ABN-AMRO E-dentifier2 reverse engineering

Master of Science Thesis
Supervisor: Erik Poll

Arjan Blom
a.blom@student.ru.nl

Radboud University Nijmegen
Computing Science Digital Security Faculty

December 19, 2011

Contents

Acknowledgements

I would like to thank Erik Poll for being my thesis supervisor, giving me directions and feedback that I needed to successfully complete the research and thesis.
I also would like to thank the Digital security group, especially Gerhard de Koning Gans, Joeri de Ruiter and Roel Verdult for their insights and help.

Chapter 1

Introduction

Securing online banking transactions nowadays is commonly done by means of a handheld smartcard reader which, in combination with a bankcard, is used to authenticate transactions by generating transaction confirmation codes. A security code is a code displayed on the internet banking website of the bank as a string of numbers when a user needs to confirm a transaction. Most security tokens are autonomous devices that require a smart card to operate and only interface with the outside world by means of a keypad and a screen. This means that when an user wants to login or authorize a transaction, the user has to type in the security code and his pin code on the security token; the security token then displays the transaction confirmation code which, in turn, the user has to enter at the banking website to login or authenticate a transaction. The process offers a relatively high level of security, but the process of retyping the security code and transaction confirmation code is often considered a burden. Also the security code itself is meaningless to the customer because the security code appears as a non-descriptive collection of numbers (often information is included in this collection of numbers, e.g. a destination account or the transfer amount). To counter these issues, banks have devised a security token that can be connected to the computer by means of an USB connection. This removes the need to retype the different codes and enables the banks to offer more useful information (relative to the security code) on the screen of the security token. The problem with this new device is that it is connected via USB with a home PC which may be infected with all sorts of malware). The connectivity with a home PC requires more protection of the USB security token than its non-connected counterpart that only accepts input by the keypad and bankcard. The USB security token investigated in this thesis is an e-dentifier developed by Gemalto used for internet banking by ABN-AMRO, see also figure 1.1. In this Master Thesis, the USB security token (e-dentifier2) is investigated to answer the following main question:

> Has the USB security token implemented Sign What You See (SWYS) correctly?

In essence, SWYS mandates that the text displayed on the USB security token is used in the sign process (cryptogram generation) and the text displayed on the screen of the USB security token is understood by the customer. This property is discussed in more detail below. The question is answered by investigating the following sub questions:

1. Which flows of communication are distinguishable between the home pc, the USB security token and the smartcard and how are the commands translated into the different device protocols?
2. What additional logic is present in the USB security token?

3. How susceptible is the USB security token for malicious use (by means of malicious code or commands)?
4. Is it possible to use the USB security token for different purposes than it was originally designed for? E.g. as a general secure signature device.

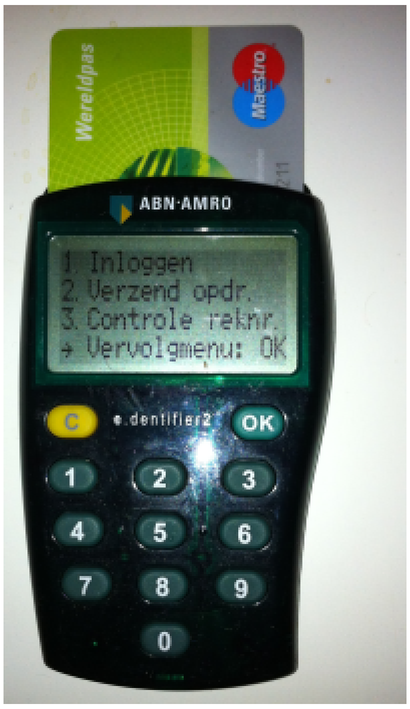


Figure 1.1: E-dentifier2

1.1 Sign What You See (SWYS)

In this thesis, the abbreviation SWYS is often used. This refers to Sign What You See, as used by the manufacturer Gemalto. This term is also known as the more commonly used What You See Is What You Sign (WYSIWYS), a property that is crucial for the edentifier2. A bank requires a process that guarantees when the customer orders a transfer of money on their internet banking website the following statements are true:

- The customer is the owner of the bank account;
- The customer has indeed ordered the transfer;
- The transfer amount is correct;
- The destination account is correct.

All of these requirements are met by the Sign What You See process. The Sign What You See process involves message authentication codes, generated by the bankcard that is provided by the bank and a trusted device with a (trusted) display. The process involves the following:

- The trusted device displays the transfer details (destination account, transferred money).
- The customer agrees with the transfer on the trusted device.
- The trusted device sends the displayed data to the bank card to be signed with a personal key proved by the bank.
- The signed data is send to the bank.

It is crucial that the data displayed on the device is part of the data that is signed. If this is not the case, Sign What You See is not true. An additional demand for Sign What You See is that the displayed data is understood by the customer. Otherwise the bank cannot ensure that the customer has confirmed that the amount and the destination account are correct. In order to prevent replay attacks the signed data should also contain some random nonce or a transaction counter.

Chapter 2

Background

2.1 Tools

In order to answer the main question and sub questions, general tools and specific attack tools were used, which are described in this section.

2.1.1 General tools

The general tools that did not have a specific attack or reverse engineering goal are described in this section. These tools mainly support the attack tools used for this research.

- Windows 7 (64 bit); For this research Microsoft Windows 7 was used. (I do not recommend the use of the 64 bit version of Windows 7 for doing research. The problem with the 64 bit edition is the need for 64 bit drivers that do not come with every tool. The 32 bit version is better suited for research purposes because it is backwards compatible with older windows versions and therefor there are more tools available. I countered this problem mainly in the search for a suitable USB sniffing tool.)
- Microsoft Visual Studio 2010; An integrated development environment. Visual studio was used to write the custom USB driver.

2.1.2 Attack tools

The following tools were used specifically for the attack and reverse engineering.

- USBtrace; Sniffing tool specially designed to capture commands sent over the USB port.[8]
- RealTerm; terminal client used to capture the data sent from the "rebbelsim". Any terminal client can be used as long as it supports the custom setting of the baud rate. To be able to eavesdrop the card communication, the baud rate should be set to 5200 and the display has to be set to "hex". [7]

- LibUsbDotNet; This library offers a basic USB driver logic to ease the writing of custom USB drivers. This specific library is created for the .NET platform but other implementations exist, if required. [6]
- Rebbelsim; Hardware that enables to eavesdrop on the communication between the USB security token and the smartcard.
- Smartlogic; Software that enables to eavesdrop on the communication between the USB security token and the smartcard, like Rebbelsim but has a nicer output.[1]
- Fiddler2; HTTPS transparent proxy which can be used as a Man In the Middle tool. This was used to intercept and change the data sent from the bank.[3]
- Custom USB driver; custom written USB driver for the USB security token, to accommodate the experiments.

2.2 Standards

2.2.1 EMV-CAP

EMV-CAP are two complimenting standards (EMV and CAP) that enable online banking transactions. This section provides some background information on the two standards.

EMV

EMV stands for Europay, MasterCard and Visa. This consortium has created specifications for Integrated Circuit Cards for payment systems to ensure interoperability between chip based consumer payment applications and acceptance terminals to enable payment. The standard is publicly available[1].

CAP

CAP stands for Chip Authentication Program. As an initiative from MasterCard and later adopted by Visa, CAP enables smartcards that comply with the EMV standard to authenticate users and approve online transactions. This standard is not publicly available , it has been reverse engineered in [2]

2.2.2 USB communication

The following section contains general background information about the USB communication between USB hosts (computer) and USB clients (USB security token). As depicted in figure 3.2, the driver and the home PC communicate by means of the USB standard. USB communication takes place between a host, in this case a home PC and an endpoint in a device, the device being the USB security token. An endpoint is a uniquely addressable portion of the device that can either be a receiver or transmitter of data. Endpoint addresses are 4 bits in size and also indicate the direction of the transfers.

Connections are made by “virtual pipes”. These pipes connect the device endpoint with the host. When the connection is established between the endpoint of a device and the host, the endpoint returns

[1]Available at http://www.emvco.com/

a descriptor that contains the configuration settings of that endpoint. These descriptors include settings about the transfer type, data packet size, interval of data transfers and bandwidth requirements.

USB supports four data transfer types: control, isochronous, bulk, and interrupt messages.[4]

- Control messages; These messages exchange configuration, setup and command information between the device and the computer. CRC checks will ensure data integrity
- Isochronous messages; isochronous messages handle streaming data. Streaming data is time sensitive and as such these messages have priority on the USB bus. No error handling occurs on these type of messages, message loss can occur.
- Bulk messages; Bulk messages move large amounts of data when the USB bus has excess bandwidth. CRC checks will ensure data integrity, but there exists no guarantees on timing.
- Interrupt messages; USB is a poll only protocol; devices cannot initiate communication on their own. Interrupt messages are messages which indicates that the device needs urgent attention.

Chapter 3

Protocol

In this chapter internet banking with the e-dentifier2 is outlined and elaborated in a top-down approach. The protocol is reverse engineered by means of intercepting all communication over the communication channels. The communication is captured by the tools in chapter 2. The outline of the communication helps in understanding the communication flow in the protocols used by the USB security token. By understanding the communication flow the (possible) weaknesses are identifiable. The protocol consists out of two types of transactions:

- Login; the part of the protocol that is utilized in the logon procedure of the banking website;
- Transaction; the part of the protocol that is used for bank transaction confirmation.

3.1 General

This section describes the actors involved in internet banking with the e-dentifier2. To clarify the actors used and their role in the processes, as depicted in figure 3.1.

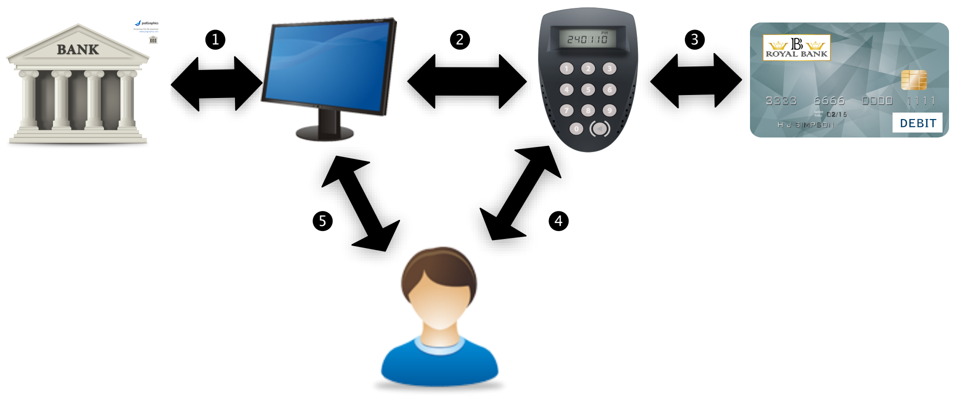


Figure 3.1: Communication overview

From left to right:

- **Banking website:** This is an application running on a webserver. This webserver is located in a secure environment in an off-site location. All the backend processes and administrative tasks of the bank take place at this location. This actor is a black box for this thesis. The banking website uses the HTTPS protocol to communicate with the home PC of the customer.
- **Home PC:** Used by the customer to interact with the banking website. It is located at the customers home or other facility accessible by the customer. The home PC utilizes a browser to communicate with the banking website and the customer. A plugin/driver is utilized to communicate with the USB security token over an USB connection as depicted in figure 3.2.

