DISCOmfort: A Tool For Practical Security Analysis Of Mifare RFID Systems

Markus Fuchs

Bachelorarbeit

DISCOmfort: A Tool For Practical Security Analysis Of Mifare RFID Systems

vorgelegt von

Markus Fuchs

16. Februar 2011

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Distributed Computer Systems

Betreuer: Dr. Ivan Martinovic
Prüfer: Prof. Dr. Jens B. Schmitt

Abstract

Der Gegenstand dieser Bachelorarbeit ist die Entwicklung von Tools zur Sicherheitsanalyse von RFID Systemen. Im Zuge dessen wurde DISCOmfort entworfen und implementiert, ein Werkzeug zum entschlüsseln und Analysieren mitgeschnittener Datenübertragungen von Mifare Classic Systemen. Die zugrunde liegende Hardware ist ein sog. "Open Source" Produkt, dessen Baupläne und benötigte Software frei verfügbar sind.
Mit Hilfe dieser Entwicklungen wurden verschiedene Systeme auf den Einsatz der allgemein als unsicher geltenden Mifare Classic Technologie überprüft und eine Fallstudie an der TU Kaiserslautern durchgeführt. Dabei konnten sämtliche Sicherheitsmaßnahmen erfolgreich überwunden werden. Es wird ausserdem gezeigt, dass immernoch RFID Infrastruktur im Einsatz ist, die keinerlei Möglichkeiten zum Schutz der übertragenen Daten nutzt oder sogar vermeidet, obwohl vorhanden. Desweiteren werden Gegenmaßnahmen präsentiert, um dem Missbrauch der Systeme entgegen zu wirken.

The subject of this thesis is the development of tools for RFID security analysis. In this context, DISCOmfort, a tool to decrypt and analyze recorded Mifare Classic data transmissions, has been designed and implemented. The underlying hardware is a so called "Open Source" product, whose bauplans and required software are publicly available. By means of these developments, different RFID systems have been checked for their utilization of Mifare Classic, which is generally known to be insecure. Furthermore, a case study at TU Kaiserslautern has been conducted, where all security barriers have been successfully circumvented. In addition to that it is shown that there are still RFID infrastructures which do not offer any possibility to protect transmitted data or even avoid the usage of security features, although available. Furthermore, counter measures are presented to counteract abuse.

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Alle wörtlich oder sinngemäß übernommenen Zitate sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Kaiserslautern, den 16. Februar 2011

Markus Fuchs

Ein Dank gilt Prof. Schmitt, der diese Arbeit erst ermöglicht hat, Dr. Ivan Martinovic für seine Betreuung, sowie Patrick Ernst und Tim Stöber für das Korrekturlesen.
Desweiteren möchte ich mich beim Studierendenwerk Kaiserslautern für die Erlaubnis und die freundliche Kooperation beim Durchführen der Fallstudie bedanken. Ausserdem gilt ein Dank all denjenigen, die mir bei dieser Arbeit mit gutem Rat zur Seite standen: Markus Engel für den exzellenten C-Crashkurs und Michael Arndt für die zusätzliche Hardware.

Contents

1 Introduction

1.1 Motivation

First evolved in World War II, RFID devices have become an ubiquitous phenomenon. Practically used in different applications, such as access control systems for cars and buildings, passports, labeling of goods, credit cards etc. they are attackable. Due to the fact that sensitive data could possibly be stolen from a certain distance without physically interfering with the victim, they demand measures to protect the stored data.

Of course, it will depend on the purpose of RFID systems whether they need security features. There are a number of systems which do not have to offer any functionalities to ensure data integrity and confidentiality like animal tracking devices. People using these products must be aware of this fact, if they applied these technologies in sensitive scenarios like access control. As hardware became cheaper in the last few years, more and more people are able to circumvent the low security barriers of such systems. There are RFID readers publicly available for less than 50 Euro. On the other hand, manufacturers offer products which they praised to be "very secure". Mostly, systems are based on the *security by obscurity* approach. There are no open specifications about their functional principles. Consequently, one has to trust the manufacturer about the security of his product, whereas standardized techniques and algorithms like AES have established themselves to be secure for several years. A common example for a failed security by obscurity approach is the Mifare Classic system by NXP, broken a few years ago as seen in [1]. With more than 2 billion cards and 25 million sold readers, according to [2], NXP's Mifare Classic technology is still one of the most popular contactless chip cards. The Vendor once praised this technology as "very secure" until by time, scientists have overcome the *security by obscurity* approach and revealed the truth about Mifare and its cryptographic functions. More secure approaches, like Mifare DESFire evolved, but require the replacement of whole infrastructures. This can be very expensive for companies already using Mifare Classic. It is a double-edged sword if they ran into potential financial damage because of weak security measures or invested money to harden their architecture. In fact, there are some countermeasures to be applied to the deployed systems which could improve security even if the devices' security itself is broken. We will present such countermeasures later on. Furthermore, one has to consider the fact that even secure devices don't necessarily imply secure systems. Given an RFID device with all its content being encrypted, except the unique identifier, a system administrator can still decide to avoid using cryptography, e.g., by just using this clear-text unique identifier which is built into most tags. One can argue about spirit and purpose of this method, but nevertheless we will see that it is used in practice.

1.2 Goals Of This Thesis

The goal of this thesis is the design and implementation of *DISCOmfort*, a security analysis tool for RFID systems. This tool is built upon affordable hardware which offers total control over all transmissions and data on the channel. For this reason the whole architecture is open and allows changes with respect to current protocols and standards and offers the ability to extend implemented capabilities. The toolset is intended to detect types of devices and exploit the vulnerabilities of Mifare Classic. Information revealed in this process can be used to gain an overview over the whole RFID infrastructure. To the best of our knowledge, this is the first tool implementing the attack described in [1].
As Mifare Classic vulnerabilities are known for more than two years, we want to investigate which countermeasures have been applied. Our findings should highlight that security systems considered to be broken for more than two years are totally obsolete. This must vendors and maintainers give a cause for thought, if they still utilized those devices in security-critical applications.

1.3 Related Work

A general security and privacy survey of RFID systems has been done by Ari Juels from RSA Labs. The paper [3] outlines different risks and attack scenarios. To enhance privacy and protect users' data on tags from being stolen Rieback et al. came up with the *RFID Guardian*, a battery powered device which should ensure that only authorized readers may talk to the protected tags. Illegitimate reading attempts are prevented by selective jamming where the guardian decides by considering an access control list of legitimate readers. In the end, a reader authenticates against the tag. The whole work can be read in [4].
Eavesdropping describes a technique to passively listen on a channel to collect information. By means of this approach, G.P. Hancke presented his results in [5]. Considering HF Mifare tags he was able to listen on the forward channel up to ten meters and the backward channel up to three meters. These results correlate with a similar study by the BSI stated in [6] that they were able to passively sniff communications in more than 1.5 meters. Furthermore, Hancke investigated the possibility of active attacks in [7]. In the end he was able to actively scan PICCs in a distance not larger than 15 centimeters. Besides that, he showed that relay attacks are possible in up to

50 meters.
In order to secure communications, research is focused on the development of energy-saving crypto algorithms. Martin Feldhofer and his group presented an AES implementation in [8] and a public key scheme in [9]. The latter is based on the Rabin crypto system [10] which has the advantage that the tag must not store a private key. In fact it was shown that sufficient cryptography can even be implemented on passive RFID tags.
An approach towards tag cloning has been presented in [11]. The authors extracted physical fingerprints from RFID tags which make it possible to detect spoofing. They used different spectral features in combination and were able to distinguish 50 identical devices with an error of 2.43%.

Whereas researchers mentioned above proposed techniques to improve security, others tried to despise commercial products which rely on their own proprietary security features. Stephen Bono et al. were one of the first who reverse-engineered and exploited a wide spread RFID device in [12]. They analyzed the *Digital Signature Transponder* by Texas Instruments which was widely used as automobile incognition key and payment transponder. They managed to dismantle the proprietary block cipher and found out that it uses a weak key length of 40 bits. Using an FPGA array it takes less than 1 hour to break such a key. Moreover the group built up a device to emulate tags and allowed them to perform passive and active attacks. Finally they have proven their findings at a gas station where they were able to emulate a valid device.

The weaknesses of Mifare Classic systems were first revealed by Karsten Nohl and Henryk Plötz, who presented a partial overview of the Mifare Classic Technology on Chaos Communication Congress 2007 in Berlin[1]. They managed to slice down the chip and reconstruct its algorithms by analyzing the hardware. Their work is also published in [13]. Independently, Gerhard de Koning Gans et al. made a different approach in [14]. Without any knowledge about the protocol they were able to discover weaknesses in the pseudo-random number generator such that they could reconstruct keystream from a given ciphertext and replay authentication. Moreover, they managed to read and modify memory blocks on the card. Furthermore Flavio Garcia et. al from the Radboud University, Nijmegen reverse engineered the whole authentication process and presented vulnerabilities in [15]. This work describes two attacks. The first one is intended to recover the keys without a card by just using a valid reader. The second one is an improved version, where one card-to-reader communication is needed. Moreover, they cover the topic of multi-sector authentications where tag nonces are sent encrypted. This will be extensively discussed in the third chapter. Another paper by the same authors, [1], introduces attacks where only a card is needed. This overcomes the fact that an attacker must have access to a legitimate reader and eavesdrop communication. The authors claim, that by using their attacks it was possible to recover all sector keys within seconds.

