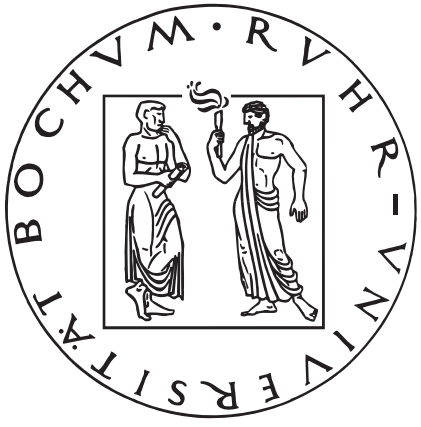
Security Analysis of Contactless Payment Systems in Practice

Michael Silbermann

November 2, 2009

Diplomarbeit
Ruhr-Universität Bochum



Lehrstuhl für Embedded Security
Prof. Dr.-Ing. Christof Paar
Supervisor: Dipl.-Ing. Timo Kasper

Generated on January 19, 2010 at 15:25.

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Bochum, den 2. November 2009

_____

Michael Silbermann

Contents

Glossary

3DES . . . . . Triple-DES

ADC . . . . . Analog to Digital Converter
AES . . . . . Advanced Encryption Standard
API . . . . . Application Programming Interface
ASK . . . . . Amplitude-Shift Keying
ATQA . . . . . Answer To Request A
ATS . . . . . Answer To Select

CL . . . . . Cascade Level
CRC . . . . . Cyclic Redundancy Check

DES . . . . . Data Encryption Standard
DoS . . . . . Denial of Service

ECC . . . . . Elliptic Curve Cryptography
EEPROM . . . Electrically Erasable Programmable Read-Only Memory
EOC . . . . . End Of Communication

FDT . . . . . Frame Delay Time
FSM . . . . . Finite-State Machine

GUI . . . . . Graphical User Interface

HLTA . . . . . HaLT A

IDE . . . . . Integrated Development Environment
ISP . . . . . In-System Programming
ISR . . . . . Interrupt Service Routine

LED . . . . . Light-Emitting Diode
LFSR . . . . . Linear Feedback Shift Register

MAC . . . . . Message Authentication Code
MITM . . . . . Man-In-The-Middle

OOK . . . . . On-Off-Keying
OS . . . . . Operating System

PCB . . . . . Printed Circuit Board
PICC . . . . . Proximity Integrated Circuit Card
PPS . . . . . Protocol and Parameter Selection
PPSR . . . . . Protocol and Parameter Selection Request
PRNG . . . . . Pseudo-Random Number Generator

RAM . . . . . Random Access Memory
RATS . . . . . Request for Answer To Select
REQA . . . . . REQuest A
RF . . . . . Radio Frequency
RFID . . . . . Radio-Frequency IDentification
RGT . . . . . Request Guard Time
RISC . . . . . Reduced Instruction Set Computer

SAK . . . . . Select AcKnowledge
SOC . . . . . Start Of Communication
SRAM . . . . . Static Random Access Memory

UID . . . . . Unique IDentifier
USB . . . . . Universal Serial Bus

VCP . . . . . Virtual COM Port

WUPA . . . . . Wake-UP A

List of Figures

List of Tables

List of Algorithms

1. Introduction

1.1. Motivation

1.1.1. RFID Technology

Today many applications use Radio Frequency (RF) chip technology to provide an easy way of automatically identifying various objects and people. Those applications include tracking animals and tagging goods for automatic inventory control, as well as contactless payment and access control. While these applications all use radio waves to transport information, the chip architectures used for each of them differ considerably, concerning the storage, range, computational power and security requirements. In general, Radio-Frequency IDentification (RFID) tags are used for object identification and tracking while contactless smartcards are used for ID cards and cashless payments. Possible

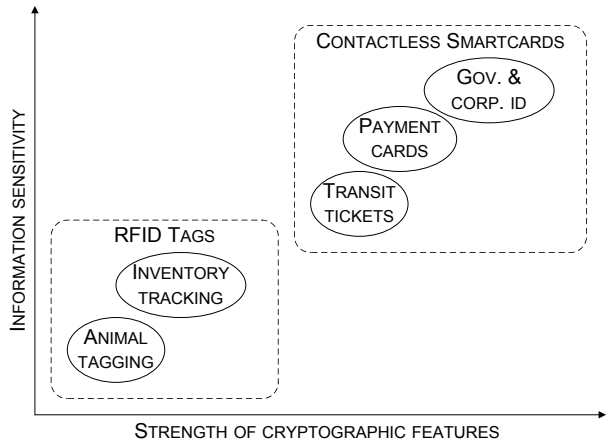


Figure 1.1.: Comparison of RFID tag and contactless smartcard technologies.

applications and their requirements are illustrated in Figure 1.1.

Because RFID tags are mostly deployed in applications with low security requirements, such as goods tracking, they typically have no security features, i.e., encryption or data integrity mechanisms. In contrast, contactless smartcards are often used in sensitive access control and payment applications. They offer a whole wide range of security features, including cryptographic co-processors, Message Authentication Code

(MAC) generation and more. The main focus of this work is the contactless smartcards technology and the associated security considerations.

1.1.2. Contactless Smartcard

A contactless smartcard in its state-of-the-art form includes an embedded smartcard microcontroller or a digital control unit, some sort of internal memory — typically non-volatile memory, such as Electrically Erasable Programmable Read-Only Memory (EEPROM) — and an analog antenna front-end for power supply and communication with an RFID reader device through a contactless RF interface.

Contactless smartcards are deployed in a wide range of fields where there is a need to protect personal information or deliver fast and secure transactions, such as transportation, government or corporate ID, and access control applications. Moreover, a growing number of payment systems incorporates contactless technology, as it offers additional benefits in terms of flexibility, convenience and reliability over its contact-based counterpart. Contactless chips can be embedded in a variety of forms, e.g., plastic cards, key fobs, watches, documents and even mobile phones. A detailed description of the operation of contactless smartcards follows in Chapter 2.

Operation Principle

Similar to the contact-based smartcards, contactless smartcards store information in the internal memory of the chip. However, unlike the contact-based smartcard, the power supply to the card and the data exchange between the card and the reader are achieved without the use of contacts, but by means of an electromagnetic field to both power the card as well as to transmit and receive data.

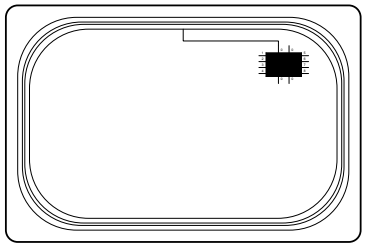


Figure 1.2.: Basic layout of a contactless smartcard.

In its common form, the contactless smartcard illustrated in Figure 1.2 contains a chip and a coil acting as an antenna embedded within the plastic body of the card. When the card is brought into the electromagnetic field of the reader, the chip in the card is powered up and a data transfer can be started through a wireless communication protocol.

The following sequence of events describes at a high level what happens when a card is brought into the vicinity of a reader:

1. Energy and clock transfer *reader* → *card* for powering the integrated circuit.
2. Data transfer *reader* → *card*.
3. Data transfer *card* → *reader*.

1.1.3. Contactless Payment Systems

Contactless technology currently enjoys a great popularity amongst payment system integrators due to its convenience and ease of use. Purchases, charges and other types of money transactions speed up considerably, as there is no cash involved, hence the name cashless payments. Contactless payment transactions require no physical connection between the card and the reader device at the checkout. Instead of swiping or inserting a card, as it is the case with magnetic stripe cards and contact-based smartcards, the contactless smartcard is tapped on or held near a contactless reader device at the cash register, resulting in wireless debit or credit of the card balance.

To allow for rapid operation, the cash registers typically work autonomously without an on-line connection to a database in the back-end, and synchronize their data, for instance, blacklisted card numbers or the balance on the account of the cash register, only infrequently, e.g., once per day. This scenario also applies to the payment system analyzed in this thesis. For some systems, so-called "shadow accounts" exist for each card, containing the amount of money stored at the instant of the last synchronization. These shadow accounts can be used for detecting functional errors in the system or fraud.

1.1.4. ID Card

In this work we focus on a specific implementation of a contactless payment system. According to the manufacturer, approximately one million users in Germany hold the so-called "ID Card"[1], a dual-interface employee ID and payment card, i.e., both contact-based and contactless interfaces are provided. It replaces and combines functions of the traditional paper identity card, the cafeteria pay card, the copy and print card, and is also used for access control.

The contactless chip in the ID Cards can be charged at dedicated terminals or directly at a cash register with a maximum amount of €150 and allows for fast, convenient payments without the need to give change. The staff encourages the customers to hold

[1]Note that the name has been changed for anonymization

their wallets close to the reader, that interacts with the card wirelessly and carries out the payment.

The wireless technology used for the payment functionality implies new threats compared with contact-based systems, for instance, a card could be read out from the pocket or wallet without the owner taking note of it. Thus, we examine the security of the ID Card as an example for a widespread payment system and intend to answer the question: How secure are today's electronic payment systems really?

1.2. Security Issues

As with every new technology, there are always two sides to a coin. With the recent trend in Germany and other European countries towards issuing contactless smartcards in corporations and educational facilities, a number of privacy- and security-related concerns have been raised. The over-the-air communication channels used by the contactless smartcards can be exploited by an adversary in an unwanted manner. Some of these risks and considerations along with common remedies are summarized in the following.

1.2.1. Eavesdropping

The data transmitted over the air can be easily intercepted and/or modified by a third party. Eavesdropping describes a passive scenario, where an attacker can record a communication between a legitimate card and a genuine reader without the card owner being aware of it, e.g., by setting up a disguised antenna in vicinity of a genuine reader. This type of threats can be impeded considerably or even eliminated completely by implementing cryptographic mechanisms, both in the card and the reader, allowing for mutual authentication and encryption of transmitted data.

1.2.2. Covert Transactions

The owner of a contactless smartcard does not notice when a reader enters into a communication with the card. An adversary in possession of a fake reader can carry out covert transactions with the card to, e.g., debit the stored balance, or otherwise modify the data stored on the card. Therefore, a strong mutual authentication should precede any transaction in order to prevent such attacks. For not cryptographically-enabled smartcards an optional countermeasure would be to provide some means of user interaction whenever a transaction is about to occur, e.g., by requiring the card owner to press a button on the card, or make the card produce a sound to alert the owner about an unauthorized transaction.

1.2.3. Tracking

Each contactless smartcard has a unique identification number that is often transmitted at the beginning of communication in plaintext, even in cryptographically enabled smartcards. In most cases this is a fixed, factory-programmed number that cannot be changed after manufacturing. This fact enables an adversary to track a person holding the card by recording the ID number of the card. Furthermore, smartcards without cryptographic functions transmit data in plaintext over the air, thus facilitating tracking based on the actual data transmitted. Some of the newer and more sophisticated cryptographically enabled smartcards are capable of generating a random serial number in every transaction, thus preventing this attack, but these cards are still rather rare on the market.

1.2.4. Man-in-the-Middle

In a *passive Man-In-The-Middle (MITM)* attack[2], two adversaries work together to initiate the communication between a legitimate reader and a genuine card. The transmitted data is redirected in real-time via a separate channel, such that both parties will assume that they are in their direct vicinity and, e.g., carry out payments, while in fact they can be miles apart.

For an *active MITM* attack, some of the relayed information — e.g., a credit or debit value involved in a monetary transaction — may be altered or replaced with arbitrary data blocks by an adversary. Note that MITM is a very strong attack, as it operates on the bit level, and hence even allows for spoofing many systems with cryptographically secure authentications and strong encryption.

1.2.5. Side-Channel Attacks

Side-channel attacks make use of the information gained from the concrete physical implementation of a cryptosystem, rather than the theoretical weaknesses of the algorithm. This information can be obtained from various side channels existing in the system, such as the execution time of the algorithm, the power consumption for the individual algorithm steps or, particularly in case of the contactless smartcards, fluctuations in electromagnetic emanations. The information can be used to (partially) recover the secret key or compute the plaintext to a given ciphertext. Many of the cryptographically enabled contactless smartcards currently on the market are susceptible to side-channel cryptanalysis, even those based on proven secure encryption algorithms [26].

[2] A passive MITM attack is also termed “relay attack” or “mafia fraud”.

1.3. Scope

This work is organized as follows. Chapter 2 provides a basic overview over the ISO/IEC 14443 standard for contactless smartcards and gives a brief introduction into the communication principle of the ISO/IEC 14443-A compliant smartcards. The smartcards solution portfolio from NXP termed Mifare is introduced in Chapter 3. A specific focus is put on the most widespread member of this portfolio, the Mifare Classic, which is also used in the here analyzed payment system.

Chapter 4 describes the existing custom-built RFID equipment [25] used for the security analysis of the system. The embedded software framework developed for the RFID tool implements a combined attack on Mifare Classic to extract the secret keys and allows for comfortable reading and writing of the card content. It is presented in Chapter 5.

The practical system tests carried out in the course of this work and the discovered security flaws are given in Chapter 6. This chapter also discusses the implications of the vulnerabilities found in the contactless payment system. Some low-cost countermeasures and improvements are proposed that increase the overall security of the system. Finally, Chapter 7 summarizes the results and shows possible directions for future research.

2. Standards

2.1. Overview

ISO/IEC 14443 [21] is an international standard for contactless smartcards operating at $f_c = 13.56$ MHz in close proximity with a reader antenna. This is why the contactless smartcards complying to ISO/IEC 14443 are also called "proximity cards". Proximity Integrated Circuit Cards (PICCs) are intended to operate within approximately 8-15 cm of the reader antenna and support high-speed bit rates of up to 848 kbit/s. The wavelength of the emitted electromagnetic field is $\lambda = \frac{c}{f_c} = 22.1$ m, where c denotes the speed of light. The wavelength is several times greater than the typical operating distance between reader and card and therefore the communication and power link is purely inductive [12]. Virtually all contactless smartcards on the market are compliant to ISO/IEC 14443[1] which is further detailed in Section 2.2.

An alternative to ISO/IEC 14443, ISO 15693, is an ISO standard for "vicinity cards", i.e., cards which can be read from a greater distance as compared to proximity cards. ISO 15693 systems also operate at the 13.56 MHz frequency, and offer a maximum read distance of 100-150 cm. As the vicinity cards have to operate at a greater distance, the strength of the magnetic field necessary to establish a communication link is less (0.15 to 5 A/m) than that for a proximity card (1.5 to 7.5 A/m) and the maximum data rate is reduced to 26.48 kbit/s. One of the major target markets for ISO 15693-compliant contactless chips are the so-called Smart Labels for labeling drugs and medicines, and other goods tagging applications.

2.2. ISO/IEC 14443

The standard comprises four parts:

Part 1 specifies the size and physical characteristics of the card, as well as a list of environmental stresses that the card must be able to withstand without permanent damage to the functionality.

[1] According to data from NXP: http://mifare.net/about/standards.asp

Part 2 defines the RF power and signal interface for data transmission. Part 2 of the standard is divided into two half-duplex[2] signaling schemes denoted Type A and Type B. Both communication schemes provide a data bit rate of up to 848 kbit/s after initialization is completed. Data transmitted by the card is load modulated with a 847.5 kHz subcarrier. The card is powered by the RF field and no battery is required.

Part 3 gives specification of the initialization and anticollision protocols for Type A and Type B. The anticollision commands, responses, data frame, and timing are defined in Part 3. The initialization and anticollision scheme is designed to permit the construction of multi-protocol readers capable of communication with both Type A and Type B cards. An anticollision procedure ensures that, if there are multiple cards in reader's proximity, only one card at a time is selected for transaction.

Part 4 is optional and specifies the high-level data transmission protocols for Type A and Type B. The card reports to the reader if it supports the Part 4 commands in the response to the polling command (as defined in Part 3). The protocol defined in Part 4 is also capable of transferring application protocol data units as defined in contact-based smartcards standard ISO/IEC 7816-4 and of application selection as defined in ISO/IEC 7816-5.

Type A is by far the most widespread type of contactless smartcards on the market[3]. Its operation principle is described in the following.

2.3. Communication Principle Type A

The bit duration during the initialization of the card is always

$$t_b = \frac{128}{13.56 \text{ MHz}} \approx 9.44 \ \mu\text{s}.$$

After the initialization the bit duration depends on the mutually agreed data rate between the reader and the card and can be up to 848 kbit/s.

2.3.1. Communication Reader → Card

Modified Miller Coding

Type A signaling scheme utilizes 100% Amplitude-Shift Keying (ASK) of the RF field for communication from the reader to the card with Modified Miller encoded data.

[2] Communication only in one direction at a time
[3] http://mifare.net/about/standards.asp

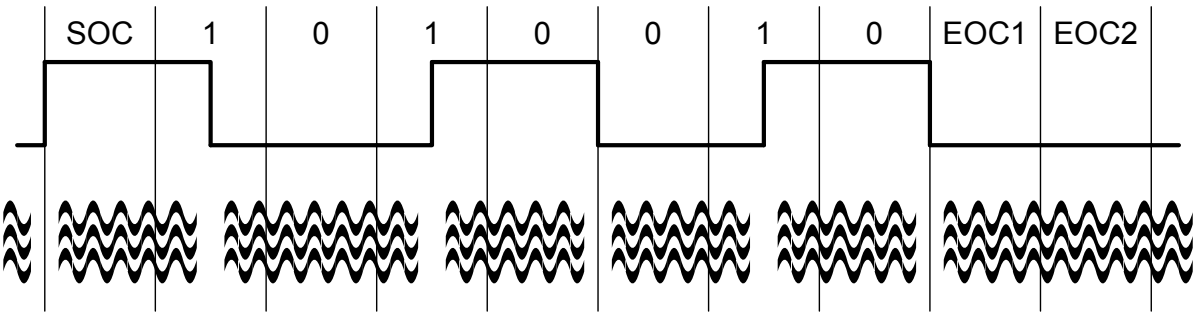


Figure 2.1.: Sample Miller-encoded data sequence transmitted from reader to card and the schematic representation of the resulting 100% ASK field modulation.

The data bits to be transmitted from reader to card are first encoded with the Miller Code. In the Modified Miller encoded data all transitions of the original Miller Code are represented by short (≈2.5 μs) pauses in the RF field. The field is turned off for short periods of time when the reader is transmitting. Therefore, the integrated circuit of the card must store enough energy in the internal capacitors to continue functioning while the RF field is temporarily switched off during field modulation.

Coding of the Data Bits

Start Of Communication (SOC), End Of Communication (EOC) as well as logic 1 and 0 levels are coded as follows.

SOC A pause at the beginning of the bit duration.

EOC Logic 0 followed by no modulation for a full bit period.

Logic 0 A pause at the beginning of the bit period, if the previous bit was logic 0 or SOC.
No modulation for a full bit period, if the previous bit was logic 1.

Logic 1 A pause after $\frac{t_b}{2}$.

A sample Miller-encoded bit sequence generated by the reader and the resulting waveform of the transmitted signal are illustrated in Figure 2.1.

2.3.2. Communication Card → Reader

Manchester Coding

Communication from card to reader utilizes On-Off-Keying (OOK) modulation of an 847.5 kHz subcarrier with Manchester encoded data. Thereby the subcarrier is switched

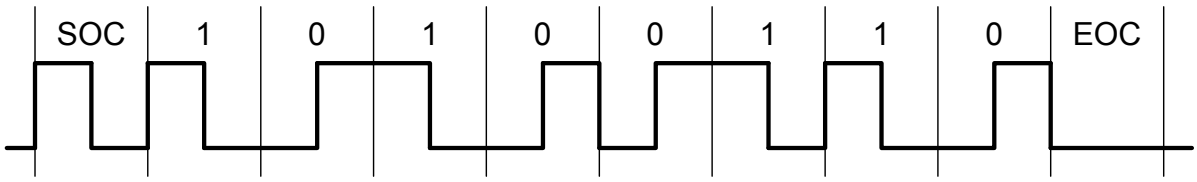


Figure 2.2.: Sample Manchester-encoded data sequence transmitted from card to reader.

on and off depending on the current logic value of the data bit. The card therefore draws more or less energy from the field leading to the 13.56 MHz carrier sine wave being "load modulated" with the resulting subcarrier signal in a way that the reader can sense the change.

Coding of the Data Bits

SOC, EOC as well as logic 1 and 0 levels are coded as follows.

SOC The carrier is modulated with the subcarrier for the first half of the bit duration $0 \leq t \leq \frac{t_b}{2}$.

EOC The carrier is not modulated with the subcarrier for one bit duration.

Logic 0 The carrier is modulated with the subcarrier for the second half of the bit duration $\frac{t_b}{2} \leq t \leq t_b$.

Logic 1 The carrier is modulated with the subcarrier for the first half of the bit duration $0 \leq t \leq \frac{t_b}{2}$.