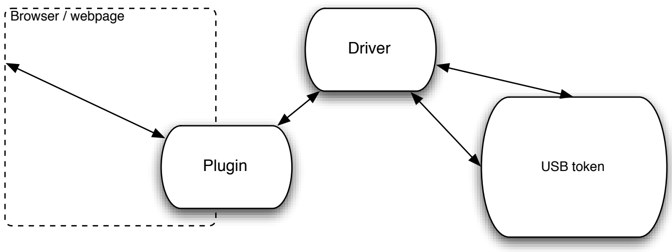


Figure 3.2: Overview browser-usbtoken

  The communication via the browser, from and towards the banking website and from and towards the USB security token, is performed at a level not visible to the customer (and of non-importance to the customer). Communication with the customer is done via the screen connected to the home PC and via the screen of the USB security token.

- **USB Security Token**: This device is used as a trusted interface between the bank card and the customer. The device communicates with the customer via the onboard display and the keypad. The device communicates with the bankcard via APDUs [5].
- **Bankcard:** The bank card contains a microchip which is capable of doing computational tasks, as specified in the EMV-CAP standard. The card communicates via APDUs with the outside world.
- **Customer:** The customer is a user of the banking website and an initiator of the transaction.

3.1.1 JavaScript

As depicted in figure 3.2 the home PC actor consists of multiple items on a lower level. The communication from and towards the banking website (run in the browser of the customer) and the driver of the USB security token is done by means of JavaScript in conjunction with a browser plugin. The JavaScript communicates with the browser plugin. The browser plugin communicates with the USB security token driver. Among other JavaScript files, which are not relevant to this thesis, the BECON.js takes care of this communication. The following functions in the BECON.js file are used to communicate with the USB security token driver.

- **function BECON_CheckConnection()** ; checks if the USB security token is connected and a bankcard is inserted in the device.

- **function BECON_CheckCard()**; returns the bankaccount number and the card number of the smart card that is present in the USB security token.
- **function BECON_GetResponse(p_sSignData)**; this method is used for the cryptogram generation. The argument p_sSigndata is displayed on the USB security token's display and then used as input for the cryptogram generation. The method returns a cryptogram.
- **function BECON_GetMode1Response(p_sChallenge, p_iCurrency, p_sAmount)**; not used, EMV-CAP mode1 is a less complicated operation than EMV-CAP mode2.
- **function BECON_GetMode2Response()**; not used.
- **function BECON_DisplayResult(p_iMessageId, p_sLastLogon)**; Used to show the last login date when the login procedure is completed. Uses the DisplayResult(p_iMessageId, p_sLastLogon, l_Language) method of the driver. p_iMessageId, are predefined messages. p_sLastLogon is a variable to send the date and time. l_Language is used to define the language for the predefined language.
- **function BECON_Cancel()** Used when the user has pressed the cancel button.
- **function BECON_GetObject()**; Returns a handle to the USB security token driver object.
- **function BECON_GetLanguage()**; this function returns the language identifier used in the page, e.g. "NL" or "EN".

3.2 Banking process at high-level

This section describes the login procedure and the transaction procedure of internet banking with the e-dentifier2 with the USB security token at an abstract level.

3.2.1 Prerequisites

In order to be able to do the login and transaction procedures (below), the following items need to be in place:

- Working Internet connection;
- Working E-dentifier2 (USB security token);
- Correct drivers installed for the USB security token;
- Active bank account and bankcard from the ABN-AMRO;
- PIN code that corresponds with the bankcard.

3.2.2 Login procedure

Described below is the high level procedure for logging on the internet banking website with the e-dentifier2.

1. The customer opens his browser.

2. The customer plugs the USB security token in a free USB slot of the PC.
3. The customer inserts the bankcard in the USB security token.
4. The customer navigates to https://www.abnamro.nl/nl/logon (Banking application)
5. The USB security token asks for a PIN code.
6. The customer types the PIN code on the keyboard of the USB security token.
7. The customer presses the OK button on the USB security token.
8. USB security token displays last login and the bankcard number.
9. The customer presses the OK button on the USB security token.
10. ABN-AMRO website loads.
11. The customer is logged in.

3.2.3 Transaction procedure

The following section describes the procedure to transfer an amount to a bank account using the USB security token.

1. The customer is logged onto ABN-AMRO website.
2. The customer navigates to transactions.
3. The customer fills in the forms.
4. The customer presses the "OK&Send" button on the PC.
5. The customer selects "Select all transfers & send" on the PC.
6. The customer types the PIN code on the keyboard of the USB security token.
7. The customer presses the OK button on the USB security token.
8. The USB security token displays the number of transactions and total amount of the transfer.
9. The customer presses the OK button on the USB security token.
10. Transaction has occurred.

3.3 Banking process at lower level

This section will explain what occurs on a lower level of the communication between the actors. First the communication for a login and a transaction will be explained for the communication from and towards:

- The banking application and the home PC
- The home PC and USB security token.

Below are the two procedures that form the basis of the internet banking website communication with the e-denitifier2. The top level procedure, this is what the customer sees, is displayed in normal font. *In italics is the communication between the banking website and the home PC is displayed*, **in bold is the communication between the home PC and the USB security token displayed (See section 3.5 of a indepth description of the USB communication).**

3.3.1 Login procedure

1. The customer opens his browser.
2. The customer plugs the USB security token in a free USB slot of the home PC.
   2.1 **home PC sends handshake**
   2.2 **USB security token answers**
3. The customer inserts the bankcard in the USB security token.
4. The customer navigates to https://www.abnamro.nl/nl/logon
   4.1 *The home PC sends HTTP-GET command to retrieve the webpage from the banking application.*
   4.2 *Banking application sends a webpage requesting the browser to check if the USB security token is inserted in the home PC.*
   4.3 **The home PC sends "INIT"**
   4.4 **USB security token answers with bankcard number and account number**
   4.5 *The home PC sends HTTP-POST with the account number and bankcard number.*
   4.6 *The banking application sends webpage requesting to authenticate.*
   4.7 **The home PC sends "ASK PIN"**
5. The customer USB security token asks for a PIN code.
6. The customer types the PIN code on the keyboard of the USB security token.
7. The customer presses the OK button on the USB security token.
   7.1 **USB security token sends "OK"**
   7.2 *The home PC sends HTTP-GET*
   7.3 *Banking application sends a webpage with signdata which is sent to USB security token via javascript interface.*
   7.4 **The home PC sends "SHOW TEXT" and signdata to the USB security token, see section 3.5.1 for more details of the signdata**
8. The USB security token displays last login and the bankcard number.
9. The customer presses the OK button on the USB security token.
   9.1 **The USB security token sends cryptographic data**
   9.2 *The home PC sends HTTP-POST with generated authentication response.*
   9.3 *The banking application sends a webpage with current balance information.*
10. ABN-AMRO website loads.
11. Logged in.

