The SmartLogic Tool: Analysing and Testing Smart Card Protocols

Gerhard de Koning Gans and Joeri de Ruiter
Institute for Computing and Information Sciences
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
{gkoningg,joeri}@cs.ru.nl

***Abstract*—This paper introduces the SmartLogic, which is a flexible smart card research tool that gives complete control over the smart card communication channel for eavesdropping, man-in-the-middle attacks, relaying and card emulation. The hardware is available off-the-shelf at a price of about 100 euros. Furthermore, the necessary firm- and software is open source. The SmartLogic provides essential functionality for smart card protocol research and testing. This is demonstrated by reproducing two attack scenarios. The first attack is on an implementation of the EMV payment protocol where a payment terminal is forced to do a rollback to plaintext PIN instead of using encrypted PIN. The second attack is a relay of a smart card payment over a 20 km distance. We also show that this distance can be increased to at least 10.000 km.**

***Keywords*-Smart card testing; Man-in-the-Middle; Protocol analysis; Relay attack; EMV**

I. Introduction

Since the introduction of smart cards back in the early seventies [1], a lot of progress has been made in the capabilities of smart cards. Nowadays there are cards on the market that support computationally expensive cryptographic operations [2]. These impressive capabilities allow a wide range of applications that involve complicated protocols. A tool that allows to test these protocols in all kinds of adversarial setups is essential to smart card research and testing. However, to the best of our knowledge such tools are not widely available or are limited in their capabilities. This paper introduces the SmartLogic, which is a smart card research tool that can be used in different modes such as eavesdropping, card emulation, man-in-the-middle attacks (or so-called "wedge" attacks) and relaying.

We demonstrate the capabilities of the SmartLogic by two attack scenarios. First, we reproduce the man-in-the-middle attack of Barisani et al. [3] on an implementation of the EMV[1] protocol in the Netherlands. The success of this attack depends on the system specific parameters of the terminal in the field. In the attack a rollback from encrypted to plaintext PIN is forced on a genuine payment terminal. In other words, the terminal is convinced to send the PIN in plaintext to the card while this should not happen. In our second demonstration we use the SmartLogic to relay a smart card payment over a distance of approximately 20 km. Furthermore, we show that this distance can be increased to at least 10.000 km. The standard for smart cards, ISO/IEC 7816, does not contain any countermeasures against relay attacks.

A. Related Work

Here we describe smart card tools that are related to the SmartLogic. Some of the tools are not publicly available, while others are limited in their functionalities or are targeted at specific card types like SIMs[2]. Table I gives an overview of non-commercial smart card research tools. The first column indicates whether the tool is publicly available. By *eavesdropping* we mean that it can passively overhear the smart card communication. Support for an active man-in-the-middle setup is indicated by the column *active MitM*. The column *baudrate detection* refers to the ability of the tool to automatically detect the data transfer rate. This is essential for a correct interpretation and manipulation of the intercepted data. Some tools explicitly need to be configured at the right speed as they cannot detect this automatically. Furthermore, *distance relaying* means that the terminal and card are not required to be at the same physical location, e.g. the communication is relayed over the Internet. Finally, a tool supports *sharing* when it is possible to use one smart card simultaneously at multiple locations.

The *RebelSim APDU Scanner* [4] can be used to passively sniff the communication between a smart card and a reader. As the name suggests its main focus is on SIM cards, presumably to analyse and undo SIM locking. It provides SIM interfaces as depicted in Figure 1. The communication is intercepted with an UART[3] chip and can be read out using standard terminal software. A drawback of the RebelSim is that the baudrate needs to be set beforehand in order to capture the communication.

The *Osmocom SIMtrace* [5] is a piece of hardware and software in the OsmocomBB project. This project aims to

[1]The EMV standard contains smart card specifications for payment systems by Europay, MasterCard and VISA.

[2]Subscriber Identification Module

[3]Universal Asynchronous Receiver/Transmitter

Tool	Publicly Avail.	Eavesdropping	Active MitM	Baudrate Detection	Distance Relaying	Sharing
1) RebelSim APDU Scanner	✓	✓	-	-	-	-
2) Osmocom SIMtrace	✓	✓	-	✓	-	-
3) Leon Device	-	✓	✓	✓	-	-
4) Season3	✓	✓	✓	-	-	-
5) Smart Card Detective	✓	✓	✓	✓	-	-
6) SmartLogic	✓	✓	✓	✓	✓	✓

Table I
SMART CARD RESEARCH TOOLS

produce an open source GSM baseband software implementation. The Osmocom SIMtrace tool is used within this project to eavesdrop on communication between a SIM card and a mobile phone. It uses the SIM connector from the RebelSim (see Fig. 1).

