returned in step 2. In most cases it does not expose any problems, but when the procedure byte is equal to a hexadecimal value of '61', the terminal is supposed to send a *GET RESPONSE* command. A *GET RESPONSE* command is defined statically in the EMV standard with a CLA of '00', an INS of 'C0', P1 == P2 == '00' and an expected length byte as indicated by the card in the second possible byte. However, in ISO 7816, the *GET RESPONSE* command is defined as to use the CLA of the *previous* command, and *not* the static value of '00'.

Since most libraries (including the javax.smartcardio library provided by Sun) follow the ISO 7816 standard, this can lead to problems. Since many of the commands in the EMV standard have a CLA byte of '00' this is usually not a problem.

However, when one of the other commands (such as the *GET PROCESSING OPTIONS* command that uses a CLA byte of ’80’) is used, this can (and will!) lead to problems: the CLA byte is copied from the previous command APDU, and since this CLA was ’80’ the resulting APDU will be ’**80**C000XX’, where XX is the expected length specified after the procedure byte. Since the card expects a *GET RESPONSE* instruction to be of the form ’**00**C000XX’, it will not understand this APDU and respond with a ’6D00’ error message: INS not supported.

Luckily this problem could be solved because the source code of the JDK is available. The fix could be applied, and since the JDK makes it possible to load this fixed class instead of the normal class, it could be used for the development of this application information program and the software-based e-dentifier.

3.4 The software-based e-dentifier

Since the passive interface did not work properly, reverse engineering the protocol used by the e-dentifier had to be done in an alternative manner. Two applications were developed: one to simulate the e-dentifier (called the software-based e-dentifier) and one to simulate an ABN-AMRO bank card called the bank card simulator.

The bank card simulator was developed for a JCOP 41 card with 72k EEPROM memory, and was developed using the JCOP tool set. This application simply responds to known requests using predefined responses, and logs the commands (in a raw APDU format) it does not recognise that are sent before giving an error message. The unknown requests can be retrieved from the card using a single command, allowing an overview of what the e-dentifier sends to real bank cards.

The software-based e-dentifier is a Java application that acts as a terminal, and was integrated into the earlier developed application that extracts application information from EMV-compatible bank cards. It can generate most of the e-dentifier responses that can be used on the ABN-AMRO Internet banking website, and thus can emulate the e-dentifier device.

3.4.1 How the protocol was reverse engineered

Reverse engineering the protocol was quite easy, even though it was a bit tedious. It starts with getting a command from the e-dentifier by inserting the bank card simulator into the e-dentifier. Initially the bank card simulator had no responses available to any commands, so the first command sent by the e-dentifier will be logged, after which the bank card simulator gives an error and the communication will stop. This command was then extracted from the bank card and added to the software-based e-dentifier.

The next step was to see how a real bank card responds to this command of the e-dentifier, so a real bank card was inserted into a card reader and the software-based e-dentifier executed the currently known sequence of commands. The responses of the bank card were then extracted and added to the bank card simulator, which would then be used to collect the next command sent by the e-dentifier. This process was repeated until the complete protocol was known.

The bank card that was used for this operation was an ABN-AMRO *Wereld-pas*, which is a bank card that is usable in nearly all countries because of the affiliation it has with MasterCard. Because of this affiliation it contains the MAESTRO logo. The bank card is not very new, and is scheduled to expire in April 2012.

3.4.2 Analysing the protocols used by the e-dentifier

Figure 3.3 is a detailed description of the protocol as it is used by the e-dentifier. Note that it first tries to select a variety of applications (most of these were found by searching for the AID on Google, others were present on a bank card and thus provided an application label):

- 2FFD (unknown)
- A0000000032010 (VISA Electron)
- A0000000031010 (VISA credit)
- A0000000043060 (MAESTRO)
- A0000000041010 (Mastercard Credit)
- D5280050218002 (unknown)
- A0000000038002 (unknown)
- A0000000048002 (SecureCode Aut)

This sequence of application/file selection commands is *always* executed, even if one of the earlier applications (such as the *MAESTRO* application which is present on the ABN-AMRO bank card used to extract meaningful responses) is successfully selected. Since the *SecureCode Aut* application is always the last one to be selected, the e-dentifier will use that application to perform its Internet banking steps if it is present on the card. Since this application *was* present on the ABN-AMRO bank card, all the experiments listed in this document have been done using the *SecureCode Aut* application unless otherwise specified.

When it has tried to select these applications, it will display the selection menu. The selection menu contains the following items:

1. Log on
2. Send transaction
3. Check account number
4. Check input
5. SecureCode
6. Chip balance
7. Choose Dutch/English language

The first four options are the interesting ones, as they are the ones that are used during Internet banking transactions. However, when looking at the APDUs that will be sent when either of those four selections has received its required input, it will always have the same structure. What this means in practice will not be explained here, but will be explained in subsection 4.2.3.

The e-dentifier has been abbreviated as E, while the card has been abbreviated as C in the protocol overview of figure 3.3. Note that as previously has been stated, the protocol is the same for every option that is selected. The only difference exists in the value of the data field in step 13.

```
01) E => C: SELECT A0000000048002

02) C => E: Selection data.

03) E => C: GET PROCESSING OPTIONS

04) C => E: A template containing at least the Application Interchange
            Profile (AIP) and Application File Locator (AFL).

05) E => C: READ RECORD based on the AFL response. Might be done for
            multiple records.

06) C => E: A record containing the data elements. For the e-dentifier
            only the CDOL1/2 and CAP bit filter are of interest.

07) E => C: GET DATA 9F17, which is a request for the PIN try counter.

08) C => E: The PIN try counter.

09) At this point the e-dentifier requests a PIN entry.

10) E => C: VERIFY command with the entered PIN code.

11) C => E: 63Cx, where x is the number of PIN tries left, or 9000.

12) At this point the e-dentifier will request a challenge given by the
    Internet banking application. In case option 1: log-in has been
    selected, it will use a challenge value of 0000 0000. In case
    another option is selected, the challenge will be encoded.

13) E => C: GENERATE APPLICATION CRYPTOGRAM, requesting an ARQC with
            the (encoded) challenge in the data field.

14) C => E: ARQC: cryptogram type (CID), transaction counter (ATC),
            the cryptogram (AC) (which is a MAC over the transaction data)
            and the issuer application data (IAD).

15) At this point the e-dentifier can calculate the final response with
    the bit filter from step 06 and the ARQC from step 14.

16) E => C: Another GENERATE APPLICATION CRYPTOGRAM, this time requesting
            an AAC.

17) C => E: AAC: same structure as the ARQC.
```

Figure 3.3: The e-dentifier protocol

This protocol contains three steps that are particularly interesting: step 12, step 15 and step 16.

Step 12 - transaction data

Step 12 is normally used to provide transaction data that a card will use in the generation of the cryptograms. The e-dentifier provides a challenge that is generated by the bank and communicated to the user through the Internet banking website. It is likely that this challenge is somehow based on the current transaction, such as the amount of money involved, the target bank account number and a transaction counter, but this is currently unknown. What is interesting is that for the log-in procedure the user does not need to enter a random challenge displayed by the Internet banking website. This *was* the case with the old version of the e-dentifier and with the CAP readers described in [5], and because of this change it will be much easier for a user to distinguish between logging in to the ABN-AMRO Internet banking website and actually authorising a transaction.

Step 15 - Applying the bit filter

The second interesting step is also the most technical one. In this step the e-dentifier will calculate the response it will show on the display. It does by selecting the bits defined by the bit filter (which was returned by the card in step 06) from the ARQC. As an example, take the bit filter and the ARQC from an actual trace taken from a real bank card and e-dentifier as shown in figure 3.4:

```
           CID ATC  AC               IAD
ARQC:      80  0042 C14D71DBAFA79FED 0012A5000302000000000000000000000000FF
Bitfilter: 00  007F FFFFE00000000000 00

                   CID      ATC              AC
Binary ARQC:       10000000 0000000001000010 1100000101001101011100...
Binary bit filter: 00000000 0000000001111111 1111111111111111111000...
```

Figure 3.4: Applying the bit filter (additional trailing zeroes have been removed)

In this example the bit filter is shorter than the ARQC, and in order to make them the same size it is padded with zeroes at the end. In order to extract the response, the bit filter is applied to each individual bit of the ARQC by looking at the corresponding bit in the bit filter:

- If the bit in the bit filter is a 0, remove the bit in the ARQC.
- If the bit in the bit filter is a 1, preserve the bit in the ARQC.

Applying this filter leads to the following bits being preserved (note that trailing zeroes and ARQC values have been removed for brevity in figure 3.4): 10000101100000101001101011. Converting this binary value to a decimal value leads to the value displayed on the e-dentifier as the response: 34998891.

Even though this example was taken from a session with an ABN-AMRO bank card, cards from other banks that have been examined have shown to use a different bit filter. An important question that remains is whether or not the chosen bit filter is a good choice.

Of course the best course of action would be to have the user fill in the complete response to the *GENERATE AC* command, as the bank can then verify with 100% certainty that a transaction is correct. However, for user interface considerations this is not a good idea, as a lot of characters would have to be entered by a user even if the binary result is transformed into a hexadecimal representation. In order to accommodate this, a selection has to be made, and this is done through the bit filter.

A good bit filter should select a large part of the AC, as this is the result of the MAC algorithm being applied to the provided transaction data and card data. If enough bits of the AC are selected it would be very hard for an attacker that doesn't know the key to the MAC algorithm to find a different input that leads to the same output for the part that is selected by the bit filter (and thus find a collision that might be usable in an attack).

However, even though the bank knows most of the input provided to the MAC algorithm, not everything is known. The card also provides some input in the form of a transaction counter, which means the bank has to know this input as well in order to verify that the input specified by the user is correct. In the case of the ABN-AMRO, the bit filter selects 7 bits of the transaction counter, meaning it will know the transaction counter up to the value of 128. Due to the fact that it is not possible to use older responses that have not been entered during an Internet banking transaction, it seems obvious that the bank stores the last known transaction counter for a particular card. Since the transaction counter can only go up, the bank should be able to guess whether or not the transaction counter has "overflown" past 128 (and if needed past the next bit as well), and can thus simply add this bit to its own internal MAC algorithm to see if the (partially) provided cryptogram is correct.