[1]slides available on http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html

2 RFID Communication Basics

This chapter is intended to give an overview of currently existing RFID technologies. We will introduce physical properties and encoding schemes which form the basic of wireless communication. Due to the fact that RFID is a very heterogeneous technology, i.e. there are a lot of different procedures for modulation and encoding, we will cover the most popular ones necessary for our purposes. Eventually, the ISO-14443A/B standards are introduced. This is only one out of many specifications, but also the basis for Mifare Classic devices focused on in this document.

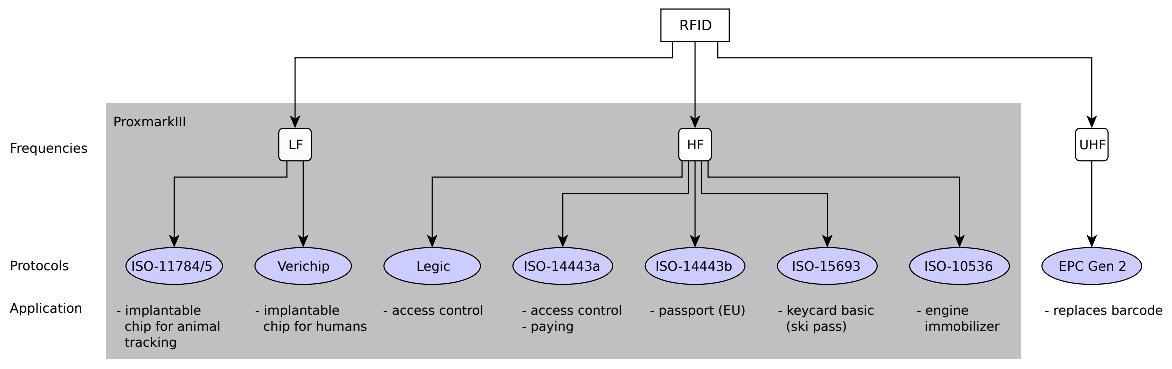


Figure 2.1: RFID technology overview.

2.1 Frequencies

As stated in figure 2.1, we can arrange RFID Technologies into three categories:

- Low Frequency (*LF*), usually 125/133kHz
- High Frequency (*HF*), usually around 13.56MHz
- Ultra High Frequency (*UHF*), usually around 865MHz

Different frequencies are used in different application scenarios. Whereas UHF devices are mostly used for the tracking of goods, LF and HF frequencies are utilized in animal tracking, access control and paying systems. Basically, the communication range depends on the frequency. LF tags usually have a short range of few centimeters while UHF tags can be scanned meters away. Moreover, in [16] it is stated that LF tags better cope with humidity and metallic environment. This benefits their application in the industry. HF transponders are cheap and mostly utilized for access control. Furthermore, there are the so called *Smart Tags*. These are very flat, passive transponders, which have the benefit of very low production costs. UHF technology is mostly used for the tagging and tracking of goods. Because of their low costs, they have also made their way into clothes and packets. What all of those technologies have in common is their application of rather simple modulation and encoding schemes.

2.2 Modulation

The most widely used modulation schemes concerning all standards in RFID communication are *Amplitude Shift Keying* (ASK), *Frequency Shift Keying* (FSK) and *Binary Phase Shift Keying* (BPSK). All of them have in common, that they are easy to perform and not very expensive when it comes to energy. Figure 2.2 illustrates these different techniques.

Amplitude Shift Keying ASK encodes symbols by varying the amplitude of a signal while frequency and phase stay constant. In the simplest case high amplitudes represent 1's while low amplitudes can be interpreted as logic 0's. This is known as *On Off Keying* (OOK). Furthermore multiple bits can be put into one symbol using different amplitude levels. In fact, ASK is very sensitive towards atmospheric noise and distortions [17].

Frequency Shift Keying FSK modulation transmits information by discrete frequency changes. The simplest form is *Binary FSK* (BFSK) which represents logical 0's and 1's by a set of two different frequencies. The signal's phase and amplitude are always kept constant [18].

Binary Phase Shift Keying *Phase Shift Keying* uses a finite set of phases to encode symbols. While frequency and amplitude are kept constant, the phase can vary. Each phase is assigned an equal number of bits. Simple 0's and 1's can be modulated with *Binary PSK* (BPSK) where phases are separated by 180 degrees. In contrast to ASK, this type of modulation is less susceptible against distortion and noise [19].

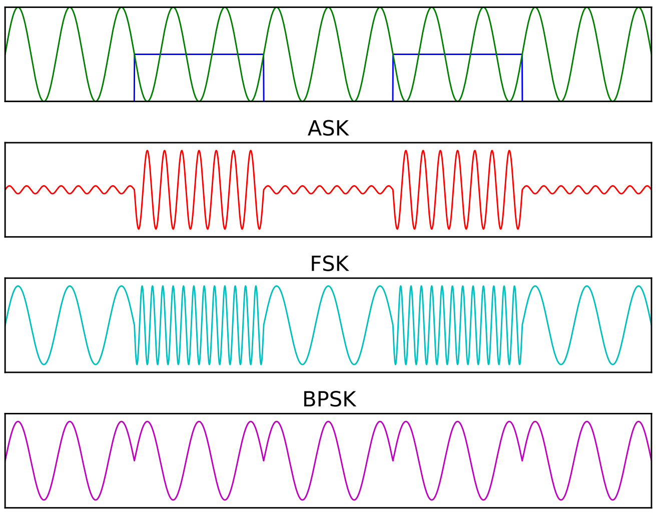


Figure 2.2: Comparison of different modulation schemes.

2.3 Encoding Schemes

NRZ Encoding with NRZ is one of the simplest schemes, where different logical states are given as different states of the communication medium during a bit duration. The NRZ-I scheme encodes bits as changes in the logical state of the medium. Hence, the state stays the same if a sequence of similar bits occured. This requires synchronization [20].

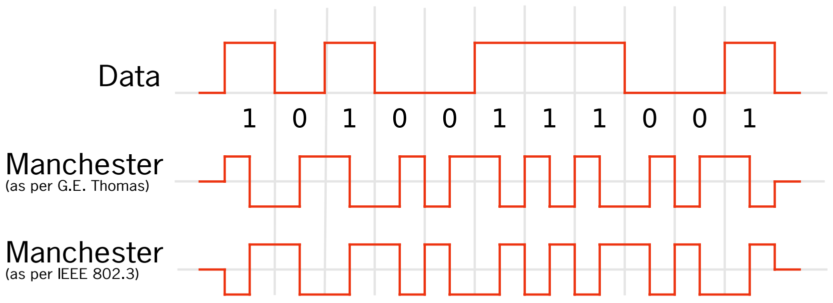


Figure 2.3: Manchester Encoding (taken and modified from [21]).

Manchester Like NRZ-I, the Manchester Code represents a bit by changing the signal level. Nevertheless, two different states of the communication medium are needed. A logical 1 is represented by a transition from higher to lower level whereas a logical 0 is expressed by changing the level from low to high. The same can be applied vice versa. For this reason, Manchester requires twice the bandwidth in comparison to NRZ [21].

Modified Miller This encoding scheme symbolizes a logical level by a position of a pulse within a bit-frame. This requires time synchronization methods to be applied beforehead. In practice this is done with a special sequence of changing levels to indicate the start of a communication. Figure 2.4 illustrates the sequences used in ISO-14443 where t_b is the time period for a bit-frame [22].

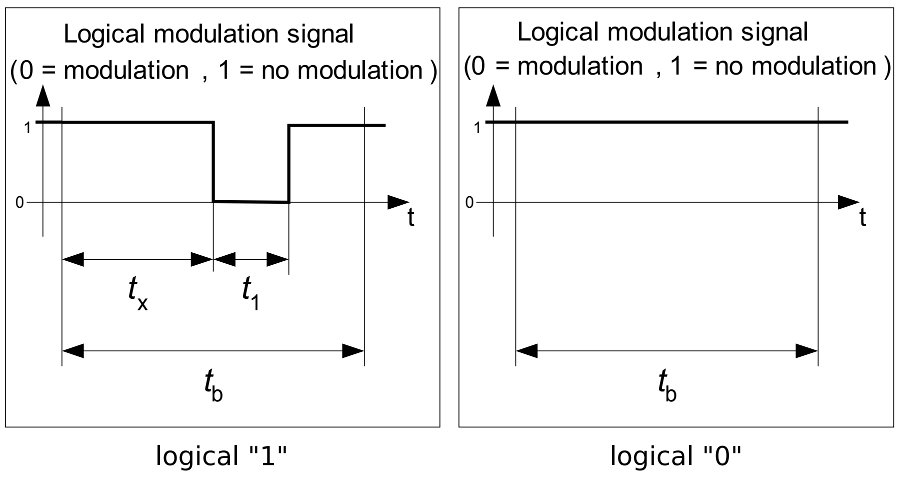

Logical modulation signal
(0 = modulation , 1 = no modulation)
1
0
t
t_x
t_1
t_b
logical "1"
Logical modulation signal
(0 = modulation , 1 = no modulation)
1
0
t
t_b
logical "0"

Figure 2.4: Modified Miller encoding (taken from [22]).