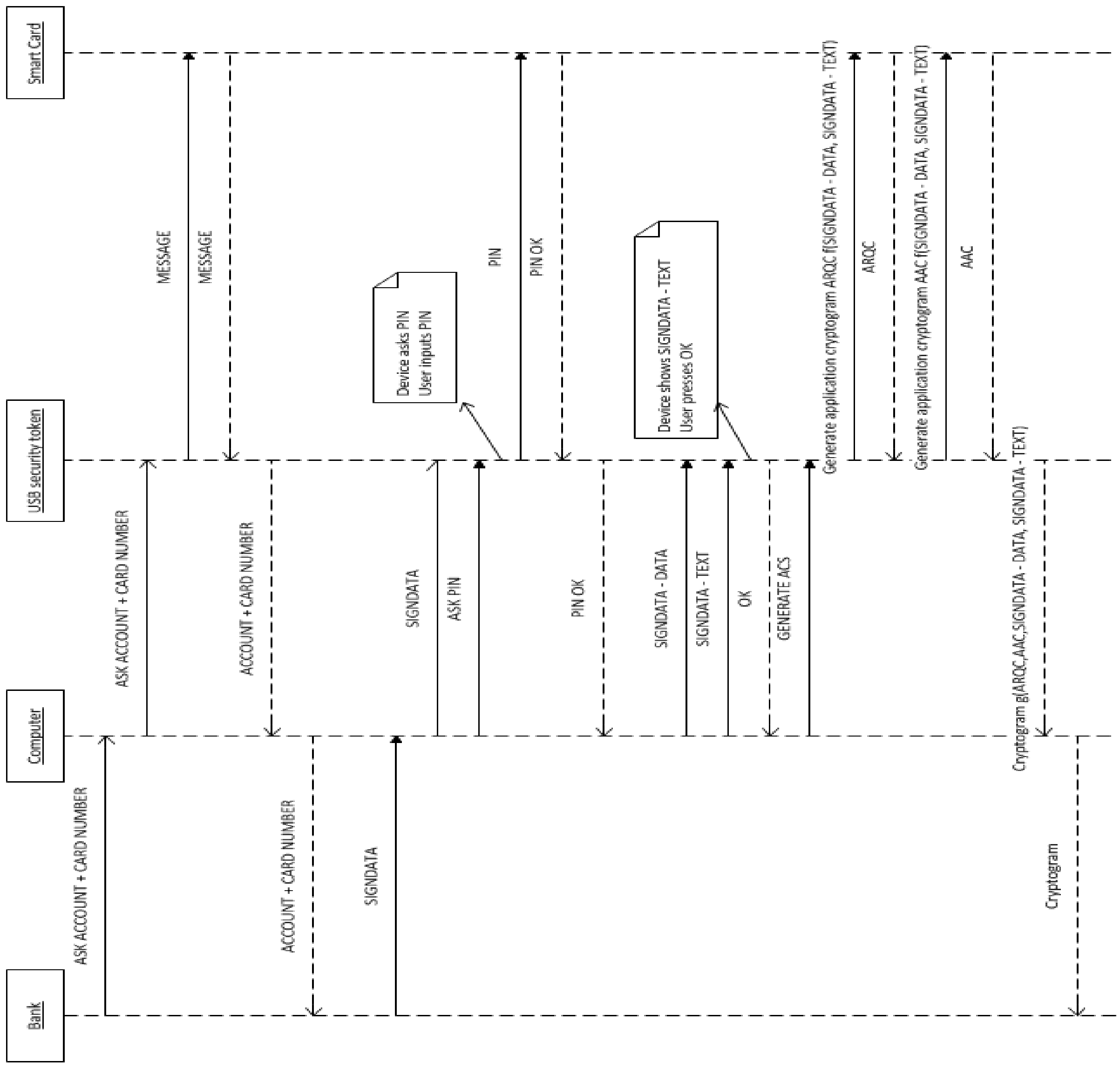
3.3.2 Transaction

1. The customer is logged onto ABN-AMRO website.
2. The customer navigates to transactions.
   2.1 *The home PC sends HTTP-GET*
   2.2 *Banking application sends a webpage with a form.*
3. The customer fills in the forms.
4. The customer presses the "OK&Send" button on the PC.
   4.1 *The home PC sends HTTP-POST with form data.*
   4.2 *Banking application sends webpage with transaction information.*
5. The customer selects "Select all transfers & send" on the PC.
   5.1 *The home PC sends HTTP-POST with form data.*
   5.2 *Banking application sends webpage requesting to authenticate.*
   5.3 **The home PC sends "INIT"**
   5.4 **USB security token answers with bankcard number and account number**
   5.5 **The home PC sends "ASK PIN"**
6. The customer types the PIN code on the keyboard of the USB security token.
7. The customer presses the OK button on the USB security token.
   7.1 **USB security token sends "OK"**
   7.2 *The home PC sends HTTP-GET*
   7.3 *Banking application sends a webpage with sign data which is sent to USB security token via javascript interface.*
   7.4 **The home PC sends "SHOW TEXT" and signdata data, see section 3.5.1 for more details of the signdata**
8. The USB security token displays the number of transactions and total amount of the transfer.
9. The customer presses the OK button on the USB security token.
   9.1 **USB security token sends cryptographic data**
   9.2 *The home PC sends HTTP-POST with generated authentication response (that came from the USB security token)*
   9.3 *Banking application sends a webpage that informs the customer.*
10. Transaction has occurred.

3.4 Sequence diagrams

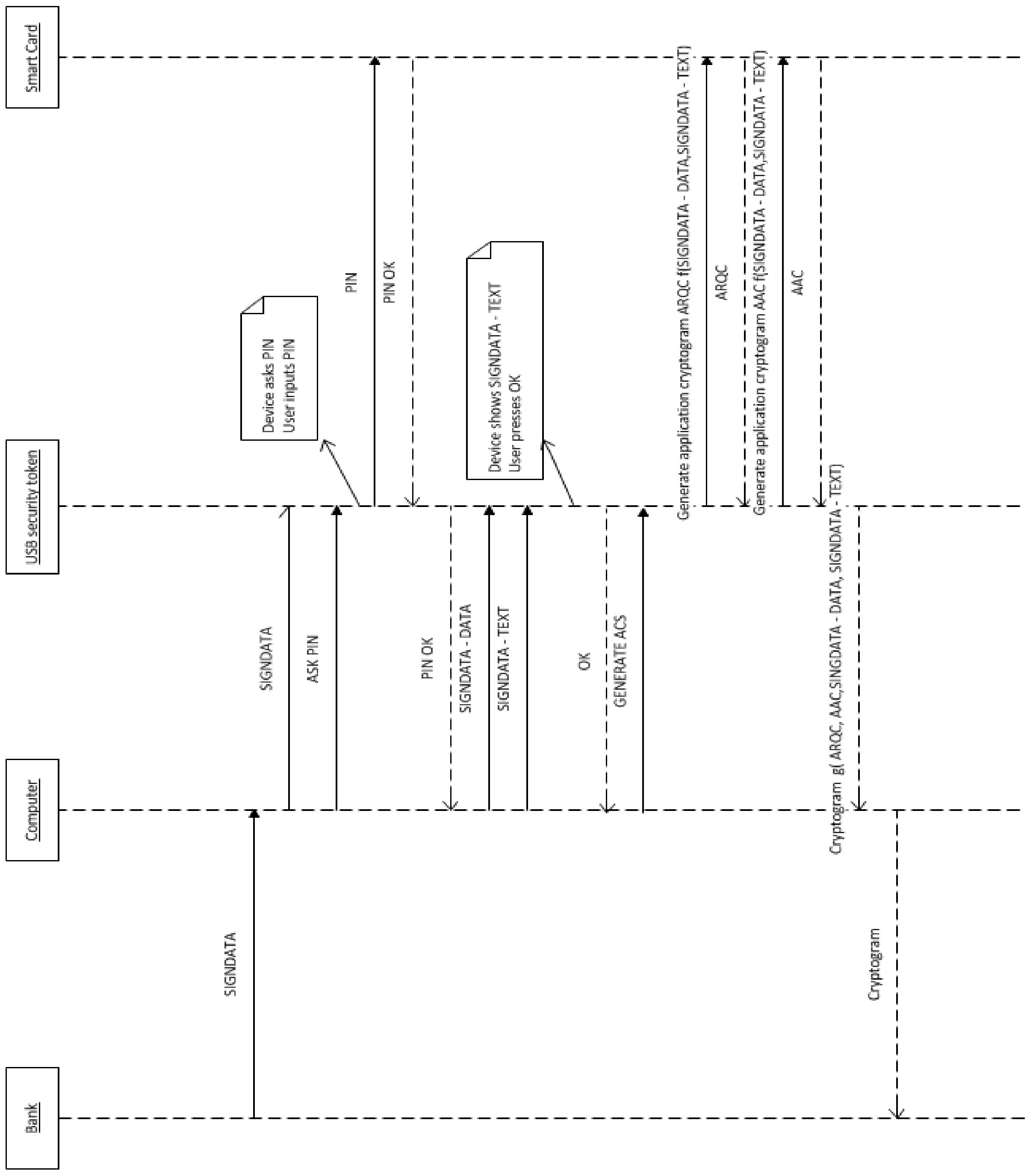
In this section the banking process is depicted as a sequence diagram. These diagrams depict all, including the USB security token and the smartcard, communication between the different actors as depicted in figure 3.1 and described in section 3.3.

The SIGNDATA is one of the core messages used for the sign process. Message SIGNDATA is sent from the bank to the customer and contains data and text. The message is sent to the USB security token twice, namely first as a whole in message SIGNDATA and in parts in messages SIGNDATA-DATA and SIGNDATA-TEXT. Where SIGNDATA-DATA consists out of the data part and SIGNDATA-TEXT out of the text part of the original SIGNDATA message.



Smart Card
USB security token
Computer
Bank
ASK ACCOUNT + CARD NUMBER
ASK ACCOUNT + CARD NUMBER
MESSAGE
MESSAGE
ACCOUNT + CARD NUMBER
ACCOUNT + CARD NUMBER
SIGNDATA
SIGNDATA
ASK PIN
Device asks PIN
User inputs PIN
PIN
PIN OK
PIN OK
SIGNDATA - DATA
SIGNDATA - TEXT
OK
Device shows SIGNDATA - TEXT
User presses OK
GENERATE ACS
Generate application cryptogram ARQC f(SIGNDATA - DATA, SIGNDATA - TEXT)
ARQC
Generate application cryptogram AAC f(SIGNDATA - DATA, SIGNDATA - TEXT)
AAC
Cryptogram g(ARQC,AAC,SIGNDATA - DATA, SIGNDATA - TEXT)
Cryptogram

Figure 3.3: Login sequential diagram, with for each message a name, e.g. SIGNDATA.



Smart Card
USB security token
Computer
Bank
SIGNDATA
SIGNDATA
ASK PIN
Device asks PIN
User inputs PIN
PIN
PIN OK
PIN OK
SIGNDATA - DATA
SIGNDATA - TEXT
Device shows SIGNDATA - TEXT
User presses OK
OK
GENERATE ACS
Generate application cryptogram ARQC f(SIGNDATA - DATA,SIGNDATA - TEXT)
ARQC
Generate application cryptogram AAC f(SIGNDATA - DATA,SIGNDATA - TEXT)
AAC
Cryptogram g(ARQC, AAC,SINGDATA - DATA, SIGNDATA - TEXT)
Cryptogram

Figure 3.4: Payment sequence diagram. Difference with figure 3.3 is that the first communication to read the card data is skipped and the content of the signdata is different.

3.5 Communication between the computer and USB security token

In this section the communication between the computer and the USB security token is described on a lower level.

3.5.1 Signdata

The banking application sends data to the USB security token that contains random data and relevant transaction text. The data, called “Signdata” in the javascript file, is sent by the function BECON_GetResponse(p_sSignData) to the driver of the USB security token. Both modes, login and transaction, use the BECON_GetResponse function. The data is Type-Length-Value (TLV) encoded.

In case of the mode login, the bank sends the following TLV encoded hexdata, data parts are marked in bold (SIGNDATA in figure 3.3):

00 04 **4D D0 D2 DC** 01 44 **55 20 6C 6F 67 74 20 69 6E 20 6D 65 74 3A 20 20 20 52 65 6B 20 35 30 2E 31 32 2E 38 34 2E 34 38 36 20 50 61 73 20 32 31 31 20 20 20 20 20 20 20 20 20 20 42 65 76 65 73 74 69 67 20 6D 65 74 20 4F 4B 20 20 6E 6C**

which consist of:

- type and length:

  00 04

- UNIX time part “Mon May 16 2011 07:31:40 GMT+0200 (CEST)” (SIGNDATA-DATA in figure 3.3) :

  4D D0 D2 DC

- type and length

  01 44

- 68 bytes of text “U logt in met: Rek 50.xx.xx.486 Pas 123 Bevestig met OK nl” (SIGNDATA-TEXT in figure 3.3):

  55 20 6C ... 20 6E 6C

And for the mode “transaction” (BC1 in figure 3.4) :
Hexdata :

00 14 **2C 90 92 02 E9 A9 39 DB 43 8C 1F 47 5A 74 AE 44 FE 98 BB F6** 01 44 **42 65 74 61 6C 65 6E 20 20 20 20 20 20 20 20 20 20 31 20 74 72 61 6E 73 61 63 74 69 65 28 73 29 20 20 45 55 52 20 31 2C 30 30 20 20 20 20 20 20 20 20 20 42 65 76 65 73 74 69 67 20 6D 65 74 20 4F 4B 20 20 6E 6C**

Which consist of:

- type and length:

  00 14

- 14 bytes unidentified part (CU4 in figure 3.4) :

  2C 90 92 02 E9 A9 39 DB 43 8C 1F 47 5A 74 AE 44 FE 98 BB F6

- type and length:

  01 44

- 68 bytes of text “Betalen 1 transactie(s) EUR 1,00 Bevestig met OK nl” (CU5 in figure 3.4):

  42 65 74 ... 20 6E 6C

3.5.2 USB instructions

In the table below are the USB instructions that are send to the USB security token. These are non-standard commands and derived by reverse engineering. The meaning of the commands are derived by educated guesses from the context where they are used and by means of executing them via a custom created driver.