To conclude, the bit filter chosen by the ABN-AMRO seems very sensible, as it selects plenty of information from what it does not know (the transaction counter) and what it should know to authorise the transaction (a large part of the generated cryptogram). However, it will be interesting to look at how a new bank card behaves when its transaction counter is raised past 128 without actually logging in to its Internet banking application, as the Internet banking application then has not yet stored a transaction counter. It will be interesting to see whether or not codes that have been generated earlier can then be used, even if only for log in purposes. This question is sadly out of the scope of the research project, and is suggested as a possibility for future work in Section 5.

Step 16 - Finishing the EMV transaction

Step 16 is only interesting because of its meaning: why does this step occur if the final output can already be determined, and any subsequent data that is received will be discarded? Generating this AAC *should* occur according to the EMV specifications, but is not needed in the context of the e-dentifier. The theory posed by Drimer et. al. in [5] is likely: to remain compatible with EMV, and to prevent cards from locking up due to too many failed (consecutive) attempts the transaction is ”aborted” and the card can reset its security parameters.

Comparing EMV definitions and the actual protocol

Now that the different steps of the protocol are known, it can be compared to how the EMV standard has defined the possible steps and what the e-dentifier has actually implemented. In section 3.1 the definitions have been given by the EMV standard, and figure 3.2 is relevant in particular. Removing the unneeded steps from the original image leads to the image as shown in figure 3.5.

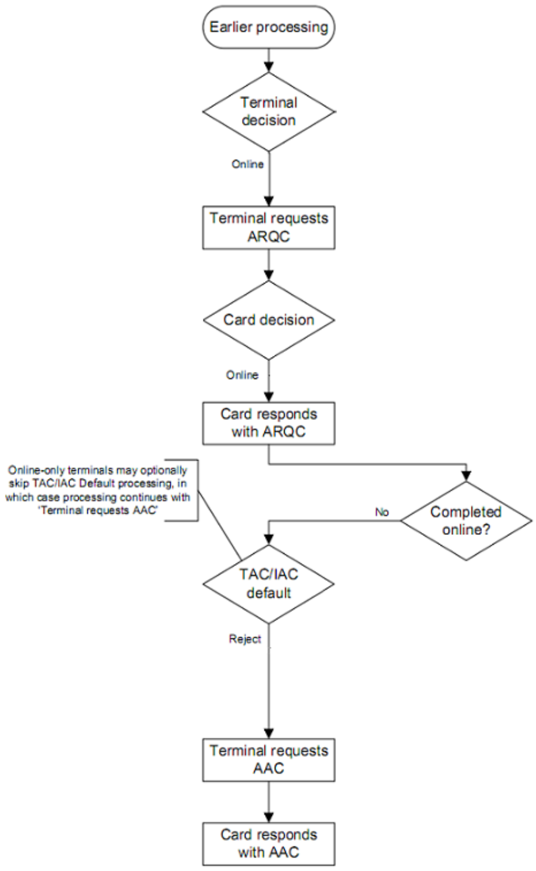


Figure 3.5: Online/offline e-dentifier

3.4.3 APDU data values in the GENERATE AC command

Step 12 of figure 3.3 has already been described in the previous subsection, but what has not been described is what happens to the challenge entered into the e-dentifier. When not using the log-in procedure, a challenge must be entered and while in [5] the challenge was literally copied into a particular section of the data field, something else occurs when using the e-dentifier.

Instead of copying the challenge directly, it is encoded before being used in the appropriate data field. When using the *send transaction* option, this leads to the APDU contents as shown in figure 3.6.

```
Challenge entered with send transaction: 2466 1140

Header:        80AE80002B
Data:          00000000000000000000000000008000000000000000000000
Data (cont.): 661D7D5934000000000000000000000010002
Le:            00
================================================================
Challenge entered with send transaction: 1234 5678

Header:        80AE80002B
Data:          00000000000000000000000000008000000000000000000000
Data (cont.): EFD22EDB34000000000000000000000010002
Le:            00
```

Figure 3.6: Two APDUs for the GENERATE AC command - send transaction

When comparing the APDU structures in figure 3.6 to the one in [5], four differences can be found:

1. The challenge is encoded somehow.
2. The (encoded) challenge is somewhere near the middle, and not at the end.
3. The challenge is followed by the (static) value of '34'.
4. The static value of '010002' can always be found at the end of the data field.

In order to determine what this means, one should look at what *should* be in the data field by looking at the CDOL 1 value sent earlier by the card. This object list determines what the terminal should send to the card in a *GENERATE AC* command's data field (see section 2.2.2 for the details on this). Table 3.1 lists the CDOL1 value of the card used in the same session, and how it relates to the values provided by the e-dentifier in the *GENERATE AC* command.

With this information a proper comparison can be made with [5]. Even though the position has changed, the unpredictable number field is still used to hold the

```
CDOL 1: 9F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403
```

Tag	Meaning	Length	Value in transaction
9F02	Amount, Authorised (Numeric)	06	000000000000
9F03	Amount, Other (Numeric)	06	000000000000
9F1A	Terminal Country Code	02	0000
95	Terminal Verification Results	05	8000000000
5F2A	Transaction Currency Code	02	0000
9A	Transaction Date	03	000000
9C	Transaction Type	01	00
9F37	Unpredictable Number	04	661D7D59
9F35	Terminal Type	01	34
9F45	Data Authentication Code	02	0000
9F4C	ICC Dynamic Number	08	0000000000000000
9F34	Cardholder Verification Method Results	03	010002

Table 3.1: CDOL 1 value and corresponding data in GENERATE AC command

challenge. Dutch bank cards also request a terminal type and the e-dentifier sets this to the value of '34', which signifies an unattended online-only terminal according to Annex A of Book 4 of the EMV specifications. The final difference lies in the card requesting the Cardholder Verification Method (CVM) results, and Annex A4 of Book 4 of the EMV specifications [10] (which references Annex C3 of Book 3 [9]) specify how this should be interpreted.

The first byte signifies which CVM has been performed, and according to table 39 in Book 3 of the EMV specifications, the value of '01' value means *Plaintext PIN verification performed by ICC.*

The second byte indicates the condition codes that were applicable during cardholder verification (such as a minimum transaction value before CVM has to be performed). A value of '00' signifies that CVM always has to be performed.

The third and last byte is not clearly specified in Book 3, but Book 4 Annex A4 defines three possible values:

- 0 = Unknown, for example when using a signature
- 1 = Failed, for example for offline PIN
- 2 = Successful, for example for offline PIN

3.4.4 Encoding of the challenge

Since three out of four differences listed between the APDU values found during this research and the ones found in [5] have already been explained in section 4.2.3, there is only one difference remaining that needs to be explained: how and why are they encoded?

How is it encoded?

Even though the size of the encoded challenges are always the same due to fact that they are placed in the unpredictable number field (which has a static length of 4 bytes), the input entered into the e-dentifier leading to those encoded challenges can differ. For example, when selecting the *send transaction* option, the e-dentifier expects a challenge of up to 8 different numbers. When selecting the *Verify account* or *Check input* option, an input can be entered of up to 36 numbers. Entering the same value for different options leads to different encodings.

Due to this variety, the fact that entering the same value for different options leads to different values, and the fixed length output it is likely that some sort of hash function is applied, and that the selected option is somehow concatenated to the input of this hash function to make sure it can be distinguished from the other options. It is also likely that the output of this hash function is not 4 bytes, which means a selection of bits is made from the output to be used in the unpredictable number field. How this is done is currently unknown, and is one of the suggestions made for future work in chapter 5.

Without knowing the details, figure 3.7 gives an abstract overview on how this is likely done. Note that it is likely that an existing cryptographic hash function has been used such as SHA-1. The bit filter also has no influence on the selection function S, as changing this bit filter does not change the encoded values. An example overview of some of the encodings can be found in appendix B.

```
E = Encoded challenge
x = challenge entered into e-dentifier
H = hash function
y = unique identifier of the option selected in the e-dentifier menu
| = symbol representing concatenation
S = selection function that selects 4 bytes of the output

E = S(H(y|x))
```

Figure 3.7: How the challenge might be encoded

Why is it encoded?

Why the challenge is encoded is more easily answered than *how* it is encoded. One of the vulnerabilities found in [5] was the fact that the unpredictable number field is overloaded with the challenge, and that in different modes (called *sign* and *respond* mode in [5], here it is called a *log-in*, *send transaction*, *check account nr.* or *check input* selection) you could select the input in such a way that it would be valid for another mode. This is no longer possible, as the log-in procedure should be the only one that sends an encoded challenge of '00000000' and the same input for different options leading to different encoded challenges. With this method the only concern a bank has in terms of being able to use input from one option for another option is when collisions occur, but the chances of this being an exploitable vulnerability are low since the bank chooses the challenge.

3.4.5 The semantics of the ACs in the e-dentifier

Figure 2.1 contains the semantic definitions of the different Application Cryptograms in the EMV specification, but in the e-dentifier their definitions are different. This is because they are no longer used as proofs that a transaction is accepted or rejected, but instead are used to provide proof of ownership of the proper bank card to an Internet banking application, as well as intent of performing a transaction. This is especially apparent in the fact that an ARQC cannot be generated if the PIN verification step has been skipped. This *is* possible according to the EMV standard, but when this is tried on the SecureCode Aut application (which is the one used for Internet banking) the card will instead reject this request and generate an AAC instead.

Knowing this, the semantics for the e-dentifier can be defined for the different application cryptogram types as they are defined in table 3.2.

AC type	**Semantics**
TC	Not used, and therefore does not have any meaning any more.
AAC	Primarily used to make sure the card does not lock up to complete the EMV compliant transaction, but can also indicate the PIN verification step has not been performed and the transaction is rejected by the card.
ARQC	Used to prove ownership of the card and transaction intent to an Internet banking application, who can deduce the relevant ARQC information based on response shown by the e-dentifier.