Examples of active man-in-the-middle tools are the *Leon Device* [6], developed at the University of Michigan, and the *Season3* [7]. As far as we know no hardware design or software for the Leon Device has been made public. The Season3 can be controlled over a serial connection where the baudrate needs to be pre-configured.

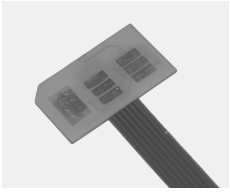

Figure 1. SIM Interface

Lastly, the *Smart Card Detective* (SCD) [8] is a more recent tool that supports active man-in-the-middle attacks. The SCD has been developed by Choudary as a hand-held EMV interceptor. The resulting traces can be stored in EEPROM which is read out over a USB connection. Although the SCD was designed specifically for EMV protocols it might be used for other protocols as well. However this requires modification of the firmware. Both the hardware design and the software for the SCD are publicly available.

B. The SmartLogic Tool

The main focus of the SmartLogic is to provide a highly flexible setup. Part of its flexibility is achieved by its client-server architecture. The server gives the client, equipped with SmartLogic Hardware, access to either a real smart card, connected via a standard reader, or one emulated by the server. This architecture allows to easily setup relay attacks over the Internet and makes it possible to share a smart card between different clients, at different locations. The SmartLogic Hardware consists of an FPGA[4] and USB microcontroller. No knowledge of the hardware is required in order to implement the different attack scenarios, i.e. no microcontroller or FPGA programming is needed.

The FPGA firmware, USB firmware and PC software for the SmartLogic are released [9] under the terms of the GNU General Public License version 3. The hardware is available [10] off-the-shelf at a price of about 100 euros.

[4]Field-Programmable Gate Array

C. Outline of this Paper

This paper first describes the basic design of the SmartLogic in Section II. This section starts with a brief explanation of the ISO/IEC 7816 standard on smart cards. Then, the SmartLogic setup and hardware components are discussed. The section concludes with the software and functionalities of the SmartLogic. To demonstrate these functionalities, Section III describes an attack on the EMV protocol and Section IV describes relaying smart card traffic over larger distances. Finally, we conclude with Section V and discuss further research that can be done using the SmartLogic.

II. SMARTLOGIC DESCRIPTION

This section discusses the different components of the SmartLogic. First, a short introduction to the ISO/IEC 7816 standard is given. This standard specifies the physical characteristics and the different protocol levels of smart cards. The implementation of the SmartLogic Hardware follows the ISO/IEC 7816 protocol specification.

A. ISO/IEC 7816

The ISO/IEC 7816 standard defines *identification cards* as integrated circuit cards with contacts. This section presents a quick overview on the standard. For a more detailed description we refer to the standard [11], [12]. ISO/IEC 7816 was introduced in 1998 and defines circuit cards (or smart cards) on different levels. We will mainly focus on Part 3 [11] where the smart card contact interface is explained. It can be said that this part mainly covers the physical layer of the protocol. Besides normative references, electrical characteristics and some basic card operation procedures, the ISO/IEC 7816 standard defines the transmission protocols T=0 and T=1.

1) Answer to Reset: The *Answer to Reset* (ATR) is always sent after a signal on the reset pin of the card. The ATR conveys information about the supported protocols and possible configurations. Protocols are referred to by T=x where x stands for a transmission protocol.

2) T=0 and T=1 Protocols: The SmartLogic supports both the T=0 and the T=1 protocol. The T=0 protocol is widely used in smart cards and implements half-duplex transmission of characters. It is configured in a master-slave setting where the reader is always first to initiate a command and the card is in slave mode. A reader command consists of five bytes. This is defined in more detail in ISO/IEC 7816-3 [11].