Hex-value	Command
01 03 01 01 00 00 00 00	??UNKNOWN
01 03 01 02 00 00 00 00	Init
01 03 02 00 00 00 00 00	OK
01 03 03 00 00 00 00 00	??UNKNOWN
01 03 04 00 00 00 00 00	ASK PIN
01 03 07 00 00 00 00 00	CANCEL
01 03 05 16 00 00 00 00	SEND SIGNDATA-DATA-transaction
01 03 05 05 00 00 00 00	SEND SIGNDATA-DATA-login
01 03 05 46 00 00 00 00	SEND SIGNDATA-TEXT
01 03 06 00 00 00 00 00	GENERATE CRYPTOGRAM
01 03 08 15 00 00 00 00	LAST LOGIN MESSAGE ? Guess
00 02 6E 6C 00 00 00 00	SET LANG TO NL
00 04 4F 4B 20 20 74 20	END OF TEXT WITH OK.

Chapter 4

Experiments

This chapter describes the experiments performed to determine whether the USB security token indeed signs that which is shown on the display, signs what you see. In short, this means requires validating the following properties:

- That the text displayed on USB security token is genuine and unaltered from the bank.
- At least the displayed data on the USB security token must be used for the sign process. Preferably some random nonce or transaction counter is also included.
- The bank uses the result from the sign process to authorize the transaction.

The following experiments to validate these properties are outlined in this chapter:

- **Manipulating the sign process**; to verify whether the SIGNDATA-DATA and SIGNDATA-TEXT are both used in the sign process and whether SIGNDATA-TEXT is displayed on the USB security token.
- **Double signdata in USB trace**; to discover which payload is used for signing and which one is displayed at the USB security token.
- **Clear display attack**; by sending USB commands without confirmation the display can be cleared.
- **Move Text part attack**; The signdata is split in two separate parts, namely data and text. Both are part of the sign process,a possible attack is to move the text to the data part.

4.1 Manipulating the sign process

This experiment proves that the data sent from the bank (signdata) is actually used in the sign process. Two functions on the USB security token have been identified that are of importance. One of these functions utilizes the USB input, including the signdata and possibly other smartcard related data to create a 4 byte input for the cryptogram creation. For the purposes of this thesis this function is called f. The second function, here called g, has as input the output of the cryptogram creation, the signdata and possibly other smartcard related data. These functions itself are seen as black boxes but their arguments are of significance. To answer the question whether SWYS is true for the USB security token, the arguments of the functions are important: SIGNDATA-TEXT should be one of the arguments, since it is the message that contains the text that is displayed on the USB security token and therefore must be included in the argument of the function. In this section, the arguments of the functions are reverse engineered, in order to validate that at least the SIGNDATA is part of the arguments.

4.1.1 Mapping from signdata to cryptogram input

As discussed in the previous section and in figure 3.3 the bank sends data “Signdata”. This data is used for the cryptogram creation. It should be noted that the payload of the messages for cryptogram generation is only 4 bytes, while the Bank sends SIGNDATA that is at least 76 bytes long. The signdata is transformed by the function, earlier named as “f”, to a 4 byte input for the cryptogram generation. By examining the communication between the home PC and the USB security token one can extrapolate which data the function f uses as arguments for generating the input for the cryptogram generation. The function f possibly relies depend on:

- SIGNDATA-DATA; the data part of the signdata.
- SIGNDATA-TEXT; the text displayed on the USB security token.
- Other data; undefined.

As seen in the communication sequence diagrams in section 3.3 the possible arguments used for function f can be manipulated by altering the SIGNDATA message between the bank and the home PC.

4.1.2 Manipulating the arguments for function f

The SIGNDATA can be manipulated by a man in the middle attack between the bank and the browser. The altered SIGNDATA between the bank and the home PC yields an altered message SIGNDATA between the home PC and the USB security token. Depending on which part of the SIGNDATA is altered, both the fata (SIGNDATA-DATA) and the text part (SIGNDATA-TEXT) or either one is affected. By reviewing the outcome of the function one can determine of which arguments the function is minimally depending on. The result of various manipulations is summarized in the following table:

Nr	Test	Outcome
T1	Two runs with fixed SIGNDATA	The outcome of f remains equal in the two runs.
T2	Two runs with manipulated SIGNDATA-DATA	The outcome of f changes in relation to T1 but remains equal in the two runs.
T3	Two runs with manipulated SIGNDATA-TEXT	The outcome of f changes in relation to T1 and T2 but remains equal in the two runs.
T4	Partially manipulate SIGNDATA-TEXT	The outcome of f changes in relation to T1,T2 and T3

As a results of the performed tests above the following is concluded for the function f:

- Varying with SIGNDATA-DATA and maintaining SIGNDATA-TEXT constant demonstrates that function f depends on SIGNDATA-DATA.

- Varying with SIGNDATA-TEXT and maintaining SIGNDATA-DATA constant demonstrates that function f depends on SIGNDATA-TEXT.

From the test results is concluded that function f has the data and text parts as arguments. Thus the function is depending at least on **(SIGNDATA-DATA,SIGNDATA-TEXT)** as arguments.

4.1.3 Mapping cryptograms to USB response

After both the cryptograms have been generated, a second function, named function “g”, is used to transform (both) the 22 bytes long cryptograms to one 12 bytes long response that is sent back to the home PC and bank. The cryptogram generation is executed twice, GENERATE ARQC which is the Authorisation Request Cryptogram and GENERATE AAC which is the Application Authentication Cryptogram. Both cryptograms depend on function f. Derived from the usage of the function, the function g might depend on:

- SIGNDATA-DATA; the data part of the signdata.
- SIGNDATA-TEXT; the text displayed on the USB security token.
- ARQC; The outcome of the ARQC cryptogram generation.
- AAC; The outcome of the AAC cryptogram generation.
- Other data; undefined.

Manipulating the possible arguments of function g is a daunting task. Sequence diagram 3.4 shows that the data (SIGNDATA-DATA) and text (SIGNDATA-TEXT) part can be manipulated by manipulating the SIGNDATA message between the bank and the home PC but the ARQC and AAC arguments can only be manipulated by altering the message GENERATE ARCQ and GENERATE AAC. This requires manipulating data between the USB security token and the smart card. By reviewing the outcome of the function one can determine the arguments on which the function is depending.

4.1.4 Manipulating the arguments for function g

The SIGNDATA can still be manipulated by the man in the middle attack utilized in section 4.1.1. The ARQC response and AAC response will be altered by using a MITM device that can manipulate messages between the USB security token and the smart card. By manipulating these variables the input of function “‘g” is altered. The result of various manipulations is summarized in the following table:

Nr	Test	Outcome
T1	Multiple runs with fixed ARCQ and AAC responses	Result does not remain equal in the runs.
T2	Two runs with fixed SIGNDATA, ARCQ and AAC	Result remains equal in the two runs.
T3	Two runs with manipulated SIGNDATA-DATA, fixed SIGNDATA-TEXT, ARCQ and AAC	Result changes in relation to T2 but remains equal in the two runs.
T4	Two runs with manipulated SIGNDATA-TEXT, fixed SIGNDATA-DATA, ARCQ and AAC	Result changes in relation to T2 and T3 but remains equal in the two runs.
T5	Partially manipulate SIGNDATA-TEXT, fixed SIGNDATA-DATA, ARCQ and AAC	Result changes in relation to T2, T3 and T4

With the results of the performed test the following is concluded for the function g:

- Varying ARQC and keeping SIGNDATA-TEXT, SIGNDATA-DATA and AAC constant shows that function g depends on cryptogram ARCQ.
- Varying AAC and keeping SIGNDATA-TEXT, SIGNDATA-DATA and ARQC constant shows that function g depends on cryptogram AAC.
- Varying SIGNDATA-DATA and keeping SIGNDATA-TEXT, ARCQ and AAC constant shows that function g depends on SIGNDATA-DATA.
- Varying SIGNDATA-TEXT and keeping SIGNDATA-DATA, ARCQ and AAC constant shows that function g depends on SIGNDATA-TEXT.

From the test results is concluded that function g depends at least on SIGNDATA-DATA, SIGNDATA-TEXT and both the cryptograms as arguments.

conclusion

- The sign process does include the correct arguments for SWYS to be true. The data which is displayed on the screen is included in the cryptogram generation.
- The text displayed on the display of the USB security token is sent in plaintext over USB.
- The inner workings of functions f and g remain unknown. (This severely limits the reuse of the USB security token as a general purpose security token, because these functions are bank specific)

4.2 Double signdata in USB trace

Whilst reverse engineering the USB communication flows from and towards the home PC and the USB security token, some “odd” communication was discovered. The “signdata” is sent twice. In figure 3.3 this is depicted as message SIGNDATA and both SIGNDATA-DATA and SIGNDATA-TEXT. The reason as to why the signdata is sent twice is unclear, but it could indicate that one of the messages is utilized for the display of the USB security token and the other message is used for the signing process. Would this be true, one of the messages could be altered, resulting in that the customer does not sign what is displayed on the screen of the USB security token, thus yielding that SWYS is not true.

In order to further investigate this observation the communication between the home PC and the USB security token, SIGNDATA or SIGNDATA-DATA and SIGNDATA-TEXT, has been altered.

4.2.1 Home PC - USB communication

To alter the USB communication flows from and towards the USB security token, a MITM attack on the communication flows from and towards the home PC and the USB security token is performed. This attack happens at the USB communication level to be successful. To perform this MITM attack I have written a custom driver that made it possible to alter the USB messages. It was no easy endeavour to build such a driver because of the properties of the USB protocol. The USB protocol is designed as a protocol that supports a broad subject of devices. This design choice makes the protocol not strict in the communication flow and enables many design choices to device builders. This level of freedom complicates writing a custom USB device driver for an USB device without device documentation.

An alternative solution for a MITM attack is to relay the USB communication via an USB sharing application. This application relays the communication via the TCP protocol between two operating systems. This way a MITM attack can be executed on the TCP packets, which is easier to do.

4.2.2 Custom USB driver

Publicly available libraries exist that supply the basic needs for USB communication. For the purposes of this thesis *LibUSBdotnet* is used. The goal is to replay the communication of a normal login and transaction run, as captured by USBtrace. Having achieved this the messages SIGNDATA, SIGNDATA-DATA and SIGNDATA-TEXT can be altered.

4.2.3 Altering messages

In order for SWYS to be true, the following properties require validating:

- The SIGNDATA control message should not be of any importance to either the text on the display of the USB security token or the sign process.
- Any alteration to SIGNDATA-DATA and SIGNDATA-TEXT should reflect on both the display of the USB security token and in the function f (sign process).

After altering the SIGNDATA control message, SIGNDATA-DATA and SIGNDATA-TEXT, the purpose of the control message SIGNDATA remains unclear. Altering the control message SIGNDATA appears to have no impact. However, any alteration of SIGNDATA-TEXT is displayed on the screen and reflected in the sign process.

conclusion

A custom USB driver is built that alters the SIGNDATA control message, SIGNDATA-DATA and SIGNDATA-TEXT messages, yielding the following observations:

- Only the SIGNDATA-DATA and SIGNDATA-TEXT are used for the text displayed on the screen of the USB security token and data for the cryptogram generation. The double signdata in the USB trace could be an idiosyncrasy of the USB trace software or it was sent twice. In either case the signdata in the control message is unused.
- Both SIGNDATA-DATA and SIGNDATA-TEXT are displayed and used for the cryptogram generation, as needed for SWYS.