Table 3.2: Cryptogram semantics for the e-dentifier

3.4.6 Usefulness of the simulators

Now that the protocols used by the e-dentifier are known, it can be debated on how useful the final resulting applications are. Obviously the software-based e-dentifier is currently more useful than the bank card simulator, as it can be used to get correct responses for Internet banking applications from real bank cards.

In its current state the bank card simulator *can* be used, but it can only simulate a single instance of the protocol. What this means in practice is that even though it can successfully convince an e-dentifier that it is a real bank card, it will always produce the same responses, as the responses are based on the ARQC that is produced by the card. Since this ARQC includes a MAC over a transaction counter, and the secret key used by real bank cards to generate this MAC is not known, it can only repeat the correct responses it does know. Until it is possible to extract the secret keys from a real bank card the usefulness of the bank card simulator is limited.

3.5 Overview of Dutch banks

Since the ABN-AMRO bank is only one of the major banks in the Netherlands, it is interesting to see how other Dutch banks have implemented their challenge/response devices and applets on their bank cards. Figure 3.8 gives an overview of what this research considers the most interesting: what a bank card requests from a terminal through its CDOL 1 data object list, and which data is selected from a generated cryptogram through the bit filter.

```
CDOL 1:
ABN:  9F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403
ING:  9F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403
RABO: 9F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403
SNS:  9F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403

Bit filters:
      CID ATC  AC               IAD
ARQC: 80  0042 C14D71DBAFA79FED 0012A50003020000000000000000000000FF
ABN:  00  007F FFFFE00000000000 00
ING:  0F  0000 7FFFFFE000000000 00
RABO: 07  0000 7FFFFF0000000000 00
SNS:  07  0000 7FFFFF0000000000 00
```

Figure 3.8: CDOL 1 and bit filters for different Dutch cards

3.5.1 ING bank

What is interesting about the ING is that while the ING bank provides the possibility for Internet banking, it does not use a challenge/response device in combination with a bank card to do this. Instead it has a user name/password combination to log in to the web site, and offers its customers two possibilities when performing a transaction:

1. Use a predetermined sheet of response codes (so called TAN codes) which have to be entered to confirm a transaction. Each code is only usable once.

2. Have the ING bank send the TAN code through an SMS to the mobile phone numbers specified by the customer when a transaction needs to be confirmed.

Even though the ING does not use EMV for Internet banking, bank cards from the ING still contain EMV applets such as MAESTRO and even the SecureCode Aut applet which is an EMV CAP applet that can be used in Internet banking applications. Both the MAESTRO and SecureCode Aut applet have the same CDOL 1 as the ABN-AMRO card, which means they request the cardholder verification method results as input to the cryptogram generation. Because they do this, the man-in-the-middle attack as described in [12] to perform transaction without PIN verification should not be possible (assuming the back-end of the ING bank properly verifies the transaction data, see Section 4.1 for details on this analysis).

CDOL and bit filter

What *is* different for ING banks, is the bit filter exposed by the SecureCode Aut applet. This bit filter does not select anything from the transaction counter, which means that a theoretical ING Internet banking application will have to brute force the transaction counter in order to be able to calculate the cryptogram. It also selects the four least significant bits of the CID, which in all EMV cases will be four zeroes making it quite useless information to select.

3.5.2 Rabobank

The Rabobank has an Internet banking application that uses a challenge/response device called the *random reader*. The protocol that the Rabobank random reader uses is identical to the one used by ABN-AMRO (see figure 3.3). However, there are some differences in the applications it tries to select, and also in the way the challenge that has to be input from the Internet banking application is processed.

How the Rabobank random reader works

The Rabobank random reader is slightly different from the ABN-AMRO e-dentifier. Instead of providing a menu on a large display, it uses a small single-line display. There is a button labelled l to start the log-in procedure, and a button labelled s to start the send transaction procedure. Both prompt the user for a PIN, and if the PIN verification is successful for the log-in procedure the display will show the response that needs to be entered into the Internet banking application. If the send transaction procedure has been selected the display will request up to two challenges, of which the second challenge can usually be skipped by pressing the *OK* button.

Challenges entered into the Rabobank random reader are usually 10 digits in size, which is more than the 8 digits usually requested by the ABN-AMRO e-dentifier.

Application selection

The Rabobank random reader first executes the following commands:

- BCA40000022901 (?)
- 00A4040007A0000000031010 (select VISA credit)
- 00A4040007A0000000032010 (select VISA electron)
- 00A4040007A0000000043060 (select MAESTRO)
- 00A4040007A0000000041010 (select MasterCard Credit)
- BCA40000022F00 (?)
- BCA40000022FFD (?)
- 00A4040007A0000000038002 (select V i s a R e m A u t h e n)
- 00A4040007A0000000048002 (select SecureCode Aut)

Apart from the obvious application selection commands, it tries to have the card execute several strange, unknown commands. While the e-dentifier tries to select a strange application or file (2FFD) the Rabobank random reader tries to execute a completely other type of command. What this command should do is currently unknown, as bank cards give error codes when attempting to execute these commands.

Alternative challenge processing

The biggest difference between the e-dentifier and the Rabobank random reader can be found in how they process the challenges given by the Internet banking website. The previous sections have established that the ABN-AMRO uses the static value of 0000 0000 as the challenge code when logging in, while the challenge entered to, for example, send a transaction is encoded and is transmitted to the card in the unpredictable number field for the card to be included in the MAC.

The Rabobank random reader *always* sends a static value of 0000 0000 in the unpredictable number field, even when selecting the *send transaction* option. This is because the *GENERATE AC* command that is sent to the card to generate an ARQC is sent *before* the challenge is entered. When using the bank card simulator to make sure the response to the Rabobank random reader is always the same, it was discovered that entering a different challenge leads to a different response.

What this means in practice is that the challenge that is entered is somehow used to scramble the bit-filtered result. How this is done is currently unknown, but given that similar inputs lead to such similar outputs (see appendix C for some sample challenges and its given output for the bank card simulator) it is likely that it is not extremely hard to reverse engineer.

Another consequence of this choice is that the problem of type confusion introduced in [5] is even more obvious for the Rabobank random reader. Since the challenge is not used in the generation of the cryptogram at all by the bank card, proper responses collected through the log in procedure of the Rabobank random reader can also be used to perform transactions once it is clear how the challenge influences the output of the Rabobank random reader. See Chapter 4 for the detailed analysis of this problem.

Getting the PIN try counter

As a final note, if the GET DATA for the PIN try counter gives an error code, the Rabobank random reader just continues anyway. It is likely that the result of this command is completely disregarded, since the amount of PIN tries that are left will be returned by the card if an invalid PIN is entered anyway.

Different bit filter for different Rabo cards

Figure 3.8 has two entries for the Rabo bank. This is because one particular card specified the second bit filter, while other cards exposed the first one. The card with the second bit filter was an extremely new card, and had never been used yet for both Internet banking transactions or payments at regular terminals.

3.5.3 SNS bank

The SNS bank also uses a challenge/response device called the *Digipas*, but this device is fundamentally different to the one used by the ABN-AMRO and Rabobank.

The first major difference is that every individual device is linked to the bank account of the owner, making it impossible for the device to be lent to another customer. This device contains a code that is unique for every customer, and is linked to the bank account customer. This is signified at the log-in process, in which the customer can choose to use a user name and password combination to log in to the Internet banking application, but can also choose to log in using this unique code found on the challenge/response device. When entering this code the customer still has to verify ownership of the device through PIN verification, by entering a challenge and typing in the response on the Internet banking website.

The second major difference is that the SNS Digipas does not allow a bank card to be inserted. Instead, the customer has to enter the challenges posed by the Internet banking website into the device, which will then generate the correct response assuming the PIN verification was successful. This means that the customer's bank card is not used to sign transactions, which consequently means that investigating the SNS Digipas falls out of the scope of this research project.

Whether or not the SNS Digipas *does* use elements from the EMV specifications internally is unknown. What can be deduced is that it is extremely likely that the Digipas contains some sort of secret tied to the account of the customer. It is therefore suggested that the SNS Digipas is investigated in a future research project, as stated in Section 5.

3.5.4 ABN-AMRO - the old e-dentifier

As a final comparison, the old e-dentifier was also looked at to see if there were any noticeable differences in the protocol. However, problems were encountered when trying to reverse engineer this device, as even though it first tries to select the SecureCode Aut application, a correct response to this selection causes the old e-dentifier to lock up and stop sending commands. However, this was because the JCOP 41 programmable smart card that was used during the project to log instructions sent by the different devices uses the T=1 protocol, and the old ABN-AMRO e-dentifier expects cards to use T=0. Because of this the old e-dentifier could not understand the response sent by the card and would lock up.

Luckily there was also a JCOP 31 programmable smart card available, which still uses the T=0 protocol. However, even though initially the suspicion that it was caused by a difference in the protocols used was confirmed, the card only managed to send one other instruction (a strange *BCA4000002000000* instruction) before locking up once more. A bank card that can be used in the old e-dentifier responds with *6E00 (CLA not supported)*, but when this response is sent by the programmable smart card it will lock up again and stop sending commands.

This new problem could not be solved in the time span of this research project. The only noticeable difference that was found between an actual ABN-AMRO bank card and the programmable JCOP 31 smart card was that the ATR of the JCOP card sets a parameter of *N=-1* whereas the bank card has a value of *N=0.*

3.5.5 Data authentication

So far, sub-research question 2 has only been addressed in terms of protocols. One issue that has not been addressed is the issue of data authentication.

Data authentication deals with the security issue of validating whether or not the data on a specific card is authentic; i.e. whether or not the card was actually issued by its issuer. Without this it would be possible for attackers to create cards themselves with its own data to be used during valid transactions.

The EMV specifications define three different possibilities when dealing with data authentication: Static Data Authentication (SDA), Dynamic Data Authentication (DDA) or Combined DDA/Application Cryptogram Generation (CDA). Chapter 2 deals with the details of the different types of data authentication, while a simple overview of the main differences between the different types of data authentication can be found in Section 2.3 of this document.