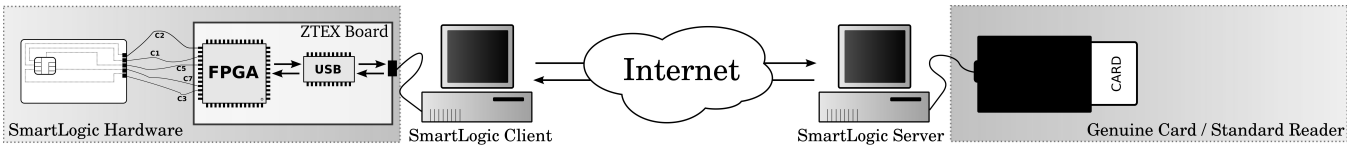

Figure 2. SmartLogic Setup

CLA	INS	P1	P2	P3

Here CLA is the class byte which indicates the class of the command. Then, INS is the instruction byte followed by two parameter bytes P1 and P2. Finally, P3 is the length byte which indicates the length of a possible data message. Depending on the command, a data message can be send by either the reader or the card. The direction might differ from application to application. This means that in an active setup, like the SmartLogic, one should learn these directions in order to successfully relay a protocol run. If there is any data to be sent or received in response to a command execution the card indicates that it is ready to send or receive data by repeating the instruction byte.

The T=1 protocol specifies a half-duplex transmission of blocks. It is very similar to T=0, the main difference is that commands and their data are now wrapped into blocks that contain header information and a final check byte.

B. SmartLogic Setup

The SmartLogic consists of hardware and software parts. Figure 2 shows a typical SmartLogic setup where the ZTEX Board is the hardware that emulates the smart card communication on the lowest level. This board is directly connected to a smart-card-sized circuit board that can be inserted into a genuine terminal. A PC that controls the SmartLogic Hardware, the SmartLogic Client, relays the smart card communication to the SmartLogic Server. The SmartLogic Server controls a standard smart card reader with a genuine smart card and forwards all communication to it. Note that it is also possible to run both the SmartLogic Client and Server on the same host machine. This is in most cases sufficient enough to set up a simple man-in-the-middle attack.

There are some major advantages of the SmartLogic setup. First, the application layer of the protocol is implemented as host software in the SmartLogic Client and Server. This means that all protocol logic can be programmed in Java. The low-level parts like the firmware for the hardware can remain untouched, i.e. there is no need to flash or re-program the hardware. Another advantage is that the SmartLogic Server is able to emulate a card or may function as a proxy that caches smart card communication in order to limit smart card access. Furthermore, an inventive new idea of this architecture is the concept of smart card sharing at different locations at the same time. This can help to find out whether a system checks for location-based conflicts. In order to do this the SmartLogic Server can handle multiple SmartLogic Clients simultaneously.

C. SmartLogic Hardware

The general purpose hardware (see Fig. 2) that is used is an FPGA evaluation board from ZTEX [10]. The ZTEX Board is a small programmable device that is equipped with an FPGA and USB microcontroller chip. There are different versions of the ZTEX Board available[5]. This section describes the main hardware components which is a combination of a *smart card circuit board* and the ZTEX Board.

1) Smart Card Circuit Board: The smart card circuit board is a smart-card-sized board that contains the smart card connectors on the physical locations as described in the ISO/IEC 7816 standard. The connectors of the smart card circuit board are depicted in Figure 3. The power (VCC) and ground (GND) connectors are not logically connected but only use the circuitry and ground from the ZTEX Board. Connectors C4, C6 and C8 are not used and the remaining three connectors, reset (RST), clock (CLK) and input-output (I/O), are logically connected to the ZTEX Board.

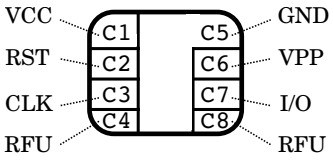

Figure 3. Smart Card Interface

2) ZTEX Board: The main component of the ZTEX Board is an FPGA which can be seen as a piece of hardware that is able to simulate a hardware design. Since the FPGA chip can be flashed it allows to change hardware designs and test different configurations which makes it multi-functional. The hardware design can be described in a hardware description language such as VHDL or Verilog. The input-output line is a half-duplex channel which makes use of a pull-up register. This means that both parties pull the line to a high voltage when they are not communicating. The party that communicates pulls the line to a low voltage in order to

[5]We tested the SmartLogic with ZTEX USB-FPGA-Module 1.2 and 1.11c.

send a '0' and pulls it up again to send a '1'. Since the line is half-duplex, only one party can communicate at a time.