A sample Manchester-encoded bit sequence generated by the card is presented in Figure 2.2. Note that the presence of multiple cards can be detected by the reader, as the not allowed state of the carrier being modulated with the subcarrier for a full bit duration may occur. This requires, however, that all cards in the field reply in a synchronous manner, which is one of the requirements of ISO/IEC 14443.

2.3.3. Card Initialization

Unique Identifier

Every ISO/IEC 14443-A compliant contactless smartcard contains a Unique IDentifier (UID) number, which can be a fixed or random number of a single (4 bytes), double (7 bytes) or triple (10 bytes) size. A fixed UID is stored in the read-only section of the EEPROM or some other read-only or one-time programmable memory on the card and, as the name suggests, uniquely identifies the smartcard often programmed by manufacturer. A random UID is randomly generated for every transaction with the reader.

Anticollision Procedure

During the initialization process the reader issues the REQuest A (REQA) command and all cards present in the field must reply with an Answer To Request A (ATQA) in a synchronous manner, so that the reader can detect the presence of multiple cards and choose one of them for further communication. To select one of the cards the reader starts the anticollision procedure consisting of one, two or three Cascade Levels (CLs) CL_n, depending on the UID size. In each CL a part of the UID is transferred from card to reader. Initially, the reader issues the ANTICOLLISION command, prompting all cards in the field to reply with their respective UIDs for CL_1. In case of more than one smartcards in the field, a bit collision can be sensed by the reader as the field is modulated for a full bit duration — the state that is not allowed in the Manchester coding scheme. In this case the reader modifies the ANTICOLLISION command accordingly by toggling the bit position where the collision has occurred in order to reduce the number of smartcards that reply with their UIDs and resends the ANTICOLLISION command.

After acquiring the collision-free UID for CL_1, the reader issues the SELECT command to select *all* cards with the same UID for CL_1, while cards with a different UID for CL_1 do not reply. All selected cards respond with a Select AcKnowledge (SAK) for CL_1. In case of the single-size UID the selection process ends here. For double- and triple-size UID the selection process has to be repeated for higher cascade levels CL_2 and CL_3 until the complete UID is acquired by the reader and one card in the field is selected.

In case the selected card supports the higher-level communication protocol specified in ISO/IEC 14443 part 4, the reader issues the Request for Answer To Select (RATS) command, indicating the maximum frame size supported by the reader. The card replies with the Answer To Select (ATS) which contains information about the maximum frame size and the bit rate in both directions supported by the card. The optional Protocol and Parameter Selection (PPS) sent by the card following the Protocol and Parameter Selection Request (PPSR) from reader allows to individually select the communication baud rate between reader and card.

In order to communicate with more than one card simultaneously, the reader can issue a HaLT A (HLTA) to put the currently activated card on hold. The card will remain in the suspended state until it receives the Wake-UP A (WUPA) command which reactivates the card for communication. The implementation details for all aforementioned commands can be found in the ISO/IEC 14443 specification.

Frame Formats

There are two types of frame formats for commands specified in ISO/IEC 14443-A: The short frame and the standard frame. The short frame format, visualized in Figure 2.3, consists of the SOC bit and seven data bits followed by the EOC bit. The standard

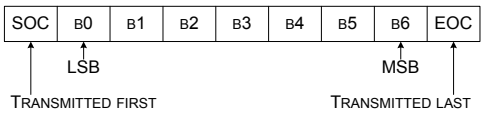


Figure 2.3.: Short frame format.

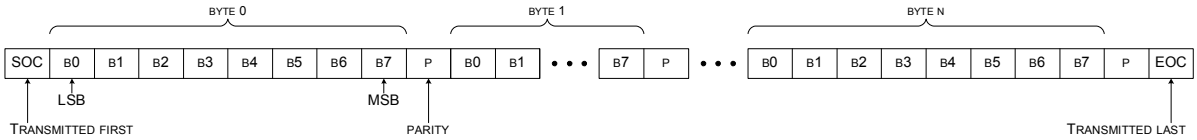


Figure 2.4.: Standard frame format.

frame, shown in Figure 2.4, consists of the SOC bit, data bytes, each one followed by a parity bit, and the EOC bit in the end. Note that all information bytes are transmitted with the least significant bit (LSB) first.

The parity bit appended to every byte is odd, i.e., the XOR of all 1's in the byte and the appended parity bit is always 1. The parity bit enables detection of an odd number of transmission errors, i.e., bit flips.

Only two commands are transmitted in the short frame format: REQA and WUPA. All other commands specified in ISO/IEC 14443 are transmitted in standard frames.

Frame Delay Time

The definition of the Frame Delay Time (FDT) is the time elapsed between two frames transmitted in opposite directions. For the commands REQA, WUPA, ANTICOLLISION and SELECT the FDT between the end of the last pause transmitted by the reader and the first modulation edge within the start bit transmitted by the card must be exactly $\frac{1172}{f_c} = 86.43\ \mu$s if the last bit sent by the reader was logic 0, and $\frac{1236}{f_c} = 91.15\ \mu$s if the last bit sent by the reader was logic 1. The exact timing, and hence the synchronous response of all cards in the field, is required for the anticollision procedure described above. For all other commands the card must ensure that its response is aligned to the bit grid. In this case the FDT is given by $\frac{128n+20}{f_c},\ n \in \mathbb{N},\ n \geq 9$ if the last bit sent by the reader was logic 0, and $\frac{128n+84}{f_c},\ n \in \mathbb{N},\ n \geq 9$ if the last bit sent by the reader was logic 1.

For the FDT in the opposite direction, i.e., the time elapsed between the last modulation transmitted by the card and the first pause transmitted by the reader, only the minimum time of at least $\frac{1172}{f_c} = 86.43\ \mu$s has to be considered.

Request Guard Time

The minimum time between the start bits of two consecutive REQA commands termed Request Guard Time (RGT) has the value $\frac{7000}{f_c} \approx 516\ \mu$s.

2.4. ISO/IEC 14443 Compliant Card Systems

2.4.1. Calypso

An ISO/IEC 14443 B compliant Calypso card [20] has been introduced and patented in 1993 by a French company Innovatron, with the first implementation following in 1996. The Calypso technology offers microprocessor-based contactless interface, which is currently deployed in public transit systems of some major European cities, including Paris, Lisbon and Brussels.

The data stored on the card is organized in hierarchical files compliant to ISO/IEC 7816-4 [22]. Calypso employs a proprietary secure session mechanism that allows to mutually authenticate reader and card, encrypt all data transmitted, and also deal with the problems occuring when the card accidentally leaves the RF field of the reader — the mechanism called "ratification". Different types of Calypso compatible cards offer different levels of security. Calypso allows the use of the Data Encryption Standard (DES), DES-X and Triple-DES (3DES) algorithms with the 64, 120 and 112 bit keys, respectively.

2.4.2. HID iClass

Launched in 2002 by HID Global, the HID iClass contactless smartcard [17] is delivered in versatile packages offering compliance to both ISO/IEC 14443 A and B types. In addition to physical access control, HID iClass cards provide support for multiple applications such as biometric authentication, cashless payment and PC log-on security.

Each card is offered in 256 byte, 2 kB and 4 kB memory configurations and can be used for multiple applications simultaneously. The maximum data rate for communication with the card is set to 212 kbit/s. A mutual authentication between the card and the reader and encryption of the transmitted data is based on the non-disclosed proprietary encryption algorithm using 64-bit diversified keys. Additionally, according to manufacturer, higher levels of security may be achieved with the supported DES and 3DES encryption.

2.4.3. Legic Advant

Legic Advant transponder chips compliant to ISO/IEC 14443 A are offered in three memory packages: 512 byte, 2 kB and 4 kB. The memory can be subdivided in up to 127 applications. The data rate for communication is 106 kbit/s. The Legic safeID feature allows for secure (authenticated) transfer of the UID of the card, which can be single (4 byte) or double (7 byte) size.

Encrypted data transfer and encrypted data storage can be defined per application. Support for DES, 3DES as well as the proprietary LEGIC encryption algorithm is included for this purpose.

2.4.4. e-Passport

An e-Passport is a combined paper and electronic passport that has been introduced in the European Union and many other countries worldwide. Technical specifications for the new passports published by the European Commission [11] are binding for the Schengen-agreement countries. All e-Passports issued in the EU contain an embedded contactless chip that holds the same information that is printed on the identity information page of the passport: the name of the holder, date of birth, and a facial image. Optionally, an e-Passport can contain biometric identifiers, e.g., a fingerprint or an iris scan of the pass holder.

Many countries across the world adopted e-Passport technology, as it promises to reduce fraud, ease identity checks, and enhance security. Other optional services that can be facilitated with the introduction of the e-Passport include registering a change of address, registering a vehicle, filing a tax return (eGovernment), casting a vote in elections (eVoting) and other services provided by the retail, banking and insurance sectors (eBusiness). To insure interoperability between different countries, all e-Passports comply to ISO/IEC 14443 specification.

In Germany, e-Passports have been introduced in November, 2005. The first generation chip contained only a photograph as a biometric identifier, with the second generation following in 2007, which now additionally contains images of two fingerprints of the card holder. German e-Passports are based on the microcontroller-based SmartMX [37] chips from NXP with 72 kB EEPROM. Security mechanisms employed are based on public-key Elliptic Curve Cryptography (ECC) and symmetric Advanced Encryption Standard (AES) thus permitting usage of short key lengths of, e.g., 224 bit[4].

A rigorous research has been spurred by the introduction of e-Passports around the globe, which is however out of scope of this thesis. An interested reader is referred to [5, 18, 23, 29].

[4]Source (German): `http://www.teli.de/pdf/DuD_3_2008_Sicherheitsmechanismen_Personalausweis_pdf.pdf`

2.4.5. Mifare

The Mifare product line consisting of Mifare Classic [39, 38, 40], Mifare Ultralight (C) [35, 43], Mifare DESFire (EV1) [36, 41] and Mifare Plus [42] is now maintained by the Philips semiconductor spin-off, NXP. All Mifare cards comply with ISO/IEC 14443 A. Mifare cards are memory cards, i.e., the information is stored in the internal EEPROM with an integrated digital control unit to handle the communication with a reader, which is for some cards secured with cryptography.

Mifare Ultralight is the smallest member of the Mifare family without any cryptographic mechanisms implemented. All sectors can hence be easily read out or modified and most applications using this card rely on the fixed, factory programmed UID that allows to distinguish the cards.

Mifare Classic chips are fairly cheap and therefore made their way into many public transportation systems such as the Octopus card in Hong Kong [45], the Oyster card in London [53], the OV-chipkaart in the Netherlands [48] and the CharlieCard in Boston, USA [30]. Mifare Classic promises to prevent replay attacks, cloning and eavesdropping due to the integrated proprietary CRYPTO1 stream cipher. That is why many universities, corporations and even government agencies use Mifare Classic chips for access control and cashless payments. Mifare Classic cards make an integral part of this thesis and are therefore introduced in detail in Chapter 3.

The high-end class of Mifare is termed DESFire and in the latest version is advertised with a special protection against so-called power-analysis attacks that use measurements of the actual implementation of cryptography to reveal secret keys of even mathematically secure ciphers. These cards are more expensive than their Classic counterpart (more than €2 as compared to less than 50 cents for a Classic) and to our knowledge, are not yet widely deployed in the field.

3. Mifare Classic

3.1. Introduction

The document "MIFARE Type Identification Procedure" [44] published by NXP describes how to differentiate between the members of the Mifare card IC family. It shows how to use the information obtained from the card during the initialization procedures to deliver the chip type information for all Mifare ICs. Analyzing the security of the payment system, we observed that the ID Card replies with `ATQA = 00 04` and `SAK = 08`, indicating that it contains a Mifare Classic chip [38]. Hence, this chapter aims to describe the technical characteristics and the security architecture of the Mifare Classic, and present a survey of the existing attacks.

The Mifare Classic chip was introduced in 1995[1] by Philips and is today the most widespread type of contactless smartcards. According to manufacturer, more than one billion Mifare Classic chips being currently deployed in ticketing, access control and payment systems worldwide covering more than 70%[2] of the contactless smartcard market.

3.2. Memory Structure

The memory of a Mifare Classic 1K chip is partitioned into 16 sectors, each consisting of four blocks, as illustrated in Figure 3.1. The read-only block 0 of the first sector contains the factory-programmed UID of the card and other fixed data of the manufacturer. The fourth block of each sector, termed sector trailer, contains two 48-bit keys (key A and key B) used for the authentication to that sector and a set of bits defining the access conditions[3] for each individual block. All other memory blocks of the card can be used to store data according to their access conditions. The default key A or B on a Mifare Classic card from NXP or Philips is `0xFFFFFFFFFFFF`, whereas Infineon cards have `0xA0A1A2A3A4A5` as the default key A and `0xB0B1B2B3B4B5` as the default key B.

Two types of blocks are supported by the Mifare Classic system: data blocks and value blocks. Value blocks are especially well suited for payment applications. While normal

[1] Source: `http://www.nxp.com/products/identification/mifare/classic`
[2] According to NXP.
[3] For details on the coding of the access condition bits, refer to [38].

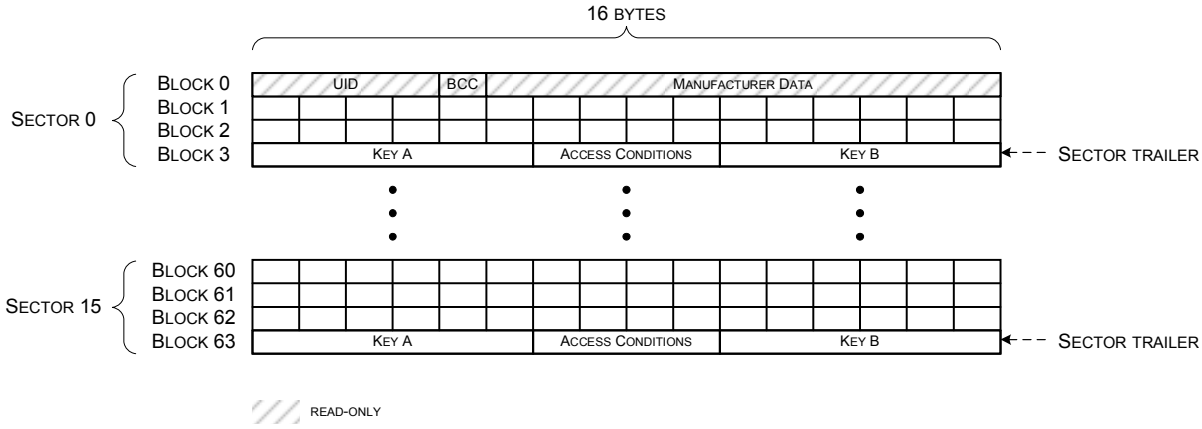


Figure 3.1.: The memory structure of a Mifare Classic card.

data blocks can be written with arbitrary data, value blocks need to be conform to a special format. Each value block contains a 4-byte unsigned value which is stored once inverted and twice non-inverted. The last 4-byte segment of the block may be optionally used to store a 1-byte address pointer, which saves the storage address of a block, when implementing a backup management. The address pointer is stored four times: twice inverted and twice non-inverted.

Prior to performing any read or write operations, the reader must perform an authentication with one of the sector keys of type A or B. The authentication is only valid for one sector at a time. Once the reader wants to access blocks of another sector, it has to repeat the authentication process with the appropriate sector key, as detailed in Section 3.4.

3.3. Command Set

The Mifare Classic high-level transmission protocol diverges from part 4 of ISO 14443. It consists of a proprietary Mifare command set that has been reverse-engineered in the course of this work. The commands for a Mifare Classic reader are summarized in Table 3.1. Note that all higher-level commands in standard frame format are appended with two Cyclic Redundancy Check (CRC) bytes according to ISO/IEC 14443 A for data integrity. The coding of the three short-frame card responses is as follows:

- Acknowledged: `ACK = 0x0A`
- Not acknowledged (not allowed): `NACK = 0x04`
- Not acknowledged (transmission error): `NACK = 0x05`

Table 3.1.: Proprietary command set of a Mifare Classic reader.

Type	#	Command	Description
AUTHENTICATION	1	`0x60 0xXX`	Authenticate for block `0xXX` with key A
	2	`0x61 0xXX`	Authenticate for block `0xXX` with key B
DATA BLOCKS	3	`0x30 0xXX`	Read block `0xXX`
	4	`0xA0 0xXX`	Write block `0xXX`
VALUE BLOCKS	5	`0xC0 0xXX`	Decrement value block `0xXX`
	6	`0xC1 0xXX`	Increment value block `0xXX`
	7	`0xC2 0xXX`	Restore value block `0xXX`
	8	`0xB0 0xXX`	Transfer to value block `0xXX`

The authentication process with the associated commands is further discussed in Section 3.4.

Reading a data/value block is accomplished by sending the read block command followed by a block number to be accessed. The card replies with 16 data bytes or NACK. In order to write a block a reader issues the write command, waits on the response of the card (ACK/NACK), sends 16 data bytes and, again, waits on the response of the card (ACK/NACK). When writing a value block, the data format is checked by the integrated circuit and, if it does not comply to the format of a value block, the protocol aborts.

Four commands are available for value blocks only: decrement, increment, restore and transfer. To increment/decrement a value stored in the value block, the reader first issues the increment/decrement command, waits on the card response (ACK/NACK), then sends the 4-byte unsigned integer value to increment/decrement by, and finally receives ACK/NACK from the card. Increment and decrement operations load the value stored in the value block to a temporary register inside the integrated circuit of the card, where all value operations are performed. In order to write the processing result back to the EEPROM the transfer command must be issued after all value operations are completed. The restore command restores the original value of the value block discarding all previously performed operations.

3.4. The CRYPTO1 Cipher and Communication Protocol

All members of the Mifare family comply to Parts 1-3 of the ISO/IEC 14443 A [21]. While ISO 14443 also specifies higher data rates, Mifare Classic cards communicate with

a fixed rate of 106 kbit/s and use a higher-level communication protocol that diverges from Part 4 of the standard. This proprietary protocol for authentication and subsequent data encryption promises to prevent replay attacks, cloning, and eavesdropping by means of NXP's CRYPTO1 cipher.

3.4.1. Cipher

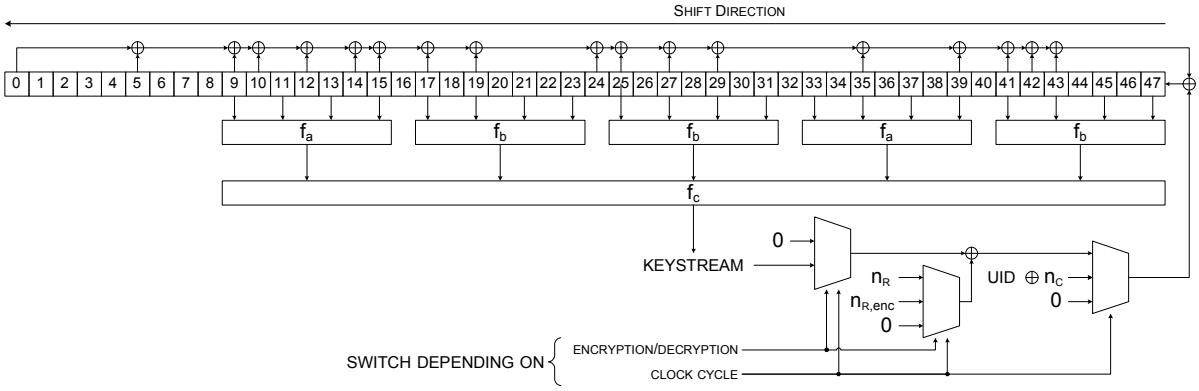


Figure 3.2.: CRYPTO1 top-level structure.

The CRYPTO1 stream cipher, illustrated in Figure 3.2, is based on a 48-bit Linear Feedback Shift Register (LFSR) with the feedback polynomial $x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^{9} + x^{7} + x^{6} + x^{5} + 1$ and six non-linear filter functions, of which there are only three different types f_a, f_b and f_c. They are defined by:

$$
\begin{aligned}
f_a(x_0, x_1, x_2, x_3) &= ((x_0 \vee x_1) \oplus (x_0 \wedge x_3)) \oplus (x_2 \wedge ((x_0 \oplus x_1) \vee x_3)),\\
f_b(x_0, x_1, x_2, x_3) &= ((x_0 \wedge x_1) \vee x_2) \oplus ((x_0 \oplus x_1) \wedge (x_2 \vee x_3)),\\
f_c(x_0, x_1, x_2, x_3, x_4) &= (x_0 \vee ((x_1 \vee x_4) \wedge (x_3 \oplus x_4))) \oplus\\
&\qquad ((x_0 \oplus (x_1 \wedge x_3)) \wedge ((x_2 \oplus x_3) \vee (x_1 \wedge x_4))),
\end{aligned}
$$

where $\vee, \wedge, \oplus$ denote logical OR, AND and XOR operations, respectively. The cipher is outdated and has to be regarded as insecure, since it is possible to recover the small 48-bit state of the LFSR by means of an exhaustive search. In addition, several other weaknesses in the cipher as well as in its implementation have been found as detailed in Section 3.5.