Determining which form of data authentication a card supports is easy: the card specifies this through the AIP which is returned in the *GET PROCESSING OPTIONS* command. Annex C1 of Book 3 of the EMV specifications specify how the AIP should be interpreted. Figure 3.9 gives an overview of the AIP values that have been found for the different bank cards that have been investigated. An important note, however, is that all these values were taken from other applets than the SecureCode Aut applet, since this applet always exposes an AIP with the value '1000'.

```
ABN:  3800 == 0011100000000000
ING:  3800 == 0011100000000000
RABO: 3800 == 0011100000000000
SNS:  3800 == 0011100000000000
```

Figure 3.9: Application Interchange Profile for different Dutch bank cards

As can be seen in Figure 3.9 all bank cards that were investigated during this research project have the same AIP, and thus provide the same capabilities. When looking up what those capabilities are the following is found:

- SDA is *not* supported (bit 7 of the leftmost AIP byte)
- DDA *is* supported (bit 6 of the leftmost AIP byte)
- Cardholder verification *is* supported (bit 5 of the leftmost AIP byte)
- Terminal risk management *should* be performed (bit 4 of the leftmost AIP byte)
- Issuer authentication is *not* supported (bit 3 of the leftmost AIP byte)

- CDA is *not* supported (bit 1 of the leftmost AIP byte)

Chapter 4

Exploring potential weaknesses

Now that it is clear how at least the ABN-AMRO has implemented its Internet banking application and how this relates to the EMV specifications, it will be interesting to see if any (exploitable) flaws or weaknesses can be found. This can be related in particular to the situation in the U.K. as it has been described in [4, 5, 12] (of which [5] is the most important one) in order to answer research sub-question 3.

This Chapter looks at these reported vulnerabilities and relates them to the situation for Dutch banks as described in this document, but also elaborates on (new) flaws that have been discovered during this research project.

4.1 Weaknesses reported in the U.K.

There has already been a fair amount of research on the use of EMV in general and its application in Internet banking, and currently published works have mainly focussed on its application in the U.K. Three papers [4, 5, 12] in particular list various vulnerabilities of not only the EMV specifications and implementations, but also the devices used for Internet banking. For each of these papers the situation in the U.K. will be compared to the one described here in previous sections, and an analysis will be made to see if the implementations in the Netherlands suffer from the same listed vulnerabilities and flaws.

This Section is divided into three Subsections, one relating to each paper. Every section contains a description of the flaws/weaknesses (labelled as *Attack description*), the defences proposed in that particular paper (labelled as *Proposed defences*) and how this particular weakness relates to the situation found in the Netherlands as described in the rest of this document (labelled as *Comparing the U.K. with the Netherlands*).

4.1.1 Relay attack [4]

This flaw does not directly relate to Internet banking, but is instead an attack on EMV applications in general. In Internet banking applications the landscape is drastically different compared to "normal" EMV applications, which usually occurs in a store at a terminal that is not controlled by the owner of the bank card. Internet banking is usually done in a more controlled environment, since the device used for Internet banking is owned and controlled by the owner of the bank card.

Since this research project focussed mainly on EMV in the context of Internet banking and not EMV applications in general, the scope of this research and the paper describing the relay attack is different. However, due to nature of the vulnerability and the knowledge on the subject matter hat has been gained during this research project an educated opinion can be formed on how this relates to the situation in the Netherlands.

Attack description

Instead of performing a regular EMV transaction at a terminal, an attacker instead uses a fake terminal that interacts with the card of the victim. This fake terminal relays the communication to a second attacker that has a counterfeit card, and this counterfeit card is used at a real terminal. The fake terminal is in a shop that sells low-value goods (such as a terminal for a diner or a flower shop), while the second attacker to which the communication is relayed attempts to buy something more expensive (such as a new TV or jewelry). By relaying the signal from the fake terminal to the counterfeit card, and by relaying the responses from the real terminal back to the real card through the counterfeit card and fake terminal, an unsuspecting victim will think he or she is paying for something he or she is not. Instead of buying a cheap bouquet of flowers, the victim will actually buy an expensive diamond ring without being able to tell the difference.

Proposed defences

In [4] several proposed defences have already been elaborated upon, along with why they will most likely not be a cost-effective measure of defence. Instead they propose to implement a distance bounding protocol, which is an additional step into the existing protocols during which a terminal attempts to estimate the physical distance between the card and the terminal itself. When adding this estimation, cards that do not respond fast enough or make too many errors during this response time will be rejected, as they are likely being used in a relay attack.

Even though this distance bounding protocol is relatively simple to both understand and implement, it would still require an addition to the EMV specifications, as well as alterations to all existing card implementations and terminals. While most existing terminals will allow for this addition to be made with a software update, it is inevitable that cards will have to be replaced, as their tamper-resistant nature makes it easier to replace them than to update them. As they seem to have a lot of focus at the costs of the hardware that is needed, and disregard the need for changes to be made to the specifications as well as the need to replace existing cards, we believe it to be infeasible as a short-term solution.

As a long term solution, however, it can be feasible as it provides a solution on the technical level, which means nothing has to change in the existing procedures for a customer who wishes to make a purchase and the merchant operating the terminal.

One thing that is disregarded is that the proposed solution is that it is a solution to a *symptom* of the underlying problem. The actual problem lies in the fact that the customer has no foolproof way of verifying that the transaction displayed at the terminal is the actual transaction that is being authorised by the card. Other solutions, such as providing customer-owned trusted devices (called the *electronic attorney*) that can mediate between a card and a terminal to display the actual transaction data are disregarded as being too expensive, too complex and perhaps not approved by banks.

As an example that this is merely the solution to a symptom of the problem consider a fake terminal that is connected to a nearby laptop. This laptop runs a special program that automatically logs in to a victim's Internet banking website and transfers all the money from the victim's account to an account of the attacker. The distance bounding protocol will not help in this case, as the laptop is extremely close to the victim's card.

Comparing the U.K. with the Netherlands

Relay attacks are hard to counter, and since this particular attack does not depend on a particular setting or mode in which EMV is used it is likely that this attack can be used in the Netherlands. In fact, the authors of the paper describing this attack have been in the Netherlands to demonstrate it for a TV program called *Goudzoekers* [*]. This attack was performed less than a year ago at the time of writing this document, and it is unlikely that this problem has been (temporarily) fixed in the meantime.

[*]Footage available at `http://weblogs.vpro.nl/goudzoekers/2009/12/14/afl-5-het-nieuwe-pinnen/`

4.1.2 Bypassing PIN verification [12]

Out of the three papers this section focusses on this weakness appears to be most critical one, as it allows for an attacker to freely use a bank card to which he does not know the PIN. Being able to steal a bank card and use it freely without having to know the PIN code would be considered the jackpot for criminals. In the light of Internet banking, it would be especially bad as it allows for a much larger amount of control of the attacker.

Attack description

The proposed attack is extremely simple, especially when looking at figure 3.3. It would also be the "holy grail" for criminals, as it allows them to simply steal a bank card and use it freely without having to know the PIN belonging to that bank card.

By using a man-in-the-middle (MitM) attack message number 10 can be suppressed, thus skipping the PIN verification step. The MitM would simply intercept this message, discard it and return status code '9000' to pretend that the card accepted the PIN. Every other message in the protocol is still sent, and since the EMV specifications allow for this to happen (some terminals do not have a PIN pad, or allow for a transaction to be performed by verifying a physical signature) the card does not sound any alarm bells.

The EMV specifications accounted for this possibility by keeping track of the transaction through several lists maintained by the card and the terminal, such as the Terminal Verification Results (TVR) and the Issuer Application Data (IAD). However, when combining these two data lists it does not provide a complete overview of what has occurred during the transaction. For example, the TVR does not specify that PIN verification *has* occurred, but merely it has failed at one point for what reason. Even though the IAD often does specify whether PIN verification has occurred or not it is usually not in a public format, which means that the terminal cannot verify it by itself.

Proposed defences

One obvious solution in [12] would be to provide terminals with the ability to parse the IAD. Since this specifies whether or not PIN verification has occurred, the terminal can immediately verify both parties have the same idea on what happened during this transaction. However, the IAD is usually in a propriety format, and equipping all the terminals with this ability would require banks and terminal vendors to establish a common format. Even if this has happened, the IAD can still be changed by the MitM, and only if CDA was used (in which a signature would be added over the IAD (among other data) to help detect this change) would help prevent this in offline situations.

Another, much more feasible solution that has been proposed was to add the Cardholder Verification Method Results (CVRM) be added to the card's CDOL. Doing this, the card would receive what the terminal believed to be the cardholder verification method as input to the MAC algorithm, thus allowing both the card and the issuer to detect the MitM attack. Of course the issuer still needs to verify both the data it gets from the card and terminal, and reject the transaction if the terminal believes PIN verification was performed while the card reports it has not done so.

Comparing the U.K. with the Netherlands - Internet banking

As previously indicated, this particular attack would be a powerful tool to a criminal if it could work with Internet banking. Especially for banks such as the ABN-AMRO and the Rabobank, which rely exclusively on the card-reader devices to provide the log-in and transaction tokens this would be a nightmare. Having a user name/password to prevent an attacker from logging in to the Internet banking application is an option, but that can be considered a very weak line of defence as they can relatively easily be extracted through other means. However, this would mean attacks would have to be targeted at individuals (in particular those of which the log in name and password are known), and could not be massively exploited through, for example, fake terminals.

Since the Dutch banks all seems to use SecureCode Aut as the application of choice for Internet banking, attempting to perform this attack on this application should be enough to see if it works. Luckily it does not work, at least not for the cards that have been used during this research project. If the PIN verification step has not been performed when the first *GENERATE AC* command is used (step 13 in figure 3.3) the card will not return an ARQC, but will instead return an ACC indicating it has rejected the transaction.

Comparing the U.K. with the Netherlands - regular terminals

Since the attack does not seem to work with the SecureCode Aut application which is used for Internet banking, it is interesting to see what happens with other EMV applications present on Dutch bank cards, such as the MAESTRO application.