2.4 The ISO-14443 Standard

ISO-14443 describes a standard for *proximity coupling smart cards* in a distance up to 10 centimeters. In general it is divided into four parts:

1. Physical characteristics [23]
2. Radio frequency power and signal interface [22]
3. Initialization and anticollision [24]
4. Transmission protocol [25]

		PCD to PICC	PICC to PCD
Type A	Modulation	ASK	ASK, BPSK at higher bit rates
	Encoding	Modified Miller	Manchester, NRZ-I at higher bit rates
Type B	Modulation	ASK	BPSK
	Encoding	NRZ-I	NRZ-I

Table 2.1: ISO-14443A/B radio properties as in [22].

Readers are called *proximity coupling device* (PCD) whereas tags are denoted as *proximity card* (PICC). The devices communicate at an operating frequency $f_c = 13.56MHz$ with bit rates beginning from $106\frac{kbit}{s}$ up to $848\frac{kbit}{s}$. Furthermore the standard proposes two types of communication interfaces with different modulation and encoding schemes. Their different properties have been summarized in Table 2.1.

Both of them use ASK for transmissions from PCD to PICC. Type A specifies different modulation and encoding schemes depending on the bit rate. ASK and Manchester are only used at the lowest rate of $106\frac{kbit}{s}$. At higher bit rates BPSK and NRZ-I are applied. In addition to that both communication interfaces demand all bits of a byte to be sent in reverse order.

The layer above is specified in [24]. Data is sent as frames of variable size. Usually, an odd parity bit is sent after each byte.
To select a card both PICC and PCD perform an anticollision and initialization protocol. For our purposes we cover a part for communication interface A. Here, the ATQA and SAK are of particular importance as the standard specifies the frame format, but not all bits in there. Their allocation is left to the manufacturer. A Mifare Classic tag by Infineon, for instance, replies with ATQA 04 00 and SAK 88 XX XX. The values X depend on the manufacturing date and other factors only known to the vendor. It is worth mentioning that the *UID* is always transmitted during this protocol. This happens after the reader has received the ATQA and can be observed in Figure 2.5.

In [25], a transmission protocol is specified which features commands to transfer data between PICCs and PCDs. We will not cover those commands as they are not needed in the scope of this thesis.

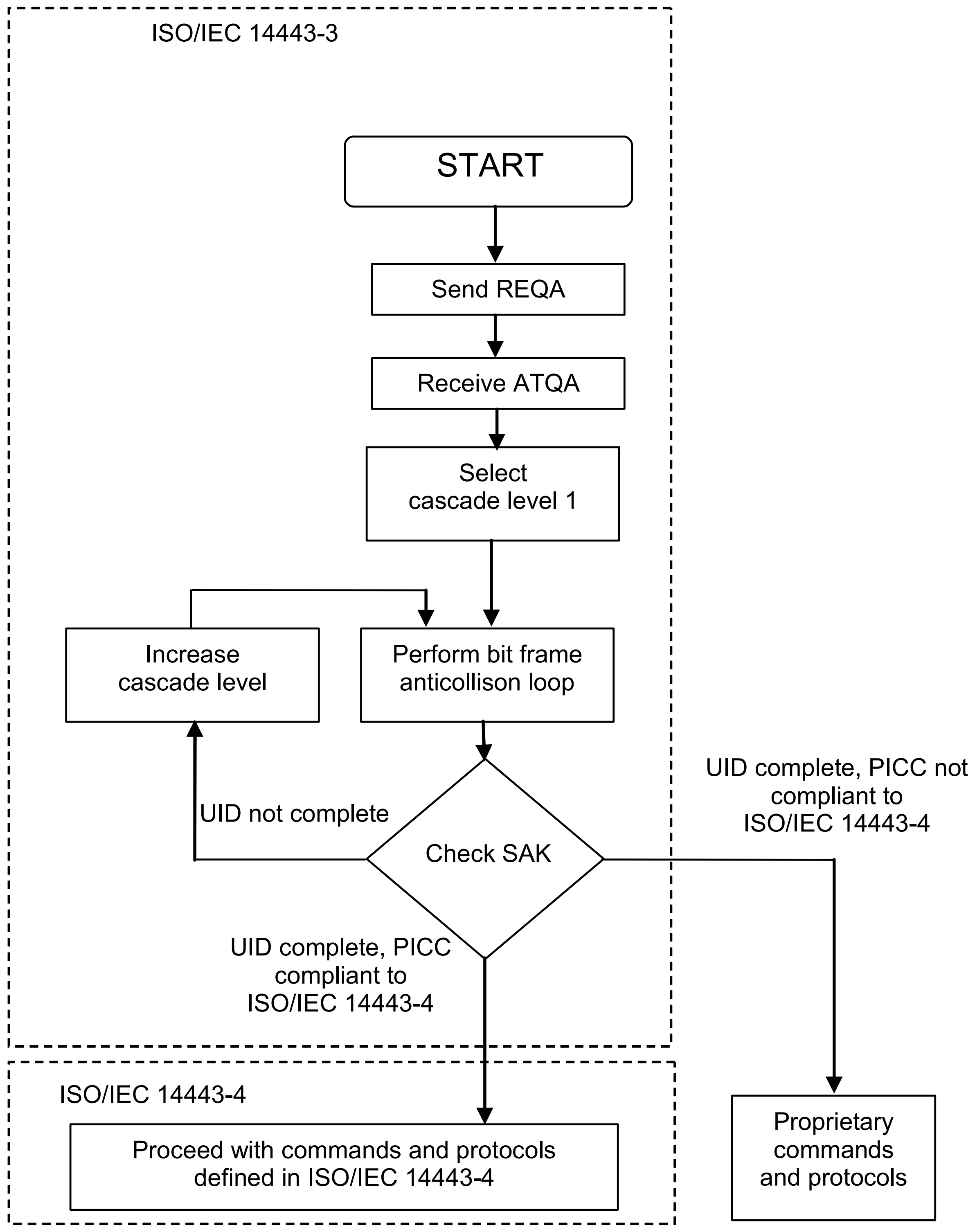

ISO/IEC 14443-3
START
Send REQA
Receive ATQA
Select cascade level 1
Increase cascade level
Perform bit frame anticollison loop
UID not complete
Check SAK
UID complete, PICC not compliant to ISO/IEC 14443-4
UID complete, PICC compliant to ISO/IEC 14443-4
ISO/IEC 14443-4
Proceed with commands and protocols defined in ISO/IEC 14443-4
Proprietary commands and protocols

Figure 2.5: Initialization and anticollision flowchart for a PCD (taken from [24]).

3 Mifare Classic Security Scheme

As mentioned in the intrdoductory chapter of this work about Mifare Classic there is already extensive research. In order to become familiar with this technology this chapter will have a closer look at the functional principles of these devices. After general properties are presented, we will inspect its cryptography and authentication protocol. Finally, some major vulnerabilities are shown which form the fundamentals for a successful attack.

3.1 Communication Basics And Memory Organization

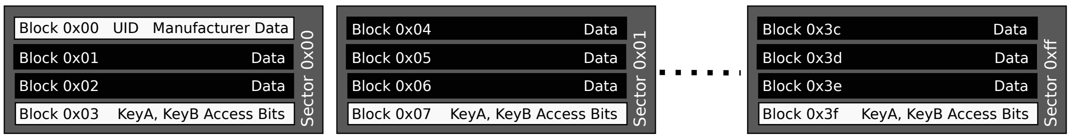


Figure 3.1: Mifare Classic 1k memory layout.

Mifare uses the data link and the MAC layer as specified in ISO-14443A. The application layer is essentially a proprietary protocol, partially available as online document [26]. Table 3.1 is an excerpt of the commands used in communications.

Hex Code	Description
60	Authentication with Key A
61	Authentication with Key B
30	Mifare Read Data Block
a0	Mifare Write Data Block
c0	Mifare Decrement Data Block
c1	Mifare Increment Data Block
c2	Mifare Restore
b0	Mifare Transfer

Table 3.1: Mifare Classic application layer commands.

Considering a Mifare Classic 1k device, where one can store 768 Bytes of user data. The memory is divided into 16 sectors each of them contains three blocks for user data and one sector trailer. The latter provides two sector keys and the access control bits which regulate read/write access for specific sectors. Hence, it is possible to use key A for reading and key B for writing purposes. A data block has a size of 16 bytes. There is one block which cannot be overwritten, namely, the first one which holds the *unique identifier* (UID) of each card. That's exactly where the problem arises from, if we try to clone a card. This issue is further investigated in the following chapters.

Data blocks can adopt a special format which makes them *value blocks*. They allow only 4 byte values stored as standard 2's-complement in little endian. Furthermore, the values are saved three times, twice normal and once inverted. As stated in [26], this enhances data integrity and security. At the end of the block there is a byte representing a backup address also stored normal and inverted.

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Description	Value				$\overline{\text{Value}}$				Value				Adr	$\overline{\text{Adr}}$	Adr	$\overline{\text{Adr}}$

Figure 3.2: Mifare Classic value blocks.