3.4.2. Pseudo-Random Number Generator

An integrated Pseudo-Random Number Generator (PRNG), clocked simultaneously with the cipher, is utilized to generate nonces for the authentication between a reader and a card. It is based on a 16-bit LFSR with feedback polynomial $x^{16}+x^{14}+x^{13}+x^{11}+1$. The state of the LFSR is expanded to 4-byte values by simple shifting as illustrated in

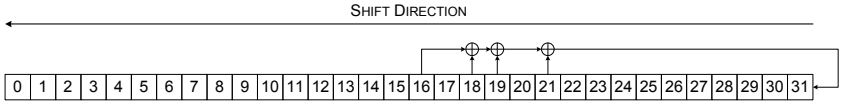


Figure 3.3.: Structure of the internal Mifare Classic Pseudo-Random Number Generator.

Figure 3.3. Therefore the resulting pseudo-random numbers contain only 16 bit entropy instead of the possible 32 bit.

3.4.3. Operation and Authentication

Table 3.2.: A typical authentication between a reader R and a Mifare Classic card C.

#	Direction	Hex Bytes	Explanation
1	R → C	26	REQA
2	C → R	04 00	ATQA
3	R → C	93 20	ANTICOLLISION
4	C → R	74 82 F9 D7 D8	UID || BCC
5	R → C	93 70 74 82 F9 D7 D8 BB AD	SELECT || UID || BCC || CRC
6	C → R	08 B6 DD	SAK || CRC
7	R → C	60 00 F5 7B	AUTH || sector number || CRC
8	C → R	CD 98 4F AE	n_C
9	R → C	73 09 C4 F8 10 2D 97 4E	$n_R \oplus \mathrm{ks}_1 \;\|\|\; a_R \oplus \mathrm{ks}_2$
10	C → R	B7 52 8C 02	$a_C \oplus \mathrm{ks}_3$
		...	
		AUTHENTICATION ESTABLISHED, BLOCK READ/WRITE COMMANDS FOLLOW	
		...	

A three-pass authentication between a reader (R) and a Mifare Classic card (C) is presented in Table 3.2, where || denotes a concatenation. After the initialization and anticollision part of the ISO 14443 protocol in Steps 1 to 6, the reader sends the command 0x60 or 0x61 for an authentication with key A or key B, respectively, followed by the

number of the sector that it wants to access. The card replies with a random nonce n_C. The LFSR of the cipher is loaded with the respective secret key, and during the clock cycles $i = 0 \dots 31$ the value UID $\oplus\, n_C$ is clocked in, while no keystream is sent to the output. Starting with the message in Step 9, the communication is encrypted by XORing the bits of the plaintext with the keystream bits for $i > 31$, which are grouped and denoted as ks_1 for $i = 32 \dots 63$, ks_2 for $i = 64 \dots 95$, ks_3 for $i = 96 \dots 127$, and so on. The reader generates a nonce n_R and concatenates it with its response to the challenge of the card a_R, which is obtained by loading n_C into the PRNG and clocking it 64 times. The card receives the encrypted nonce of the reader $n_{R,enc} = n_R \oplus \mathrm{ks}_1$ and the (decrypted) n_R is input to the cipher during the clock cycles $i = 32 \dots 63$. Note that, due to the feedback structure of the cipher, the later bits of $n_{R,enc}$ are affected by the earlier bits of n_R. At this point the initialization is complete, and the LFSR autonomously produces one bit of keystream per clock cycle. The card finally replies with a_C, generated by the PRNG 96 clock cycles after n_C was output. The mutual authentication is successful if the received values for a_C and a_R are identical with those generated by the internal PRNGs.

Cipher Rollback

Note that once the internal state of the LFSR becomes known at any point in time during the operation of the cipher, it can be easily rolled back by computing the bits that "fall out" in each clock cycle and shifting the LFSR in opposite direction. This approach can be used to obtain the initial value loaded into the LFSR, which is the secret key. In addition to the known internal state, values n_C and n_R that are input to the cipher for the first 64 clock cycles are required to perform the rollback operation.

3.5. Previous Attacks on Mifare Classic

Since its invention in 1995 by Philips [34], all details of the Mifare Classic chip had been kept secret until 2007, when Nohl et al. [33] succeeded in reverse-engineering the cipher. They manually grinded layer by layer of the silicon chip, took microscopic pictures, and deducted from the layout of the chip to the cipher. The authors instantly discovered numerous weaknesses in CRYPTO1, as well as flaws in the PRNG. In the following, we summarize the state-of-the-art of the research on Mifare Classic that is the basis for our system break.

3.5.1. Keystream Recovery

The first discovered weakness [33] is that the random nonce generated by the card is only dependent on the time elapsed between the power-up of the card and the issuing of the

authentication command by the reader. Hence, by controlling the timing, the authors are able to reproduce the same nonce with a certain probability. The consequence of this weakness is the keystream recovery attack described by de Koning Gans et al. [8]. Prerequisite for this attack is a recorded authentication session between a genuine reader and a Mifare Classic card. Afterwards, several queries to the card are issued by a specially prepared reader. As for all stream ciphers, a part of the keystream can be recovered from a known pair of plaintext and ciphertext. Using the fact that key A of each sector is not readable and that trying to read it returns encrypted zeros as a response, the keystream used to encrypt the first six bytes of each block of that sector can be recovered. Often key B is not readable as well, which reveals another six bytes of the keystream. Furthermore, the first five bytes of the manufacturer data (see Figure 3.1) are known, altogether allowing to recover all 16 keystream bytes for sector 0 and up to 12 keystream bytes for higher sectors.

With this approach, the attacker is also able to replay the previously eavesdropped communication session between a genuine reader and a genuine card, which can be used to, e.g., increase the credit balance stored on the card multiple times. The card will accept the encrypted commands, given the card and the reader nonce are the same as in the originally recorded authentication session.

The attack does not enable card cloning, as the cryptographic keys remain secret. However, if an attacker is able to eavesdrop on the communication between card and reader, and can afterwards access the card long enough for the second part of the attack, it is possible to read or modify the whole sector 0 and partially the higher sectors.

3.5.2. Key Recovery from Genuine Authentications

The possibility to recover the keystream led to an attack proposed by Garcia et al. [14], where one can extract the secret key from just two eavesdropped authentications between a card and a genuine reader. The goal is to retrieve the state of the LFSR at some point in time.

First, 32 bit of keystream ks_2 used to encrypt a_R are computed using the ciphertext $a_R \oplus \mathrm{ks}_2$ from a recorded authentication session. Note that the plaintext value a_R can be obtained from n_C which is transmitted in plaintext. Afterwards, the filter functions of CRYPTO1 are inverted to calculate on average $2^{48-32} = 2^{16} = 65536$ key candidates for the state of the LFSR, of which the correct one is obtained offline by rolling back the cipher (cf. Section 3.4.3) and checking it against a second authentication. No precomputation is required and, after recording the two authentication sessions, it takes only 0.1 s to recover the secret key of one sector using ordinary computers, as the authors claim. Note that eavesdropping can be nontrivial in a real-world scenario.

This attack is the first to exploit another weakness of Mifare Classic cards concerning the parity bits. According to ISO 14443 [21], each transmitted byte has to be appended

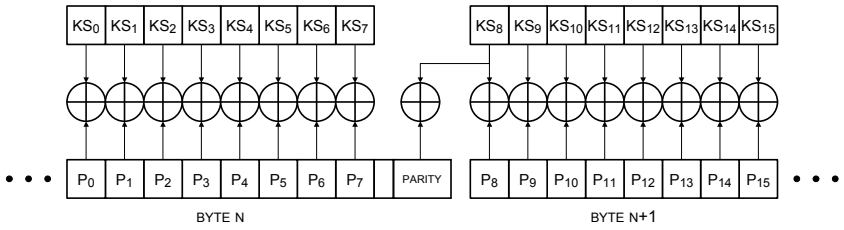


Figure 3.4.: Principle of the parity bit encryption with the CRYPTO1 keystream.

with a parity bit in order to detect transmission errors. However, the parity bits in the Mifare Classic system are computed over the plaintext instead of the actual bits transmitted. As shown in Figure 3.4, the parity bit of byte N of the plaintext is furthermore encrypted with the same keystream bit KS_j that is used to encrypt the first bit P_j of the next byte $N + 1$. This vulnerability contradicts a generally accepted stream cipher paradigm stating that the keystream bits should never be reused to encrypt multiple plaintext bits [52].

3.5.3. Card-Only Key Recovery

After the previous findings, the first card-only attacks emerged that do not require to eavesdrop on a communication with a genuine reader and are hence more realistic. An attacker only needs to be close enough to the targeted Mifare Classic card to activate it and communicate with it by means of a special-purpose reader, in order to retrieve the state of the LFSR and thus recover a secret key.

Garcia et al. [15] propose four different attacks to obtain the secret key of one sector. Their first proposal is an offline brute-force attack on the entire 48-bit key space of the cipher, after carrying out 1500 authentication attempts. The exhaustive key search takes $\approx$ 50 min with a special-purpose code breaker COPACOBANA [28]. For their second attack, the authors need to keep n_C constant and vary n_R to gain some knowledge about the internal cipher state. On average, 28500 queries to the card with a fixed card nonce are required, which takes about 15 min. The offline brute-force search takes approx. 1 min on a standard PC. To perform the third attack, an attacker needs to precompute a table with a size of 384 GB. Then, she keeps n_R constant and varies n_C, ultimately revealing a special internal cipher state. The required 4096 authentication attempts on average can be obtained in about two minutes. Finally, the fourth attack assumes that one sector key is already known. An authentication to that sector is decrypted to obtain the corresponding n_C. The subsequent n_C that is used for authenticating to another sector can be guessed due to the weak PRNG. Since the new n_C is encrypted with the key of the new sector, an attacker recovers 32 bits of keystream. This attack is very efficient, as it requires three authentication attempts and only 2^{16} key candidates need

to be checked, which takes under a second on a standard desktop PC.

In addition to fixing n_C and exploiting the parity weakness described above, the authors make use of the fact that in Step 10 of the protocol (see Table 3.2) the card sometimes replies with four bits instead of four bytes. This happens if and only if the parity bits sent along with the eight-byte reader cryptogram $n_{R,enc}||a_{R,enc}$ are correct and a_R is incorrect, i.e., not accepted by the card. The four bits sent in this case are an encrypted `NACK = 0x5` command. This knowledge allows to establish a side channel to recover four bits of the keystream for each failed authentication attempt with correct parity bits, occurring with a probability of 1/256.

The latest and most efficient attack is proposed by Courtois [6]. It requires only 300 queries to the card on average and a few seconds of off-line computation to find the sector key. The time required to collect data from the queries is reported to be 5 min due to the card nonce fixing issues. Additionally to the weaknesses described in the previous attacks, Courtois exploits a mathematical vulnerability of the filter functions used in CRYPTO1 to mount a differential attack on the cipher. The following steps are involved in the attack:

1. First, after receiving n_C we need to find the right combination for the eight bytes of the randomly chosen reader cryptogram $n_{R,enc}||a_{R,enc}$ that is kept fixed in each iteration, and the eight parity bits that are variable in each iteration, such that the card replies with four bits (encrypted `NACK`), indicating that the parity bits were correct. In case the parity bits are not correct the card does not reply at all. For this step on average $\frac{2^8}{2} = 128$ queries to the card are required.
2. Then we keep the first 29 bits of $n_{R,enc}$ and the first three parity bits from step 1 fixed. Additionally we fix the card nonce n_C from step 1 through precise timing control. We alter only the last three bits of $n_{R,enc}$ ($2^3 = 8$ cases) choosing $a_{R,enc}$ and the last five parity bits at random in each authentication attempt. After on average $\frac{2^5}{2} = 16$ queries, the card replies with encrypted `NACK` giving 4 bit of keystream. This process is repeated eight times so that finally eight encrypted `NACK` replies from the card are obtained, giving $4 \cdot 8 = 32$ bit of keystream.
3. The first four keystream bits that stem from the first authentication attempt are used to reduce the key space by a factor $2^4 = 16$ and to compute candidates for the state of the LFSR prior to encryption of `NACK`. Then, the corresponding states for the other seven authentications are predicted and compared with the received keystream bits. A match is found, if and only if the corresponding output of the filter functions ks_1 does not depend on the last three bits of n_R. Due to the bad non-linear properties of the boolean functions f_b and f_c (cf. Section 3.4.1), the output keystream does not depend on the last three bits of the LFSR state with a probability of 70%. Therefore the success rate for this attack is 70%. If a contradiction is found at any stage then the attack was unsuccessful and must be repeated starting with step 1. The offline computations are negligible, as on

average only $2^{48-32} = 2^{16}$ key candidates need to be tested in order to recover a secret key.

The most practical and efficient card-only attack known to date is the combined differential and nested authentication attack. The first key is recovered using the differential attack, which takes a few minutes depending on the implementation platform. The remaining 15 keys can be recovered with the nested authentication attack almost instantaneously requiring less than a second of offline computation. Note that the authors of the above summarized papers report difficulties in fixing the nonce of the card n_C in practice, which is crucial for the attacks.

4. Attack Hardware

The hardware equipment used for the tests with the contactless payment system is presented in this chapter.

4.1. RFID Tool

For the security analyses in this thesis we used a self-built, freely programmable device termed "RFID Tool" [25]. The tool consists of a low-level RFID reader and an additional card-emulation device, presented in the following.

4.1.1. Reader

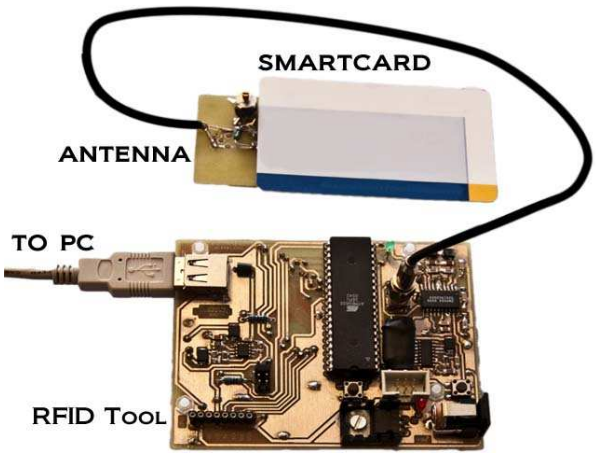


Figure 4.1.: RFID tool.

Figure 4.1 shows the low-level reader integrated on a Printed Circuit Board (PCB), that can be built for less than €40 from off-the-shelf equipment, is based on an Atmel ATMega32 [1] microcontroller (μC) clocked at 13.56 MHz. ATMega32 features, amongst others, 32 kB Flash Random Access Memory (RAM), 2 kB internal Static

Random Access Memory (SRAM), 1 kB non-volatile EEPROM and an Analog to Digital Converter (ADC). For applications with high timing requirements two 8-bit timers and one 16-bit timer are provided that can be operated in different modes. The software running on the μC can be easily updated from a PC via a Universal Serial Bus (USB) using the In-System Programming (ISP) interface, without the need to remove it from the PCB.

The analog front-end is provided by the EM 4094 RF-transceiver [10] which is capable of 100% ASK required for ISO 14443 A compliant operation and costs less than €5. Generally, the ISO 14443 B compliant operation is also supported although it is not currently implemented. A high input level on the DIN pin switches off the field instantenously, while a low level switches it on. Thus, in contrast to commercially available products, the RFID tool reader allows to fully control the RF field with a high timing accuracy of approx. 75 ns, corresponding to the duration of one clock cycle. This function was extensively used for our attacks on Mifare Classic as described in Chapter 6. Various antennas and power amplifiers can be connected to the reader via an output socket that is matched to 50 Ω, in order to extend the eavesdropping and activation range out of the bounds of the specification.

To disburden the μC the creation of pauses in the field required for transmitting Modified Miller encoded data is carried out by a 74123 [50] monoflop, that creates a pulse on every rising edge on its input, driven by the μC. The output pulses are applied to the EM 4094 RF-transceiver DIN input pin resulting in the field being switched off for a pulse duration ($\approx 2.5\ \mu$s). Alternatively, the pulses can be created with two additional monoflops which are capable of converting the Miller encoded data to Modified Miller without the μC, e.g., during a MITM attack with a fake tag, as described in Section 4.1.2.

The Manchester coded data modulated with the subcarrier (cf. Section 2.3.2) output by the EM 4094 is demodulated using an envelope detector circuit. The signal is rectified by a diode and then fed into a low-pass filter. The presence of the subcarrier is detected by an LM 311 [32] voltage comparator, which outputs Manchester encoded data, that can be further processed by the μC.

A circuitry for short adjustable time delays has been added to the RFID tool in order to compensate for a small time delay introduced by the relaying of data in the context of a MITM attack. This step is required to synchronize the response of the card with the bit grid as specified in ISO 14443.

USB communication with a PC or other USB-enabled hardware is enabled by the on-board FTDI FT245R [13] USB chip. Eight data bits are transmitted in parallel in either direction by controlling read or write input pins from the μC. Using the supplied Virtual COM Port (VCP) driver, the USB port can be accessed as a standard serial COM port from a PC, while maintaining a maximum possible data rate of 1 MB/s.

Besides the reset button, there is an additional used-definable button integrated on the PCB, which can be used for interaction with the software of the μC. It is particularly

practical in case of a stand-alone operation of the RFID tool, where no input to the μC via USB is possible. Furthermore, two Light-Emitting Diodes (LEDs) — green and red — are provided as indicators for the status of the reader or for the in-system debugging purposes.

Every bit sent via the RF interface — including chosen challenges for the authentication protocol, intentionally wrong calculated checksums and parity bits — is fully controlled by the attacker. In case of a typical commercial reader, nonces are randomly chosen and parity bits are automatically generated, without that an attacker can directly control the process – some parameters may even not be available for her.

4.1.2. Fake Tag

Figure 4.2.: Card emulation device “fake tag”.

A device termed “fake tag” shown in Figure 4.2 has been developed that cooperates with the reader and allows to emulate any contactless smartcard according to the ISO 14443 standard. The fake tag can be powered by an external power supply, e.g., a small lithium battery, to allow for stand-alone operation in the field, which was crucial for the practical security analysis.

The subcarrier required for the load modulation of the RF field is derived from the field generated by the reader by means of a 4-bit binary counter 74393 [49] that toggles every $2^3 = 8$ clock cycles, thus delivering the desired frequency of $\frac{f_c}{16} = 847.5$ kHz. A transistor switches the resistor in parallel to the antenna of the fake tag depending on the incoming Manchester coded data, thus achieving load modulation with the 847.5 kHz subcarrier.

The RFID reader and the fake tag allow to conduct different types of analyses and attacks on real-world systems. All further details about the hardware and practical applications are described in [24, 25].

4.1.3. Modes of Operation

The RFID reader and the fake tag support three main modes of operation, either controlled by a PC via USB or in a stand-alone battery-operated mode:

- Reader mode: freely programmable ISO 14443 (and hence Mifare-compliant) reader.
- Card mode: emulation of any ISO 14443 compliant contactless smartcard, e.g., Mifare, by means of the fake tag.
- MITM mode: both reader and fake tag are utilized to relay (and maybe modify) the data between a genuine reader and a genuine contactless smartcard in real-time, e.g., to pretend the vicinity of a genuine card to a genuine reader, which in reality can be miles apart from each other.

While the reader is used for our security tests and attacks on the ID card, as detailed in Chapter 6, the fake tag facilitates emulation of any Mifare Classic card, e.g., with an *arbitrary* UID, or combined with the reader in the context of a MITM attack. In order to reverse-engineer unknown protocols, the data exchanged between the two entities in all operation modes can be stored in the EEPROM on the reader or directly sent to a PC via USB.

4.2. AVR Dragon

Figure 4.3.: Atmel AVR Dragon programming device, Source: [2].

The programming of the ATMega32 μC was accomplished by means of a multi-purpose programming device from Atmel termed AVR Dragon [2] illustrated in Figure 4.3. The AVR Dragon is a low-cost development tool that supports all programming modes for the AVR device family, including the ATMega32 of the RFID tool. It also includes complete emulation and debugging support for devices with 32 kB or less Flash memory.

There is a number of programming interfaces supported by AVR Dragon:

- ISP
- High Voltage Serial Programming (HVSP)

- Parallel Programming
- Joint Test Action Group (JTAG) Programming

Table 4.1.: Pinout match of the AVR Dragon ISP interface and the programming header on the RFID tool.