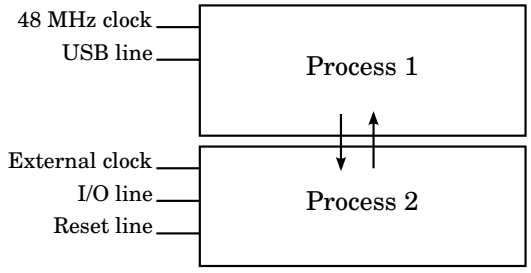

Figure 4. FPGA Processes

The FPGA is responsible for the communication on bit-level. Since it uses the clock of the genuine terminal it is able to snap in on the correct bitrate by counting the clock cycles. By default one bit period is 372 clock cycles according to the ISO/IEC 7816. Concretely, the FPGA runs two processes, see Figure 4. The first process controls the input and output buffers that contain the raw bits for sending and receiving. This process runs at the main clock of 48 MHz. The second process snaps to the clock frequency of the external genuine terminal. The speed of this process varies but is typically a couple of megahertz and it makes sure that the bits are communicated at the right speed on the I/O line (C7). The communication between the FPGA and the SmartLogic Client is controlled by an USB chip.

D. SmartLogic Software

Apart from the firmware for the ZTEX Board, the software for the SmartLogic consists of a client and server implementation in Java. The SmartLogic Client controls the ZTEX Board and establishes a connection with the SmartLogic Server. In its turn the SmartLogic Server controls a standard smart card reader and forwards all communication to a genuine smart card. Alternatively, the SmartLogic Server can be programmed such that all smart card responses are emulated.

Figure 5 shows two abstract classes *ProtocolMitm* and *ProtocolEmulator* that are used in the SmartLogic Server. These classes can be extended for a specific application like the man-in-the-middle attack on the EMV protocol that is presented in this paper. By default there are some example implementations available like *ProtocolMitmDefault* that intercepts and displays plaintext PIN codes in an alert box. In order to set up new man-in-the-middle or emulation setups, only an extension on either the *ProtocolMitm* or *ProtocolEmulator* class has to be written.

1) Smart Card Sharing: One of the features of the SmartLogic Server is that it can accept multiple connections. Depending on the smart card application that is being shared, some additional checks are needed to keep the output consistent. For example, the SIM application contains a directory structure. First the directory is chosen by a *select* command before a *read* or *write* command is issued. To prevent inconsistent output the server needs to keep track of which client wants to access which directory.

2) Set Answer To Reset: According to ISO/IEC 7816 a smart card needs to send an Answer to Reset when it receives a reset signal from the reader. This ATR should be sent within 400 to 40.000 clock cycles after the reset signal. The ATR is cached in the SmartLogic Hardware after startup in order to keep control over the response time. All other communication is relayed to the genuine card or emulated by the SmartLogic Server.

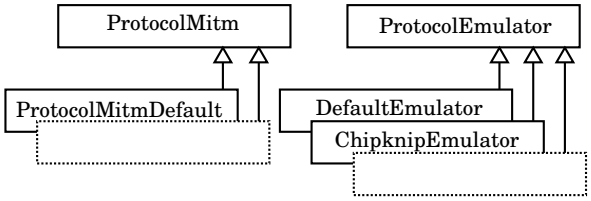

Figure 5. Man-in-the-middle and Emulator Java classes

3) Speed and Baudrate Detection: Unlike with tools as the RebelSim it is not necessary to set the baudrate for the SmartLogic since it connects directly to the clock of the genuine terminal. Tools such as the RebelSim need to be pre-configured before eavesdropping with a baudrate that varies for different smart card readers. By using the internal clock of the ZTEX Board, which runs on 48 MHz, and comparing its difference in cycles with the clock of the genuine terminal it is possible to approximate and communicate the clock speed of the terminal to the PC.

III. Investigating the EMV Protocol

In this section we discuss a man-in-the-middle attack on an EMV protocol. The attack was presented by Barisani et al. [3] using dedicated hardware. Usually these kind of attacks are not reproduced because it is a time consuming and tedious job. The SmartLogic lowers the effort needed to mount such an attack. By writing an extended class *ProtocolMitmEMV* in Java we were able to easily reproduce the attack.

A. The Protocol

EMV is a standard for electronic payments using smart cards [13]–[16]. The initiative for EMV was taken by Europay, MasterCard and Visa in the 1990s. Currently the EMV standard is maintained by EMVCo, a company jointly owned by MasterCard, Visa, American Express, and JCB. According to EMVCo, the number of EMV cards in use reached 1 billion in 2010.