Beside basic read and write operations, it is possible to increment and decrement value blocks. Temporary values are written to a volatile memory and applied only after a successful *transfer* command. It is intended that this type of blocks is used for digital purse applications.

3.2 CRYPTO1

Trying to guarantee confidentiality, a stream cipher called *CRYPTO1* is used. Essentially, it consists of a 48 bit *linear feedback shift register* (LFSR) and some filter functions for the encryption. The LFSR uses the generator polynomial $g(x) = x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^{9} + x^{7} + x^{6} + x^{5} + 1$ computing the feedback bit while the register is shifted left at every clock tick. Thereby, the leftmost bit is discarded while another bit from an input value is XOR-ed with the feedback bit and fed into the LFSR. The input value consists of the UID, the key and a few pseudo random numbers, exchanged during authentication. It is the *Initialization Vector* for this stream cipher. However, feedback bits are revealed by applying the filter function

$$\begin{aligned} g(x_0, x_1, \cdots, x_{47}) = f_c(&f_a(x_9, x_{11}, x_{13}, x_{15}), f_b(x_{17}, x_{19}, x_{21}, x_{23}), \\ &f_b(x_{25}, x_{27}, x_{29}, x_{31}), f_a(x_{33}, x_{35}, x_{37}, x_{39}), \\ &f_b(x_{31}, x_{35}, x_{37}, x_{39})) \end{aligned} \tag{3.1}$$

These are used to encrypt a message. An overview of the cipher can be found in Figure 3.3.

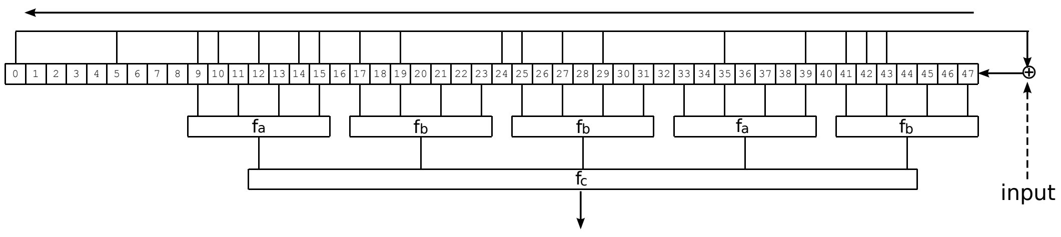


Figure 3.3: 48 bit LFSR of the CRYPTO1 cipher.

To initialize the cipher, a pseudo random number generator is used. This is also implemented as an LFSR with a width of 16 bits and generator polynomial $x^{16} + x^{14} + x^{13} + x^{11} + 1$. The following function describes the feedback bit produced at each clock cycle:

Definition 1. *The feedback function* $L_{16} : \mathbb{F}_2^{16} \to \mathbb{F}_2$ *of the pseudo random number generator is defined by*
$L_{16}(x_0 x_1 \cdots x_{15}) := x_0 \oplus x_2 \oplus x_3 \oplus x_5.$

So we are able to define the a function $succ()$ that returns the next 32 bit pseudo random sequence computed by the LFSR:

Definition 2. *The successor function* $succ : \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ *is defined by*
$succ(x_0 x_1 \cdots x_{31}) := x_1 x_2 \cdots x_{31} L_{16}(x_{16} x_{17} \cdots x_{31}).$

In fact there are only 2^{16} possible values generated by the pseudo random number generator. These are used as *tag nonces* during authentication.

3.3 Authentication And Authorization

Card and reader mutually authenticate, essentially using a challenge-response protocol. This is done in three steps.

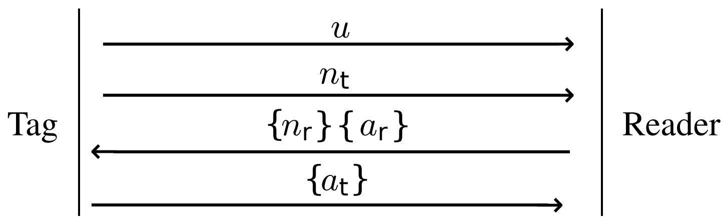


Figure 3.4: Mifare Classic authentication scheme.

1. The tag sends its UID and a random 32 bit tag nonce n_t to the reader. The values k and $n_t \oplus UID$ are fed into the LFSR where k is the sector key. The following communication is performed encrypted.
2. The reader computes its response $a_r := succ^{64}(n_t)$ and sends it back to the card, including its own nonce n_r. Now n_r is also shifted into the LFSR.
3. In the end, the tag answers with $a_t := succ^{96}(n_t)$.

Being able to calculate the responses a_t and a_r, reader and tag can verify that their LFSR is in the same state. This ensures mutual authentication.

After that, data can be read from blocks belonging to the authenticated sector. If the reader wants to apply operations on another sector, further authentication processes have to be made. This is called *nested authentication*. The protocol used is essentially the same as the original, except that the tag nonce is sent encrypted. Since both communication partners know which key to use (they are already sending encrypted), the specific sector key can be used to encrypt the nonce. This procedure makes it harder to break the cipher.

In fact, authorization is fulfilled by the knowledge of a sector key. Because there are different keys for each sector, entities can only access data for which they know this secret.

3.4 Weaknesses And Exploits

In this section we will introduce some weaknesses as stated in [15] and [1]. The LFSR Rollback Attack enables us to recover sector keys by eavesdropping a communication. The Parity Weakness reveals additional information about clear text messages thus making it possible to speed up attacks.

Generally, there are five facts which made it possible to break the cipher

1. **Gerneration of feedback bits** For the generation of feedback bits and while shifting the LFSR, not all registers are used. In fact, it can be observed that just

odd values are fed into the feedback function. Furthermore the leftmost bit is discarded at all.

2. **PRNG has low entropy** As [14] found out, the tag nonce strongly depends on the power of the electric field. It can be observed that an attacker can predict nonces if he can control its strength. Furthermore the 16 bit LFSR is just able to produce 2^{16} possible random numbers.
3. **Misuse of a One-time Pad** Re-usage of keystream bits for the parity bits violates the stream cipher's one-time-pad property. A possible attacker may be able to get information out of this.
4. **Protocol weakness on failed authentication** While checking the reader response a_r it may occur that parity bits are correct and the response itself is not. This results in an error message. On the other hand, if parity bits were wrong, the tag does not respond at all. As a consequence, an attacker may be able to gain information about the responses if, additionally, he was able to control the electric field.
5. **Default keys** Deploying systems with publicly known default keys is a more general problem, but it also effects Mifare Classic.

The first weakness enables us to recover sector keys with known values for the initialization vector. This is described by the *LFSR Rollback Attack*. In addition to that, attacks may be fascilitated using 2 and 3. The protocol weakness is rather useful for active attacks than for passive ones we would like to perform.

LFSR Rollback Due to the fact that not all registers of the LFSR are used to generate key stream bits, it is possible to compute previous states. Given the state at time k as $x_k x_{k+1} \cdots x_{k+47}$ and the input bit i, then the state at time $k+1$ is $x_{k+1} x_{k+2} \cdots x_{k+48}$ where

$$\begin{aligned} x_{k+48} = x_k &\oplus x_{k+5} \oplus x_{k+9} \oplus x_{k+10} \oplus x_{k+12} \oplus x_{k+14} \oplus x_{k+15} \oplus x_{k+17} \oplus x_{k+19} \\ &\oplus x_{k+24} \oplus x_{k+27} \oplus x_{k+29} \oplus x_{k+35} \oplus x_{k+39} \oplus x_{k+41} \oplus x_{k+42} \oplus x_{k+43} \oplus i \end{aligned} \tag{3.2}$$

This equation enables us to get the previous state. However, if we can recover the state of the LFSR, i.e. by eavesdropping a communication, we are able to recover previous states. In fact, the reader nonce is encrypted. To recover it, we have to shift the LFSR to the right by using equation 3.2. The rightmost bit vanishes and for the leftmost we can choose an arbitrary value since it is not used in the filter function g. Then the filter function gives us a feedback bit which was used to encrypt the last bit of the reader nonce. Now that we know the last bit of the reader nonce, the leftmost bit can be set to its correct value such that we know the state before feeding in the last bit of the reader nonce. Repeating this procedure 31 more times reveals the LFSR's state before the reader nonce has been entered. Tag nonce and UID are sent in plain which makes it easy to roll back. In the end, the LFSR state results in the sector key.

Parity Weakness As dictated by the ISO-14443 standard, parity bits are sent after each byte of data. Mifare cards compute parity bits over the plain text. These are encrypted with one bit of the key stream which is reused for the next bit of the messages. To clarify our intention we have to define parity bits and the encryption scheme formally, focusing on n_t.

Definition 3. *Let $n_t = n_{t,0} \cdots n_{t,31}$ be the tag nonce, $\oplus$ be the bit wise XOR operation and b_i with $i \in \mathbf{N}$ an arbitrary bit of the key stream.Then we define*

1. *the parity bits for the tag nonce by $p_j = n_{t,8j} \oplus n_{t,8j+1} \oplus \cdots \oplus n_{t,8j+7} \oplus 1 \; \forall j \in [0,3]$*
2. *the encrypted parity bits by $\{p_j\} = p_j \oplus b_{8+8j} \; \forall j \in [0,3]$*
3. *the encrypted tag nonce by $\{n_{t,j}\} = n_{t,j} \oplus b_j \; \forall j \in [0,31]$*

Now we can use the following theorem taken from [1].