AVR Dragon	⟷	**RFID Tool**	**Description**
PIN-1	⟷	PROGRAM-4	MISO
PIN-2	⟷	PROGRAM-3	VTG/VCC
PIN-3	⟷	PROGRAM-6	SCK
PIN-4	⟷	PROGRAM-2	MOSI
PIN-5	⟷	PROGRAM-1	RESET
PIN-6	⟷	PROGRAM-5	GND

The AVR Dragon is powered via USB and is also capable of sourcing an external target. For our purposes the AVR Dragon was connected via USB to a host PC and through a customized 6-pin cable to the ISP header mounted on the PCB of the RFID tool. The pinout of the ISP header of the AVR Dragon as described in [2] and the corresponding pins on the PCB of the RFID tool are presented in Table 4.1.

5. Software Framework

The software framework developed in the course of this thesis consists of two main parts: Embedded Software for the μC and Desktop Software running on the host PC. Both were used for the subsequent security analysis of the system, and are therefore presented in detail in this chapter.

5.1. Toolchain

The toolchain that was used for developing the embedded and desktop applications is presented in this section.

5.1.1. Atmel AVR Studio

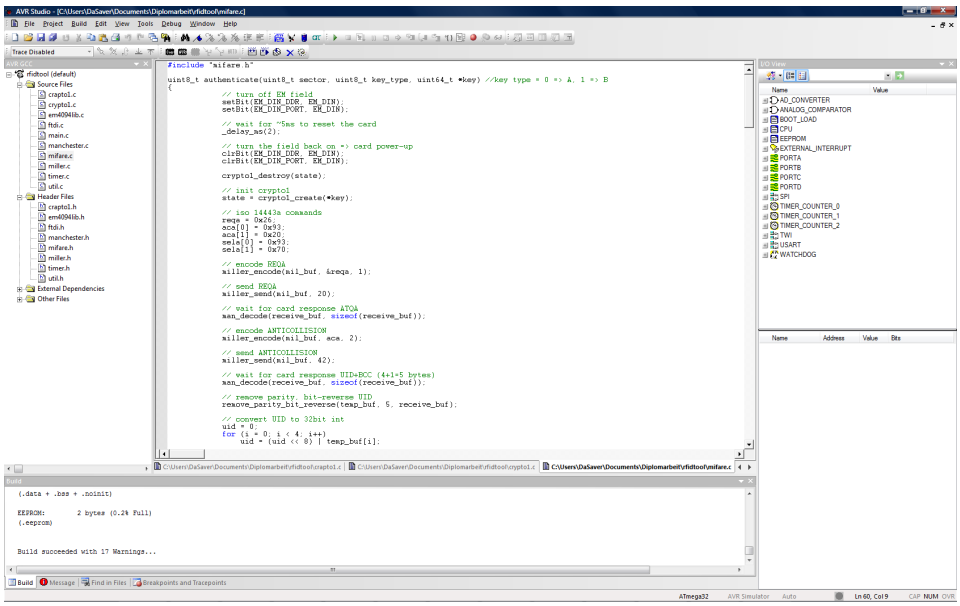


Figure 5.1.: Main screen of Atmel AVR Studio 4.

AVR Studio [3] version 4.16 from Atmel shown in Figure 5.1 was used for developing embedded software for the ATMega32 μC. AVR Studio is an Integrated Development Environment (IDE) facilitating developing of AVR applications in Windows environments. It includes a set of tools allowing for fast and comfortable writing and debugging of software for Atmel AVR microcontrollers: An integrated project management tool, a source file editor and a chip simulator. AVR Studio can also interact with in-circuit emulators, such as AVR Dragon, and development boards available for the AVR 8-bit Reduced Instruction Set Computer (RISC) family of microcontrollers. Prior to compiling the software project the oscillation frequency of the μC needs to be configured in the corresponding Project Options tab, in this case for the ATMega32 μC of the RFID tool the value 13560000 Hz must be selected.

The simulation tool of AVR Studio is particularly handy, since it allows to perform time measurements of critical code segments without the need to actually program the μC. This feature was used to provide performance characteristics for the implemented cryptographic mechanisms, as detailed in Section 5.2.4.

WinAVR

While AVR Studio provides support for C, Pascal, BASIC and Assembly languages, the software described in Section 5.2 is written completely in C. WinAVR [46], a suite of executable, open-source software development tools, was used in conjunction with AVR Studio to compile and debug the embedded C code. WinAVR includes the avr-gcc compiler for C and C++, the programmer avrdude and the debugger avr-gdb.

Programming of the Microcontroller

The ISP mode of the AVR Dragon device introduced in Section 4.2 was used for programming of the μC. After compilation two binary files are generated: `<filename>.hex` and `<filename>.eep`. The former one contains the program code and is intended for the Flash memory, whereas the latter file contains the default values for the EEPROM. Both files need to be programmed to the memory of the μC with the integrated programming interface of the AVR Studio, without changing any of the default parameters for the AVR Dragon.

5.1.2. Microsoft Visual Studio

The PC-side software code was written in C using the latest version of Microsoft Visual Studio 2008 Professional [31] illustrated in Figure 5.2 and can be executed on any Windows platform. The code also compiles successfully with the freely available Microsoft Visual C++ 2008 Express. In order to run the compiled code on a machine that does

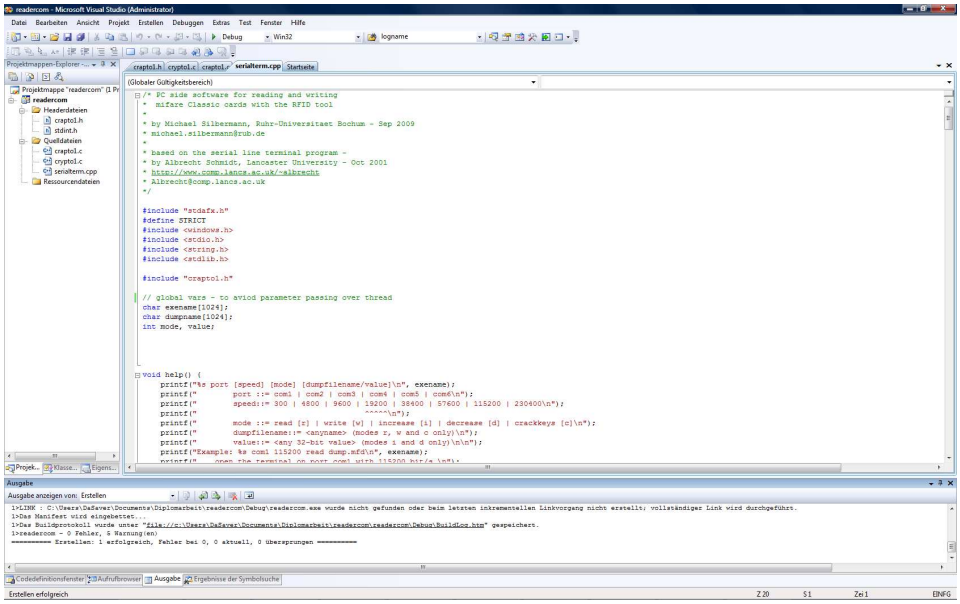


Figure 5.2.: Main screen of Microsoft Visual Studio 2008 Professional.

not have MS Visual Studio or MS Visual C++ Express, the additional Microsoft Visual C++ 2008 Redistributable package[1] must be installed.

5.2. Embedded Software

This section details the embedded part of the C software framework for the Atmel ATMega32 μC.

5.2.1. Project Structure

The embedded part of the software is organized in multiple C source and header files. The source files are summarized in Table 5.1 along with a brief description of the contained functionality. Each source file, except the `main.c`, has a corresponding header file where all function prototypes are defined.

[1] http://www.microsoft.com/downloads/details.aspx?FamilyID=9b2da534-3e03-4391-8a4d-074b9f2bc1bf

Table 5.1.: List of source files of the project with a brief description of their content.

File	Functional Description
`main.c`	Main file of the project containing the `main()` function that is called upon reset of the μC.
`em4094lib.c`	Library of low-level functions for initialization of the EM 4094 RF-transceiver and sending of Manchester and Miller encoded data.
`ftdi.c`	Library of low-level functions for the FTDI FT245R USB chip.
`miller.c`	Contains routines for en- and decoding of Miller data.
`manchester.c`	Contains routines for en- and decoding of Manchester data.
`mifare.c`	Provides API for accessing a Mifare Classic card, including authentication, reading and writing routines.
`crypto.c`	Implementation of the CRYPTO1 encryption algorithm.
`crapto.c`	Contains two routines for rolling back the LFSR of the CRYPTO1.
`timer.c`	Library of low-level functions for setting-up and accessing the Timer/Counters of the μC.
`util.c`	Auxiliary routines for parity, CRC calculation, etc.

5.2.2. Low-level and Auxiliary Functions

Several libraries with low-level functions have been developed to provide API for direct access to the underlying hardware system.

EM 4094 Library

The source file `em4094lib.c` contains three low-level functions responsible for initialization of the chip hardware and sending of Miller and Manchester encoded data:

```
1 void EM_Init(void);
2 void miller_send(uint8_t *data, uint16_t length);
3 void man_send(uint8_t *data, uint16_t length);
```

The initialization routine `EM_Init()` prepares the EM 4094 for signal modulation and demodulation according to ISO 14443 A by setting the appropriate chip parameters and also initializes three timer/counters of the μC required for transmitting of coded data later on.

The function `miller_send()` used to send commands from reader to card receives two arguments: the array with Miller-coded data pointed to by `*data` and its length `length`. The Interrupt Service Routine (ISR) of the 8-bit timer/counter 0 of the μC is used to achieve precise timing during data transmission. The ISR is called after each half-bit period, i.e., 64 clock cycles of the μC, and can trigger the `EM_DIN_PULSE` pin responsible

for creation of short ($\approx 2.5\ \mu$s) pauses in the field. The pause is created if the special register flag `ISR_PAUSE` is set, which is done in the `miller_send()` function depending on the current value in the Miller array.

The function `man_send()` used to send data from fake tag to reader receives two arguments: the array with Manchester-coded data pointed to by `*data` and its length `length`. The operation is very similar to the one described above for `miller_send()` with two differences:

- The ISR is now used in combination with the 8-bit timer/counter 2.
- The ISR controls the `EM_MAN_OUT` pin for transmission of Manchester encoded data.

Note that the least-significant byte corresponding to one bit information in the respective array is transmitted first.

Miller Functions

Two routines for Miller encoding

```
1 uint16_t miller_encode(uint8_t *result, uint8_t *data, uint8_t length);
2 uint16_t miller_encode_fake_parity(uint8_t *result, uint8_t *data,
     uint8_t length, uint32_t fake_parity);
```

and one routine for Miller decoding

```
1 uint8_t miller_decode(uint8_t *result, uint8_t max_length);
```

are provided in the source file `miller.c`. While the decoding routine operates on the physical layer accessing the EM 4094 for decoding of Miller data received from a reader, both encoding routines do *not* actually send the data. Instead they prepare sending of bitstreams via the `miller_send()` provided by the EM 4094 library `em4094lib.c` described above.

The `miller_encode()` routine Miller encodes raw data of a reader command with the least-significant bit of the least-significant byte transmitted first, and stores the result in an internal array of maximum length $2^{16} = 65536$ bytes, which is then passed to the `miller_send()` function. The routine receives three arguments: The array with raw data to be encoded pointed to by `*data`, its length in bytes `length`, and the pointer to an array to store the encoded data `*result`. The length of the latter is the return value of the function. Time-memory tradeoff is utilized for encoding: one bit of Miller-encoded data is stored in one byte of the array. This step is necessary in a subsequent call to `miller_send()`, since it is more efficient in an 8-bit μC architecture to check the whole byte register, as compared to accessing each individual bit in that register, which would take too long for the ISR.

An additional parameter `fake_parity` is passed to the `miller_encode_fake_parity()` function. In contrast to `miller_encode()` where correct parity bits are calculated automatically, the `miller_encode_fake_parity()` function is used for encoding of encrypted data in the Mifare Classic protocol, where parity bits are calculated over the plaintext instead of the actual bits transmitted. Therefore, precomputed parity bits are passed in `fake_parity` with the least-significant bit being the parity bit for the first transmitted data byte.

Two arguments are passed to the `miller_decode()` routine: The pointer to an array to save the decoded data `*result`, and the maximum length of the array `max_length` which can be up to $2^8 = 256$ bytes. The function returns the actual number of bytes decoded. Upon call the function waits infinitely on a transition on input pin `EM_MIL_IN` indicating an incoming command from a reader. The function can be either terminated by pressing the user button or upon successful decoding of the received command. Timer/counter 0 is utilized to distinguish the time periods between two transitions on the `EM_MIL_IN` pin, and thus determine the correct bit transmitted. The Finite-State Machine (FSM)

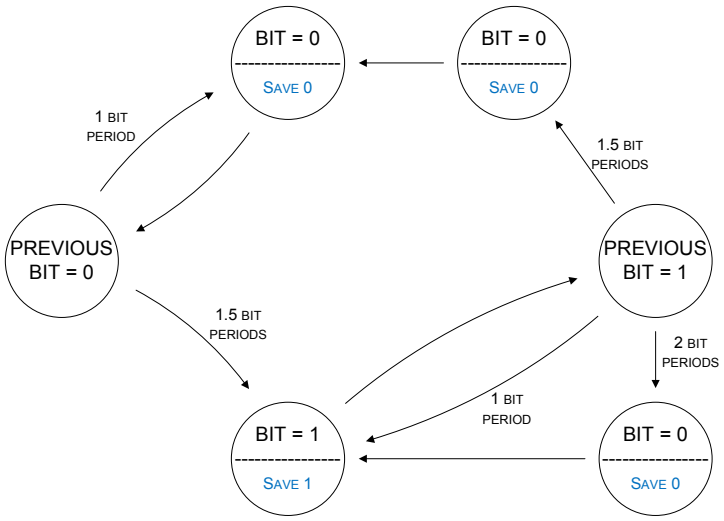


Figure 5.3.: Decoding of Miller-coded data visualized as a finite-state machine, based on Figure 8 in [9].

describing the decoding process is presented in Figure 5.3.

Manchester Functions

The source file `manchester.c` contains two routines for Manchester encoding and one for Manchester decoding:

```
uint16_t man_encode(uint8_t *result, uint8_t *data, uint8_t length);
uint16_t man_encode_fake_parity(uint8_t *result, uint8_t *data, uint8_t
    length, uint32_t fake_parity);
uint8_t man_decode(uint8_t *result, uint8_t max_length);
```

The Manchester functions are very similar to the respective Miller functions described above. One important difference worth mentioning is the fact that the `man_decode()` routine does not wait infinitely for incoming data from a smartcard or the fake tag. A timeout using the 8-bit timer/counter 2 is utilized instead, making the function return after a certain amount of time if no response is received from the card/fake tag.

FT245R Library

Three functions

```
void ftdi_init(void);
void ftdi_send(const uint8_t * data, const uint8_t length);
uint8_t ftdi_receive(uint8_t * data, const uint8_t max_length, const
   unsigned int timeout);
```

are provided for initializing the FT245R chip from FTDI, as well as for sending and receiving data via USB.

The byte array pointed to by `*data` and its length `length` are passed to `ftdi_send()` which transmits the data by pulling the `WR` pin of the FT245R high and low after the current byte is output at the appropriate port of the μC. The receive function `ftdi_receive()` gets three arguments: The byte array to store the received data `*data`, the maximum length of the array `max_length` and the timeout value `timeout` indicating how long should be waited on the incoming data. In the current version the timeout is set to 14000 clock cycles @13.56 MHz which equals to approx. 1 ms. The data is read by pulling the `RD` pin high and low.

CRYPTO1 Functions

The source files `crypto1.c` and `crapto1.c` contain the software implementation of the proprietary stream cipher CRYPTO1 used for authentication and data encryption in Mifare Classic smartcards. The code is based on the open-source project publicly available on the Internet [7]. It has been adapted and optimized for the 8-bit runtime environment of the μC and can be viewed as a "light-weight" version of the referenced implementation. The listing of CRYPTO1 related data structures and functions is given in the following.

```
struct Crypto1State {uint32_t odd, even;};

struct Crypto1State* crypto1_create(uint64_t key);
```

```
4  void crypto1_destroy(struct Crypto1State* s);
5  void crypto1_get_lfsr(struct Crypto1State* s, uint64_t* lfsr);
6  uint8_t crypto1_bit(struct Crypto1State* s, uint8_t in, int fb);
7  uint8_t crypto1_byte(struct Crypto1State* s, uint8_t in, int fb);
8  uint32_t crypto1_word(struct Crypto1State* s, uint32_t in, int fb);
9  uint32_t prng_successor(uint32_t x, uint32_t n);
10 void lfsr_rollback_bit(struct Crypto1State* s, uint32_t in, int fb);
```

The data structure `Crypto1State` holds the current state of the LFSR. The cipher is initialized with the 48-bit key passed as an argument to `crypto1_create()` which returns a pointer to the cipher state on completion. This pointer is used later on to clock the LFSR and obtain the keystream output by the cipher. The functions `crypto1_word()`, `crypto1_byte()` and `crypto1_bit()` return 32, 8 and 1 bit of keystream, respectively, i.e., in each subsequent function call new keystream bits are generated and returned. During the initialization of the cipher (cf. Section 3.4.3) the values UID $\oplus$ n_C and n_R are input to the cipher. These values can be optionally provided through the `in` parameter of the respective function. This parameter can be omitted by providing 0 for an autonomous feedback operation of the cipher without any input. `fb` indicates whether the input `in` is encrypted, e.g., the encrypted reader nonce n_R. If the input is encrypted, it is automatically decrypted and input to the cipher.

The PRNG successor function `prng_successor()` clocks the LFSR of the PRNG initialized with `x` for `n` times. Note that clocking in this case refers to the internal clock of a Mifare Classic card and not to the clock cycles of the μC. This function is amongst others used to compute both responses to the card nonce a_R and a_C.

The function `lfsr_rollback_bit()` rolls back the state of the LFSR by one bit. This function is used in the context of encrypting parity bits: After encrypting the parity bit the cipher is rolled back so that the first bit of the next data byte will be encrypted using the same keystream bit as the parity.

It is worthwhile noting that for all CRYPTO1 functions applies the rule that the values supplied to the functions must be in the big-endian format, i.e., the most-significant byte of the variable is the least-significant byte in the algorithm, while bit order in each byte is in little-endian format, i.e., the least-significant bit in the variable is the least-significant bit in the algorithm.

Mifare Classic Protocol Functions

The library of Mifare Classic functions can be found in the source file `mifare.c`. It provides basic functionality for sector authentication, as well as accessing the memory of a Mifare Classic card, and contains the following functions:

```
1 uint8_t authenticate(uint8_t sector, uint8_t key_type, uint64_t *key);
2 uint8_t read_block(uint8_t block, uint8_t *result);
3 uint8_t write_block(uint8_t block, uint8_t *data);
```

```
4 uint8_t inc_dec_value_block(uint8_t block, uint8_t *value, uint8_t
     is_dec);
5 uint8_t transfer(uint8_t block);
```

On success all of the above functions return a positive value, while in case of an error a zero value is returned.

The function `authenticate()` accepts three arguments: Number of the sector `sector` to authenticate to, type of the key to use `key_type` which is $= 0$ for key A and $\neq 0$ for key B, and a pointer to the 48-bit key `*key`.

The functions `read_block()` and `write_block()` accept as arguments the block number `block` in the range 0..63, and a pointer to a data array in the μC where the data should be stored in or retrieved from. The value block operations are supported by the `inc_dec_value_block()` function, which receives the pointer `*value` to a value to increase or decrease the value block by, and the flag `is_dec` indicating what type of operation should be carried out ($= 1$ for decrease, $= 0$ for increase). After all value block operations are performed and the result is stored in a temporary byte register of the card (cf. Section 3.3), the function `transfer()` must be called in order to save the result permanently in the EEPROM of the Mifare card.

Timers and Utilities

Additionally to the convenience functions developed in [24, 47], allowing for comfortable control of the ATMega32 timers, the LEDs and the user button, three additional routines have been implemented:

```
1 void compute_crc(uint8_t *data, uint8_t length, uint8_t *first, uint8_t
     *second);
2 void remove_parity_bit_reverse(uint8_t *result, uint8_t length, uint8_t
     *raw);
3 uint8_t bit_reverse(uint8_t byte);
```

The function `compute_crc()` enables generation of CRC bytes appended to transmitted data according to ISO/IEC 14443. The code is largely based on the sample function available in [21]. The function accepts the pointer `*data` to the data buffer of length `length`, and two pointers to memory locations `*first` and `*second` where the resulting CRC bytes are to be stored.