An EMV session consists of four phases:

1) *initialisation*
2) *data authentication* (optional)

Sender		Original Run	Modified Run	Info
READER	:	00 B2 01 0C 8A	00 B2 01 0C 8A	← READ RECORD
CARD	:	B2 70 81 87 5F 25 03 10 06	B2 70 81 87 5F 25 03 10 06	
		17 5F 24 03 15 04 30 9F 07	17 5F 24 03 15 04 30 9F 07	
		02 FF C0 5A 0A XX XX XX XX	02 FF C0 5A 0A XX XX XX XX	
		XX XX XX XX XX XX 5F 34 01	XX XX XX XX XX XX 5F 34 01	
		08 8E 12 00 00 00 00 00 00	08 8E 12 00 00 00 00 00 00	
		00 00 **42 01** 02 04 04 03 02	00 00 **01 00** 02 04 04 03 02	Two CVM bytes 42 01 are adjusted to 01 00
		03 01 00 9F 0D 05 B8 70 BC	03 01 00 9F 0D 05 B8 70 BC	
		80 00 9F 0E 05 00 00 00 00	80 00 9F 0E 05 00 00 00 00	
		00 9F 0F 05 **B8** 70 BC 98 00	00 9F 0F 05 **FF** 70 BC 98 00	One Action Code - Online byte B8 is adjusted to FF
		8C 21 9F 02 06 9F 03 06 9F	8C 21 9F 02 06 9F 03 06 9F	
		1A 02 95 05 5F 2A 02 9A 03	1A 02 95 05 5F 2A 02 9A 03	
		9C 01 9F 37 04 9F 35 01 9F	9C 01 9F 37 04 9F 35 01 9F	
		45 02 9F 4C 08 9F 34 03 8D	45 02 9F 4C 08 9F 34 03 8D	
		0C 91 0A 8A 02 95 05 9F 37	0C 91 0A 8A 02 95 05 9F 37	
		04 9F 4C 08 5F 28 02 05 28	04 9F 4C 08 5F 28 02 05 28	
		9F 4A 01 82 90 00	9F 4A 01 82 90 00	
READER	:	00 88 00 00 04		← Card Authentication
CARD	:	88		
READER	:	36 25 2E 81		
CARD	:	61 87		
READER	:	00 C0 00 00 87		
CARD	:	C0 77 81 84 9F 4B 81 80 79		
		0F 64 83 96 9D FC 5F 17 09		
		1B 6E ...98 CC B3 18 83 E0		
		63 A5 90 00		
READER	:	00 84 00 00 00		← GET CHALLENGE
CARD	:	6C 08		
READER	:	00 84 00 00 08		
CARD	:	84 5A 6F E6 FA A5 78 87 9D		
		90 00		
READER	:	00 20 00 88 80	00 20 00 80 08	← VERIFY PIN
CARD	:	20	20	
READER	:	51 62 E3 B7 98 D6 42 79 58	24 **12 34** FF FF FF FF FF	← Plaintext PIN 1234
		54 EB 9B D1 46 53 62 3C BA		
		6A EF ...17 3C A9 2A B8 58		
		A1 22 DA 9B		
CARD	:	90 00	90 00	

Figure 6. EMV Payment: Card Authentication and PIN Verification

3) *cardholder verification* (optional)
4) the actual *transaction*

In the *initialisation*, the terminal selects the EMV application on the smart card and retrieves the data necessary for the transaction.

The *data authentication* phase is used for authentication of data that is residing on the smart card. This at least includes a signature over static data, for example, the account number and expiry date of the card. The optional dynamic part of the authentication is achieved by either running a challenge-response protocol or retrieving a signature over the transaction data. If the verification of the signature on the static data fails, the dynamic part is not performed and the data authentication is aborted.

There are several methods for *cardholder verification* of which only two involve the card in the verification process: off-line encrypted PIN and off-line plaintext PIN. Off-line plaintext PIN is supported for older smart cards, that do not support asymmetric cryptographic operations. In the initialisation phase the card provides a list with supported cardholder verification methods (*CVM List*) in order of preference and with possible additional conditions. For example, Dutch banking cards prefer off-line encrypted PIN over off-line plaintext PIN.