4.3 Clear display & evade customer confirmation of transaction

As mentioned in the previous section a custom USB driver has been written to manipulate the USB messages. An interesting observation was made while trying to recreate a protocol run with the custom USB driver. It is discovered that by sending USB commands which do not strictly adhere to the discussed communication flow, the USB security token would respond differently than expected. In this section the “clear display” attack is explained, which is a result of this observation. The goal of this attack is to avoid the need for a customer to confirm or cancel a money transaction by pressing the “OK”button or Cancel button.

4.3.1 Attack

In a normal run, as depicted in figure 3.3, the home PC waits for the USB security token to reply to the commands sent by the home PC. By slightly adjusting the protocol, as depicted in figure 4.1, the message that should be displayed after the message SIGNDATA-TEXT is no longer displayed.

The USB security token is instructed by the home PC to perform certain tasks and is then regularly asked by the home PC (polled) whether the task is finished. The driver of the USB security token is designed to wait for the USB security token to respond to certain tasks before sending new messages.

By manipulating the timing of messages to the USB security token one can manipulate whether the user has to confirm a transaction. In a normal sequence a customer enters his pin and is then presented with a message on the display of the USB security token. This message has to be confirmed or cancelled by the customer in order to continue with the transaction sequence. In the attack scenario the driver is altered to directly send the GENERATE AC command without waiting for the customer’s confirmation of the transaction. As a result the message on the display asking the customer for a confirmation of the transaction is cleared and the transaction sequence continues as normal. This results in a situation where any transaction can be automatically confirmed without the customer knowing. For the bank this attack is not different than a normal transaction run and hence cannot be detected. Depending on the speed of cryptogram generation by the bank card, the confirmation message will disappear from the display. Normally this will take less than a second (somewhere in the order of half of a second). This attack is depicted in figure 4.1.

4.3.2 Custom USB driver

For the aforementioned attack the custom USB driver is utilized as discussed in section 4.2.2. The driver replays the pre-captured login or transaction sequence. This sequence is then adjusted to ignore the ok message that is sent by the USB security token and directly send the GENERATE AC command. The custom driver is able to replay and adjust the communication flow. Unfortunately the custom driver is not able to obtain the generated cryptogram returned by the USB security token. However, the logging data (between the USB security token and bank card) shows that the cryptogram is generated by the bank card. This proves the attack to be effective for 90% of a sequence. The remaining 10%, being the returned cryptogram message, has not been covered by the author.

To complete a real world scenario and thus fully proving the validity of the attack, Gerhard de Koning Gans presents an alternative solution. He has created a filter application by utilizing an USB sharing application. The filter application is able to intercept and alter the USB communication in real-time by directing it via the TCP stream. The result of the combined work is a fully functional real world attack scenario which demonstrates the effectiveness of the discussed attack.

4.3.3 Prevention

An important question that immediately surfaces is how to prevent this attack. This attack exists because the driver instructs the USB security token to generate the cryptograms after the customer has

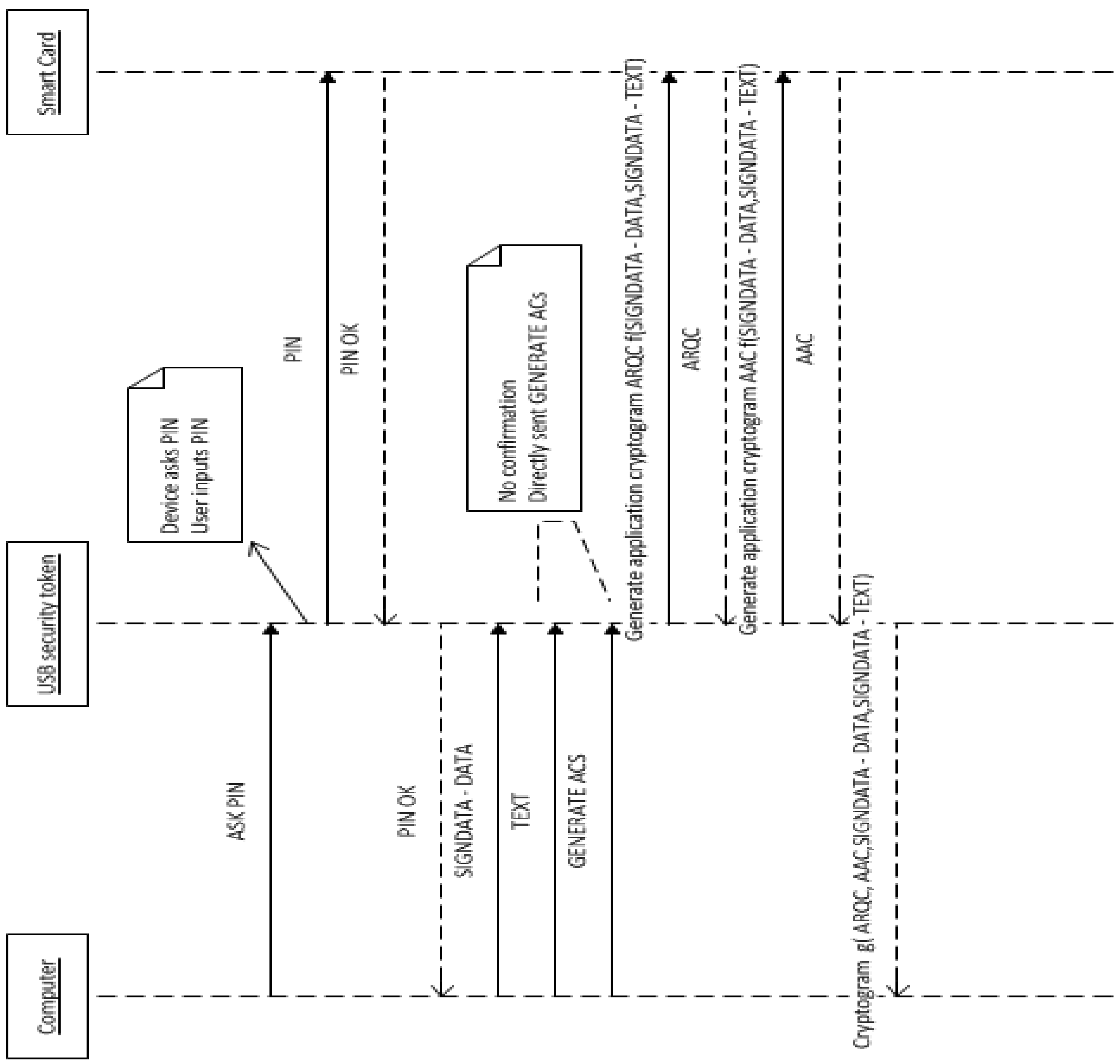

Computer
USB security token
Smart Card
ASK PIN
Device asks PIN
User inputs PIN
PIN
PIN OK
PIN OK
SIGNDATA - DATA
TEXT
GENERATE ACS
No confirmation
Directly sent GENERATE ACs
Generate application cryptogram ARQC f(SIGNDATA - DATA,SIGNDATA - TEXT)
ARQC
Generate application cryptogram AAC f(SIGNDATA - DATA,SIGNDATA - TEXT)
AAC
Cryptogram g(ARQC, AAC,SIGNDATA - DATA,SIGNDATA - TEXT)

Figure 4.1: Atack sequential diagram. After SIGNDATA-TEXT the GENERATE AC is directly send.

pressed the “ok” button. This means that the driver is in control where the driver should not be in control. One possible solution is that after the signdata is sent, the USB security token decides if the cryptograms are generated or a cancel message is returned. The cryptograms should only be generated when the user has pressed the OK button. External influences are no longer possible, and the only way the cryptograms are generated is when the user has confirmed the transaction.

conclusion

- The attack is successful. By directly sending the GENERATE AC command after the SIGNDATA-TEXT, the confirmation text on the display disappears and the USB security token generates the cryptograms without customer confirmation.
- Because the attack is successful SWYS is **not** true for the USB security token.

4.4 Move Text part attack

As discussed in section 3.5.1 both signdata parts are encoded using TLV. Also both SIGNDATA-DATA and SIGNDATA-TEXT are used as input of the cryptogram creation but only the SIGNDATA-TEXT is utilized to display TEXT on the display. Because of this property there could be a possibility that the SIGNDATA-TEXT could be moved and added to the SIGNDATA-DATA part by altering the TLV encoding. One hypothesis is that by moving the SIGNDATA-TEXT and adding it to the SIGNDATA-DATA part could result in a correct cryptogram without any text on the display of the USB security token. In order for SWYS to be true the modification to the SIGNDATA-TEXT and SIGNDATA-DATA parts should not result in a valid cryptogram.

4.5 Manipulating the SIGNDATA messages

For this experiment the TLV encoding of the signdata messages are altered. The signdata messages start with 2 bytes of TLV encoding. The first byte describes the section that follows, 00 for the SIGNDATA-DATA part and 01 for the SIGNDATA-TEXT part. The second byte defines the number of bytes that follow. By chancing the second byte, the length of the message can be altered. In the experiments the second byte is changed in the SIGNDATA-DATA part to add the SIGNDATA-TEXT part. The SIGNDATA-TEXT part was left empty. The USB security token did not accept this altered message. Small alterations to the message length where accepted by the USB security token but also had their reflection on the outcome of the cryptogram generation.

conclusion

- This experiment did not have the desired results, the USB security token did not accept the altered messages.
- SWYS is true as for this experiment.

Chapter 5

Future work

This chapter contains the work that could not be finished in time for this thesis but seems interesting to investigate further.

- **Function f and g**; The functions f and g are used in the USB security token for the input of the cryptogram generation and response generation. For this Master thesis the inner workings of the functions where seen as a black box (see chapter 4.1). It is very interesting to find out what the functions actually do to transform the input. Also the arguments of the functions where determined in section 4.1. I could not rule out that there were any other arguments where the function f and g might depend on. In future research the inner workings of the functions are interesting investigate further and to rule out any other dependencies of arguments of the functions would be interesting. This could eventually lead to a bank independent card reader.
- **USB MITM software**; Write software that makes it possible to intercept and change USB messages in real time. This would be beneficial for doing research in USB protocols and devices that use the USB as a communication line.
- **USB instructions**; Further research the 8 byte USB instructions sent between the computer and the USB security token for unkown working instructions.

Chapter 6

Conclusion

The presented work discusses the following main research question:

Has the USB security token implemented Sign What You See (SWYS) correctly?

In order to answer the research question, the first step comprised the reverse engineering of communication flows from and towards the USB security token. The exact communication flow for the login and transaction procedures are depicted in Section 3.4.