In order to obtain the meaningful commands from the data actually transmitted over the air, a conversion must take place where the parity bits are truncated and the bit order is reversed for each byte. For commands in the standard frame format the function `remove_parity_bit_reverse()` is provided, which truncates the parity bits and reverses the bit order for all bytes in the array of length `length` pointed to by `*raw`, and stores the result in the buffer pointed to by `*result`. As the short-frame commands are only one

byte long and do not contain parity bits, the extra function `bit_reverse()` just reverses the byte order in the byte supplied as argument and then returns the result.

5.2.3. Operating System and Modes

The `main()` function acts as the simplest form of an Operating System (OS). After initializing utility functions and both hardware chips EM 4094 and FT245R, the OS outputs a "Hello" message on the serial interface and enters an infinite loop in the menu mode. In order to choose a specific mode a 1-byte command is sent via USB from a host PC. Each mode subroutine is terminated either automatically or through user interaction by returning to the menu where another mode can be selected. In general, every mode can be terminated manually by pressing the user button which leads to the OS returning to the menu mode. In the following a brief description for each individual mode is presented.

Menu Mode

Command (hex : ascii) = `0x6D : 'm'`

This is the default entry point for the OS. The μC waits for user input to enter a specific mode. Once the command indicating the desired mode is received via USB, it is saved in an internal variable and to the EEPROM, which allows for stand-alone operation, as the selected mode is entered directly after a reset of the μC.

Relay Mode

Command (hex : ascii) = `0x72 : 'r'`

The relay mode is used to conduct a MITM attack. Once entered, the green LED lights up and the data exchanged between reader and card is relayed in real time. At the same time the data is saved in a temporary buffer in SRAM. Once the button is pressed the relay subroutine terminates and the saved data is sent to PC via USB. The temporary buffer is needed for performance reasons, as the data must be relayed and *decoded* "on-the-fly".

Stand-Alone Relay Mode

Command (hex : ascii) = `0x73 : 's'`

In contrast to the normal relay mode the data is not sent to PC but saved to EEPROM after pressing the user button. The process of writing data to EEPROM is signaled by the switched-on red LED. Note that access to EEPROM takes approx. 623 μs which is way too slow for storing decoded data to the EEPROM "on-the-fly".

Active Relay Mode

Command (hex : ascii) = `0x61 : 'a'`

This mode is a proof-of-concept for an active MITM attack. This function serves for compatibility with contactless smartcards compliant to part 4 of the ISO/IEC 14443 defining the high-level transmission protocol. Some of these smartcards support bit rates higher than 106 kbit/s, e.g., Mifare DESFire [41] from NXP provides support for bit rates of up to 848 kbit/s. Due to the fact that the RFID tool can conduct reliable relaying of communication only for the bit rate 106 kbit/s, the command for the negotiation of the bit rate exchanged between card and reader must be replaced with a fake one. ISO/IEC 14443 defines this command as ATS which codes the maximum possible data rates in both directions supported by the smartcard.

In order to exchange this command for the fake one with the coded value for 106 kbit/s, the RFID tool "listens" (i.e. decodes) on the incoming commands from a genuine reader. Once the RATS command is received, the direct relay path is deactivated and the timer 0 is initialized and started. After the exact amount of time has passed, the faked ATS command with the coded bit rate of 106 kbit/s is forwarded to the EM 4094 RF-transceiver and sent to the reader instead of the original ATS with a coded higher bit rate. The waveform of the Manchester-encoded data captured by an oscilloscope

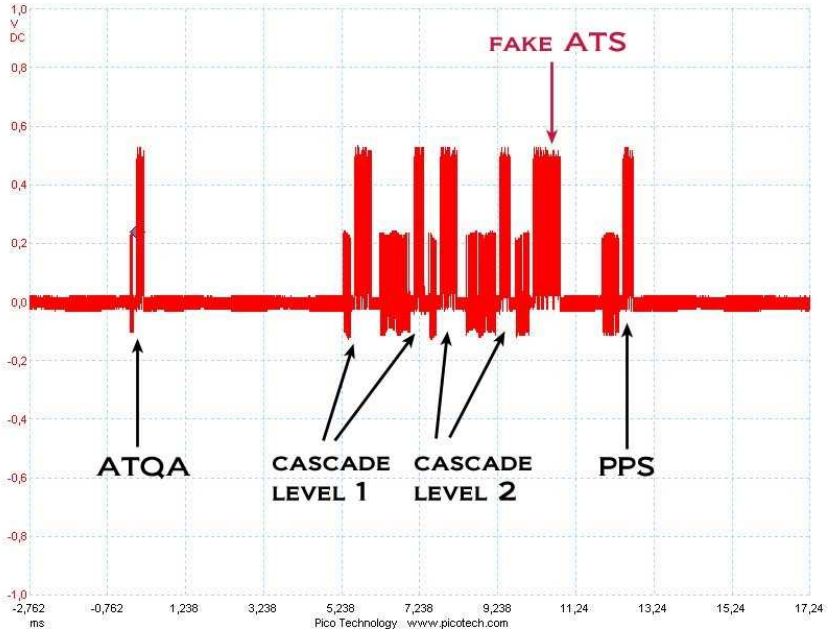


Figure 5.4.: Manchester-encoded card responses in a typical card activation sequence with a faked ATS command.

illustrates this process in Figure 5.4. This way it can be ensured that both parties

continue to communicate at 106 kbit/s and the RFID tool can relay communication in a reliable fashion.

Relay Output Mode

Command (hex : ascii) = `0x6F : 'o'`

This mode is closely coupled to the stand-alone relay mode described above. Once the communication data is saved to the EEPROM it can be extracted and sent to a PC by entering the output mode and pressing the user button.

Generic Reader Mode

Command (hex : ascii) = `0x65 : 'e'`

The generic reader mode, to a large extent based on the code developed in [47], is intended to provide a flexible reader functionality for sending arbitrary commands to card and switching the field on and off. This mode can be completely controlled from the serial terminal HTerm described in Section 5.3.1. The commands entered by the user are Miller-encoded and sent to the card. The response of the card is Manchester-decoded and forwarded to the PC via USB. The command interface for the reader mode is defined by:

- `'s'cmdhex` : Send command specified in `cmdhex` to the card.
- `'w'1` or `'w'0` : Switch the electromagnetic field on (1) or off (0).

Note that `'s'` and `'w'` denote the ASCII characters 's' and 'w', respectively, whereas the values `cmdhex`, `0` and `1` are hex values. Note further that timing is not deterministic, since USB communication in Windows induces an unpredictable delay.

Mifare Classic Key Recovery Mode

Command (hex : ascii) = `0x66 : 'f'`

This mode can only be operated in conjunction with the developed desktop command-line tool described in Section 5.3. This mode is the implementation of the combined differential attack on Mifare Classic secret keys as proposed by Courtois [6] and the nested authentication attack by Garcia et al. [15] (cf. Section 3.5.3). The combined attack is the fastest card-only attack known to date. The purpose of this mode is to recover all secret keys from a Mifare Classic card.

In the first step of the attack the secret key of Sector 0 is cracked using the differential attack by issuing multiple authentication requests to the card. In each authentication

attempt, the concatenation $n_{R,enc}||a_{R,enc}$ is sent to the card. At stage one (cf. Section 3.5.3), the values in $n_{R,enc}||a_{R,enc}$ are randomly generated and each of the eight associated parity bits has a fixed value of 1. At stage two, the 29-bit prefix of $n_{R,enc}$ as well as the first three parity bits are kept fixed, while iterating over all eight cases for the three last bits of $n_{R,enc}$. Four bytes of $a_{R,enc}$ and the five last parity bits are chosen at random.

First, the initialization and anticollision procedure is started to get the UID of the card. After the card is selected and the UID is retrieved, the key recovery of the first sector (Sector 0) of Mifare Classic is initiated. Afterwards, an inner while-loop is entered with each iteration initiating the anticollision procedure and performing the Mifare Classic authentication protocol, either for key A or key B of the respective sector. The key type and the keys are hard-coded in the current version and can only be modified through the firmware update.

In order to reliably reset the PRNG and thereby fix the nonce of the card the electromagnetic field is switched off completely for at least 40 ms, and for some cards for at least 70 ms, by driving the `DIN` pin of the EM 4094: Setting it to a high level through

```
1 setBit(EM_DIN_DDR, EM_DIN);
2 setBit(EM_DIN_PORT, EM_DIN);
```

switches off the field and setting it to a low level with

```
1 clrBit(EM_DIN_DDR, EM_DIN);
2 clrBit(EM_DIN_PORT, EM_DIN);
```

switches the field back on. While the field is switched off the computations that do not require direct interaction with the card are performed, e.g., calculating the new reader nonce.

During a normal protocol run, the response times of the Mifare Classic card vary. Hence, using commercial readers, the same card nonce n_C can only be reproduced with a relatively low probability during an authentication. In order to precisely fix the timing, independently of the behavior of the card, the capability of the μC to wait a fixed multiple of 75 ns between the power-up of the card and the issuing of the authentication command is used. Additionally to the running time of the code, a microsecond delay is utilized for tweaking the timing of issuing the authentication command in order to fit the time grid of the PRNG and hence force it to reproduce the same card nonce n_C in every authentication. The nonces generated by the PRNG of the card can be observed for debugging purposes on a PC and the delay adjusted accordingly to fulfill this requirement.

Once the trace data containing eight encrypted 4-bit `NACK` replies from the card required for the differential attack is collected, it is sent to the desktop software via USB for further processing (cf. Section 5.3.2). As the attack is successful only with a certain probability (approx. 75%) the μC waits for the response from PC indicating whether the

attack has been successful and the sector key has been cracked or whether another set of trace data must be collected from the card. If the attack is unsuccessful, the process of collecting data is restarted for Sector 0. In case of a success the cracked key is saved to RAM and the remaining 15 keys are computed with the nested authentication attack.

The second step of the attack is the nested authentication attack that starts with the initial authentication to the so-called exploit sector, i.e., the sector for which the key has been cracked by the differential attack in the first step. The following authentication to the target sector delivers an encrypted card nonce. Due to the fact that the PRNG of the card keeps running in a deterministic fashion throughout the power-on period, the total number of possible plaintext nonces 2^{16} is greatly reduced by fixing the time period elapsed between the first and the second authentication, leaving only 600-900 possible nonces. This number is further reduced to approx. 75 nonces by checking the three parity bits sent along with the encrypted card nonce offline. Finally, 32 bit of keystream are obtained from the encrypted card nonce (cf. Section 3.5.3) that are used offline to compute on average $2^{48-32} = 2^{16}$ candidates for the internal state of the LFSR. The process of authentication to the exploit sector and to the target sector is repeated 16 times to obtain $4 \cdot 16 = 64$ bit information about the key from the parity bits transmitted with the card nonces. The correct key is then computed offline from the 2^{16} candidates. The FSM that visualizes this process on a high level is presented in Figure 5.5.

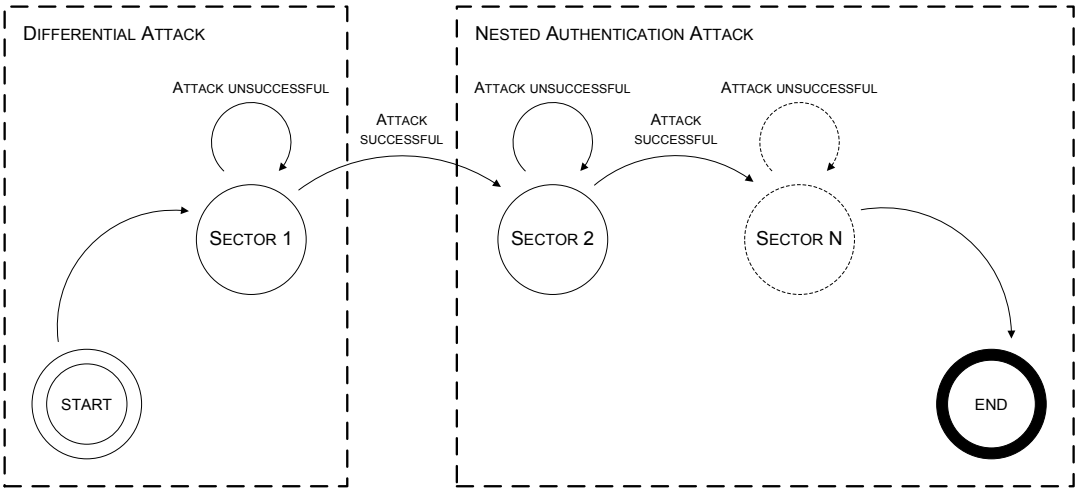


Figure 5.5.: Finite-State Machine visualizing the implementation of the combined differential and nested-authentication attack on Mifare Classic.

Notes:

- The first 35 nonces (can be adjusted for each individual card) are skipped for the differential attack due to transient effects leading to some (undesired) variation of the card nonces generated.

- A constant running time is required for the critical code sequence in the inner while-loop of the differential attack. Otherwise, it is not possible to recreate the same card nonce in multiple authentication attempts.
- The randomness involved in the differential attack is provided by two library functions `srand()` and `rand()`. The first one receives an integer value ("seed") as an argument. The seed is increased in every for-loop iteration. Depending on the seed value a different sequence of random numbers is output by the `rand()` function, which is used to generate random values for the attack.
- Some newer models of Mifare Classic cards are not susceptible to the differential attack, as they have a new PRNG that does not allow to fix the card nonce. These newer cards can still be cracked with our attack, however, due to the issue of fixing the nonce the time required to perform the attack may grow considerably.

Mifare Classic Reader Mode (Dump Card)

Command (hex) = `0x00`

As the name suggests, this mode is used to make a dump of the ID Card or any other Mifare Classic 1K card, provided the secret keys for the card are known, and save it to a dump file. The secret keys A and B of the ID Card are stored in the Flash memory in the current implementation. Cooperation with the desktop software is required for this mode.

The for-loop iterates over all 16 sectors of a Mifare Classic 1K. For each sector a complete authentication with key A stored in the Flash memory of the μC is carried out with subsequent reading accesses to all four blocks of the respective sector. It must be noted that secret keys stored in each sector trailer are not readable and return logical 0's when read. Therefore, in order to generate a complete dump of the card, the secret keys A and B stored in the Flash memory are inserted in each sector trailer in a separate step by the μC and then sent to the PC for saving to dump file.

Mifare Classic Reader Mode (Write Card)

Command (hex) = `0x01`

This mode allows for writing of a previously obtained dump to a blank Mifare Classic card. The dump file can be generated with the "Mifare Classic Reader Mode (Dump Card)". The Mifare Classic card used to write data to does not necessarily need to be blank, it is however required that the access conditions stored in each sector trailer permit overwriting of the existing data/value blocks. Depending on the access conditions, either key A or key B must be used for authentication to each sector on the card. In the

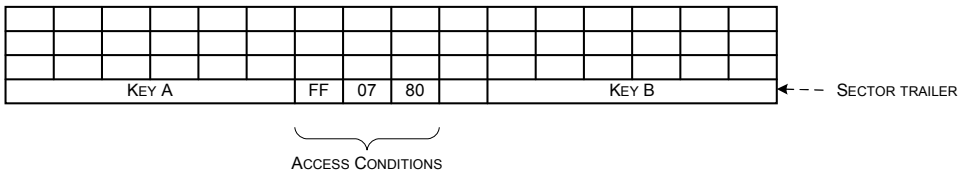


Figure 5.6.: Default configuration of the access condition bits permitting rewriting of a Mifare Classic.

current implementation the keys for accessing the card to write data are stored in the Flash memory of the μC.

A critical issue with writing blank Mifare Classic card is making sure that the configuration of the card will not become frozen afterwards, i.e., rewriting of the card remains possible in the future. If no extra measures are taken, writing of the unchanged dump of an ID Card to a blank Mifare Classic card will lock it for rewriting through the changed access condition bits. Hence, the access condition bits must be modified accordingly prior to writing the card to leave the card open for future rewrites. This is done through setting the access condition bits to default values as shown in Figure 5.6.

Mifare Classic Reader Mode (Increase & Decrease Balance of ID Card)

Command (hex) = `0x02` (increase), `0x03` (decrease)

This mode is used to increase or decrease the credit value blocks stored in Sector 1 of an ID Card. The unsigned 32-bit integer value by which the balance is to be increased or decreased is automatically received from the command-line tool (see Section 5.3.2) via USB.

Mifare Classic Reader Mode (Stand-alone ID Card Manipulation)

Command (hex) = `0x04`

Using this mode it is possible to demonstrate the capability of the RFID tool to manipulate the credit balance stored on the ID Card in a stand-alone battery-operated operation, i.e., without connection to PC. The current implementation runs an infinite loop and wirelessly sets the credit balance of an ID Card in vicinity of the reader antenna to a fixed value of €100. A green LED indicates a successful completion.

Generic Card Mode

Command (hex : ascii) = `0x64 : 'd'`

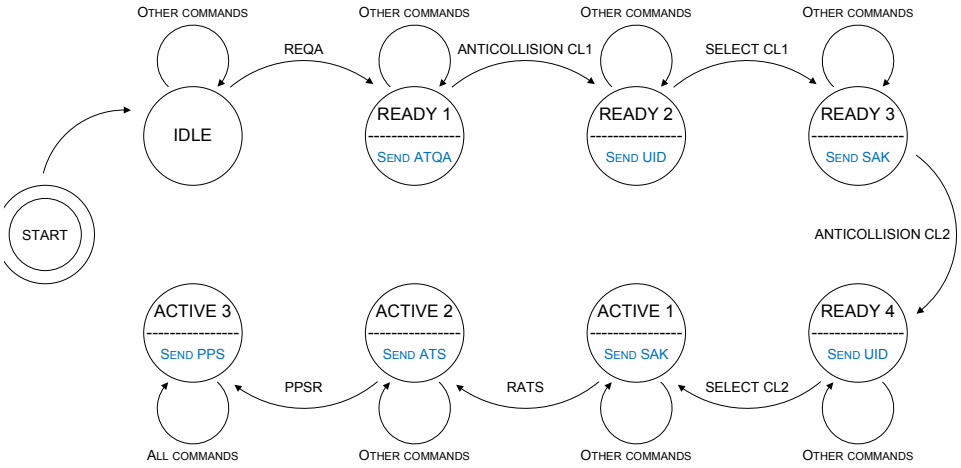


Figure 5.7.: Finite-State Machine defining state transitions in the card emulation mode.

This mode demonstrates the card emulation capabilities of the RFID tool allowing for emulation of any ISO/IEC 14443 compliant contactless smartcard with the fake tag introduced in Section 4.1.2. The current implementation is functional only for smartcards with a double-size UID, e.g., a Mifare DESFire. The operation of the card emulation mode is defined by a FSM illustrated in Figure 5.7. An infinite loop polls for the incoming commands from reader and performs a state transition based on the current state stored in the variable `picc_state`, and the received command.

There are strict timing requirements for the response of the card specified in Part 3 of ISO/IEC 14443. The FDT *and* the bit grid have to be considered to make the reader accept the response. For this purpose the Timer/Counter 0 of the μC is utilized to wait the exact amount of time required for reliable operation. Note that all Manchester-encoded card responses are precomputed before entering the FSM loop for simplicity of the timing control with the utilized timer. However, this can be changed to allow for encoding of arbitrary commands "online" if the timer values for FDT are modified accordingly. The provided timing values have been determined experimentally using one commercially available RFID reader and may vary for different reader implementations.

Mifare Classic Card Mode

Command (hex : ascii) = `0x63 : 'c'`

Similar to the Generic Card Mode, this mode is designed to demonstrate the card emulation capabilities of the RFID tool. This mode provides support for emulation of a Mifare Classic protocol and in its current implementation allows for authentication to a sector and reading of one data block (block 0). Every parameter transmitted by the

fake tag, including the UID of the card and the card nonce n_C, can be chosen and set individually.

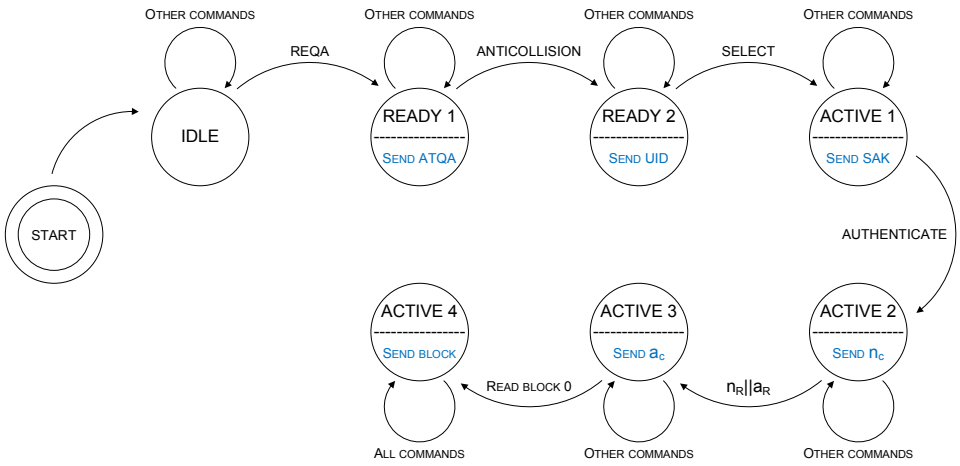


Figure 5.8.: Finite-State Machine defining state transitions in the card emulation mode for a Mifare Classic.