Theorem 1. *For every $j \in \{0,1,2\}$ we have*

$$n_{t,8j} \oplus n_{t,8j+1} \oplus \cdots \oplus n_{t,8j+8} = \{p_j\} \oplus \{n_{t,8j+8}\} \oplus 1 \tag{3.3}$$

Consequently, the parity bits of encrypted data leak three bits of information when it comes to tag nonces. Usual ISO-14443A PICCs show a parity error if such a nonce is received. These are used in our attack described on page 24 to speedup the process of decrypting a communication.

4 Design And Implementation Of DISCOmfort

In this chapter we present our tool for decrypting Mifare Classic communications. Besides this feature there is also a graphical user interface (GUI) which allows to inspect the collected data in order to reverse-engineer the sector content. Levereging the GUI we can identify critical parts of the infrastructure. Before we have a closer look at the implementation, the necessary hardware is presented.

4.1 Hardware Equipment

There is a vast amount of proprietary RFID devices on the market just offering high-level programming interfaces. If you want to analyze traffic and gain detailed information for a specific RFID system, this is not enough. The goal was to find a device which is capable of different protocols and standards. Moreover, it is sufficient to be able to eavesdrop transmissions. For this purpose we have chosen ProxmarkIII, a flexible reader when it comes to the possibility for customizations. In fact, we could have chosen any device which is capable of recording low level ISO-14443A traffic

- in hexadecimal or binary representation,
- including parity bits and
- including timing information for further optimizations.

4.1.1 ProxmarkIII

Jonathan Wethues built up the RFID device for testing and analyzing. The full documentation and plans as well as the firmware are available on google code [1]. Essentially, it consists of two send and receive circuits to process low-frequency (125/133MHz) or high-frequency (13.56MHz) communications. The basic architecture is illustrated in Figure 4.1. For signal reception, the *analog-digital-converter* (ADC) receives signals straight from an external antenna and passes it to the Spartan-II FPGA. The latter

[1]http://code.google.com/p/proxmark3/

demodulates it and hands the symbols to the firmware where decoding takes place. The resulting bits are interpreted according to the underlying protocol. Sending is essentially the same just the other way around, where the ADC is replaced by *digital-analog-converter* (DAC). All the data can be sent to a computer connected via USB. A client application is available to control ProxmarkIII. Due to the fact that the device is also powered by the USB, a computer is necessary nearly for all tasks.
The firmware is running on an ARM7 CPU (AT91SAM) with 64k SRAM and 256k NAND Flash integrated. Like the whole hardware architecture it is released under an open source license. It is written in C and thus can be extended to the needs of our purpose. Basically its features are divided into *hf* and *lf* commands, specifying if either high or low frequencies are used. Table 4.1 illustrates which tags can already be handled. With ProxmarkIII we can read, write, eavesdrop and emulate cards. All the supported tags do not offer any cryptography. However, some manufacturers built up their devices on top of already standardized protocols. The Mifare Classic system, for example, uses ISO-14443A physical and transport layer as well as the anti-collision protocol and adds its own commands when it comes to data transmission. These imply cryptographic functionalities and mutual authentication. In effect, ProxmarkIII can eavesdrop even more different RFID systems as stated here.

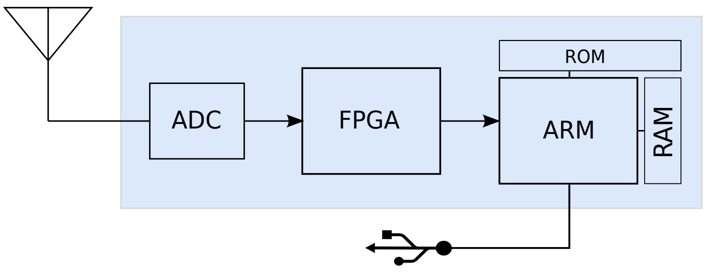


Figure 4.1: Architecture of ProxmarkIII.

The device comes with two loop antennas, one for each frequency domain. The range is restricted to a few centimeters depending on the frequency. We have observed that reading a HF card is possible up to four centimeters and LF devices up to five centimeters without bit errors.

Modifications

In practice, the actual firmware is quite unstable. We experienced a vast number of crashes while reading from cards and collecting data. We have applied some fixes in the firmware and thereby made some extensions to facilitate security analysis. First of all the software lacked a function to safe eavesdropped communication into files. These files serve as input for our software described in the next section.

Tag	Freq.	Read	Write	Emulate	Eavesdrop
EM410x	LF	X	-	X	X
EM4x50	LF	X	-	-	-
FlexPass	LF	X	-	-	-
HID	LF	X	-	X	-
Indala	LF	X	-	-	-
VeriChip	LF	X	-	-	-
LEGIC	HF	X	X	X	-
ISO-14443a/b	HF	X	-	X	X
SRI512	HF	X	-	-	-
SRIX4K	HF	X	-	-	-
ISO-15693	HF	X	-	X	-

Table 4.1: Capabilities of the ProxmarkIII firmware (February 2011).

Furthermore we added the ability to classify Mifare Classic cards. We can identify those devices by ATQA and SAK. These bytes reveal information about the manufacturer, card model and amount of storage. Mifare Cards read out with the *hf 14a reader* command are automatically classified to gain information, whether they have already been exploited. The following table shows known ATQA and SAK responses as well as the corresponding devices.

ATQA	SAK	Device Type	Vendor
04 00	09 00 00	Mifare Mini	NXP
04 00	08 00 00	Mifare Classic 1k	NXP
02 00	18 00 00	Mifare Classic 4k	NXP
44 00	00 00 00	Mifare Ultra Light	NXP
44 03	20 00 00	Mifare DESFire	NXP
44 03	20 00 00	Mifare DESFire Ev1	NXP
48 00	20 00 00	Mifare JCOP v2.4.1	NXP
48 00	28 00 00	Mifare JCOP v2.3.1	NXP
04 00	88 00 00	Mifare Classic 1k	Infineon
02 00	98 00 00	MPCOS	Gemplus
00 0c	00 00 00	Jewel	Innovision
00 02	38 00 00	Mifare Classic 4k	Nokia
00 08	38 00 00	Mifare Classic 4k	Nokia

Table 4.2: Typical values for ATQA and SAK and the corresponding devices.

4.2 DISCOmfort - The DISCO Mifare Classic Offline Reading Tool

Mifare Classic Cards are wide spread all over the world and used in various security-sensitive scenarios. As its crypto is considered to be broken since 2008 (see [1]) it is recommended to upgrade these security systems. For the analyzation of such systems, we have designed and implemented a tool which enables its user to fully decrypt a Mifare Classic communication. It is initially designed to run on the Linux operating system, but may be easily ported to other platforms.

The tool is called *DISCOmfort* and split up into four parts: a parser, a basic decryption and key recovery module, a module which processes nested authtentications and a graphical user interface (GUI). The pereson using this tool slips into the role of a passive attacker (*eavesdropper*). For this reason, it is necessary to record a communication of a Mifare Classic Card with an authorized reader. This can be done using ProxmarkIII with its included *hf 14a snoop* and *hf 14a list* features. Our modifications allow the traces to be saved automatically. This results in a text file similar to the one in Listing 4.1 with the data represented in hex code. The file is divided into four columns. The first contains information about the time elapsed after the last transmission. The second one should give the signal strength. In the following, the sender is represented by "TAG" for the card and an empty string for the reader. Finally, the last column shows the actual data transmitted over the air in hexadecimal representation. Having a closer look at this, we can find information about parity bits, clarified by exclamation marks. The ISO-14443A standard dictates that a parity bit has to be sent after each byte of data. The exclamation marks highlight wrong parity bits for the preceding byte. This phenomenon is caused by the *Parity Weakness* described in the previous chapter. After all, the string "!crc" denotes a wrong CRC checksum over the received data. This is also related to the crypto scheme. Like parity bits, the checksum is calculated over the plain text. However, this has not got a particular relevance for us. Now, that we had a look at the input data, we will inspect the components of *DISCOmfort*.

Listing 4.1: A Mifare Classic Communication recorded with ProxmarkIII.

```
+       0:     0: TAG 04  00
+     992:      :     93  20
+      64:     0: TAG 61  2f  d2  a2  3e
+    2176:      :     93  70  61  2f  d2  a2  3e  c4  30
+      64:     0: TAG 88  be  59
+  220212:      :     60  00  f5  7b
+     112:     0: TAG 4b  80  52  e4
+    1392:      :     b6  e2  8c  f7  f9  49  ed  67      !crc
+      64:     0: TAG 49! a6! 06! 74
+   15492:      :     21  0a  33  82      !crc
                ...
```

sender	data	description
READER	60 00 f5 7b	reader wants to authenticate (60) against sector 0 (00)
TAG	4b 80 52 e4	tag sends tag nonce n_t (all the 4 bytes)
READER	b6 e2 8c f7 f9 49 ed 67	reader responds with encrypted reader response $\{a_r\}$ and encrypted reader challenge $\{n_r\}$ (4 bytes each)
TAG	49 a6 06 74	tag responds with encrypted tag response $\{a_t\}$ (all the 4 bytes)

Table 4.3: Mifare Classic mutual authentication.