Even though these applications *do* allow the request of an ARQC if the PIN verification step has been bypassed, the attack is still detectable through the value requested in the CDOL 1. Table 3.1 gives an overview of the CDOL 1 values that have been found in all Dutch bank cards that have been investigated (see Figure 3.8 for the complete overview), and this indicates the card requests the CVRM of the terminal as transaction data input to the MAC algorithm. If a MitM would change this value, the MAC would be incorrect, and if the values are correct it will allow both the card and the issuer to verify that the attack has occurred.

Of course if the banks do not properly verify these results the attack might still be possible, but should such an attack occur it is at least possible to look at the "paper trail" and confirm that it has occurred. In terms of liability this is extremely positive for a victim, which is one of the more important warnings given in [12].

As a final note on this, the IADs that have been returned by cards during this research project were in a propriety format that could not be parsed. However, the IAD would be consistent during different transactions, and bypassing the PIN verification step would always return a different IAD, indicating that the card *does* send this value to the terminal.

4.1.3 Optimised to fail [5]

The third paper that illustrated weaknesses in the situation in the U.K. relates to the devices used for Internet banking. In [5] these devices are called CAP devices, as they are based on a (closed) standard that builds upon the EMV specifications called the Chip Authentication Program, or CAP in short. Several weaknesses have been found with these devices, and because the Dutch situation in terms of protocols is almost identical, a comparison can easily be made.

Another aspect that makes these devices easier to analyse and compare is that you are not as dependant on the back-end systems of banks as you are with the flaws described in the two other papers. Because of this you can run tests in a much more controlled environment with fewer risks.

Attack descriptions

The paper does not specify a single attack, but instead lists a whole range of different attacks. Some of these attacks are not technical in nature, but instead are an attack on a broader scale. For example, the first vulnerability is one concerned with mugging: in the past when muggers wanted to make sure the PIN provided by a victim was correct they would take them to an ATM and force them to demonstrate it works. With a CAP device they will run less of a risk of getting caught, as they can simply use the device to verify it is correct in an environment that does not pose such a big risk to them.

However, many of these vulnerabilities will be considered to be out of scope for this research project, mainly because some of the suggestions offered in [5] are already applicable to the situation in the Netherlands. Instead only the technical vulnerabilities, such as protocol weaknesses will be covered in this subsection. They will be numbered so references can be made to them.

Attack 1: No server freshness. There is no server freshness for CAP responses, which means responses can be requested a (long) time before they will actually be used.

Attack 2: Overloading of the Unpredictable Number (UN) field. Because the UN field is used to hold the challenge, the same number might semantically mean something different in a different mode. In [5] an example is given with *respond* and *sign* mode. Because the respond mode always loads the UN field with a value of all zeroes, a zero value transaction created in sign mode can be used as a correct token in respond mode.

Attack 3: Collisions on nonces. NatWest uses a nonce as the first four digits of the respond-mode challenge. However, a malicious terminal can request a large amount of responses from the card with random nonces, and eventually a collision will occur.

Proposed defences

Attack 1: A server provided nonce, even when logging in will help prove the freshness of CAP generated response codes. Doing this, the codes cannot be requested a long time before they are used.

Attack 2: The type confusion create by loading the challenge into the UN field can be prevented by including a response-type flag in the *GENERATE AC* command. This will cause the card to also sign this flag, and thus prevent attackers from being able to use one response in another mode. An alternative is to use an equivalent as used in Germany, in which a prefix is added for the relevant selection, for example the prefix *Account number* when signing an account number.

Attack 3: Because CAP tries to limit the number of characters a user has to type, possibly important information concerning the context of a transaction is omitted in a CAP transaction. A reader that has a bigger display and can be connected to a computer to display the full transaction data can help with this.

Comparing the U.K. with the Netherlands

Luckily, the situation in the Netherlands is better than in the U.K. Even though some issues remain, others are resolved in manner that does not influence the way users interact with the Internet banking website.

Attack 1: This is one issue of the issues that remains in the Dutch versions of CAP devises, but this only applies to the log-in step. It is possible to collect multiple log-in response codes in advance and use them at a later point. However, at least for the ABN AMRO bank the sequence in which they are used is important, as the selection of a part of the transaction counter allows the bank to estimate the current transaction counter of the card. It has been checked and confirmed that correct responses that have a lower transaction counter of the currently last known transaction counter are rejected by the ABN AMRO Internet banking application.

Attack 2: The type confusion present in the U.K. is no longer present when using the ABN-AMRO e-dentifier, as they have developed an encoding scheme that makes sure challenges from different modes (such as logging in and verifying a bank account number) no longer can be used for other modes. Of course the chance of a collision still exists, both for different challenges for the same mode and challenges that can be used for another mode. It seems unlikely that an attacker can exploit this in practice, as it would require the attacker to find a collision for a given challenge.

However, the Rabobank uses a different scheme that is worse than the scheme described in the U.K., as it does not send the challenge from the bank to the card to be signed. Instead it always uses the same value of 0000 0000 in the Unpredictable Number field for both the log-in procedure and the send transaction procedure. Even though the challenge somehow scrambles the final output, an attacker who knows how this scrambling is done can use log-in responses to make payments. When combined with the first attack that is specified here (collecting multiple log-in response codes beforehand) it can lead to an attacker having a wide range of usable responses.

Attack 3: This particular attack is not relevant for the ABN-AMRO, as they use a different scheme than NatWest. It is, however, unknown what the semantics are of the challenges posed by the ABN-AMRO Internet banking website, and thus it cannot be determined whether or not this includes a nonce.

4.2 New and open issues

Even though the Dutch banks generally seem to have created better implementations compared to the implementations in the U.K., there are still some issues that raise questions or can be improved. It is important to know that whenever a test was done on a Dutch Internet banking website this has always been the ABN-AMRO website using the e-dentifier 2.0, as that was the main focus of this research project. Due to time constraints not all Internet banking websites could be investigated.

4.2.1 Collecting valid log-in tokens ahead of time

Logging in to the ABN and Rabobank Internet banking websites does not require the use of a challenge, but instead a single option (log in) has to be selected. The needed response code is immediately shown on the display, and can be used to log in. However, when collecting multiple of these codes, they can be used in sequence after one another, making it possible to collect them before hand.

Even though [5] lists this as a vulnerability, the possibilities of exploiting this are limited, as performing a transaction will already send a new transaction counter to the bank, and thus render the harvested log in responses useless. However, this particular issue is more of a problem for the Rabobank than it is for banks in the U.K.. This is because the Rabobank does not send any challenges to the card for the card to sign. Instead, it always uses the same static value of 0000 0000, which means valid log-in tokens can also be used as valid confirm-transaction tokens assuming it is known how the entered challenge affects the bit-filtered output.

4.2.2 Unknown semantics of challenges

When receiving a challenge to enter into the e-dentifier from the ABN-AMRO Internet banking website, it is currently unknown what those challenges represent. It is likely that they are related to the transaction that is about to occur, but no correlation could be found so far.

It is also currently unknown if sequential challenges are (slightly) correlated to each other. If they are correlated, it might be possible to collect the correct responses beforehand, just like what is possible with the log in procedure. This is of course extremely far-fetched, as it is likely that the bank somehow includes transaction data and nonces. Nonetheless it is important to consider these aspects.

4.2.3 Possibility of collisions in challenges

One of the biggest suggestions that is made for future work based on this research project lies in the encoding of the challenge into the value that is actually sent to the card. Section has elaborated on this, and it is currently the reason why it is only possible to log in to the ABN AMRO Internet banking website using the software base e-dentifier, while it is not possible to perform transactions.

Even though encoding the challenge is a good idea, especially when combined with the fact that different options lead to different encodings thereby fixing the type confusion vulnerability found in [5], it will inevitably lead to collisions for some values as the input size can be much greater than the output size (since, for example, the e-dentifier allows for an input of up to 32 digits when verifying a bank account number). While it is doubtful that this can lead to an easily exploitable vulnerability (for example through once again introducing type confusion), not knowing how this encoding is done leaves this as an open issue.

4.2.4 Not rejecting a PIN-less transaction

Even though Dutch bank cards request the CVMR through its CDOL, cards that have been investigated so far do not seem to parse and validate this information.

If, for example, the MAESTRO application is used on an ABN AMRO card and the man-in-the-middle attack described in [12] is applied to bypass the PIN verification, the card will still receive data from the terminal that indicates PIN verification has been attempted and has been successful. The card should know that this has not been performed (the MitM suppressed that command so the card would not receive this), and instead of returning an ARQC it should return an AAC, rejecting the transaction. While the back-end of the bank would still be able to detect this and reject the transaction (due to the card including the CVMR in the MAC over the transaction data), it would be much safer if the card immediately rejects the transaction instead of doing what it currently does: generate the ARQC and let the bank verify it. If the card immediately detects it and rejects the transaction it would not have to rely on a proper implementation of the back-end system of a bank.

4.2.5 Rabobank: Not having the card sign the challenge

As has already been mentioned in the Subsection on collecting log-in tokens beforehand, the Rabobank has a potentially worse situation than the situation it was compared with in the U.K. in [5]. The type confusion mentioned in this paper is even more obvious for the Rabobank, as it does not send challenges entered into the challenge/response device to the card to be signed. Instead it always uses a static value of 0000 0000, which is the same for the log-in procedure, and if a transaction has to be confirmed through entering a challenge the entered challenge is somehow used to scramble the output received from the card.

This means valid log-in tokens can potentially be used as valid confirm-transaction tokens and the other way around. While this still relies on knowing how the challenge is used to scramble the output from the card, this is relying on a scheme that provides security through obscurity, which is a violation of Kerckhoffs' principle.

Chapter 5

Future work

Not every element that was uncovered during this research project could be investigated. Sometimes it was out of the scope of this research project, while it is also possible it would take too much time to complete along with the rest of the research. This Chapter therefore gives recommendations for future research that can be done based on the research that is presented in this document.