The FSM shown in Figure 5.8 defining the operation of this mode is similar to the one of the Generic Card Mode, with two differences:

- The UID of Mifare Classic cards is 4 bytes long, i.e., single-size, thus there is only one CL.
- Mifare Classic does not comply to the part 4 of ISO/IEC 14443, thus the ACTIVE states are different.

Timing Mode

Command (hex : ascii) = `0x74 : 't'`

The timing mode was developed for testing of the standard compliance of the various Mifare readers. ISO/IEC 14443 puts strict timing requirements on the response of a card: a maximum time is defined and the response must be aligned to the bit grid. The operation of this mode is very straightforward: First, an infinite loop is entered that waits on the REQA command from the reader and starts a timer with a pre-specified delay value for FDT once REQA is received. Once the delay time elapses, the ATQA is sent to the reader that either accepts the response or not. The indicator for a success is the subsequent ANTICOLLISION command issued by the reader. The tested delay value is sent to the PC and subsequently increased for another iteration.

5.2.4. Performance Measurements and Limitations

Table 5.2.: Running times of critical code segments with functions required for encryption and data transmission in the Mifare Classic Card Mode.

Code	Running Time
`state = crypto1_create(key);`	1.7 ms
`ks32 = crypto1_word(state, 0, 0);`	2.5 ms
`ks8 = crypto1_byte(state, 0, 0);`	0.5 ms
`ks1 = crypto1_bit(state, 0, 0);`	94.4 μs
`lfsr_rollback_bit(state, 0, 0);`	33 μs
`man_encode_fake_parity(mil, cmd, 4, par);`	132 μs

There is a limitation to the Mifare Classic Card Mode described in Section 5.2.3: In the current version it only works for reader implementations that allow for adjusting the timeout value, i.e., the maximum time the reader waits on the response of the card. The operation has been successfully tested with the HF Multi ISO RFID Reader [19] from ACG which has a default timeout value of 4.8 ms but can be adjusted up to 76.5 ms.

The reason for this limitation is the long running time of specific library functions pertinent to the encryption and transmission of card responses. Critical function calls together with the approximate running times measured on the ATMega32 μC operated at 13.56 MHz are presented in Table 5.2. The critical code segment while reading a data block from memory is

```
1 for (i = 0; i < 18; i++)
2 {
3   block0[i] ^= crypto1_byte(state, 0, 0); // encrypt data
4   par_bl ^= (uint32_t)crypto1_bit(state, 0, 0) << i; // encrypt parity bit
5   lfsr_rollback_bit(state, 0, 0); // roll back LFSR after parity
6 }
7 man_encode_fake_parity(block0man, block0, 18, par_bl); // encode with Manchester
```

which is responsible for encrypting 18 bytes of block data as well as the corresponding parity bits, and encoding the result with the Manchester code. The execution of this code segment takes approx. 11.7 ms. For instance, for a Mifare reader that has a 4 ms[2] timeout the oscillation frequency of the μC needs to be increased by at least fourfold, i.e., it has to be set to 54.24 MHz.

5.3. Desktop Software

The desktop part of the software framework that communicates and cooperates with the μC is detailed in this section.

[2] As recommended in part 4 of ISO 14443

5.3.1. HTerm

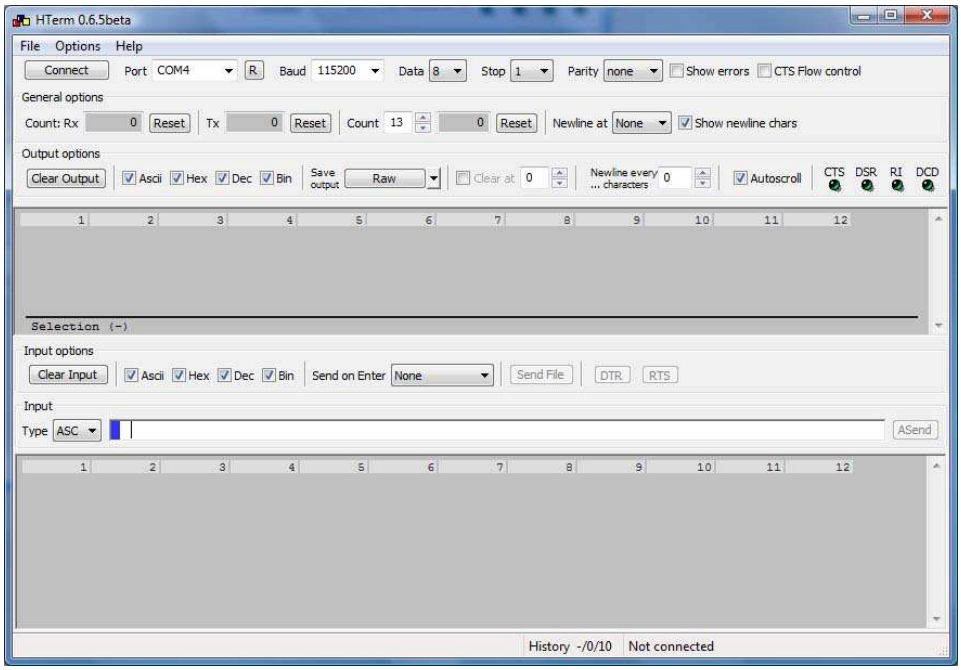


Figure 5.9.: Main screen of HTerm v0.6.5(beta).

HTerm [16] illustrated in Figure 5.9 is a serial terminal program for Windows with a Graphical User Interface (GUI) front-end. It can be used to send and receive data bytes in ASCII, hex, decimal and binary formats through the VCP communicating with the FTDI FT245R USB chip of the RFID tool. Optionally, the data may be retrieved from or written to a text log file. The functionality of HTerm v0.6.5(beta) is used in combination with the developed command-line tool described in Section 5.3.2 for some of the modes detailed in Section 5.2.

5.3.2. Command-line Tool

In the course of this work a command-line tool `readercom` was developed using Microsoft Visual Studio (see Section 5.1.2) to facilitate interaction between the RFID tool and a Mifare Classic smartcard, with a main focus on the ID Card deployed in the here analyzed payment system. The tool based on the open-source serial terminal software [51] makes use of the Windows API to access the serial COM port. Five modes of operation are supported, corresponding to the respective modes of the embedded software running on the μC:

Listing 5.1: Usage of the command-line tool `readercom`.

```
PC side software for reading and writing Mifare Classic cards with the RFID tool.

Version 0.1 by M. Silbermann, Sep 2009, Ruhr-Universitaet Bochum
Based on Commandline Serial Terminal - Version 1.1 by A. Schmidt, Oct 2001
Lancaster University - http://www.comp.lancs.ac.uk/~albrecht/

readercom.exe port [speed] [mode] [dumpfilename/value]
        port ::= com1 | com2 | com3 | com4 | com5 | com6
        speed::= 300 | 4800 | 9600 | 19200 | 38400 | 57600 | 115200 | 230400
                                     ^^^^^
        mode ::= read [r] | write [w] | increase [i] | decrease [d] | crackkeys [c]
        dumpfilename::= <anyname> (modes r, w and c only)
        value::= <any 32-bit value> (modes i and d only)

Example: readercom.exe com1 115200 read dump.mfd
    open the terminal on port com1 with 115200 bit/s,
    read card content to file dump.mfd
Example: readercom.exe com1 115200 write dump.mfd
    open the terminal on port com1 with 115200 bit/s,
    write dump file content from dump.mfd to blank card
Example: readercom.exe com2 19200 increase 1000
    open the terminal on port com2 with 19200 bit/s,
    increase balance on card by 1000 c = 10 eur
Example: readercom.exe com2 19200 decrease 200
    open the terminal on port com2 with 19200 bit/s,
    increase balance on card by 200 c = 2 eur
Example: readercom.exe com4 115200 crackkeys sectorkeys.log
    open the terminal on port com4 with 115200 bit/s,
    crack Mifare secret keys and save them to sectorkeys.log
```

- Dump the content of a Mifare Classic smartcard to a file
- Write the dump file to a (blank) Mifare Classic card
- Increase the credit balance on the ID Card
- Decrease the credit balance on the ID Card
- Crack secret keys of an arbitrary Mifare Classic card

When executed without any parameters a short help is displayed on the screen explaining the usage of the program. The output of the program is presented in Listing 5.1.

Making Card Dumps

In the current implementation, in order to make a complete dump of a Mifare Classic card the keys for memory access to every sector must be pre-stored in the Flash memory of the μC. E.g, the command

```
C:\>readercom.exe com4 115200 r carddump.mfd
```

would result in accessing the VCP COM4 at a baud rate of 115200 for reading out all 16 sectors of a Mifare Classic card and storing it in the hex dump file `carddump.mfd`.

The data is sent *block-wise* from μC to PC, i.e., 16 byte at a time are written to the file. Once the data is saved to the file, it can be opened with any free or commercial hex editor for data manipulations. Dumping of the entire memory of a Mifare Classic card takes approx. 1.2 s in the current implementation.

Writing Card Dumps to a Mifare Classic

Any Mifare Classic smartcard that has not been locked through modified access condition bits (cf. "Mifare Classic Reader Mode (Clone Card)" in Section 5.2.3) can be written, e.g., with the command

```
1 C:\>readercom.exe com4 115200 w carddump.mfd
```

which is semantically analog to the read command, with the data being *extracted* from the dump file `carddump.mfd` and sent to the μC. The data is transmitted to μC *sector-wise*, i.e., 64 byte at a time, until the whole file has been processed. Writing the entire EEPROM of the card takes approx. 1.5 s in the current implementation.

Increasing & Decreasing Value Blocks

The mode allows for increasing and decreasing of value blocks of an arbitrary Mifare Classic card. However, the current implementation of this mode is tailored to suit the scenario of the payment system analyzed in this thesis and the particular memory configuration of the ID Card. The sample commands to increase and decrease the credit balance stored on the ID Card by €5 are:

```
1 C:\>readercom.exe com4 115200 i 500
2 C:\>readercom.exe com4 115200 d 500
```

The specified value is transmitted to the μC which then accesses the memory of the card to modify two value blocks according to the supplied value.

Cracking Mifare Classic Keys

The set of data required for the offline key search on the PC, as detailed in [6], consists of the values summarized in Table 5.3, where x_{enc} denote encrypted values and $i \in [0, 7]$. Once the data is received by the PC and converted to the appropriate format, the offline computation can commence. There are two steps involved in the process of finding the correct key:

1. Find key candidates. With the 32 bit of keystream calculated from N^i_{enc} with a known plaintext ($N^i = \texttt{0x5}\ \forall i$), the key space is reduced by 32 bit, i.e., on average we can expect $2^{48-32} = 2^{16}$ key candidates.

Table 5.3.: Set of data required for the differential attack on the Mifare Classic sector key.

Value	Size (in Bytes)	Description
u	4	UID of the card
n_C	4	Card nonce fixed by the μC
$n^0_{R,enc}$	4	The first encrypted reader nonce with a fixed 29-bit prefix
$a^i_{R,enc}$	$4 \cdot 8 = 32$	Eight encrypted reader responses to the card nonce
P^i_{enc}	$1 \cdot 8 = 8$	Eight encrypted parity bytes with P^i holding parity bits for $n^i_R \| \| a^i_R$
N^i_{enc}	$1 \cdot 8 = 8$	Eight encrypted 4-bit card responses (NACKs) with N^i_{enc} being the response to $n^i_{R,enc} \| \| a^i_{R,enc}$

2. Check the key candidates and find the correct CRYPTO1 key. This is done by checking whether the parity bits in $\tilde{P}^i$ calculated over $n^i_R||a^i_R$ (decrypted values) match the (decrypted) parity bits in P^i.

Two functions

```
uint32_t gen_odd_candidates(uint32_t *list, uint8_t *nack_enc);
uint32_t gen_even_candidates(uint32_t *list, uint8_t *nack_enc);
```

have been developed that accept an array of eight encrypted 4-bit card responses N^i_{enc}, pointed to by `*nack_enc`, as an argument. Both `gen_odd_candidates()` and `gen_even_candidates()` compute each 2^8 candidates on average for the odd and even bits of the internal state of the LFSR, respectively. In total $2^8 \cdot 2^8 = 2^{16}$ candidates for the internal state of the LFSR are obtained, sorted by odd and even bits. Two lists of possible states with respect to odd and even bits are saved in the previously allocated memory segments, pointed to by `*list`, and the length of each list is returned.

Subsequently, both lists with the key candidates and their respective lengths are passed to the

```
int find_key(uint64_t *key, uint32_t *odd, uint32_t odd_size, uint32_t
    *even, uint32_t even_size, uint32_t uid, uint32_t nt, uint32_t
    nr_enc, uint32_t *ar_enc, uint8_t *parity_enc);
```

function, together with the rest of the data collected from the card (cf. Table 5.3), which is required to reject the wrong key candidates. The total number of key candidates is $\sigma =$ `odd_size` $\cdot$ `even_size`. Please note that although only $n^0_{R,enc}$ is passed to the function, the other values $n^i_{R,enc}$ for $i \in [1, 7]$ can be easily obtained by binary incrementing the three last bits of $n^0_{R,enc}$.

The Algorithm 1 is executed inside the function `find_key()` to find the correct key, where K_O and K_E denote the lists of odd and even key candidates computed in the

Algorithm 1 Find the secret key K from a set of candidate keys

Input: K_O, K_E, σ, u, n_C, $n^0_{R,enc}$, $a^i_{R,enc}$, P^i_{enc}
Output: Correct secret key K or `FAIL`

Derive $n^i_{R,enc}$ for $i \in [1, 7]$ from $n^0_{R,enc}$
for $j = 0$ **to** $\sigma - 1$ **do**
    i $\leftarrow$ 0
    wrong_key $\leftarrow$ 0
    while wrong_key $= 0$ **and** $i < 8$ **do**
        $P^i \leftarrow \text{Decrypt}(P^i_{enc}, K^j)$
        $n^i_R \leftarrow \text{Decrypt}\left(n^i_{R,enc}, K^j\right)$
        $a^i_R \leftarrow \text{Decrypt}\left(a^i_{R,enc}, K^j\right)$
        Compute $\widetilde{P}^i$ for $n^i_R || a^i_R$
        if $P^i \neq \widetilde{P}^i$ **then**
            wrong_key $\leftarrow$ 1
        end if
        i $\leftarrow$ i+1
    end while
    if wrong_key $= 0$ **then**
        Save K
        return 1
    end if
end for
return 0

previous step. The success probability of the algorithm is $\approx 75\%$. The return value of the `find_key()` function is forwarded to the μC to indicate whether it should proceed with the nested authentication attack or attempt to collect data for the same sector once again.

Table 5.4.: Set of data required for the nested authentication attack on the Mifare Classic sector key.

Value	Size (in Bytes)	Description
u	4	UID of the card
n_C^1	4	Plaintext card nonce from the 1st authentication
$n_{C,enc}^2$	4	Encrypted card nonce from the 2nd authentication
P_{enc}^2	1	4 encrypted parity bits for $n_{C,enc}^2$
$n_{C,enc}^i$	$4 \cdot 16 = 64$	16 encrypted card nonces from the i^{th} authentication
P_{enc}^i	$1 \cdot 16 = 16$	64 encrypted parity bits for $n_{C,enc}^i$

The set of data required for the nested authentication attack is summarized in Table 5.4, where x_{enc} denote encrypted values and $i \in [3, 18]$. There are three steps needed for cracking the key using the nested authentication attack:

1. Compute a list of possible nonces that correspond to the encrypted value $n_{C,enc}^2$. Depending on the hardware there can be several hundred of possible nonces. In the current implementation the list contains on average 75 nonces.
2. Find key candidates for each possible nonce. With the 32 bit of keystream calculated from XORing $n_{C,enc}^2$ with the candidate decrypted nonce, the key space is reduced by 32 bit, i.e., on average we can expect $2^{48-32} = 2^{16}$ key candidates.
3. Check the key candidates and find the correct CRYPTO1 key. This is done by checking whether the parity bits in $\widetilde{P}^i$ calculated over n_C^i (decrypted values) match the (decrypted) parity bits in P^i.

Note that u from the differential attack is reused for the nested authentication.

The helper function

```
1 uint32_t compute_nonce_candidates(uint32_t *filtered_nonce_cands,
    uint32_t nt, uint32_t nt_new_enc, uint8_t nt_new_par_enc, int
    median, int tolerance);
```

is used to compute nonce candidates corresponding to the encrypted value `nt_new_enc`. For this purpose the successor nonces to `nt` in the range from `median−tolerance` to `median +tolerance` are calculated which are subsequently sieved using the information from the three parity bits in `nt_new_par_enc`. The resulting possible nonces are stored in the array pointed to by `*filtered_nonce_cands` and the size of the array ς is returned.

The computed nonce candidates are passed to

```
1 int find_key_nested(uint64_t *key, uint32_t nonce_size, uint32_t *
    filtered_nonce_cands, uint32_t nt_new_enc, uint32_t uid, uint32_t *
    enc_nonces, uint8_t *enc_parities);
```

where for each possible card nonce in the list `*filtered_nonce_cands` 2^{16} (on average) possible LFSR states are obtained. These state candidates are checked against 64 parity bits stored in the array pointed to by `*enc_parities` for 16 encrypted card nonces obtained from the card, stored in the array pointed to by `*enc_nonces`. If the correct key is found, it is saved to `*key` and the function returns a non-negative status value. Otherwise, the state candidates for the next possible nonce are computed and searched for the correct key. Failure to find a correct key in the end of the routine indicates the too small range of tolerance used in `compute_nonce_candidates()`; in this case the function returns a zero value. The internal operation of the function is defined by the Algorithm 2, where

Algorithm 2 Find the secret key K from a list of possible card nonces

Input: ς, N, u, $n^2_{C,enc}$, $n^i_{C,enc}$, P^i_{enc}
Output: Correct secret key K or `FAIL`

1: **for** $j = 0$ **to** $\varsigma - 1$ **do**
2: Compute 32 bit of keystream $ks_j = n^2_{C,enc} \oplus n_j$ with $n_j \in N$
3: Compute list of key candidates K_C from ks_j
4: **for all** $k_C \in K_C$ **do**
5: wrong_key $\leftarrow 0$
6: **for** $i = 3$ **to** 18 **do**
7: $P^i \leftarrow \text{Decrypt}(P^i_{enc}, k_C)$
8: $n^i_C \leftarrow \text{Decrypt}\left(n^i_{C,enc}, k_C\right)$
9: Compute $\widetilde{P}^i$ for n^i_C
10: **if** $P^i \neq \widetilde{P}^i$ **then**
11: wrong_key $\leftarrow 1$
12: **end if**
13: **end for**
14: **if** wrong_key $= 0$ **then**
15: Save K
16: **return** 1
17: **end if**
18: **end for**
19: **end for**
20: **return** 0

N denotes the list of possible card nonces. The return value of the `find_key_nested()` function is forwarded to the μC to indicate whether it should proceed with the next sector or collect data for the same sector once again.

Listing 5.2: Log file containing the recovered sector keys.

```
SECTOR KEY  0 = 44253d5a24f0
SECTOR KEY  1 = 6c2f620354ff
SECTOR KEY  2 = 5bfb60a2ad04
SECTOR KEY  3 = c8eedfbe5482
SECTOR KEY  4 = 2b18dbf03927
SECTOR KEY  5 = a04b0e84a40f
SECTOR KEY  6 = 7d8c140284e4
SECTOR KEY  7 = 6ad71821f0ed
SECTOR KEY  8 = 91029bc139ad
SECTOR KEY  9 = 94730c690b4b
SECTOR KEY 10 = 4d989f41da27
SECTOR KEY 11 = b669cb162971
SECTOR KEY 12 = e0a5bd8059ee
SECTOR KEY 13 = b8afcde7f83b
SECTOR KEY 14 = 15b2922a8b24
SECTOR KEY 15 = 32213e273b48
```

In case the correct key is found, it is both displayed on the screen and saved to the log file. Listing 5.2 shows the typical content of the log file after successfully recovering all 16 sector keys.

6. Tampering with a Real-World System

As observed during the security analysis of the payment system, the ID Card is based on the Mifare Classic chip [38]. Hence, in the first step a practical key-recovery was performed, as detailed in Section 6.1. The content of the ID Card was reverse-engineered, as described in Section 6.2 and the information gained used to conduct a number of practical system-level tests, as detailed in Section 6.3. Finally, some low-cost countermeasures presented in Section 6.6 are proposed to limit the implications (see Section 6.5) due to the discovered vulnerabilities summarized in Section 6.4.