Utilizing the extrapolated communication flows, several experiments have been conducted which demonstrate that additional logic exists in the USB security token. Certain message alterations are not accepted as valid communication messages and the USB security token utilizes two unknown functions during the login and transaction procedures. Whilst these functions themselves are not within the scope of the conducted research, their arguments are. The first function, named "f" in this thesis, uses at least SIGNDATA-DATA and SIGNDATA-TEXT as function arguments. The result of "f is used as input for generating the cryptogram. The second function, named "g", uses at least the generated cryptogram, SIGNDATA-DATA and SIGNDATA-TEXT as function arguments. Being able to conduct such low-level experiments demonstrates the weak security of the device. Because the messages between the bank and the USB security token are not encrypted, the communication flows are exposed and exploitable as has been shown in Section 3.4. In an ideal setting the conducted research should have ended prematurely with the result that all communication from and towards the device is encrypted.

The observation that communication flows are not encrypted enabled experiments with the communication sequences and the payload of messages. These experiments resulted in the discovery of a feasible attack on the USB security token. This attack enables a scenario in which a customer unknowingly signs a transaction. The attack is feasible because the confirmation message can be cleared from the screen of the USB security token and no customer approval is required for the generation of the cryptograms. See Section 4.3 for the specifics of the described attack.

The discussed attack answers the main research question whether SWYS has been correctly implemented adequately with: NO! In a strict sense one could argue that the customer indeed signs what he sees, because the signdata that is sent by the bank is displayed on the screen of the USB security token and the signdata plays an integral role during the cryptogram generation. These two properties are important for SWYS. It is impossible to show the customer a certain message A, whilst a different message B is signed. As has been demonstrated by means of the discussed attack, this argumentation is flawed because it is possible to show the user the signdata for a (very) brief period of time and no confirmation is required to approve the transaction. Rather than a full implementation of SWYS, this is rather an implementation of Sign What You Have Briefly Seen, Without Confirmation (SWYHBSWC). Given that the approval step is crucial for SWYS, one can only conclude that SWYS simply does not hold for the USB security token.

This quickly yields the question as to whether the vulnerability was avoidable. The answer is that the problem was indeed avoidable, as the attack is feasible due to a rookie design mistake. The confirmation step and the generation of the cryptograms shouldn't be two separate steps directed by the driver. These two steps should be an atomic operation in the USB security token. Resulting in an operation on the USB security token that receives the signdata and, depending on the action of the customer, returns either a cancel message or a cryptogram.

Appendix A

Login, USB security token - Card (USB connected)

This appendix describes APDU traces for the login procedure with the USB security token connected to the computer with a USB cable, as described in 3.3 and in figure 3.3. Section A.1 describes the difference between two runs of the protocol.

Obtained used: rebelsim / MTM device
Account number: 501284486
Card number: 211

1. #INSERT CARD#
2. USB Security token : 3B 67 00 00
3. Bank card : 29 20 00 6F 78 90 00 00
4. USB Security token: 3B 67 00 00
5. Bank card : 29 20 00 6F 78 90 00 00
6. #CUSTOMER PRESSES LOGIN#
7. ==
8. Select Application, SecureCode authentication
9. ==
10. USB Security token : 00 A4 04 00 07 A0 00 00 00 04 80 02
11. Bank card : 61 27
12. USB Security token : 00 C0 00 00 27
13. ==

14. Returns TLV code, File Control Information (FCI) Template
    FCI:securecade auth
    Lang pref: nlen

15. ==

16. Bank card : 6F 25 84 07 A0 00 00 00 04 80 02 A5 1A 50 0E 53 65 63 75 72 65 43 6F 64 65 20 41
    75 74 87 01 00 5F 2D 04 6E 6C 65 6E 90 00

17. ==

18. Get processing options

19. ==

20. USB Security token : 80 A8 00 00 02 83 00

21. Bank card : 61 0C

22. USB Security token : 00 C0 00 00 0C

23. ==

24. Returns TLV code, Response Message Template Format2.
    Application Interchange Profile : 2
    Application File Locator (AFL) : 08010100

25. ==

26. Bank card : 77 0A 82 02 10 00 94 04 08 01 01 00 90 00

27. ==

28. Read record

29. ==

30. USB Security token : 00 B2 01 0C 00

31. Bank card : 6C 62

32. USB Security token : 00 B2 01 0C 62

33. ==

34. Bank card returns EMV Proprietary Template
    Card Risk Management Data Object List 1 (CDOL1)
    9F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403
    Card Risk Management Data Object List 2 (CDOL2)
    910A8A0295059F37049F4C08
    Application Primary Account Number (PAN)
    6734000501284486219F
    Application Primary Account Number (PAN) Sequence Number
    01
    Cardholder Verification Method (CVM) List
    0000000000000000100
    9F55 Unknown tag

80
CAP Bitfilter:
00007FFFFFE0000000000000

35. ===
36. Bank card : 70 60 8C 21 9F 02 06 9F 03 06 9F 1A 02 95 05 5F 2A 02 9A 03 9C 01 9F 37 04 9F 35 01 9F 45 02 9F 4C 08 9F 34 03 8D 0C 91 0A 8A 02 95 05 9F 37 04 9F 4C 08 5A 0A 67 34 00 *05 01 28 44 86* 21 9F 5F 34 01 01 8E 0A 00 00 00 00 00 00 00 00 01 00 9F 55 01 80 9F 56 0C 00 00 7F FF FF E0 00 00 00 00 00 00 90 00
37. ===
38. Reset, following lines are the same as lines 7-21
39. ===
40. USB Security token : 00 A4 04 00 07 A0 00 00 00 04 80 02
41. Bank card : 61 27
42. USB Security token : 00 C0 00 00 27
43. Bank card : 6F 25 84 07 A0 00 00 00 04 80 02 A5 1A 50 0E 53 65 63 75 72 65 43 6F 64 65 20 41 75 74 87 01 00 5F 2D 04 6E 6C 65 6E 90 00
44. USB Security token : 80 A8 00 00 02 83 00
45. Bank card : 61 0C
46. USB Security token : 00 C0 00 00 0C
47. Bank card : 77 0A 82 02 10 00 94 04 08 01 01 00 90 00
48. USB Security token : 00 B2 01 0C 00
49. Bank card : 6C 62
50. USB Security token : 00 B2 01 0C 62
51. Bank card : 70 60 8C 21 9F 02 06 9F 03 06 9F 1A 02 95 05 5F 2A 02 9A 03 9C 01 9F 37 04 9F 35 01 9F 45 02 9F 4C 08 9F 34 03 8D 0C 91 0A 8A 02 95 05 9F 37 04 9F 4C 08 5A 0A 67 34 00 *05 01 28 44 86* 21 9F 5F 34 01 01 8E 0A 00 00 00 00 00 00 00 00 01 00 9F 55 01 80 9F 56 0C 00 00 7F FF FF E0 00 00 00 00 00 00 90 00
52. ===
53. End of reset
54. ===
55. USB Security token : 80 CA 9F 17 00
56. Bank card : 6C 04
57. USB Security token : 80 CA 9F 17 04
58. Bank card : 9F 17 01 03 90 00

59. #CUSTOMER INPUTs PIN 1234#
60. USB Security token : 00 20 00 80 08 24 12 34 FF FF FF FF FF
61. Bank card : 90 00
62. #CUSTOMER PRESS OK#
63. ===
64. Generate application cryptogram (ARQC)
65. ===
66. USB Security token : 80 AE 80 00 2B
67. USB Security token : 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00 00 00 00 00 00 ***1D B4 B4 6C*** 34 00 00 00 00 00 00 00 00 00 00 01 00 02
68. Bank card : 61 2B
69. USB Security token : 00 C0 00 00 2B
70. ===
71. Response cryptogram (ARQC)
72. ===
73. Bank card : 77 29 9F 27 01 80 9F 36 02 00 ***C6*** 9F 26 08 ***F3 D8 2E 23 34 63 4D 5C*** 9F 10 12 00 10 A5 00 03 02 00 00 00 00 00 00 00 00 00 00 00 FF 90 00
74. ===
75. Generate application cryptogram (AAC)
76. ===
77. USB Security token : 80 AE 00 00 1D
78. USB Security token : 00 00 00 00 00 00 00 00 00 00 5A 33 80 00 00 00 00 ***1D B4 B4 6C*** 00 00 00 00 00 00 00 00 00
79. Bank card : 61 2B
80. USB Security token : 00 C0 00 00 2B
81. ===
82. Response cryptogram (AAC)
83. ===
84. Bank card : 77 29 9F 27 01 00 9F 36 02 00 ***C6*** 9F 26 08 ***BB 7F 26 FC 37 D5 80 5D*** 9F 10 12 00 10 25 00 03 42 00 00 00 00 00 00 00 00 00 00 00 FF 90 00
85. #CUSTOMER lOGGED IN#

A.1 Difference between login run 1 and run 2

52. USB security token: 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00 00 00 00 00 00 ***01 F4 FF 93*** 34 00 00 00 00 00 00 00 00 00 00 01 00 02

55. Bank card : C0 77 29 9F 27 01 80 9F 36 02 00 ***C2*** 9F 26 08 ***97 DD 4A 08 33 49 DE 7E*** 9F 10 12 00 10 A5 00 03 02 00 00 00 00 00 00 00 00 00 00 00 FF 90 00

58. USB Security token : 00 00 00 00 00 00 00 00 00 00 5A 33 80 00 00 00 00 ***01 F4 FF 93*** 00 00 00 00 00 00 00 00

61. Bank card : C0 77 29 9F 27 01 00 9F 36 02 00 ***C2*** 9F 26 08 ***35 F4 05 8F E4 29 32 77*** 9F 10 12 00 10 25 00 03 42 00 00 00 00 00 00 00 00 00 00 00 FF 90 00 00

Appendix B

Login, Computer - USB security token (USB connected)

This appendix describes USB traces for the login procedure with the USB security token connected to the computer with a USB cable, as described in 3.3 and in figure 3.3. Section B.1 describes the difference between two runs of the protocol.

Obtained using: USBtrace
Account number: 501284486
Card number: 211
Request:OUT is outbound communication to the USB security token.
Request:IN is inbound communication from the USB security token.
EP: These are the USB endpoints, 0 global configuration message endpoint, 2 the USB security token endpoint that accepts data, 81 the USB Security token endpoint that transmits data.

Description of trace:

- Request bank account and card number.
- Response with bank account and card number.
- Sends time + “logt in met:Rek 50.12.84.486 Pas? 211Bevestig met OK nl”
- Ask for PIN code.
- Send data in blocks of 6 bytes
- Send text in blocks of 6 bytes
- Response with cryptogram.
- Send last login time.