4.2.1 The Parser

The parser component is necessary to extract any information needed from the given input file. It is modeled after a state machine and stores information in a data type called "RFComm" including parameters like the card UID and the particular properties for the challenge-response protocol. However, multiple communications can be extracted from one input file. The parser steps through the file line by line while applying a regular expression. A match results in its further investigation. A message like 9370*aaaaaaaaaaaabb* signals the start of a new communication with a card having the UID *aaaaaaaaaaaa*. For Mifare Classic, only the first four bytes have to be used. Subsequently the parser assumes the type of the tag to be sent. Different types of tags send different ATQA and SAK like shown on page 19. From the following line in the trace the appropriate key and sector are extracted. Remember, that 60 means key A and 61 key B. Eventually, tag nonce, encrypted reader nonce, reader response and tag response are pulled out the following lines which completes information gathering for the authentication process. Moreover, the parity bits are also stored for these transmissions. Finally the rest of the data is stored for later decryption.

4.2.2 The Initial Encryption Component

Once all the necessary data has been extracted and given a semantic by the parser, decryption can begin by recovering the sector key. A typical authentication taken from the communication in Listing 4.1 can be observed in Table 4.3. These information can be used to process a *rollback attack* as stated in [1]. The security rollback attack depends on the fact that the filter function does not use the first nine bits of the linear feedback shift register. This can be seen as a weakness of the cryptographic cipher which allows to perform the authentication backwards to reveal the secret sector key. For the implementation we used the crapto1 library[2] which already implements com-

[2]http://code.google.com/p/crapto1

mon functionalities needed for a successful attack. However, the actual key recovery has been newly implemented as shown in Listing 4.2. The code works as follows. First of all, we extract the keystreams for a_r, n_r and a_t by XOR-ing their encrypted representation with the clear text values, computed with $succ()$. These keystreams are used to recover the LFSR state after a_t has been sent. Now, the state can be rolled back two words (64 bytes) which refers to a_t and a_r without using the feedback bit as input. This is specified via the third argument of $lfsr\_roll\_back\_word()$ set to zero. The crapto1 implementation allows to rollback directly with the encrypted reader nonce n_r, as described earlier, by setting the third argument to one. Finally we rollback by feeding in $UID \oplus n_t$. The resulting LFSR state is the key in reversed byte order. Later on, we will see that this attack is very efficient and can be done within a short amount of time. After a successful recovery, the sector key is stored in the RFComm data structure for later re-usage. To decrypt the communication's following data we will have to replay authentication such that the linear feedback shift register is set to the correct state and gives the appropriate keystream bits. For this purpose, we used a CRYPTO1 implementation from Henryk Plötz et. al, taken from [27].

However, only decrypting with the first sector key eventually does not decrypt the whole trace. On page 11 it was stated that Mifare Classic cards have different blocks, each with its own two keys. During one data communication read and write operations may be done on multiple sectors. In this case reader and card have to authenticate another time for every new sector. This is done with *nested authentication* which slightly differs from the original protocol, such that sending the tag nonce encrypted being the only difference. For this reason a valid tag nonce has to be found. This is done by the component described below.

Listing 4.2: Rollback Attack.

```
uint64_t recover_key(uint32_t uid, uint32_t tag_challenge,
            printable_parity_data_t reader_msg,
            printable_parity_data_t card_response) {

        /* left out input data conversion
         * nr_enc -> encrypted reader nonce
         * reader_response -> encrypted reader response
         * tag_response -> encrypted tag response
         */

        struct Crypto1State *revstate;
        uint64_t lfsr;
        unsigned char* plfsr = (unsigned char*) &lfsr;

        /* keystream for the reader response */
        uint32_t ks2 = reader_response
                ^ prng_successor(tag_challenge, 64);

        /* keystream for the tag response */
        uint32_t ks3 = tag_response
                ^ prng_successor(tag_challenge, 96);

        /* get lfsr register states after a_t has been sent */
        revstate = lfsr_recovery64(ks2, ks3);

        /* encryption scheme ran backwards... */
        lfsr_rollback_word(revstate, 0, 0);
        lfsr_rollback_word(revstate, 0, 0);
        lfsr_rollback_word(revstate, nr_enc, 1);
        lfsr_rollback_word(revstate, uid ^ tag_challenge, 0);

        /* the current lfsr state reveals the key */
        crypto1_get_lfsr(revstate, &lfsr);
        uint64_t key = plfsr[5];
        key = (key << 8) | plfsr[4];
        key = (key << 8) | plfsr[3];
        key = (key << 8) | plfsr[2];
        key = (key << 8) | plfsr[1];
        key = (key << 8) | plfsr[0];

        return (key & 0x0000ffffffffffff);
}
```

4.2.3 The Nested Authentication Module

Now that we have two unknown properties, finding the right sector key is a bit more difficult. First of all the appropriate nonce has to be discovered, because we will use the *rollback attack* which expects the nonce as input value. The easiest way would be to try all 4 byte values. However, this is very inefficient. Initial tests have shown that on an Intel Core 2 Duo processor one can recover at most six keys per second with different nonces. In the worst case this leads to a computation time of

$$\frac{2^{32}}{6\frac{1}{s}} = 7.15 \cdot 10^8 s \approx 23 years \tag{4.1}$$

In fact, the amount of valid nonces can be reduced to 2^{16}, since they are generated by a pseudo random number generator, based on a linear feedback shift register of 16 bit width. In the implementation we use a function *prng_successor()* to compute the successor of a valid nonce. It is essentially the $succ()$ function mentioned in the last chapter. Because of the property described on page 16 we get all possible nonces while iterating $2^{16} - 1$ times. Furthermore we can use the (wrong) parity bits mentioned above. As seen in Theorem 1 on page 16, parity bits leak another three bits of information, such that in the end there are only 2^{13} possible nonces we have to search a key for and replay the authentication. Facing this property we must also take into account that ISO-14443A specifies every bit of each byte to be sent in reverse order. The corresponding code can be inspected in Listing 4.3. It is extracted from a loop which iterates over all possible pseudo random numbers. There are three pairs of parameters calculated and compared to each other,

- decr_p(j+1) corresponds to $n_{t,8j} \oplus n_{t,8j+1} \oplus \cdots \oplus n_{t,8j+8}$ and
- encr_p(j+1) representing $\{p_j\} \oplus \{n_{t,8j+8}\} \oplus 1, j \in [0,2]$.

If they did not match, the alogorithm tries the next nonce. In the other case, a key is recovered via our LFSR Rollback function and the authentication process is replayed using these values.

Moreover, the set of possible nonces is distributed over multiple threads. This allows parallel computation using all processor cores on the machine we operate. To speedup this process even more, already known keys are re-used such that key recovery is not necessary if reader and card authenticate for an already known sector. The whole procedure is summarized in the pseudo code of Listing 4.4.
Once all of the traffic has been decrypted we can extract information about the sector contents. These information are stored in a binary file in order to reuse it for an eventual card emulation.

Listing 4.3: Nonce Property Check from Theorem 1.