6.1. Recovering the Secret Keys

The attack was implemented based on the existing attacks [6, 15], as described in Section 3.5, and an open-source implementation of the CRYPTO1 cipher [7]. Using the RFID equipment and the software framework as described in Chapters 4 and 5, the same card nonce is obtained in *every* authentication run, which leads to better practical results than all card-only attacks published so far. With the current hardware setup it takes less than 30 s to perform the required authentication runs and to collect the data for revealing the first sector key of the card. Once the data is collected, it takes less than 3 s to recover the key on a standard PC. For the remaining 15 sector keys the nested authentication attack is utilized which takes on average 25 s per sector. Hence, to extract all 16 sector keys from the card less than $30 + 3 + 15 \cdot 30 = 483$ s ≈ 8 min is needed, which is by a factor ten faster than the 80 min for 16 sectors claimed in [6]. No precomputation or other data storage is required for the attack.

The implementation of the attack was tested on an ID Card and all its secret keys were successfully revealed, thereby noticing that the revealed keys differ from the factory-programmed default keys of Mifare Classic. When analyzing a second card, it turned out that identical secret keys are used for all sectors. The conjecture was that all ID Cards in the system might have the same keys. By analyzing a dozen more cards the hypothesis was verified: *All* ID Cards in the payment system have *identical* sector keys.

6.2. Reverse-Engineering the Card Content

While simply copying an ID Card is possible without knowing how the memory of the ID Card is utilized to carry out payments, more devastating attacks require more details. Hence, in the following we reverse-engineer and describe the content of the cards.

Table 6.1.: Payment management data stored on an ID Card.

Sector	Block	Block Type	Byte	Data
1	3	read/write	4-6	Card number
			10	Check byte, calculated as XOR over all bytes of blocks 0, 1 and 2
4	16,17	value	0-15	Value of the current balance
6	25	read/write	1-4	Time stamp of the last charging
			7-10	Credit amount charged last time
			12	ID of the last used terminal
	26	read/write	1-4	Time stamp of the last payment
			12	Transaction sequence number
			15	Check byte, calculated as XOR over all bytes of blocks 24, 25 and 26

From comparing the content of several ID Cards, before and after carrying out charges and payments, we learned which data the smartcard contains apart from the read-only UID, and where it is stored in the memory. We revealed that only the bytes in Table 6.1[1] are affected by monetary transactions, while the most parts of the memory remain unused. The credit value is stored twice in the value blocks 16 and 17 on the card — probably for the sake of data integrity. Note that, as specified in [38], the content of each value block is stored once inverted and again non-inverted. Note that, in order to break the ID Card, we do not need all 16 secret keys, as the relevant data is stored in only three sectors on the card, and hence it is cracked in less than 2 min.

6.3. Practical System-Level Tests

By prior agreement with the system integrator we carried out a number of tests in order to determine which threats emerge from the vulnerabilities of the system. Blank Mifare Classic cards were used for some of the payments detailed in the following. Due to the fact that these cards do not have the same imprint on the card as the genuine ID Cards, the blank cards were not removed from the wallet and the payment took place wirelessly without revealing the card imprint to the cashier. The cards used for the

[1] Sector/block/byte numbers changed due to the data protection reasons

Table 6.2.: Cards used for the practical system-level tests.

Card	UID
G: Genuine ID Card, Card# 1677489	B410DF9E
A: Anonymous payment card, Card# 12266269	E5F11915
1: Blank Mifare Classic 1k card	15C6203F
2: Blank Mifare Classic 1k card	95AD763F
3: Blank Mifare Classic 1k card	956C713F
4: Blank Mifare Classic 1k card	55EF343F
5: Blank Mifare Classic 1k card	5510713F

practical system-level tests and the corresponding UIDs are presented in Table 6.2.

6.3.1. Payments with Cloned Cards

Goal

Test if payments with cloned cards are possible, and whether the fraud is detected through shadow accounts, if any are present at the back-end of the system.

Test Procedure

Table 6.3.: Monetary transactions associated with the test "Payments with Cloned Cards".

Card	Date	Card #	Amount Available	Amount Withdrawn	Rest
1	02.10.2009	1677489	€4.85	€3.10	€1.75
2	02.10.2009	1677489	€4.85	€4.10	€0.75
3	02.10.2009	1677489	€4.85	€4.60	€0.25
G	05.10.2009	1677489	€4.85	€2.70	€2.15

The first test took place on Friday, 02.10.2009. The goal was to determine whether payments with the clones of the original ID Card are possible. We copied the genuine ID Card charged with €4.85 to three blank cards – which naturally all have different UIDs – without modifying the credit value, and then paid with these (almost) cloned cards. The possibly present shadow account associated with the card number of the card G was intentionally overdrawn through paying a larger sum of money than the original amount

of €4.85. On the next Monday, 05.10.2009, the payment with the original ID Card G was carried out to check whether it has been blacklisted. All monetary transactions with the cards are summarized in Table 6.3.

Results

The cloned cards behaved exactly as a genuine one, and the total amount of €3.10 + €4.10 + €4.60 = €11.80 was withdrawn from the three clones, which is by €6.95 more than the originally available credit value of €4.85. The test with the genuine ID Card on 05.10.2009 shows that the card was not blacklisted, despite the overdrawn shadow account. The existence of several cards with identical card numbers but different UIDs and credit amounts is *not* detected, implying that no automated checks of the UID of the card or the shadow account are performed.

Status After the Test

Shadow account for the card number 1677489 = −€9.65 (Friday evening).

6.3.2. Tampering with the Credit Balance

Goal

Test if payments using a cloned card with a manipulated credit balance are accepted. Since the previous tests were not detected, a further goal is to overdraw the shadow account with a substantial amount of money.

Test Procedure

Table 6.4.: Monetary transactions associated with the test "Tampering with the Credit Balance".

Card	Date	Card #	Amount Available	Amount Withdrawn	Rest
5	05.10.2009	1677489	€100	€18.80	€81.20

On 05.10.2009 the genuine ID Card was copied to a blank Mifare Classic card, and the credit balance was tampered with to set it to the eye-catching value of €100. Subsequently the generated €100-card was used to pay for the meals of four persons at the enterprise cafeteria. All monetary transactions with the cards are summarized in Table 6.4.

Results

Paying with the cloned card with a faked credit balance was possible without any problem.

Status After the Test

Shadow account for the card number 1677489 = −(€ 9.65 + € 18.80) = −€ 28.45 (Monday evening).

6.3.3. Generation of New Cards

Goal

Test if cards with modified card numbers and fake credit balances are detected.

Test Procedure

Table 6.5.: Monetary transactions associated with the test "Generation of New Cards".

Card	Date	Card #	Amount Available	Amount Withdrawn	Rest
1	06.10.2009	1677491	€ 13.85	€ 8.20	€ 5.65
2	06.10.2009	12120412	€ 10.25	€ 4.70	€ 5.55
3	06.10.2009	1677490	€ 9.85	€ 3.50	€ 6.35
4	06.10.2009	10395848	€ 14.85	€ 7.60	€ 7.25
1	07.10.2009	1677491	€ 5.65	€ 4.00	€ 1.65
2	07.10.2009	12120412	€ 5.55	€ 4.50	€ 1.05
3	07.10.2009	1677490	€ 6.35	€ 3.50	€ 2.85
4	07.10.2009	10395848	€ 7.25	€ 3.20	€ 4.05

On Tuesday, 06.10.2009, four new cards with fake credit balances according to Table 6.5 were generated. Two of the card numbers (2,4) are random numbers, while the other two are the numbers following the existing one of the original ID Card. Our first naive attempt failed, since we did not take the check byte at the end of block 3 (see Table 6.1) into account. Once we found out the meaning of this byte and modified it accordingly, we were able to produce cards with a fake card number that were used for payments at the cafeteria, as summarized in Table 6.5. In order to check whether the cards have been blacklisted or remained valid, another batch of payments was carried out the next day.

Results

Paying with all four cards was – once again – possible without any problem, none of the cards was blacklisted the next day. The money was withdrawn from cards with slightly modified card numbers of the existing card, and even from cards with completely random card numbers.

Status After the Test

No conclusion can be drawn with regard to the shadow accounts, as cards with randomly generated card numbers have been used. However, it is conceivable that other users of the system holding ID Cards with the same card numbers, as used in the test, were charged for our payments. Since random card numbers are accepted, it is likely that no checks are performed whether the card number is known to the system.

6.3.4. Cashback

During our tests with the ID Card we discovered that an option exists for the users of the payment system to give back their ID Cards and receive the current balance paid out at the checkout ("cashback"), e.g., in the case they do not need the card any more. Furthermore, an anonymous pay card can be borrowed for a deposit of €10. This card can be used for cashless payments, just as the regular ID Card. With these facts in mind, we wanted to find out if it is possible to use the anonymous pay card with a modified credit balance in combination with the cashback to extract "non-existing" cash money from the system.

Goal

Test if a cashback can be executed using an anonymous pay card with the modified credit balance.

Test Procedure and Results

Table 6.6.: Monetary transactions associated with the test "Cashback".

Card	Date	Card #	Cashback
M	06.10.2009	12266269	€54.00 incl. €10.00 deposit fee

On 06.10.2009 the credit balance stored on the anonymous pay card M was fraudulently increased from €4.00 to €44.00. The content of the card was saved to a dump file for future tests. Subsequently the cashback was initiated at the checkout. Through a mistake of the cashier an additional amount of €10.00 was accidentally charged onto the card (which corresponds with the deposit fee). As a result of this accident a staff member of the company operating the system had to be consulted, who prompted the cashier to pay out the full credit balance of €44.00 stored on the anonymous pay card plus €10.00 deposit fee, in cash, as depicted in Table 6.6.

Note that it is not possible to trace back a criminal who would abuse the system in this way, as the anonymous pay card contains absolutely no information about the card holder.

Status After the Test

Shadow account for the card number 12266269 = −€40.00. Anonymous pay card retained by the system operator.

6.3.5. Using a Clone After Cashback

Goal

Test if paying with a duplicate of the already returned anonymous pay card is still possible.

Test Procedure

Table 6.7.: Monetary transactions associated with the test "Using a Clone After Cashback".

Card	Date	Card #	Amount Available	Amount Withdrawn	Rest
5	07.10.2009	12266269	€44.00	€38.12	€5.88

On 07.10.2009 a new blank Mifare Classic card was generated with the content of the dump file saved in the "Cashback" test, including the credit balance of €44.00. Several payments were carried out at several cafeterias and a coffee house where the contactless payments with the ID Card are supported. The total sum of the monetary transactions for the test is presented in Table 6.7.

Results

Paying with the clone of the invalidated anonymous pay card was possible without any problems.

Status After the Test

Shadow account for the card number 12266269 = −(€ 40.00 + € 38.12) = −€ 78.12.

6.3.6. Blacklisted ID Card

Goal

On 07.10.2009 in the morning the system operator sent out a notification that the card number of the original ID Card used to generate the € 100-card for the test "Tampering with the Credit Balance", has been blacklisted. Hence, it was interesting to find out whether the card was actually blocked in all locations accepting ID Cards for cashless payments.

Test Procedure

Table 6.8.: Recharges of the genuine ID Card.

Card	Date	Time	Card #	Avail.	Amount Charged	Result
G	07.10.2009	12:13	1677489	€ 2.15	€ 8.00	€ 10.15
G	07.10.2009	13:10	1677489	€ 0.35	€ 10.00	€ 10.35
G	07.10.2009	14:40	1677489	€ 1.55	€ 10.00	€ 11.55

Table 6.9.: Monetary transactions associated with the test "Blacklisted ID Card".

Card	Date #	Card #	Available	Amount Withdrawn	Rest
S	07.10.2009	1677489	€ 10.15	€ 9.80	€ 0.35
S	07.10.2009	1677489	€ 10.35	€ 8.80	€ 1.55
S	07.10.2009	1677489	€ 11.55	€ 10.75	€ 0.80

On 07.10.2009 in the afternoon the credit balance of the genuine ID Card G was modified three times, as specified in Table 6.8, in order to carry out further payment

tests with the card (see Table 6.9). The payments took place at five different locations that accept ID Cards for contactless payments, e.g., cafeterias, coffee houses, library, etc.

Results

It turned out that the ID Card was only blocked at the main cafeteria. At all other locations where contactless payments with ID Cards are supported, the card with the blacklisted number was accepted for payments. In the follow-up talk with the system operator it was mentioned that the cause was a delay in the update of blacklisted card numbers in the system, which resulted in the described behavior.

Status After the Test

Shadow account for the card number 12266269 = −(€ 40.00 + € 38.12) = −€ 78.12.

6.4. Summary of the Vulnerabilities

From the tests described above, we conclude that the security of the analyzed ID Card payment system is extremely low based on today's cryptanalytical knowledge, because it relies solely on weak Mifare Classic cards with identical keys. This means that the security mechanisms have been very poorly realized on the system level. Once the secret keys are extracted from one card, all cards of the system can be read and written to wirelessly from up to 25 cm [27] in less than a second. According to our practical results, the one-time key-extraction is easily performed in minutes.

All data on the cards is stored in plaintext and almost no integrity checks are performed. Neither the UID of cloned cards is verified, nor are cards with the same card number but a different balance detected. Cards with new random card numbers and credit balance can be generated that are accepted for payments. The attack related to the cashback option, enabling an attacker to have the (modified) e-balance stored on the card being paid out in cash, is possible without a problem.

From the tests described above we can conclude with near certainty that no automated monitoring of the shadow accounts is taking place on a regular basis to detect fraud. Although the card number of the genuine ID Card had been blacklisted — probably due to a very eye-catching increase of credit balance to € 100 which suggests manual inspection of the shadow accounts — it was still possible to pay at multiple locations where ID Cards are accepted for payments. Minor inconsistencies with regard to card number↔credit balance stored on the card and on the shadow account were *not* detected

in a timely manner, although, according to the system operator, there are shadow accounts existing for each card, and the checks of the shadow accounts are performed on a daily basis. Seemingly, no additional automated checks are performed, neither in the front-end, nor in the back-end of the system, allowing for various attacks as illustrated in the following section.

6.5. Implications

The implications of the security flaws described in Section 6.3 are dramatic. Once the secret keys are recovered from one card, the devastating attacks described in the following can be mounted for any card. The adversary does not need to be a student or staff member to gain access to a card, as an anonymous ID Card can be obtained at any checkout for a deposit of €10. Blank Mifare Classic cards can be purchased for less than €0.50 on the Internet. All our attacks, including the one-time key extraction, require no special knowledge, but can be performed by any attacker who possesses our public-domain reader and the corresponding software, or a commercial reader that operates with software available on the Internet, e.g., mftool from the open-source libnfc library[2].

6.5.1. Impersonation

An adversary needs about 40 ms to covertly extract the card number, and the credit value from the ID Card of a victim from a distance. Then she has two options: For *digital pickpocketing*, she will lower the money on the card of the victim (requiring another 40 ms) and produce a new card with the same card number and the "stolen" amount of money on it. The adversary can now pay with an ID Card that is known to the system, using the money of the victim. Hence, the fraud will not be detected in the back-end. This option would harm only the victim, while no damage is caused for the issuing institution.

As a second option, the attacker *impersonates* the victim by generating a new card with the extracted card number and the extracted credit value, to obtain a duplicate. This time, the card of the victim remains unchanged, and all losses are on the side of the issuing institution. However, if automated checks of the shadow accounts for each card would take place in the future, the fraud could be detected and the card number could be blacklisted, i.e., blocked by the system. Note that the original card then also would have to be blocked, implying additional cost with regard to customer service, as a new card number must be generated and written to the card of the victim.

[2]Available for free download at http://www.libnfc.org/download

In addition, the amount of money on the card[3] can be increased, which will (according to our tests in Section 6.3) still not be noticed by the current realization of the payment system. The remaining content of the card can be generated for both options according to Table 6.1.

6.5.2. Trafficking ID Cards

One of the threats with a considerable incentive for the criminal world is card forgery. An adversary can produce counterfeit ID Cards with a new, random card number and a fake credit value of, e.g., €100. The cards can be anonymous cards or have an imprint that is indistinguishable from the original. By selling them to other people, a criminal could make profit and at the same time cause high losses for the institution. It is also conceivable that a criminal offers a service to charge the cards of other users. In court, these actions would be difficult to prove, because no physical traces are left when modifying existing cards.

6.5.3. Denial of Service

For a Denial of Service (DoS) attack, we assume an attacker who wirelessly resets credit balances of ID Cards in the field to a zero value, while leaving the rest of the data unchanged. Modifying the two value blocks storing the credit value takes again just about 40 ms, hence the attack can be carried out easily in practice, e.g., by hiding a manipulating device near a waiting line of the point of sales or simply by bumping into the card holder. Performing this attack would harm and confuse all affected legitimate card owners, and result in considerable costs for the customer service and loss of credibility for the contactless payment system.

6.5.4. Distributed All-You-Can-Eat

In contrast to a DoS attack, an attacker can wirelessly set credit balances of ID Cards in the field to a high value, e.g., €100. As the process again takes just 40 ms, she can do this manually, by carrying the battery-operated RFID tool with her, or by setting up a disguised manipulating reader device in the entrance to the cafeteria that "charges" the ID Cards of anyone who gets close enough to it.

The financial losses for the institution would be dramatic, because it is unlikely that a legitimate user of the payment system complains about having too much money on his card, and — as we discovered during our practical tests — even though the card number could be detected and blacklisted on the next day, payments can still be carried out for

[3]The upper limit for charging an ID Card card is €150.

quite some time at various locations that support ID Cards. In the long term, the attack would render the payment system inoperative. The issuing institution could decide to sue all victims for manipulating the content of their cards, but in practice it will not be possible to prove whether a card has been modified with or without permission or notification of the owner.

6.5.5. Cashback

The cashback option enables any ID Card holder to receive a cash payout of the e-money stored on her ID Card at any checkout. An adversary could use this option in combination with the anonymous pay card to mount the following attack on the system:

1. Get the anonymous pay card for a deposit of €10.
2. Tamper with the credit balance stored on the card to increase it to an arbitrary value.
3. Go to the checkout and receive a payout in cash corresponding to the fake balance value.

This way the attacker would extract money from the system that never "existed" on her pay card, and thus cause direct financial losses to the issuing institution. Multiple accomplices could combine their efforts and work alternately to conceal the scam.

6.5.6. Emulation of Arbitrary Cards

Employing the fake tag described in Section 4.1.2, an adversary can emulate any ID Card including its UID, e.g., to behave like an exact clone of a card. Furthermore, the fake tag can be used (hidden in the wallet) to pretend the presence of a new card with a random UID, a random card number and a random credit value for every payment. Hence, the detection of fraud and blacklisting would be impossible, even in the presence of shadow accounts. For criminals, this device would have a high market value, since it allows for unlimited payments.

6.5.7. Pickpocketing with a Relay Attack

A relay attack, as detailed in Section 1.2.4, can be performed by two attackers possessing our RFID tool. The approach is depicted in Figure 6.1. One is standing at the cashier to initiate and forward the communication with the genuine reader by means of the fake tag. The accomplice possessing our reader stands close to a victim and thereby enables using the ID Card of the victim remotely. The fake tag and the reader are connected via a communication link, such that the data is forwarded in real-time in both directions.

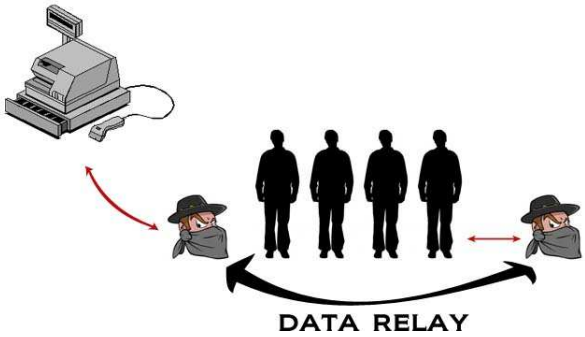


Figure 6.1.: Scenario of a relay attack at the checkout with two adversary accomplices.

The legitimate reader at the payment terminal will carry out a payment, while the money will be withdrawn from the ID Card of a victim, that can be far away. This attack, though very powerful and even working for cryptographically secure systems, is relatively difficult to conduct in practice since the accomplice needs to be close to the ID Card of the victim in the exact moment when the adversary wants to pay.

6.6. Countermeasures

In the following, we propose simple countermeasures that require little effort by the system integrators, and could prevent most fraudulent uses of the ID Card and other Mifare Classic based payment systems. The proposed improvements only require modifications of the software in the back-end and the content of the cards, while the hardware and the type of cards, i.e., Mifare Classic, can remain unchanged. Our proposals cannot provide 100% security, amongst others due to the weak CRYPTO1 cipher used in the cards, but will increase the security level such that the attacks become impractical or unprofitable. In addition to our proposals below, regular consistency checks by means of shadow-accounts for each card are mandatory, where each card should be identified by its card number *and* its UID.