Set language to “NL”

Nr	Type	Request	IO	EP	Data
001	URB	BULK	OUT	2	01 03 02 00 00 00 00 00
002	URB	BULK	IN	81	00 06 30 31 2E 30 32 00
003	DEV-CTRL	-	OUT	0	***01 00 00*** 00
004	URB	BULK	OUT	2	02 01 00 00 00 00 00 00
005	URB	BULK	IN	81	00 06 3B 67 00 00 29 20
006	URB	BULK	IN	81	00 05 00 6F 78 90 00 20
007	DEV-CTRL	-	OUT	0	01 02 08 00
008	DEV-CTRL	-	OUT	0	**6E 6C**

Returns the account number 05 01 28 44 86

Nr	Type	Request	IO	EP	Data
009	URB	BULK	OUT	2	01 03 01 02 00 00 00 00
010	URB	BULK	OUT	2	00 02 6E 6C 00 00 00 00
011	URB	BULK	IN	81	01 03 01 01 00 00 00 00
012	URB	BULK	IN	81	00 01 01 01 00 00 00 00
013	URB	BULK	OUT	2	01 03 03 00 00 00 00 00
014	URB	BULK	IN	81	01 03 04 0D 00 00 00 00
015	URB	BULK	IN	81	00 06 0A 67 34 00 **05 01**
016	URB	BULK	IN	81	00 06 **28 44 86** 21 9F 01

Hex decode: 00044D D0 D2 DC 0144 + "logt in met:Rek 50.12.84.486 Pas? 211Bevestig met OK nl"

Nr	Type	Request	IO	EP	Data
017	URB	BULK	IN	81	00 01 01 44 86 21 9F 01
018	DEV-CTRL	-	OUT	0	00 04 ***4D D0 D2 DC*** 01 44 55 20 6C 6F 67 74 20 69 6E 20 6D 65 74 3A 20 20 20 52 65 6B 20 35 30 2E 31 32 2E 38 34 2E 34 38 36 20 50 61 73 20 32 31 31 20 20 20 20 20 20 20 20 20 20 42 65 76 65 73 74 69 67 20 6D 65 74 20 4F 4B 20 20 6E 6C

"ASK PIN" command

Nr	Type	Request	IO	EP	Data
019	URB	BULK	OUT	2	01 03 01 02 00 00 00 00
020	URB	BULK	OUT	2	00 02 6E 6C 00 00 00 00
021	URB	BULK	IN	81	01 03 01 01 00 00 00 00
022	URB	BULK	IN	81	00 01 01 01 00 00 00 00
023	URB	BULK	OUT	2	01 **03 04 00 00 00 00 00**

Send 00 04 4D D0 D2 DC $SIGNDATA - DATApart$

Nr	Type	Request	IO	EP	Data
024	URB	BULK	IN	81	01 03 02 00 00 00 00 00
025	URB	BULK	OUT	2	01 03 05 06 00 00 00 00

Start to send "logt in met:Rek 50.12.84.486 Pas? 211Bevestig met OK nl" 6 bytes a time

Nr	Type	Request	IO	EP	Data
026	URB	BULK	OUT	2	00 06 00 04 ***4D D0 D2 DC***
027	URB	BULK	IN	81	01 03 02 00 00 00 00 00
028	URB	BULK	OUT	2	01 03 05 46 00 00 00 00

End of transmission

Nr	Type	Request	IO	EP	Data
029	URB	BULK	OUT	2	00 06 01 44 55 20 6C 6F
030	URB	BULK	OUT	2	00 06 67 74 20 69 6E 20
031	URB	BULK	OUT	2	00 06 6D 65 74 3A 20 20
032	URB	BULK	OUT	2	00 06 20 52 65 6B 20 35
033	URB	BULK	OUT	2	00 06 30 2E 31 32 2E 38
034	URB	BULK	OUT	2	00 06 34 2E 34 38 36 20
035	URB	BULK	OUT	2	00 06 50 61 73 20 32 31
036	URB	BULK	OUT	2	00 06 31 20 20 20 20 20
037	URB	BULK	OUT	2	00 06 20 20 20 20 20 42
038	URB	BULK	OUT	2	00 06 65 76 65 73 74 69
039	URB	BULK	OUT	2	00 06 67 20 6D 65 74 20
040	URB	BULK	OUT	2	00 04 4F 4B 20 20 74 20

received cryptogram from card 80 00 C6 12 29 8E EB 00 10 A5 00 03

Nr	Type	Request	IO	EP	Data
041	URB	BULK	IN	81	01 03 02 00 00 00 00 00
042	URB	BULK	OUT	2	01 03 06 00 00 00 00 00
043	URB	BULK	IN	81	01 03 03 19 00 00 00 00
044	URB	BULK	IN	81	00 06 80 00 ***C6 12 29 8E***
045	URB	BULK	IN	81	00 06 ***EB*** 00 10 A5 00 03

Nr	Type	Request	IO	EP	Data
047	URB	BULK	IN	81	00 06 02 00 00 00 00 00
048	URB	BULK	IN	81	00 06 00 00 00 00 00 00
049	URB	BULK	IN	81	00 01 FF 00 00 00 00 00
050	DEV-CTRL	-	OUT	0	03 03 09 00
051	DEV-CTRL	-	OUT	0	01 02 08 00
052	DEV-CTRL	-	OUT	0	03 31 36 20 6D 65 69 20 31 31 2C 20 30 ***39*** 3A ***30 34*** 75 00 6E 6C

Hex decode: “16 mei 11, 09:04u?nl”

Nr	Type	Request	IO	EP	Data
053	URB	BULK	OUT	2	01 03 08 15 00 00 00 00
054	URB	BULK	OUT	2	00 06 03 31 36 20 6D 65
055	URB	BULK	OUT	2	00 06 69 20 31 31 2C 20
056	URB	BULK	OUT	2	00 06 30 ***39*** 3A ***30 34*** 75
057	URB	BULK	OUT	2	00 03 00 6E 6C ***30 34*** 75
058	URB	BULK	IN	81	01 03 02 00 00 00 00 00
059	DEV-CTRL	-	OUT	0	03 03 09 00
060	DEV-CTRL	-	OUT	0	01 02 08 00
061	URB	BULK	OUT	2	02 0B 00 00 00 00 00 00

B.1 Difference between login run 1 and run 2

Hex decode: “logt in met:Rek 50.12.84.486 Pas? 211Bevestig met OK nl”

Nr	Type	Request	IO	EP	Data
024	DEV-CTRL	-	OUT	0	***03 03 09*** 00
120	DEV-CTRL	-	OUT	0	00 04 ***4D D0 CB F2*** 01 44 55 20 6C 6F 67 74 20 69 6E 20 6D 65 74 3A 20 20 20 52 65 6B 20 35 30 2E 31 32 2E 38 34 2E 34 38 36 20 50 61 73 20 32 31 31 20 20 20 20 20 20 20 20 20 20 42 65 76 65 73 74 69 67 20 6D 65 74 20 4F 4B 20 20 6E 6C

cryptogram 80 00 C2 61 A6 E8 E7 00 10 A5 00 03

Nr	Type	Request	IO	EP	Data
171	URB	BULK	OUT	2	00 06 00 04 4D D0 ***CB F2***
303	URB	BULK	IN	81	00 06 80 00 ***C2 61 A6 E7***
310	URB	BULK	IN	81	00 06 ***E7*** 00 10 A5 00 03

Hex decode: “16 mei 11, 08:35u?nl”

Nr	Type	Request	IO	EP	Data
345	DEV-CTRL	-	OUT	0	03 31 36 20 6D 65 69 20 31 31 2C 20 30 ***38*** 3A ***33 35*** 75 00 6E 6C

Nr	Type	Request	IO	EP	Data
367	URB	BULK	OUT	2	00 06 30 ***38*** 3A ***33 35*** 75
374	URB	BULK	OUT	2	00 03 00 6E 6C ***33 35*** 75

Appendix C

Transaction, USB security token - Card (USB connected)

This appendix describes APDU traces for the transaction procedure with the USB security token connected to the computer with a USB cable, as described in 3.3 and in figure 3.4.

Obtained used: rebelsim / MTM device
Account number: 501284486
Card number: 211

1. #INIT#
2. USB Security token : 3B 67 00 00
3. Bank card : 29 20 00 6F 78 90 00 00
4. ==
5. Select Application, SecureCode authentication
6. ==
7. USB Security token : A4 04 00 07 A0 00 00 00 04 80 02
8. Bank card : 61 27
9. USB Security token : 00 C0 00 00 27
10. ==
11. Returns TLV code, File Control Information (FCI) Template
    FCI:securecade auth
    Lang pref: nlen
12. ==

13. Bank card : 6F 25 84 07 A0 00 00 00 04 80 02 A5 1A 50 0E 53 65 63 75 72 65 43 6F 64 65 20 41 75 74 87 01 00 5F 2D 04 6E 6C 65 6E 90 00

14. ==

15. Get processing options

16. ==

17. USB Security token : 80 A8 00 00 02 83 00

18. Bank card : 61 0C

19. USB Security token : 00 C0 00 00 0C

20. ==

21. Returns TLV code, Response Message Template Format2.
    Application Interchange Profile : 2
    Application File Locator (AFL) : 08010100

22. ==

23. Bank card : 77 0A 82 02 10 00 94 04 08 01 01 00 90 00

24. ==

25. Read record

26. ==

27. USB Security token : 00 B2 01 0C 00

28. Bank card : 6C 62

29. USB Security token : 00 B2 01 0C 62

30. ==

31. Bank card returns EMV Proprietary Template
    Card Risk Management Data Object List 1 (CDOL1)
    9F02069F03069F1A0295055F2A029A039C019F37049F35019F45029F4C089F3403
    Card Risk Management Data Object List 2 (CDOL2)
    910A8A0295059F37049F4C08
    Application Primary Account Number (PAN)
    6734000501284486219F
    Application Primary Account Number (PAN) Sequence Number
    01
    Cardholder Verification Method (CVM) List
    00000000000000000100
    9F55 Unknown tag
    80
    CAP Bitfilter
    00007FFFFFE0000000000000

32. ==

33. Bank card : 70 60 8C 21 9F 02 06 9F 03 06 9F 1A 02 95 05 5F 2A 02 9A 03 9C 01 9F 37 04 9F 35 01 9F 45 02 9F 4C 08 9F 34 03 8D 0C 91 0A 8A 02 95 05 9F 37 04 9F 4C 08 5A 0A 67 34 00 05 01 28 44 86 21 9F 5F 34 01 01 8E 0A 00 00 00 00 00 00 00 00 01 00 9F 55 01 80 9F 56 0C 00 00 7F FF FF E0 00 00 00 00 00 00 90 00

34. USB Security token : 80 CA 9F 17 00

35. Bank card : 6C 04

36. USB Security token : 80 CA 9F 17 04

37. Bank card : 9F 17 01 03 90 00

38. # CUSTOMER INPUTS PIN 1234#

39. USB Security token : 00 20 00 80 08 24 12 34 FF FF FF FF FF

40. Bank card : 90 00

41. #CUSTOMER PRESSES OK#

42. ==

43. Generate application cryptogram (ARQC)

44. ==

45. USB Security token : 80 AE 80 00 2B

46. USB Security token : 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00 00 00 00 00 00 00 3B 2F 70 90 34 00 00 00 00 00 00 00 00 00 00 01 00 02

47. Bank card : 61 2B

48. USB Security token : 00 C0 00 00 2B

49. ==

50. Response cryptogram (ARQC)

51. ==

52. Bank card : 77 29 9F 27 01 80 9F 36 02 00 C7 9F 26 08 FC BE ED A5 FB 81 C5 67 9F 10 12 00 10 A5 00 03 02 00 00 00 00 00 00 00 00 00 00 00 00 FF 90 00

53. ==

54. Generate application cryptogram (AAC)

55. ==

56. USB Security token : 80 AE 00 00 1D

57. USB Security token : 00 00 00 00 00 00 00 00 00 00 5A 33 80 00 00 00 00 3B 2F 70 90 00 00 00 00 00 00 00 00

58. Bank card : 61 2B

59. USB Security token : 00 C0 00 00 2B

60. ==

61. Response cryptogram (AAC)

62. ==

63. Bank card : 77 29 9F 27 01 00 9F 36 02 00 C7 9F 26 08 7B 85 E3 33 B8 AF C7 A7 9F 10 12 00 10 25 00 03 42 00 00 00 00 00 00 00 00 00 00 FF 90 00 00 3B 67 00 00 29 20 00 6F 78 90 00 00

Appendix D

Transaction, Computer - USB security token (USB connected)

This appendix describes USB traces for the transaction procedure with the USB security token connected to the computer with a USB cable, as described in 3.3 and in figure 3.4.