ABN-AMRO challenge encoding

When entering a challenge into the ABN-AMRO e-dentifier the device sends the challenge to the card to be included in the MAC as transaction data. However, the challenge that is entered is not copied directly into the APDU that is used to transmit the challenge. Instead, it is encoded before being transmitted, obviously in an attempt to make it harder to create a software-based e-dentifier that can (for example) be used in a fake terminal.

It is interesting to know that when selecting a different option (such as *Verify account* or *Check input*) the same input will lead to a different encoding. This fact, along with the fact that the encoded output seems to be relatively unrelated to the challenge that was entered into the device suggests that some sort of hashing function is used, in which the selected option is included as input to the hash function.

Appendix B contains a few samples of the selected option, the given input, and their corresponding encoded challenges. For a more detailed description on this particular suggestion for future work, see Section 4.2.3.

Rabobank challenge scrambling

Contrary to what the e-dentifier does, the Rabobank random reader does not send the challenge entered by the user to the card to be used as transaction data. Instead it always sends a static value of 0000 0000, leading to a strong possibility of type confusion as described in Section 4.2. However, the entered challenge is still used in some manner to scramble the output received from the card, which means that without knowing how this scrambling is done the type confusion cannot be exploited.

Figuring out how this is done would allow an attacker to collect valid log-in tokens, and then use those tokens to perform transactions. Since it is unlikely that the random reader contains some sort of secret (like an encryption key), as well as the fact that the output is so similar to what the card outputs (see Appendix C for some sample outputs) it is likely that the scrambling process is a violation of Kerckhoff's Principle by being a measure of security through obscurity.

See Section 3.5 for more details on this suggestion for future work.

ABN-AMRO: Connecting an e-dentifier 2 to the PC

It is possible to connect an e-dentifier to the PC through a USB interface. This allows the ABN-AMRO Internet banking website to send data to, and receive data from the e-dentifier through a special software package. While this helps the user with verifying the transaction (the display will show transaction related data such as the amount and the account number of the target of the transaction), it might be possible to manipulate this information as well.

Investigating this USB connection to see the protocol, as well as the data that is sent would be very interesting. Will it still send EMV-compliant commands, or will it send its own set of commands which will be translated by the e-dentifier into commands that can be sent to the card? Of course it is even more interesting to see how this protocol can be influenced to see if wrong data can be sent, for example through a man-in-the-browser attack which targets the software used to communicate with the e-dentifier.

ABN-AMRO: Possibilities with a new bank card

Since the ABN-AMRO bank is the only bank investigated during this research project to select a portion of the transaction counter through the bit filter, it will be interesting to see what happens if the transaction counter is overflown past the maximum selection point of the bit filter (e.g. 128), while the bank card has never been used for Internet banking.

Since the bit filter will then select only a portion of the transaction counter, the ABN-AMRO Internet banking application might think the transaction counter is at a lower value. For example, if the current transaction counter is 130, the bank will only receive the value of 2, since the bit that represents the value 128 is not selected by the bit filter. If someone would collect a lot of different cryptograms with a lower transaction counter, it will be interesting to see if those values can be used because the Internet banking website thinks the transaction counter is lower than it actually is.

Whether or not this can be exploited for anything useful is of course debatable, but it can at least give an insight into the inner workings of the Internet banking website itself. It might give an answer to the question whether or not selecting a portion of the transaction counter is useful, or whether the ABN-AMRO would have been better off to guess it like the other Dutch banks investigated during this research project seem to do.

Multiple cards for the same bank account

It is possible to get multiple bank cards for the same bank account, for example when it is an account shared by multiple individuals or when receiving a new bank card after the old one has (nearly) expired. This is indicated on the card through a card number. For the ABN-AMRO Internet banking website it is a field that needs to be entered to log in, along with the account number of the bank account.

How this affects the generation of cryptograms is currently unknown, and it would be interesting to see how this influences the process. From what is currently known it is likely that when using Internet banking the only unknown for the Internet banking website is the transaction counter, and that the card number is not used as transaction data in the generation of the cryptograms.

Should this be true it is possible that a card with a particular card number but tied to the same bank account can be used to log in and perform transactions for a different card number, as long as the transaction counter is higher than the one currently set at the other bank card.

This could lead to several problems. Take, for example, a married couple that is going through a divorce. One of the two individuals could pretend to log in and transfer money from their shared account using the card number of the other person. The bank records would indicate that the wrong person has done this, which might lead to wrong decisions during settlement decisions.

Another example might be that a corrupt bank employee that somehow has access to the card creation process can create a fake card with a new card number and a PIN that is under his or her control. The card might not be usable for normal EMV transactions, as the card will officially not exist, but should the corrupt employee simply raise the transaction counter high enough he or she would be able to access an Internet banking account of the victim.

Of course it is likely that the cards share an account number, but still have different cryptographic keys. This would make it unlikely that such a situation can be exploited as it is explained in this particular section. However, it will still be interesting to verify whether or not this is the case.

SNS: Reverse engineering the DigiPas

The SNS bank uses a radically different device to log in to their Internet banking website: a device that is tied to a single bank account and which does not use a bank card at all. This has several consequences, one of which is that the device will obviously contain some sort of secret.

It would be interesting to see how this device works internally. For example, does the device contain a tamper-resistant chip as well and if not, how does it store its information internally? Once this is know the next logical step would be to see if information can (easily) be extracted from the device and see how this affects the overall security of the Internet banking system of the SNS as a whole.

ABN-AMRO: Reverse engineering the e-dentifier 1

The final suggestion for future work that is made is to take another look at the old version of the e-dentifier that was used during this research project: the e-dentifier 1. This e-dentifier can still be used, and even though an attempt was made to also reverse engineer this device it failed for an unknown reason (see Section 3.5.4 for the details).

Even though it should not have much of an effect, it is possible that the e-dentifier cannot communicate properly with the blank card that was used due to a difference in the ATR. However, because the old version of the e-dentifier can still be used, it is interesting to figure out how that particular device works, especially compared to the newer version. It might not use EMV, which will give all sorts of interesting possibilities in terms of attacks and vulnerabilities.

Chapter 6

Conclusions

With the completion of this research project, it is possible to answer the research questions as they have been posed in Section 1.2. The questions will once again be listed here, and instead of individually answering these questions (which would lead to having to repeat large sections of what has already been written) they will be answered by addressing them throughout the rest of the conclusions.

> *How do Dutch banks use the EMV specifications in their bank cards, in particular when using Internet banking, and what are the consequences and limitations on the security of the system as a whole?*

Sub-question 1. *How are the EMV specifications used to design an implementation of bank cards usable for Internet banking in terms of protocols and capabilities?*

Sub-question 2. *Which options did the Dutch banks choose to implement of the EMV standard in terms of protocols and data authentication (SDA, DDA or even CDA) when using Internet banking?*

Sub-question 3. *Do the Dutch EMV bank cards suffer from the same problems that have been found [4, 5, 12] in the U.K.?*

This section begins by discussing the experience of trying to understand the EMV specifications and reverse engineering the protocols. Actually answering the research questions is done after listing these experiences.

Size and complexity of the EMV specifications

One of the most important conclusions that can be made after completing this research project should be obvious when looking at Chapter 2 of this document:

the size and complexity of the EMV specifications can be called daunting at the very least. Not only are the specifications themselves very length, allowing for a wide variety of different options and different interpretations to be used, it is also based on various other standards. However, minor variations in interpreting those standards leads to possible incompatibilities, such as the problem described in Section 3.3.2 in which the procedure byte returned in a T=0 protocol session is statically defined as having a CLA byte of '00', as opposed to using the procedure byte of the previous command.

The process of reverse engineering

Reverse engineering was done to figure out which protocols are used in challenge/response devices such as the e-dentifier, as well as finding out which options of the EMV specifications Dutch banks have chosen to implement.

Even though initially it was planned to use a traffic sniffing device to reverse engineer the protocol used by the e-dentifier, the device was unable to properly interface with the card and the e-dentifier because of differences in baud rates.

Even though it would have been easier to use a traffic sniffing device, as it would provide an immediate overview of the complete protocols used by the e-dentifier, the alternative turned out to be relatively easy as well. Instead of sniffing the traffic, a programmable smart card was used that logs commands sent to it by the e-dentifier, while using a terminal application to simulate those commands sent to collect real responses from an actual bank card. The programmable smart card would then be able to give a proper response, which lead to the possibility of logging the next command sent by the e-dentifier.

Although it was more tedious to reverse engineer the protocol in this way, the nature of communication with a smart card reader and a smart card (an unexpected response usually leads to the termination of communications altogether) still made it a fairly straightforward process.

EMV in the Netherlands

Three papers [5, 12, 4] listing several flaws and weaknesses of EMV implementations in the U.K. were one of the biggest reasons this research has been done. The paper [5] that lists problems with the challenge/response devices used for Internet banking was of interest in particular, as the focus of this research was primarily on Internet banking because it provides a successful attacker with nearly limitless capabilities relating to the compromised bank account of the victim.

Internet banking

By having the bank card sign certain transaction data, the bank can verify that the owner of the card wants to perform certain actions, such as logging in to the Internet banking website or allowing a transaction to occur. Part of this transaction data is a special code that is displayed by the Internet banking website, which the user has to enter into a special device. However, the banks investigated in [5] simply have the card sign the code displayed on the website, and give the card (and more importantly: the cardholder) no method of verifying what it signs. Actions that should be different (such as verifying the account number of a transaction vs. authorising the transaction to occur) are treated in the same manner by the card, which means that an attacker could exploit this by having the card sign something which might be usable for something else.

Type confusion

In [5] the problem of type confusion is introduced. This problem is caused by using the unpredictable number field to store the challenge that is entered by the user into the challenge/response device. Because the device can be used in different modes (say: logging in and verifying a bank account number) it is possible to have the device get responses from the card for one mode that can be used in another mode. Section 4.1 of this document discusses all the details of this weakness.