Due to the high cost, a recommended upgrade of the payment system to more secure contactless cards, e.g, based on AES or 3DES, is often not feasible in practice. However, since cryptographic contactless smartcards are vulnerable to side-channel attacks [26], it is conceivable that secret keys can be extracted (with relatively large effort) even from highly secure cards. Once the keys are known, all the above detailed attacks could be conducted again for the here analyzed system — all costs and efforts for the card upgrade would be without effect. Accordingly, an appropriate key derivation, such that each card has an individual secret key, and other measures, are indispensable.

6.6.1. Organizational Measures and Back-End Improvements

Simple countermeasures can be implemented quickly on the system level without any modifications of the cards content or reader hardware. First and foremost, the back-end shadow accounts, as they are currently present in the system, do not provide enough protection against cloning attacks or malicious manipulations of the card content. In order to detect fraud and blacklist the affected card number, automated checks should be performed on a regular basis, preferably several times a day so that the adversary will not have enough time to generate substantial financial losses to the issuing institution. Secondly, checkouts at all locations where ID Card is accepted for payments need to be updated with blacklisted card numbers synchronously. Finally, an appropriate staff training could have prevented the cashback attack through a thorough manual[4] inspection of the shadow account associated with the manipulated anonymous pay card. In addition to the card number, the UID could be checked at the back-end, thus preventing fraud with the cloned cards that are not "in the system".

6.6.2. Key Diversification

One major flaw of the analyzed payment system is that identical keys are used for all ID Cards. Note that, despite the weak CRYPTO1 cipher, the most attacks that require direct contact to the victim, e.g., to extract the card number, would not be practical if each card in the field would have different secret keys, which is not the case in the current implementation of the system. In the process of initialization of the smartcards, secret keys for each sector can be personalized. We recommend to use different secret keys for each card, e.g., by generating them from the (read-only and unique) UID of the card and the sector number to be accessed. Any secure cryptographic function can be used for deriving the keys, e.g., an AES with a manufacturer key, since the computations take place in the back-end, not in the integrated circuit of the smartcard. A card in the field could be upgraded without a special user interaction by reading out its content with the old, weak keys, resetting the card to its default state, and writing the content back with the new, secure keys.

Using different keys for each card makes most of the attacks described in Section 6.5 a lot harder, if not infeasible in practice. For instance, each DoS attack would require to extract at least three sector keys from the card, which takes about 2 min, as compared to 40 ms with the knowledge of the keys. Impersonation (cloning) attacks and trafficking of smartcards on the basis of standard (blank) Mifare Classic cards are rendered impossible, because all cards have different UIDs and hence their correct secret keys cannot be guessed by an attacker. It must be noted that if the manufacturer key and the key derivation process are compromised, all of the above mentioned attacks would become practical again.

[4] In case the blacklisted card numbers have not been updated yet

6.6.3. Data Storage on the Card

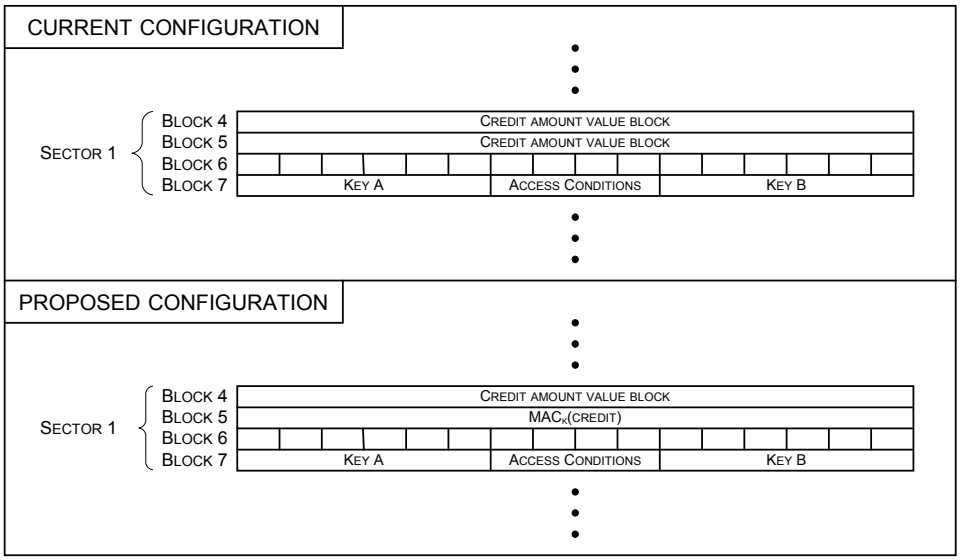


Figure 6.2.: Proposed memory configuration of an ID Card using a MAC to ensure data integrity.

To prevent an adversary from increasing the money value of his own card (or those of others), the authenticity of the credit value should be secured by a MAC, e.g., the AES-based Poly1305-AES [4]. The MAC is computed from the credit amount with a secret key that can again be derived as a function of the UID of the card, and, possibly, some additional data (e.g., card number or some random data bytes). The 16-byte output of the MAC is stored on the card as illustrated in Figure 6.2, e.g., in the block that is currently used to unnecessarily store the duplicate of the credit value. The MAC value is checked before any transaction with the card: If it is not correct, fraud is likely and the transaction should be canceled. The MAC computation and verification takes again place in the back-end, implying no additional computations for the card. This way it can be ensured that an attacker cannot copy the dump of a genuine ID Card to a blank card with another UID, as the MAC verification would fail.

However, if an attacker succeeds in reverse-engineering the MAC computation process she can still make a dump of her own card, tamper with the credit value, compute the MAC accordingly and then write the content back to the same card with the same UID. It must be noted that in this case this countermeasure does not protect from card emulation attacks. For instance, the fake tag described in Section 4.1.2 can be used by an attacker to emulate a Mifare Classic card with an arbitrary UID. Furthermore, it has been reported that contactless smartcards with a programmable UID are available on the market, enabling to produce exact clones, but these cards will be much harder to

get hold of and more expensive than the standard cards.

A better alternative to creating a MAC would be encrypting all the sensible data stored on the card (e.g., with AES, using the same key as for the MAC computation). This way it would be much harder for an attacker to figure out in what format the data is stored and what type of encryption is used.

6.6.4. Timing Control in the Reader

A MITM attack described in Section 1.2.4, although posing a great risk to the security of the payment system, still remains unaffected by the countermeasures presented above, as it operates on the physical layer. A time delay is naturally introduced through the relay of communication between two legitimate parties, depending on the type of communication link between the accomplices. According to ISO 14443 [21], the responses of a card in Steps 2, 4, and 6 of Table 3.2 must be issued in less than 92 μs after the corresponding requests of the reader. Nevertheless, many of the actual reader implementations accept a pause of more than 200 μs, hence the time delay that is naturally induced by relaying the communication is not detected. In order to impede a relay attack, we propose to strictly control the time elapsed between the end of the respective commands of the reader and the start of the response of the card. If the delay exceeds the specified value, the response should be discarded.

However, an active MITM, as described in Section 1.2.4, using our RFID tool would still be possible. In this scenario an adversary can actively intervene in the communication between reader and card. In order to meet strict timing constraints for the affected commands the card responses can be precomputed and sent out before the actual response of the card. Still, the efforts for such an attack are probably higher than the resulting profit.

7. Conclusion

7.1. Summary

Multiple attacks on Mifare Classic chips used for public transportation have been published, giving boost to the criminal activity in that sector. However, the incentive to break other electronic payment applications is at least as attractive from an attacker's perspective, as our work shows. The practical attacks described and conducted on a real-world system in this thesis are feasible for non-specialists. Our analysis of the ID Card payment application reveals obvious vulnerabilities that pose a great threat to the overall security of the system.

First and foremost, the security of any Mifare Classic based e-purse systems is at great risk, since all secret keys of all sectors can be revealed with our attack within minutes. Cloning attacks on Mifare Classic cards are feasible with only minimal time investment and cost, and hence render all Mifare Classic based payment systems insecure. However, the most dramatic threats described do not originate from the weak Mifare Classic cipher but rather from an inappropriate usage of the cryptography.

We summarized the existing attacks on the Mifare Classic and implemented the most efficient practical card-only attack to date using our custom-built equipment at a cost of less than €50. With our hardware set-up it takes less than 2 min to recover the relevant secret keys from a Mifare Classic based ID Card. Once the keys are compromised, the security of the whole system collapses instantaneously, as it turns out that all cards have identical secret keys and no additional cryptographic mechanisms or other checks are implemented on the system level.

An adversary can, in 40 ms and imperceptibly for the victim, read out an ID Card or write to it, increase or decrease its credit balance, impersonate a victim or simply clone his card, etc. Furthermore, a criminal can sell counterfeit cards or electronic devices that emulate a new random card on every usage, and hence permit an unlimited amount of payments. A fatal flaw on the organizational level allows an adversary to use the cashback option for cash payout of the credit amount stored on an anonymous pay card to extract real cash money from the system. The attacks can be carried out by anyone possessing our reader and the corresponding software, with no special knowledge.

The implications are drastic: Both the victims of our various attacks and (probably to a much larger extent) the operator of the payment system face severe financial losses and

damage, due to manipulations of the digital currency and the cost for customer service. By reason of the wireless nature of the attacks, it is difficult (or in some cases impossible) to prove that a crime has been committed. Hence, prosecution of the adversary or the victims is not promising.

Instead, the described flaws require to be instantly addressed by the system integrators. We thus proposed simple countermeasures that can minimize the damage incurred through the described attacks, without that the type of cards or the hardware of the payment system has to be changed. The countermeasures mainly establish a cryptographic binding between a card and its UID, authenticate the credit value, and scrutinize the timing during the communication.

A (cryptographic) binding between the UID of the card and the credit balance would be sufficient to stop the simplest card cloning attacks, although it would not protect against more sophisticated card emulation attacks. It is, however, advisable to all contactless payment system integrators to make a transition from Mifare Classic to the more secure 3DES- or AES-based card products, e.g., Mifare DESFire and Mifare Ultralight C chips, as soon as possible. Unfortunately, it seems like system integrators of today's payment systems do not have sufficient knowledge about cryptography and how it can be used in their payment system — one evidence is the crucial failure of using identical keys for all contactless cards in the system.

7.2. Future Work

Some research directions that can be addressed by future research are summarized in this section.

7.2.1. Mifare Classic Card Mode

In order to eliminate the limitation described in Section 5.2.4 that does not allow to emulate arbitrary Mifare Classic cards for any reader implementation, the RFID tool needs to be upgraded with a new faster μC than the currently installed ATMega32 operating at 13.56 MHz. The new operating frequency should be at least 54.24 MHz which would be sufficient to prepare and transmit the response to the command of the reader in under 4 ms required by Part 4 of ISO 14443. Additionally, a transition to a μC based on the 32-bit architecture may be considered as an option to further increase the overall performance.

Alternatively, the critical CRYPTO1 operations can be performed completely on the PC side thus relieving the μC. In this case, however, new issues with the timing of the serial interface and communication overhead may arise.

7.2.2. Active MITM

During a passive MITM attack small timing delays are inevitably introduced while relaying communication. Some reader implementations are therefore resistant to the passive MITM attack, as they have the strict timing control implemented, which is required by ISO 14443, i.e., both the response time of the card and the alignment of the response to the bit grid are checked. Hence, to circumvent this protection, the responses to the affected commands (REQA, ANTICOLLISION and SELECT) can be exchanged "online" by the μC in order to meet the timing requirements. This modification would be the second enhancement of the active MITM supported by the RFID tool.

7.2.3. Nested Authentication

Garcia et al. [15] claim to be able to predict the card nonce from the second authentication of the nested authentication attack with accuracy. In our implementation of the nested authentication attack the nonce sent by the card in the second authentication is predictable only within a certain range. This uncertainty slows down the search for the correct key considerably, as the time required for the attack grows exponentially with each additional nonce guess. A more precise control of the timing would allow to reduce the range of possible nonces from currently 75 to one, thus speeding up the attack by a magnitude: With only one possible nonce the attack would take approx. 1 s for each sector key.

A. Bibliography

[1] Atmel. ATMega32 Data Sheet. http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf.

[2] Atmel. AVR Dragon online-help. http://support.atmel.no/knowledgebase/avrstudiohelp/mergedProjects/AVRDragon/AVRDragon.htm.

[3] Atmel. AVR Studio 4. http://www.atmel.com/avrstudio.

[4] D. J. Bernstein. The Poly1305-AES Message-Authentication Code. In *FSE*, volume 3557 of *LNCS*, pages 32–49. Springer, 2005.

[5] D. Carluccio, K. Lemke-Rust, C. Paar, and A.-R. Sadeghi. E-Passport: The Global Traceability or How to Feel Like an UPS Package. In *Proc. of WISA'06*, volume 4298 of *LNCS*, pages 391–404. Springer.

[6] N. Courtois. The Dark Side of Security by Obscurity - and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime. In *SECRYPT*, pages 331–338. INSTICC, 2009.

[7] Crapto1. Open Implementation of CRYPTO1. http://code.google.com/p/crapto1, 2008.

[8] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia. A Practical Attack on the MIFARE Classic. In *CARDIS*, volume 5189 of *LNCS*, pages 267–282. Springer, 2008.

[9] H. Dettmer. A Software Extension for Practical Attacks on RFID Devices. Studienarbeit, Ruhr-Universität Bochum, 2007.

[10] EM Microelectronic. EM4094 Fact Sheet. http://www.emmicroelectronics.com/webfiles/product/rfid/ds/EM4094_fs.pdf.

[11] European Commission. EU - Passport Specification. http://ec.europa.eu/justice_home/doc_centre/freetravel/documents/doc/c_2006_2909_en.pdf, June 2006.

[12] K. Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley and Sons, 2nd edition, 2003.

[13] FTDI. FT245 USB chip data sheet. http://www.ftdichip.com/Documents/DataSheets/DS_FT245R_v105.pdf.

[14] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling MIFARE Classic. In *ESORICS*, volume 5283 of *LNCS*, pages 97–114. Springer, 2008.

[15] F. D. Garcia, P. van Rossum, R. Verdult, and R. W. Schreur. Wirelessly Pickpocketing a Mifare Classic Card. In *IEEE Symposium on Security and Privacy*, pages 3–15. IEEE, 2009.

[16] T. Hammer. Hterm. `http://www.der-hammer.info/terminal`.

[17] HID Global. 200X iCLASS Card Data Sheet. `http://www.hidglobal.com/documents/iclass_card_ds_en.pdf`, 2002.

[18] J.-H. Hoepman, E. Hubbers, B. Jacobs, M. Oostdijk, and R. W. Schreur. Crossing Borders: Security and Privacy Issues of the European e-Passport. In H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, and S. ichi Kawamura, editors, *First International Workshop on Security, IWSEC 2006*, volume 4266 of *LNCS*, pages 152–167. Springer, 2006.

[19] A. id. HF Multi ISO RFID Reader User Manual. `http://www.rfid-webshop.com/shop/download/Reader/HF%2013.56%20MHz/ACG/Multi%20ISO/TAGnology_acg_contactless_flyer_neu_ACG_1356_HFMultiManual.pdf`, 2006.

[20] Innovatron. Calypso Functional Card Application v 1.3. `http://www.calypsotechnology.net/index.php?option=com_docman&task=doc_download&gid=3&Itemid=40`, 2005.

[21] ISO/IEC 14443-A. Identification Cards - Contactless Integrated Circuit(s) Cards - Proximity Cards - Part 1-4. www.iso.ch, 2001.

[22] ISO/IEC 7816-4. Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange. `http://www.iso.org`, 2005.

[23] A. Juels, D. Molnar, and D. Wagner. Security and Privacy Issues in E-Passports. In *Proc. of SecureComm'05*, pages 74–88. IEEE Computer Society, 2005.

[24] T. Kasper. Embedded Security Analysis of RFID Devices. Master's thesis, Ruhr Universität Bochum, 2006.

[25] T. Kasper, D. Carluccio, and C. Paar. An Embedded System for Practical Security Analysis of Contactless Smartcards. In *WISTP*, volume 4462 of *LNCS*, pages 150–160. Springer, 2007.

[26] T. Kasper, D. Oswald, and C. Paar. EM Side-Channel Attacks on Commercial Contactless Smartcards using Low-Cost Equipment. In *WISA*, to appear in Springer LNCS, 2009.

[27] I. Kirschenbaum and A. Wool. How to Build a Low-Cost, Extended-Range RFID Skimmer. In *USENIX Security Symposium*. USENIX Association, 2006.

[28] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler. Breaking Ciphers with COPACOBANA - A Cost-Optimized Parallel Code Breaker. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2006.

[29] Y. Liu, T. Kasper, K. Lemke-Rust, and C. Paar. E-Passport: Cracking Basic Access Control Keys. In *Proc. of OTM'07, Part II*, volume 4804 of *LNCS*, pages 1531–1547. Springer, 2007.

[30] Massachusetts Bay Transportation Authorit. The Charlie Card Reusable Ticket System. `http://www.mbta.com/fares_and_passes/charlie`, 2006.

[31] Microsoft. Visual Studio 2008 Professional. `http://www.microsoft.com/visualstudio`.

[32] National Semiconductor. Datasheet for LM311 voltage comparator. `http://www.national.com/pf/LM/LM311.html\#Datasheet`.

[33] K. Nohl, D. Evans, Starbug, and H. Plötz. Reverse-Engineering a Cryptographic RFID Tag. In *USENIX Security Symposium*, pages 185–194. USENIX Association, 2008.

[34] NXP. Mifare Classic. `http://www.nxp.com`, 1995.

[35] NXP. Mifare Ultralight MF0ICU1 Functional specification contactless single-trip ticket IC. `http://www.nxp.com/acrobat/other/identification/M028634_MF0ICU1_Functional_Spec_V3.4.pdf`, 2003.

[36] NXP. *Mifare DESFire Short Form Specification MF3 IC D40*, 2004. `http://www.nxp.com/acrobat_download/other/identification/SFS075530.pdf`.

[37] NXP. Short Form Specification SmartMX P5CC072, Secure PKI Smart Card Controller. `http://www.nxp.com/acrobat/other/identification/096210.pdf`, 2004.

[38] NXP. Mifare Classic 1K MF1 IC S50 Functional Specification. `http://www.nxp.com/acrobat_download/other/identification/M001053_MF1ICS50_rev5_3.pdf`, 2008.

[39] NXP. *Mifare Classic 4K MF1 IC S70 Functional Specification.* NXP, 2008. `http://www.nxp.com/acrobat/other/identification/M043541_MF1ICS70_Fspec_rev4_1.pdf`.

[40] NXP. *Mifare Classic Mini MF1 IC S20 Functional Specification.* NXP, 2008. `http://www.nxp.com/acrobat/other/identification/M132211_MF1ICS20_rev1_1.pdf`.

[41] NXP. Short Data Sheet Mifare DESFire EV1 MF3 IC D41. `http://www.nxp.com/acrobat_download/datasheets/MF3ICD21_41_81_SDS_2.pdf`, 2008.

[42] NXP. *Short Data Sheet Mifare Plus*. NXP, 2008. http://www.nxp.com/acrobat/literature/9397/75016624.pdf.

[43] NXP. *Short Data Sheet Mifare Ultralight C*. NXP, 2008. http://www.nxp.com/acrobat_download/literature/9397/75016620.pdf.

[44] NXP. MIFARE Type Identification Procedure. http://www.nxp.com/acrobat_download/applicationnotes/AN10833_31.pdf, 2009.

[45] Octopus. Octopus Card in Hong Kong. http://www.octopuscards.com/consumer/products/en/index.jsp, 1997.

[46] C. O'Flynn. Downloading, Installing and Configuring WinAVR. http://winavr.sourceforge.net/install_config_WinAVR.pdf.

[47] D. Oswald. On the Feasibility of Differential Electromagnetic Analysis of Contactless Smartcards. Studienarbeit, Ruhr-Universität Bochum, 2008.

[48] OV-chipkaart. All about the OV-chipcard. http://www.ov-chipkaart.nl/allesoverdeov-chipkaart, 2005.

[49] Philips. Data sheet for 4 bit binary ripple counter 74393. http://www.semiconductors.philips.com/pip/74HC393D\#datasheet.

[50] Philips. Data sheet for monostable multivibrator 74HC/HCT123. http://www.semiconductors.philips.com/pip/74HCT123D\#datasheet.

[51] A. Schmidt. Serialterm v1.1. http://www.comp.lancs.ac.uk/~albrecht, 2001.

[52] C. E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949.

[53] Transport for London. What is Oyster? http://www.tfl.gov.uk/tickets/oysteronline/2732.aspx, 2003.