In the actual *transaction* phase, the card computes a MAC, using a key shared with the bank, over relevant data, e.g. the amount and cardholder verification results. If required for the data authentication, this is combined with an additional signature that can be verified by the terminal.

Exception Handling: When an exception occurs during an EMV session, so-called *Action Codes* determine how the terminal should react. An Action Code is basically a list of exceptions. Both the card and the terminal can contain Action Codes, the Issuer Action Codes and the Terminal Action Codes respectively. If an exception occurs, it is first checked against the *Action Code - Denial* of both the card and the terminal. When the exception is listed in one of these Action Codes the transaction is aborted. If the exception is not listed in an *Action Code - Denial*, the *Action Code - Online* is checked to determine whether the transaction should be forced online.

B. The Attack

Recently a method was shown by Barisani et al. [3] to force a rollback from encrypted PIN to plaintext PIN using a man-in-the-middle attack. Their method makes use of the fact that after a failed data authentication, the transaction might still continue and be performed online depending on the Action Codes.

In the attack the CVM List is modified such that off-line plaintext PIN is the preferred method for cardholder verification. The Issuer Action Codes are modified such that in case of a failed data authentication the transaction is not aborted but performed online. Since modifying the CVM List and Issuer Action Codes might result in a failed data authentication, it depends on the Terminal Action Codes whether the transaction is then aborted or performed online. If the transaction is not aborted, the terminal continues with the data it received from the card, including the modified CVM List, as it cannot tell which data was modified. This results in the PIN code being sent to the card in plaintext, which can then be intercepted by a man-in-the-middle.

C. Using the SmartLogic

To test whether the forced fall-back also could be applied in the Netherlands we used the SmartLogic to intercept and modify the communication between a point-of-sale terminal and a Dutch banking card. The Dutch banking card supported the challenge-response mechanism as data authentication method. After the presentation by Barisani et al. [3] the Dutch banks rolled out a fix to their terminals to prevent the attack. In the initialisation phase we modified the CVM List and Issuer Action Codes, both which are included in the static data that is retrieved during the data authentication phase.

The Action Codes are modified such that on failed data authentication the transaction is performed online. The rest of the communication is passed on unchanged. In Figure 6, part of an original transaction is compared with a modified one. The modified data in the READ RECORD command is indicated in bold. After modifying the data, the challenge-response part of the data authentication was no longer performed. This is as expected, as data authentication already fails when verifying the signature over the static data. Although the transaction was denied by the back-end, the modification of the data still resulted in a plaintext PIN code transmission (see Fig. 6) to the card, as opposed to the claim of the banks. According to the bank, the terminal we encountered was one of the few terminals that had not been patched yet.

IV. Increasing the Card-Terminal Distance

An interesting question is whether it is possible to increase the distance between a genuine card and the terminal, and moreover, to what distance. For this experiment we used the Chipknip, a Dutch smart card payment scheme. Comparable systems are, for example, the Geldkarte from Germany and Proton from Belgium. These systems are designed for micro payments and function like an electronic wallet. No PIN is needed to perform payments using a Chipknip card. The only transaction that involves a PIN is when money is transferred from a regular bank account to the Chipknip card. It is possible to buy anonymous Chipknip cards that are not linked to a bank account; these cards cannot be charged.

RTD	Sender	Location	Message
	READER	Nijmegen	BC B0 00 00 08
68 ms	CARD	Arnhem	B0 00 00 00 00 00 00 00 00 90 00
	READER	Nijmegen	BC A4 00 00 02
41 ms	CARD	Arnhem	A4
	READER	Nijmegen	29 01
66 ms	CARD	Arnhem	90 00
	READER	Nijmegen	BC B0 00 00 64
268 ms	CARD	Arnhem	B0 52 80 01 01 00 20 62 ... ... B0 7D 90 00
	READER	Nijmegen	BC B0 00 19 20
119 ms	CARD	Arnhem	B0 D2 0C 2E E9 67 30 10 ... ... 00 00 90 00
	READER	Nijmegen	E1 B4 00 01 05
76 ms	CARD	Arnhem	B4 00 06 D1 09 78 90 00

Figure 7. Relay over 20 km between Arnhem and Nijmegen

In a practical setup we were able to relay a payment between two Dutch cities that are situated about 20 km apart. In Arnhem we installed a SmartLogic Server with a genuine smart card and in Nijmegen we used a SmartLogic Client to buy candy from a vending machine. The client was connected to the Internet through a wireless connection and the server was directly connected through an ADSL connection. Round-trip times of the messages in the relayed protocol run are depicted in Figure 7.