Obtained using: USBtrace
Account number: 501284486
Card number: 211
Request:OUT is outbound communication to the USB Security token.
Request:IN is inbound communication from the USB Security token.
EP: These are the USB endpoints, 0 global configuration message endpoint, 2 the USB security token endpoint that accepts data, 81 the USB Security token endpoint that transmits data.

Description of trace:

- Sends data + "Betalen 1 transactie(s) EUR 1,00 Bevestig met OK nl"
- Ask for PIN code.
- Send data in blocks of 6 bytes
- Send Text in blocks of 6 bytes
- Response with cryptogram.

Hex decode: 00 14 2C 90 92 02 E9 A9 39 DB 43 8C 1F 47 5A 74 AE 44 FE 98 BB F6 01 44 + "Betalen 1 transactie(s) EUR 1,00 Bevestig met OK nl"

Nr	Type	Request	IO	EP	Data
Nr	Type	Request	IO	EP	Data
001	DEV-CTR	-	OUT	0	00 14 2C 90 92 02 E9 A9 39 DB 43 8C 1F 47 5A 74 AE 44 FE 98 BB F6 01 44 42 65 74 61 6C 65 6E 20 20 20 20 20 20 20 20 20 20 3120 74 72 61 6E 73 61 63 74 69 65 28 73 29 20 20 45 55 52 20 31 2C 30 30 20 20 20 20 20 20 20 20 20 42 65 76 65 73 74 69 67 20 6D 65 74 20 4F 4B 20 20 6E 6C

“ASK PIN” command

Nr	Type	Request	IO	EP	Data
002	URB	BLUK	OUT	2	01 03 01 02 00 00 00 00
003	URB	BLUK	OUT	2	00 02 6E 6C 00 00 00 00
004	URB	BLUK	IN	81	01 03 01 01 00 00 00 00
005	URB	BLUK	IN	81	00 01 01 01 00 00 00 00
006	URB	BLUK	OUT	2	01 03 04 00 00 00 00 00

Start to send SIGNDATA-DATA

Nr	Type	Request	IO	EP	Data
007	URB	BULK	IN	81	01 03 02 00 00 00 00 00
008	URB	BULK	OUT	2	01 03 05 16 00 00 00 00

end send data

Nr	Type	Request	IO	EP	Data
009	URB	BULK	OUT	2	00 06 00 14 2C 90 92 02
010	URB	BULK	OUT	2	00 06 E9 A9 39 DB 43 8C
011	URB	BULK	OUT	2	00 06 1F 47 5A 74 AE 44
012	URB	BULK	OUT	2	00 04 FE 98 BB F6 E6 21

Start to send “Betalen 1 transactie(s) EUR 1,00 Bevestig met OK nl” 6 bytes a time (SIGNDATA-TEXT)

Nr	Type	Request	IO	EP	Data
013	URB	BULK	IN	81	01 03 02 00 00 00 00 00
014	URB	BULK	OUT	2	01 03 05 46 00 00 00 00

SIGNDATA-TEXT (cont.)

Nr	Type	Request	IO	EP	Data
015	URB	BULK	OUT	2	00 06 01 44 42 65 74 61
016	URB	BULK	OUT	2	00 06 6C 65 6E 20 20 20
017	URB	BULK	OUT	2	00 06 20 20 20 20 20 20
018	URB	BULK	OUT	2	00 06 20 31 20 74 72 61
019	URB	BULK	OUT	2	00 06 6E 73 61 63 74 69
020	URB	BULK	OUT	2	00 06 65 28 73 29 20 20
021	URB	BULK	OUT	2	00 06 45 55 52 20 31 2C
022	URB	BULK	OUT	2	00 06 30 30 20 20 20 20
023	URB	BULK	OUT	2	00 06 20 20 20 20 20 42
024	URB	BULK	OUT	2	00 06 65 76 65 73 74 69
025	URB	BULK	OUT	2	00 06 67 20 6D 65 74 20
026	URB	BULK	OUT	2	00 04 4F 4B 20 20 74 20

received cryptogram 80 00 C7 C9 FF C6 B5 00 10 A5 00 03

Nr	Type	Request	IO	EP	Data
027	URB	BULK	IN	81	01 03 02 00 00 00 00 00
028	URB	BLUK	OUT	2	01 03 06 00 00 00 00 00
029	URB	BLUK	IN	81	01 03 03 19 00 00 00 00
030	URB	BLUK	IN	81	00 06 80 00 **C7 C9 FF C6**
031	URB	BLUK	IN	81	00 06 **B5** 00 10 A5 00 03

Nr	Type	Request	IO	EP	Data
032	URB	BLUK	IN	81	00 06 02 00 00 00 00 00
033	URB	BLUK	IN	81	00 06 00 00 00 00 00 00
034	URB	BLUK	IN	81	00 01 FF 00 00 00 00 00
035	DEV-CTR	-	OUT	0	01 00 00 00
036	URB	BLUK	OUT	2	02 01 00 00 00 00 00 00
037	URB	BLUK	IN	81	00 06 3B 67 00 00 29 20
038	URB	BLUK	IN	81	00 05 00 6F 78 90 00 20
039	DEV-CTR	-	OUT	0	01 02 08 00
040	DEV-CTR	-	OUT	0	00 00 00 00
041	URB	BLUK	OUT	2	02 0B 00 00 00 00 00 00

Appendix E

Login, USB security token (USB disconnected)

```
READER : #RESET#
READER : 00 A4 04 00 07 A0 00 00 00 04 80 02
CARD : 61 27
READER : 00 C0 00 00 27
CARD : 6F 25 84 07 A0 00 00 00 04 80 02 A5 1A 50 0E 53 65 63 75 72 65 43 6F 64 65 20 41 75 74 87
01 00 5F 2D 04 6E 6C 65 6E 90 00
READER : #RESET#
READER : 00 A4 04 00 07 00 00 00 04 80 02
CARD : 61 27
READER : 00 C0 00 00 27
CARD : C0 6F 25 84 07 A0 00 00 00 04 80 02 A5 1A 50 0E 53 65 63 75 72 65 43 6F 64 65 20 41 75 74
87 01 00 5F 2D 04 6E 6C 65 6E 90 00
READER : 80 A8 00 00 02
CARD : A8
READER : 83 00
CARD : 61 0C
READER : 00 C0 00 00 0C
CARD : C0 77 0A 82 02 10 00 94 04 08 01 01 00 90 00
READER : 00 B2 01 0C 00
CARD : 6C 62
READER : 00 B2 01 0C 62
CARD : B2 70 60 8C 21 9F 02 06 9F 03 06 9F 1A 02 95 05 5F 2A 02 9A 03 9C 01 9F 37 04 9F 35 01
9F 45 02 9F 4C 08 9F 34 03 8D 0C 91 0A 8A 02 95 05 9F 37 04 9F 4C 08 5A 0A 67 34 00 05 01 28 44
86 21 9F 5F 34 01 01 8E 0A 00 00 00 00 00 00 00 00 01 00 9F 55 01 80 9F 56 0C 00 00 7F FF FF E0
00 00 00 00 00 00 90 00
READER : 80 CA 9F 17 00
CARD : 6C 04
READER : 80 CA 9F 17 04
CARD : CA 9F 17 01 03 90 00
READER : 00 20 00 80 08
CARD : 20
```

```
READER : 24 XX XX FF FF FF FF FF
CARD : 90 00
READER : 80 AE 80 00 2B
CARD : AE
READER : 00 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 34
00 00 00 00 00 00 00 00 00 00 01 00 02 CARD : 61 2B
READER : 00 C0 00 00 2B
CARD : C0 77 29 9F 27 01 80 9F 36 02 00 16 9F 26 08 20 8F BA 6D A8 E7 36 1D 9F 10 12 00 10 A5
00 03 02 00 00 00 00 00 00 00 00 00 00 00 FF 90 00 READER : 80 AE 00 00 1D
CARD : AE
READER : 00 00 00 00 00 00 00 00 00 00 5A 33 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CARD : 61 2B
READER : 00 C0 00 00 2B
CARD : C0 77 29 9F 27 01 00 9F 36 02 00 16 9F 26 08 69 5D 96 FF 6C F1 09 19 9F 10 12 00 10 25 00
03 42 00 00 00 00 00 00 00 00 00 00 00 00 FF 90 00
READER : #RESET#
```

Bibliography

[1] de Koninggans, G. Smartlogic. http://gerhard.dekoninggans.nl/smartlogic, 2011.

[2] Drimer, S., Murdoch, S. J., and Anderson, R. J. Optimised to fail: Card readers for online banking. In *Financial Cryptography* (2009), pp. 184–200.

[3] Fiddler. Fiddler: web debugging proxy. http://www.fiddler2.com/fiddler2/, 2011.

[4] Group, T. G. Usb overview. http://www.ganssle.com/articles/usb.htm, 2000. seen on 23-06-2011.

[5] ISO 7816-4:1995. *Information technology – Identification cards – Integrated circuit(s) cards with contacts – Part 4: Interindustry commands for interchange.* ISO, Geneva, Switzerland, 1995.

[6] LibUsbDotNet. Libusbdotnet c# usb library. http://sourceforge.net/projects/libusbdotnet/, 2011.

[7] RealTerm. Serial terminal: Realterm. http://realterm.sourceforge.net/, 2011.

[8] sysnucleus. Usbtrace : Software-only usb protocol analyzer. http://www.sysnucleus.com, 2011.