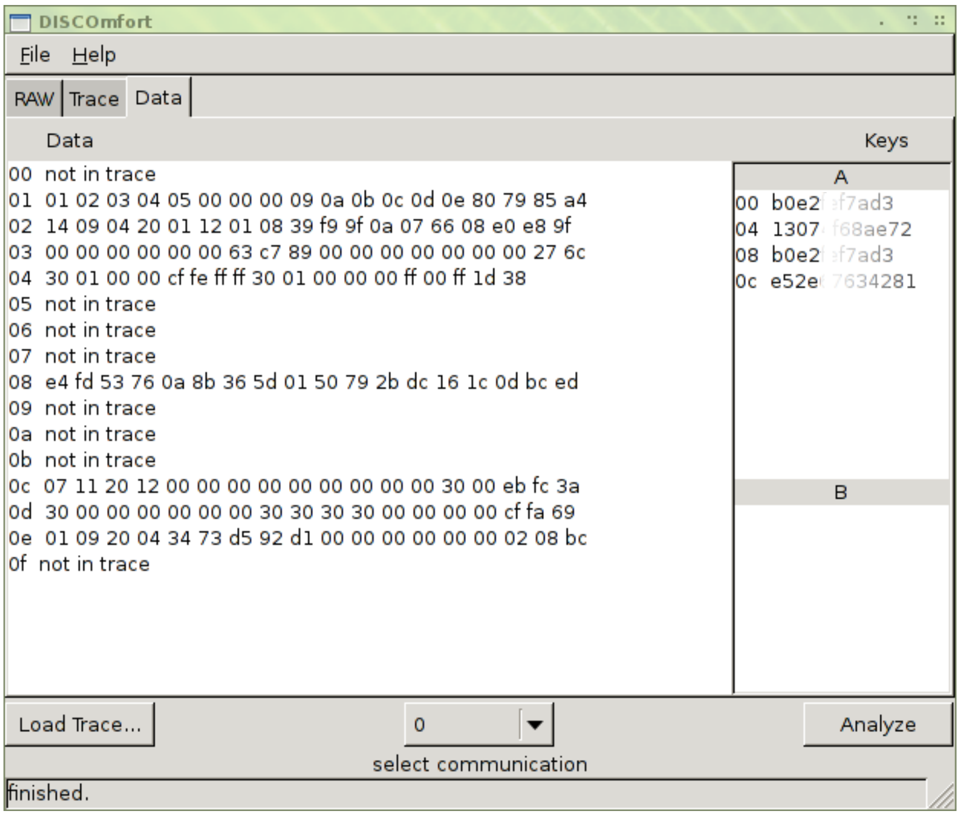
```
/* current_nonce => decrypted nonce we test
 * nonce_encrypted => encrypted nonce found in trace
 * encrypted_parities => parity bits for encrypted nonce
 */
uint8_t decr_p1 = (1 ^ (odd_parity(((current_nonce
                  & 0xff000000) >> 24))
                  ^ ((current_nonce >> 16) & 0x01))) & 0x01;
uint8_t decr_p2 = (1 ^ (odd_parity(((current_nonce
                  & 0x00ff0000) >> 16))
                  ^ ((current_nonce >> 8) & 0x01))) & 0x01;
uint8_t decr_p3 = (1 ^ (odd_parity(((current_nonce
                  & 0x0000ff00) >> 8))
                  ^ ((current_nonce >> 0) & 0x01))) & 0x01;
uint8_t encr_p1 = ((encrypted_parities >> 3) & 0x01)
              ^ ((nonce_encrypted >> 16) & 0x01) ^ 1;
uint8_t encr_p2 = ((encrypted_parities >> 2) & 0x01)
              ^ ((nonce_encrypted >> 8) & 0x01) ^ 1;
uint8_t encr_p3 = ((encrypted_parities >> 1) & 0x01)
              ^ ((nonce_encrypted >> 0) & 0x01) ^ 1;

/* property does not hold  => try next possible nonce */
 if ((decr_p1 != encr_p1) || (decr_p2 != encr_p2)
        || (decr_p3 != encr_p3)) {
                continue;
 }
```

Listing 4.4: Nonce Recovery.

```
for all n in valid_nonces do
  begin
    if Parity_Property_Holds(n, encrypted_nonce) then
     if not Sector_Key_Known then
        key := Recover_Key(uid, n, reader_response,
                                  tag_response);
     else
        key := Known_Key(sector);
     end
     auth_ok = Replay_Authentication(key, uid, n,
                       reader_response, card_response);
     if auth_ok then
        return n;
     end
   end
end
```

4.2.4 The GUI

After decrypting the sector content, we must interprete it. To support this manual task, DISCOmfort offers a graphical user interface to view the trace and sector content. This is illustrated on Figure 4.2.



```
Data
00  not in trace
01  01 02 03 04 05 00 00 00 09 0a 0b 0c 0d 0e 80 79 85 a4
02  14 09 04 20 01 12 01 08 39 f9 9f 0a 07 66 08 e0 e8 9f
03  00 00 00 00 00 00 63 c7 89 00 00 00 00 00 00 00 27 6c
04  30 01 00 00 cf fe ff ff 30 01 00 00 00 ff 00 ff 1d 38
05  not in trace
06  not in trace
07  not in trace
08  e4 fd 53 76 0a 8b 36 5d 01 50 79 2b dc 16 1c 0d bc ed
09  not in trace
0a  not in trace
0b  not in trace
0c  07 11 20 12 00 00 00 00 00 00 00 00 00 30 00 eb fc 3a
0d  30 00 00 00 00 00 00 30 30 30 30 00 00 00 00 cf fa 69
0e  01 09 20 04 34 73 d5 92 d1 00 00 00 00 00 00 02 08 bc
0f  not in trace
```

```
Keys
A
00  b0e2[redacted]
04  1307[redacted]
08  b0e2[redacted]
0c  e52e[redacted]
B
```

```
finished.
```

Figure 4.2: DISCOmfort GUI showing sector data and keys (censored).

We can see three tabs. The first one shows the RAW trace we feed in. Afterwards, the analyzation process can be started by clicking the appropriate button and the resulting decrypted data and keys are shown in the other tabs. In "Trace" the communication is presented. As in the picture, the "Data" tab summarizes up the sector contents and keys. On the bottom a drop-down menu is shown, which enables the user to switch between different decrypted traces. The interface is based on GTK+[3] thus it is also portable to platforms like MacOS X or Windows.

[3] http://www.gtk.org/

4.2.5 Remarks On Card Emulation

The UIDs of Mifare Classic PICCs are stored on a read-only sector which makes it impossible to entirely clone a card. Given a security system which matches UID and encrypted data on the card, it is not possible to circumvent these arrangements. One has to spoof the identifier by emulating the card. We have partially implemented this feature on ProxmarkIII, but it works only without cryptographic functionalities. Besides that, we have made some tests concerning the implementation of CRYPTO1 on the device which seemed to be impossible without touching the FPGA. Our software implementation is too slow to fulfill Mifare timing constraints. They prescribe that answers during the authentication phase have to be sent within an interval of $644\mu s$. Further investigations could be made by extending FPGA code with CRYPTO1 algorithms. This should enable an attacker to fully emulate a tag with the original data collected by *DISCOmfort*.

5 Evaluation

To evaluate the feasibility of security analysis using the developed software, we first have tested which types of RFID systems are still used today. As we have found out, TU Kaiserslautern still applies Mifare Classic systems which can easily be broken by *DISCOmfort*. In the following chapter we will first give a quick overview where insecure RFID technologies are still in use and then focus on TU Kaiserslautern, where nearly all security features have been exploited. Additionally we will provide an evaluation of *DISCOmfort*, inspecting how fast our method works when it comes to decrypting eavesdropped communications.

5.1 RFID Security Failures In General

We have collected some tags from national and European orginizations to find out which specific system they use. Thereby, it came up that most of them are still insecure. At the beginning we had two tags from the Swedish traffic companies "västtrafik" and "Hallandstrafiken" which are used as a replacement for paper tickets in buses. The first one is a *Mifare Ultra Light* which has no cryptographic authentication and confidentiality features. The latter is a *Mifare Classic 1k* where cryptography can be broken with our tool. Copying these tags could emerge a sensitive financial harm for those companies. Another institution which is still using *Mifare Classic* technology can be found in Kaiserslautern. The local football club *1. FC Kaiserslautern* uses a system called *Just Pay*[1] to handle all sorts of money transactions. It is not only settled in Kaiserslautern, but also in football stadiums in Cologne, Flensburg, Sinsheim, Frankfurt a.M. and Trier. In fact, people can change real to virtual money, stored on the tag. With our tool, we would also be able to decrypt the sector contents, once a communication has been eavesdropped and possibly change the amount of money on the card. There are even more examples where Mifare Classic systems can still be found [2].
Another case which does not affect finance, but access control is a popular gas company in Kassel. We have found out, that they use the *LEGIC* system to grant access to their building. This is essentially a tag without any cryptographic capabilities which stores a unique identifier. Being able to get a special tag for guests, we were able to

[1]`http://www.justpay.de`
[2]`http://en.wikipedia.org/wiki/Mifare-Other_places_that_use_Mifare_technology`

Company	Technology	Encryption
västtrafik	Mifare Ultra Light	none
Hallandstrafiken	Mifare Classic 1k	broken
1. FC Kaiserslautern	Mifare Classic	broken
Wintershall Holding GmbH/Wingas	Legic	none
...		

Table 5.1: Weak RFID systems identified during our analysis.

extract its contents and emulate it. This would also give strangers the ability to get into the building and simplifies corporate espionage.

5.2 Case Study At TU Kaiserslautern

At TU Kaiserslautern Mifare Classic 1k cards, manufactured by Infineon are used. They are utilized for cashless paying at all gastronomic facilities and for printing. There are different cards for students, employees and guests, such that prices can be staggered. The tags have a validity period of one year after which affinity to TU Kaiserslautern has to be verified by an employee of "Studierendenwerk Kaiserslautern". There are two classes of PCDs deployed:

- *Online Readers* connected to the network to handle payments and charge the credit on a tag.
- *Offline Readers* which are not connected to the network. Those only show validity period and amount of money available for paying.

From contact persons of "Studierendenwerk Kaiserslautern" we know that in general each users' credit is stored on the card. Every transaction is logged on the paying stations while data is collected everyday by a central server system. This enables administrators to have a detailed look at a customer's buying behavior. Additionally, they are able to lock a specific card and refund the money if it was lost.

The Printing System

Moreover, people can register their card at the local computation center (*RHRK*) to identify themselves against the printing and copying system. For this purpose another account is established to which a user can transfer a particular amount of money. Every copier has a PCD where the user puts his card to login. Then he is able to use the device.

Sector	Key
1	0xe251xxxxxxxx
2	0x1307xxxxxxxx
3	0x0b0exxxxxxxx
4	0xe52exxxxxxxx

Table 5.2: Mensa card sector keys (censored).