ISO/IEC 7816-3 defines the waiting time *WT* as the maximal delay between the leading edge of a character that is transmitted by the card and the leading edge of the last character of a message that was transmitted by either the card or the reader. When no signal is received within time *WT*, a card is considered to be unresponsive. It is possible to define the *WT* in the ATR of a card. We tested some terminals without setting *WT* and using 372 cycles per bit period. According to the standard the default *WT* would then be $9600 \times \frac{372}{f}$ ms where f is the clock frequency of the terminal.

Table II shows the results of the observed maximal delay times. We measured the maximal delay that could be introduced while keeping a successful protocol run. The results for the default configuration show that it is possible to run protocols over huge distances. For instance, we measured a round-trip time to an IP address in Osaka (Japan) of about 270 ms which is at a distance of approximately 10.000 km from our testing location. Note that the Chipknip terminal allows a delay that is almost twice as long. To conclude, the waiting time is not a boundary when it comes to relay attacks. If good connection speeds are available at both the

Terminal/Reader	Clk. speed	WT	Measured
VASCO DIGIPASS 810	1.05 MHz	3410 ms	3560 ms
e.dentifier2	2.00 MHz	1790 ms	1910 ms
Ingenico 5300	4.91 MHz	730 ms	1100 ms
Chipknip Charging Terminal	3.69 MHz	970 ms	1200 ms
Chipknip Payment Terminal	4.92 MHz	730 ms	500 ms

Table II
MAXIMUM MEASURED DELAY TIMES

location of the client and the server, a relay attack can be mounted from one side of the world to the other. Several solutions to this problem of relaying have been proposed and are known as distance bounding protocols [17]–[19]. Practical attacks like demonstrated in this paper show the need for such protocol implementations.

V. CONCLUSIONS

In this paper we have presented the SmartLogic, a generic and highly flexible smart card research tool, for which the hardware and software are open source and publicly available. The SmartLogic allows to execute all types of common attacks on protocols. The added value of the SmartLogic is a flexible design that can be quickly modified to a certain protocol and logical processing. In order to set up a specific attack, only a small extension needs to be written in Java, which is for most programmers more comfortable than programming a microcontroller or FPGA.

Apart from performing specific attacks, the SmartLogic can also be used to discover new security vulnerabilities. One way to do this is using real-time fuzzing. Doing this in real-time is useful if, for example, a genuine card and terminal are needed to successfully authenticate before the actual fuzzing will take place. Fuzzing is not built into the tool yet, but could easily be added. A different approach would be to try to infer the internal state machines of smart cards and terminals. Using this method, 'undesired' states, that introduce security vulnerabilities, could be discovered. In related work, models are inferred for smart cards like banking cards [20] and electronic passports [21]. Furthermore, additional testing can be done by supplying the learned model as input for model-based testing.

The use and practicality of the SmartLogic is demonstrated by mounting two practical attacks. The first one is an attack on EMV, forcing a rollback from encrypted to plaintext PIN. The second attack is a successful relay of a payment transaction over a distance of 20 km. Furthermore, we showed that these relay attacks have a high potential range and can bridge 10.000 km. Apart from the two experiments described in this paper, we also successfully used the SmartLogic to emulate smart cards in e-banking readers and shared one SIM card between two mobile phones simultaneously.

To conclude, using low-cost off-the-shelf hardware, the SmartLogic allows smart card testing in many different setups including eavesdropping, man-in-the-middle attacks, relaying, card emulation and card sharing.

REFERENCES

[1] W. Rankl and W. Effing, *Smart Card Handbook, 2nd Edition*. Wiley, 2000.

[2] N. Semiconductors, *P5Cx012/02x/40/73/80/144 Family: Secure Dual Interface and Contact PKI Smart Card Controller*, NXP, June 2010.

[3] A. Barisani, D. Bianco, A. Laurie, and Z. Franken, "Chip & PIN is definitely broken," 2011, presentation at CanSecWest Applied Security Conference, Vancouver 2011. Slides available at http://dev.inversepath.com/download/emv/emv_2011.pdf.

[4] "RebelSim APDU Scanner," http://rebelsimcard.com/network-sim-apdu-scanner.html.

[5] "Osmocom SIMtrace," http://www.osmocom.org.