ABN-AMRO Internet banking

After having reverse engineered the protocol used by the e-dentifier (see Figure 3.3) it was discovered that while the ABN-AMRO uses the same steps in the protocols as those reported in [5], the bank has taken several steps in order to prevent the possibility of type confusion.

This is solved by the ABN-AMRO by encoding the challenge that is entered into the e-dentifier. Instead of simply putting an entered code straight into the unpredictable number field, it somehow modifies it to allow at least the back-end of the ABN-AMRO Internet banking application to differentiate between the different options. Of course since the same protocol steps are still used, the bank card is still not able to determine what it signs. However, this countermeasure makes sure an attacker does not have an easy method of exploiting the potential type confusion. Of course it might still be possible due to the possibility of collisions (the potential size of the input is greater than the size of the output) but it has been made harder up to the point where it can be considered unlikely.

Rabobank Internet banking

The Rabobank also uses a challenge/response device called the random reader, and while this also uses the same protocol steps as the ABN-AMRO, it still suffers from the type confusion. What is worse, the random reader suffers from a worse version of this type confusion, since it never has the card sign the code that is displayed on their Internet banking website. Instead, they always sets the value of the unpredictable number field to ’0000 0000’, and then use the challenge entered into the device to somehow scramble the result.

Because this static value of ’0000 0000’ is used for the log in procedure, and since they chose to use this data for all transactions instead, it is possible to collect valid log in tokens and use those to perform transactions. Of course the attacker still needs to know how the scrambling is done, but this can easily be bypassed by using a relay attack to an actual random reader. A good social engineer could likely convince Rabobank customers to divulge these tokens in order to perform transactions (”*I can only log in with this data so there is nothing to worry about. As a bank employee I can already log in to your account*”), making this a big potential risk.

ING bank and SNS bank EMV cards

While these two banks do not use EMV cards for their Internet banking, they all distribute bank cards which contain EMV applications, as they will switch to EMV for payments before the end of 2011. Because of this certain data could be extracted from these cards (even data that should only be relevant for Internet banking as they seem to be introduced by the CAP standard such as the bit filters). These values can be found in Figure 3.8.

PIN-less transactions at a Point of Sale

In [12] an attack is described that allows an attacker to complete transactions without having to know the PIN of the card. This is possible because when an issuer is given the transaction data, it is not able to determine what verification methods have been performed by the terminal and see if this corresponds with what the card has specified. By using a man-in-the-middle attack an attacker can suppress the PIN verification step request to the card and simply pretend it was successful by sending a ’9000’ result to the terminal.

Luckily, the Dutch banks seem to have chosen better options in their implementation of the same system. By including the Cardholder Verification Method Results in the transaction data to be signed by the card (see Figure 3.8) an issuing bank gets the ability to compare this with the data given by the card, and detect corresponding mismatches in what the card and terminal believe has occurred.

All Dutch banks request this information, which means they should all be able to detect it when this attack occurs. Since verifying whether this is also done in practice was out of the scope of this research project, it is currently not yet known if Dutch banks are vulnerable to this attack. However, due to the fact that it should be detectable is already good news for customers of these Dutch banks, as it should be possible to reconstruct the scenario in case of disputes. Fortunately it seems unlikely that this will become an issue, as Dutch banks claim to have verified this attack is detected, leading to a rejection of the transaction in case this happens.

Data authentication

All cards that have been looked at during this research project indicated they supported DDA, and not SDA or CDA. This corresponds with what has been revealed by Dutch banks during private communications, during which they indicated they are in the process of phasing out SDA cards.

Overall

The major Dutch banks seem to have chosen better options of their implementations of the EMV specifications for both their Internet banking devices and cards, as well as their EMV payment cards when compared to the U.K. Of course there are still some open issues, but that is to be expected. For example, it is currently unclear whether Dutch banks actually verify the cardholder verification method results correspond with what the bank card reports on a particular transaction. It is also currently unknown whether the encoding used by the ABN-AMRO is done through a cryptographically secure manner such as a cryptographic hash function.

Suggestions for improvement

There is always room for improvement. For example, due to the huge potential of abuse when an attacker gains access to an Internet banking application of a victim, it should be made extremely hard for an attacker to be successful in this.

Extra step in the log-in procedure

Currently, both the ABN-AMRO and the Rabobank rely exclusively on a customer using their PIN and bank card in order to log in to their Internet banking and authorise transactions. An attacker with a fake terminal could therefore potentially attack every customer of this bank, without having to know more information than the victim provides through the fake terminal.

As an additional security measure it is possible to require a user name and password to log in. This would prevent attackers being able to use a fake terminal on a massive scale. They would have to target individuals of which they knew the log in credentials before being able to abuse it.

Of course it would also have various other security implications, but these (and possible alternatives) are out of the scope of this research. It is however a measure that should be considered for the potential gain it provides.

Rabobank: Loading transaction data in the unpredictable number field

The biggest suggestion for improvement that can be made is aimed at the Rabobank, which uses a static value of '0000 0000' in the unpredictable number field to be signed by the card. Why this choice was made is unknown, but if an attacker can figure out how the encoding scheme works (or how to create a relay to get proper responses from the random reader itself, which is not hard to do) it can lead to unwanted situations. An attacker only needs knowledge of the encoding scheme and two valid log-in tokens from a victim to be able to log-in and perform a transaction.

Having bank cards check the CVMR themselves

The final suggestion that is made is to have cards themselves compare the cardholder verification method results to what the card believes has occurred, allowing a card to reject a transaction if a terminal believes PIN verification was attempted while the card knows it was not attempted. When doing this, fraudulent transactions can be stopped by the card, and the possibility of verification errors occurring at the issuing bank becomes irrelevant for these situations.

Bibliography

[1] Ross Anderson, Mike Bond, and Steven J. Murdoch. Chip and spin, 2006. [cited at p. 6]

[2] David John Boyd. Phd thesis: E-payments: Cardholder privacy and non-repudiation, 2009. [cited at p. 38]

[3] S. Burns and G. R. S. Weir. Trends in smartcard fraud. In *Proceedings of the 4th International Conference on Global E-Security*. University of East London, June 2008. [cited at p. 3]

[4] Saar Drimer and Steven J. Murdoch. Keep your enemies close: distance bounding against smartcard relay attacks. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1–16, Berkeley, CA, USA, 2007. USENIX Association. [cited at p. 4, 6, 7, 65, 66, 79, 80]

[5] Saar Drimer, Steven J. Murdoch, and Ross J. Anderson. Optimised to fail: Card readers for online banking. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 184–200. Springer, 2009. [cited at p. 4, 6, 7, 17, 38, 39, 40, 51, 53, 54, 55, 56, 61, 65, 70, 71, 73, 74, 79, 80, 81]

[6] Gareth Ellis. Coping with new EMV initiatives. *Card Technology Today*, 19(7-8):10 – 11, 2007. [cited at p. -]

[7] EMVCo. *Integrated Circuit Card Specifications for Payment Systems - Book 1: Application Independent ICC to Terminal Interface Requirements.* EMVCo, 4.2 edition, 2008. [cited at p. 4, 9, 10, 23, 46]

[8] EMVCo. *Integrated Circuit Card Specifications for Payment Systems - Book 2: Security and Key Management.* EMVCo, 4.2 edition, 2008. [cited at p. 4, 9, 14, 15, 16, 34, 37]

[9] EMVCo. *Integrated Circuit Card Specifications for Payment Systems - Book 3: Application Specification.* EMVCo, 4.2 edition, 2008. [cited at p. 4, 9, 34, 41, 42, 45, 55]

[10] EMVCo. *Integrated Circuit Card Specifications for Payment Systems - Book 4: Cardholder, Attendant, and Acquirer Interface Requirements.* EMVCo, 4.2 edition, 2008. [cited at p. 4, 9, 55]

[11] Vorapranee Khu-smith and Chris J. Mitchell. Using emv cards to protect e-commerce transactions. In *EC-WEB '02: Proceedings of the Third International Conference on E-Commerce and Web Technologies*, pages 388–399, London, UK, 2002. Springer-Verlag. [cited at p. 38]

[12] Steven J. Murdoch, Saar Drimer, Ross Anderson, and Mike Bond. Chip and pin is broken. In *To appear at the 2010 IEEE Symposium on Security and Privacy (draft)*, 2010. [cited at p. 3, 4, 6, 7, 38, 59, 65, 68, 70, 74, 79, 80, 82]

[13] Cristian Radu. *Implementing Electronic Card Payment Systems.* Artech House, Inc., Norwood, MA, USA, 2002. [cited at p. 38]

[14] Els van Herreweghen and Uta Wille. Using emv smartcards for internet payments. In *ECIS*, 2000. [cited at p. 38]

Appendices

Appendix A

EMV commands overview

This appendix contains an overview of the commands that have been defined for financial transaction in the EMV standard. This overview is meant to be used as a reference guide when trying to get an overview of the protocols that are used in an EMV application in terms of commands and responses that are sent in order to be able to quickly determine which command has been sent by the card.

Table A.1: EMV commands overview

CLA byte	INS byte	Command name	Summary
00	20	VERIFY	Instructs to ICC to verify the PIN specified in the data field to the PIN data associated with the application
00	82	EXTERNAL AUTHENTICATE	Requests the currently selected application to verify a cryptogram
00	84	GET CHALLENGE	Used to obtain an *unpredictable number* from the ICC
00	88	INTERNAL AUTHENTICATE	Initiates computation of the Signed Dynamic Application Data
00	A4	SELECT	Selects a file, directory or application on the ICC
00	B2	READ RECORD	Instructs the ICC to read and return a file record
00	C0	GET RESPONSE	Used in the T=0 protocol to get a response from the ICC
80	A8	GET PROCESSING OPTIONS	Initiates the transaction within the ICC
80	AE	GENERATE APPLICATION CRYPTOGRAM	Sends transaction data to the ICC to be involved in creating a cryptogram
80	CA	GET DATA	Request a primitive data object that is not within a record on the ICC

Table A.1: Continued: EMV commands overview

CLA byte	INS byte	Command name	Summary
8C or 84	16	CARD BLOCK	Permanently disable all applications on the ICC
8C or 84	18	APPLICATION UNBLOCK	Reactivates the currently selected application
8C or 84	1E	APPLICATION BLOCK	Invalidates the currently selected application
8C or 84	24	PIN CHANGE/UNBLOCK	Allows issuers to change and unblock the PIN

Appendix B

Sample challenge encodings

This appendix contains sample challenge encodings for the ABN-AMRO e-dentifier 2 device. The selected option column means the option that was selected in the menu of the e-dentifier 2. Challenge means the challenge as it was entered into the e-dentifier 2, while the encoded challenge is the value of the challenge as it is sent in its APDU format by the e-dentifier 2 to the card.

Selected option	Challenge	Encoded challenge
Verify account	5609 0403	2B8B E1AA
Verify account	0000 0000	BE8F 936B
Verify account	000000000000000000000000000000000000	2616 BE93
Check input	1234 5678	C436 CEED
Check input	000000000000000000000000000000000000	7808 5E9F
Check input	0000 0000	A3D8 C36C
Send transaction	2466 1140	661D 7D59
Send transaction	1234 5678	EFD2 2EDB
Send transaction	9876 5432	542A 2C16
Send transaction	1111 2222	8C88 5B29
Send transaction	0000 0000	F0BA 6928
Send transaction	0000 0001	54F6 D1E7
Send transaction	0000 0002	C128 9A44

Table B.1: Challenge encoding examples

Appendix C

Sample challenges and their outputs for the Rabobank random reader

This appendix contains a table with some of the outputs from the Rabobank random reader for a given challenge. These responses were collected from the bank card simulator (a programmable bank card that was used to simulate a single session of a valid bank card), which always returns the same response to the *GENERATE AC* command requesting an ARQC. Because it always returns the same response, it is possible to see the effect of the challenge onto the final result.

The input column specifies the challenge that is entered into the random reader, while the result column lists the corresponding response that is shown on the display of the random reader. The result (binary) column is the result in its binary format, which is the same as the previous column which is in a decimal format.

When applying the bit-filter that is specified by the bank card simulator to the standard response, it will lead to a value of 3499 8891 (10000101100000101001101011 in binary). As can already be seen this value is extremely similar to the output presented by the random reader.

Input	Result	Result (binary)
00000 00000	3478 1624	10000100101011100110111000
00000 00001	3460 7710	10000100000001001001011110
00000 00002	3502 1539	10000101100110001011100011
00000 00003	3500 6820	10000101100010100101100100

Table C.1: Sample challenges and their outputs for the Rabobank random reader

Table C.1: Continued: Sample challenges and their outputs for the Rabobank random reader

Input	Result	Result (binary)
77777 77777	3461 1458	10000100000010000100000010

Table C.1: Sample challenges and their outputs for the Rabobank random reader

Appendix D

Applets on Dutch bank cards

This appendix contains a list of applets that have been found during this research project, either through a challenge/response device attempting to select this applet or through a bank card exposing this applets in its *1PAY.SYS.DDF01* directory.

The names of the AIDs that were *not* found on a bank card (and thus were not known) have been searched for online.

- A0000000032010 (VISA Electron)
- A0000000031010 (VISA credit)
- A0000000043060 (MAESTRO)
- A0000000041010 (Mastercard Credit)
- D5280050218002 (unknown)
- A0000000038002 (unknown)
- A0000000048002 (SecureCode Aut)
- A0000003156020 (ChipKnip)
- A00000031510100528 (PIN)

Appendix E

Trace of an Internet banking session

This Appendix contains a trace of an Internet banking session in its raw APDU format. It lists the command, the P1 and P2 values and the data that is sent, followed by the response sent by the card. Sensitive information (such as the PIN) has been replaced with an X for each character that is replaced.

This trace was made using the e-dentifier 2 of the ABN-AMRO and an ABN-AMRO bank card. Application selection as it has been specified in Section 3.4 is omitted, except for the selection of the SecureCode Aut application that is done in the final step.

```
-----------------------------------------------------------------------------------
User enters the card into the e-dentifier 2. It attempts to
select several applets that might or might not be present
-----------------------------------------------------------------------------------
SELECT APPLICATION (SecureCode Aut)
Command: 00A4
P1+P2:   0400
Data:    07A000000004800200

Response: 6F258407A0000000048002A51A500E536563757265436F646520417574
          8701005F2D046E6C656E9000
-----------------------------------------------------------------------------------
GET PROCESSING OPTIONS
Command: 80A8
P1+P2:   0000
Data:    02830000

Response: 770A82021000940408010100 9000
```

```
------------------------------------------------------------------------------------
READ RECORD
Command: 00B2
P1+P2:   010C
Data:    -


Response: 70608C219F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403
          8D0C910A8A0295059F37049F4C085A0AXXXXXXXXXXXXXXXXXXXX
          5F3401018E0A00000000000000000001009F5501809F560C00007FFFFE000000000000009000
------------------------------------------------------------------------------------
User selects option 2 (Send transact.) at the e-dentifier 2 menu
------------------------------------------------------------------------------------
GET DATA (PIN try counter)
Command: 80CA
P1+P2:   9F17
Data:    -

Response: 9F1701039000
------------------------------------------------------------------------------------
User enters the PIN
------------------------------------------------------------------------------------
VERIFY
Command: 0020
P1+P2:   0080
Data:    0824XXXXFFFFFFFFFF

Response: 9000 if the PIN is correct.
          63Cx otherwise, where x is the remaining # of PIN tries
------------------------------------------------------------------------------------
User enters the challenge 2466 1140
------------------------------------------------------------------------------------
GENERATE APPLICATION CRYPTOGRAM (ARQC)
Command: 80AE
P1+P2:   8000
Data:    000000000000000000000000000080000000000000000000000661D7D5934000000000000000
         00000010002


Response: 77299F2701809F360200429F2608C14D71DBAFA79FED
          9F10120012A50003020000000000000000000000FF9000
------------------------------------------------------------------------------------
GENERATE APPLICATION CRYPTOGRAM (AAC)
Command: 80AE
P1+P2:   0000
Data:    000000000000000000005A338000000000000000000000000000000000
```

```
Response: 77299F2701009F3602004E9F260896F166E11152A46B
          9F1012001225000342000000000000000000000000FF9000
-------------------------------------------------------------------------------------
e-dentifier display shows 3499 8891
-------------------------------------------------------------------------------------
```

Appendix F

Overview of produced code

Code has been written during this research project in order to help with the reverse engineering of the different challenge/response devices. This appendix gives a global overview of the different code projects, listing both their names and contents to help others understand it should they wish to look at or tinker with the code.

Three different code solutions have been made during this research project:

1. CardChannelBugFix
2. EMV-Bankcard
3. EMV-Bankcard-Terminal

CardChannelBugFix

The code that has been produced is Java code. However, because of the EMV specifications the *ChannelImpl* class that can be found in *sun.security.smartcardio* does not work properly when used with EMV cards (see 3.3.2 for the details of this problem). For this reason the class has been edited to properly work with EMV cards by copying the source code of the particular Java class and fixing the bug manually. This class has to be loaded *before* the Java runtime environment in order for the EMV-Bankcard-Terminal to work properly with EMV cards! This can be done in Eclipse by adding the project to the bootstrap entries in the Classpath tab of the debug/run configurations of the EMV-Bankcard-Terminal project and making sure it is higher in the priority chain than the Java runtime environment. Alternatively it is also possible to load this manually using the *Xbootclasspath* option in the Java application launcher.

EMV-Bankcard

This project contains the code that has to be loaded on a programmable smart card. This allows the card to log unknown responses from a terminal in order to help with the reverse engineering of the protocols used by these terminals.

The code for this applet contains a lot of static information in terms of data and responses, and could be used as a bank card that simulate a single session of an EMV Internet banking session. This is because a real bank card was used to extract actual responses to be used by the bank card simulator.

In order to have the card report the latest instruction it did not understand (i.e. no response was programmed for that particular command) one should send it a *0050000000* command.

EMV-Bankcard-Terminal

This application contains both the terminal application that can simulate an e-dentifier Internet banking session, as well as the code that allows EMV bank card information (such as the bank account number and AIP) to be read. If a card is entered into a card reader the application will automatically attempt to see which applications are present on the card, and will extract the information exposed by the applications that support EMV.

List of Symbols and Abbreviations

Abbreviation	Description	Definition
AC	Application Cryptogram	page 15
AAC	Application Authentication Cryptogram	page 17
AFL	Application File Locator	page 14
AID	Application Identifier	page 13
AIP	Application Interchange Profile	page 50
APDU	Application Protocol Data Unit	page 13
API	Application Program Interface	page 30
ARPC	Authorisation Response Cryptogram	page 18
ARQC	Authorisation Request Cryptogram	page 17
ATC	Application Transaction Counter	page 36
ATR	Answer to Reset	page 11
BER-TLV	Basic Encoding Rules-Tag Length Value	page 18
CAP	Chip Authentication Program	page 40
CDA	Combined Dynamic Data Authentication / Application Cryptogram Generation	page 15
CDOL	Card Risk Management Data Object Lists	page 23
CID	Cryptogram Information Data	page 36
CLA	Class byte	page 23
CVM	Cardholder Verification Method	page 55
CVRM	Cardholder Verification Method Results	page 69
DDA	Dynamic Data Authentication	page 14
DOL	Data Object List	page 22
EMV	Europay, MasterCard and Visa	page 3
IAD	Issuer Application Data	page 36
ICC	Integrated Circuit Card	page 9
IFD	Interface Device	page 28
INS	Instruction byte	page 23
MAC	Message Authentication Code	page 17
PKI	Public Key Infrastructure	page 19

Abbreviation	Description	Definition
PAN	Primary Account Number	page 18
POS	Point of Sale	page 28
SDA	Static Data Authentication	page 14
SFI	Short File Identifier	page 22
TC	Transaction Certificate	page 17
TLV	Tag-Length-Value	page 45
TVR	Terminal Verification Results	page 68
UN	Unpredictable Number	page 71

List of Figures

List of Tables