[6] "Smartcards: Leon Devices," http://www.citi.umich.edu/projects/smartcard/leon.html, Center for Information Technology Integration (CITI).

[7] "Season3," http://www.cardman.com/loggers.html.

[8] O. Choudary, "The Smart Card Detective: A Hand-Held EMV Interceptor," Master's thesis, 2010, http://www.smartcarddetective.com.

[9] G. de Koning Gans, "SmartLogic Tool," http://gerhard.dekoninggans.nl/smartlogic.

[10] S. Ziegenbalg, "ZTEX Hardware," http://ztex.de.

[11] ISO/IEC 7816-3:2006, *Identification cards — Integrated circuit cards with contacts — Part 3: Cards with contacts — Electrical interface and transmission protocols*. ISO, Geneva, Switzerland, 2006.

[12] ISO/IEC 7816-4:2005, *Identification cards — Integrated circuit cards with contacts — Part 4: Organization, security and commands for interchange*. ISO, Geneva, Switzerland, 2005.

[13] EMVCo, "EMV– Integrated Circuit Card Specifications for Payment Systems, Book 1: Application Independent ICC to Terminal Interface Requirements," 2008.

[14] ——, "EMV– Integrated Circuit Card Specifications for Payment Systems, Book 2: Security and Key Management," 2008.

[15] ——, "EMV– Integrated Circuit Card Specifications for Payment Systems, Book 3: Application Specification," 2008.

[16] ——, "EMV– Integrated Circuit Card Specifications for Payment Systems, Book 4: Cardholder, Attendant, and Acquirer Interface Requirements," 2008.

[17] S. Brands and D. Chaum, "Distance-Bounding Protocols," in *Advances in Cryptology – EUROCRYPT'93*. Springer, 1994, pp. 344–359.

[18] S. Drimer and S. Murdoch, "Keep your enemies close: Distance bounding against smartcard relay attacks," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. USENIX Association, 2007, pp. 1–16.

[19] G. Hancke and M. Kuhn, "An RFID Distance Bounding Protocol," in *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*. IEEE, 2005, pp. 67–73.

[20] F. Aarts, E. Poll, and J. de Ruiter, "Formal models of banking cards for free," 2012, unpublished.

[21] F. Aarts, J. Schmaltz, and F. Vaandrager, "Inference and abstraction of the biometric passport," in *Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part I*, ser. ISoLA'10, 2010, pp. 673–686.

APPENDIX

APPENDIX A: EMV ATTACK IMPLEMENTATION

This is the attack implementation that was used for an EMV terminal in the Netherlands. The extended class *ProtocolMitmEMV* was written to set up the man-in-the-middle attack as described in this paper using the SmartLogic.

```
import javax.smartcardio.Card;
import javax.smartcardio.CommandAPDU;
import javax.smartcardio.ResponseAPDU;

import net.sourceforge.scuba.smartcards.CardService;
import net.sourceforge.scuba.smartcards.CardServiceException;

/**
* ProtocolMitmEMV.java – EMV plaintext PIN attack
*/
class ProtocolMitmEMV extends ProtocolMitm {

        public ProtocolMitmEMV() {
                this.setActivated(true);
        }

        public void reset() {
        }

        public byte[] getResponse(CardService card, byte[]
             readerMessage) {
                byte[] empty = {};
                byte[] reply;
                CommandAPDU command;
                ResponseAPDU response;

                reply = empty;

                try {
                        command = new
                             CommandAPDU(readerMessage);
                        response = card.transmit(command);
                        reply = response.getBytes();

                        if (readerMessage.length == 5 &&
                             reply.length > 0) {
                                byte CLA = readerMessage[0];
                                byte INS = readerMessage[1];
                                byte P1  = readerMessage[2];
                                byte P2  = readerMessage[3];
                                byte P3  = readerMessage[4];

                                if (    CLA == (byte) 0x00 &&
                                        INS == (byte) 0xB2 &&
                                        P1  == (byte) 0x01 &&
                                        P2  == (byte) 0x0C &&
                                        P3  == (byte) 0x8A) {

                                        reply[46] = (byte)
                                             0x01;
                                        reply[47] = (byte)
                                             0x00;
                                        reply[75] = (byte)
                                             0xFF;
                                }
                        }
                } catch (Exception e) {
                        reply = empty;
                }

                return reply;
        }
}
```