By eavesdropping we have found out that a user is only identified by the UID of his card. In fact, no encrypted traffic takes place. One can argue that UIDs on Mifare Classic are stored on a read-only sector which prevents cloning. Due to the fact that no encryption takes place, a device which implements the ISO-14443A standard can be used to spoof the system. In particular, we used ProxmarkIII with its enhanced ISO-14443A emulation capabilities. We were able to tamper our tag's UID and grant access to our account without using our card. The data required for this attack can be collected either by eavesdropping or by stealthily reading a tag. Consequently, we have shown that it is possible to use this service at other people's expense.

The Mensa

Paying at gastronomic facilities involves cryptography. Passively sniffing transactions revealed that multiple sectors are used while checking the credit or paying. We passively collected data from both offline and online readers. The dumped communications have been processed with DISCOmfort to reveal the secret keys and sector contents transmitted over the air. We have collected data for the following actions:

- checking the fund at an offline reader
- paying
- recharging

It turned out that ten blocks of the card are used, as shown in Table 5.3. There is a different key for every sector but all Mensa cards share the same set of keys. Additionally, our recording revealed that besides for the recharging process, just key A is used. Unfortunately, we were not able to recover key B due to the fact that ProxmarkIII always crashed. It was not possible to identify the problem as it seems to be a software bug. According to Table 5.3 let us have a look at the sector contents.

It is specified in [26] that block three contains the access conditions for sector one. In this case, block 1-3 can only be read using key A. Writing to the card is only allowed with key B. We can approve that, because all readers only read sectors using key A. Writing is possible with key B. Furthermore, we have tried to manipulate data on the card which was impossible without the second key. The first block remains the same

Block	Content	Comment
01	01 02 03 04 05 00 00 00 09 0a 0b 0c 0d 0e 80 79	constant on every card
02	01 03 10 20 01 12 01 a7 43 25 9f 0a 07 66 08 e2	constant on every card
03	00 00 00 00 00 00 63 c7 89 00 00 00 00 00 00 00	access bits
04	d5 01 00 00 2a fe ff ff d5 01 00 00 00 ff 00 ff	credit
08	20 cd 1d 0a 07 da 0c 34 00 00 07 d0 00 00 00 37	?, changed by payment
09	0f 64 08 0b 07 da 0d 26 00 00 00 32 00 00 46 15	?, changed by payment
10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff	constant on every card
12	31 10 20 11 30 00 00 00 00 00 00 00 00 30 00 ef	expiration date
13	30 00 00 00 00 00 00 30 30 30 30 00 00 00 00 cf	constant on every card
14	01 03 20 10 34 b8 b2 23 f4 00 00 00 00 00 00 24	delivery date

Table 5.3: Mensa card sector contents in hex representation.

with every card we have tested. In the fourth block four bytes of data are stored, two times normally and once inverted. This indicates it being a *value block* supported by the fact that they are usually used for paying. Converting the values revealed that the credit in cents is stored in this sector. We have been able to read the access bits for sector 1 which includes permissions for the fourth data block. For the value block the following access conditions apply:

- **key A**: read, decrement
- **key B**: read, decrement, increment, write

This measure can prevent abuse if somebody was able to encrypt a paying transaction. Additionally, it is nearly impossible to record recharging because cards have to be put into a small slot, shielding all signals sent from the tag.

Block number twelve is used to store the expiration date. As it can easily be seen, it is represented in hexadecimal. The last bytes' meanings of this sector remain unknown. One could assume that it could be some kind of check sum but we can't approve that. In fact, this value differs on every card. Due to the limited amount of traces we don't have enough data to compare two cards with the same expiration date. The 14th block also contains four bytes which look like a date. It turned out that there are three series of cards:

- series 1 from 01/09/2004
- series 2 from 01/03/2008
- series 3 from 01/03/2010

We have inspected all of them but could not find any difference regarding security features or strong divergent sector contents. Finally there are some blocks we couldn't assign a proper meaning. We have observed that block 8 and 9 are changed during a payment. Finally we know that the card also stores information about the entity using it being a student, employee or guest.

Cloning A Card The information gathered in our experiments enabled us to clone a card with the particular state it had during the trace we recorded. For this purpose we used a commercial RFID reader which is available on pertinent online auction platforms. We took an empty Mifare Classic tag and wrote all the data blocks, access conditions and keys onto it. In the end we had an exact copy of one of our Mensa cards, except the key set B and block 0 which contains the *UID*. This enabled us to perform a payment without any error message. Furthermore, the credit on the original card has not been changed at all. Recharging the tag at a valid paying station was not possible and returned an error, because we do not have the writing key B on the cloned device. However, cloning a card at the state of a reading transaction is enough to cause financial damage.

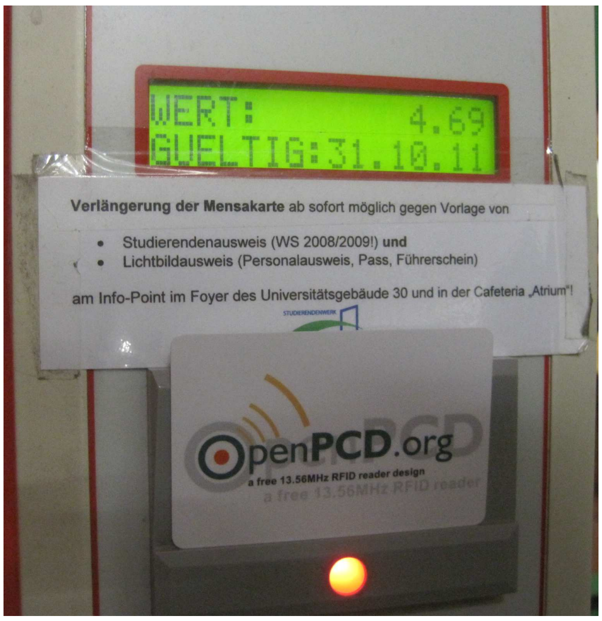

WERT: 4.69
GUELTIG:31.10.11
Verlängerung der Mensakarte ab sofort möglich gegen Vorlage von
• Studierendenausweis (WS 2008/2009!) und
• Lichtbildausweis (Personalausweis, Pass, Führerschein)
am Info-Point im Foyer des Universitätsgebäude 30 und in der Cafeteria „Atrium"!
OpenPCD.org
a free 13.56MHz RFID reader design

Figure 5.1: Mensa offline reader with our cloned card.

Although the system assigned our payment to the original card number, it didn't warn the administrator that something could be wrong with the user's account. Only by manual inspection, an employee of Studierendenwerk Kaiserslautern can detect fraud and lock the card. The person whose card has been copied doesn't even notice that somebody abuses his account as money is managed on the card itself. The computer system just gathers information about transactions for accountancy and refunding. None of the paying stations checks the credit on the card against a value on a central server.

Countermeasures And Outlook As far as the current system is concerned, there are the following countermeasures to prevent or at least detect fraud.

1. Every night, when all data from automatas has been collected, iterate over each user's account and check if his current credit is reproducible by means of payments and rechargings. This can be done automatically and warnings may be sent to the administrator if some discrepancies emerged.
2. The system could be changed such that card number in the system is correlated with a UID of a card. This prevents attackers from cloning a card, as the UID is unique and immutable on every Mifare Classic card. However, it would still be possible to emulate a card with proper equipment.
3. One could established a centralized system, where the credit is stored just on a server. Every paying station has to check the value online and decrement it. Cloning would still be possible, but users would become suspicious and a fraud could be detected.

The first proposal would be the cheapest. It could protect against and detect fraud, such that affected cards can be locked. The second one requires readers to be reprogrammed, whereas the third is a complete change of design and maybe too costly to perform.

However, a new system is going to be supplied in April 2011. It remains to be seen if things change concerning security.

5.3 DISCOmfort Performance

To evaluate the performance of DISCOmfort we used a general purpose PC with an Intel Core i7 920 CPU and 4GB of RAM running a 64 bit Linux, version 2.6.34. Thereby, the amount of memory is neglectable since we just store some temporary values which do not cover more than a few hundert kilobytes. The processor has four physical cores and features eight threads (see [28] for further information). Our software is compiled with the GNU C compiler[2] using the highest optimization level.

Performance of LFSR Rollback Applying the LFSR with valid input variables in terms of clear-text nonces revealed that this procedure takes less than a second to be completed. However, we are able to perform on average 8 key recoveries per second with one thread.

[2] `http://www.gnu.org/software/gcc/`

5.3.1 Recovering Encrypted Nonces

The nested authentication module requires the most time to perform decryptions. Basically, we have two different situations which may occur, where n_t has to be recovered:

1. A nested authentication takes place where the sector key is already known.
2. The sector key has to be recovered for every possible n_t to replay a valid authentication.

In the first case we observed that nonce recovery is done within 300 milliseconds on average. This should be sufficient to perform a full decryption of a trace in at most 19 seconds if all data blocks are transferred.

Recovering a nonce with an unknown sector key is slower. In theory we would have to test an average of

$$\frac{2^{13}}{2} = 4096 \tag{5.1}$$

possible nonces. With a rate $r = 8\frac{keys}{s}$ one thread would take 585 seconds to recover the nonce. Tests have been conducted and their results are presented in Figure 5.2. We have to take into account that we only have a limited number of traces, so that the amount of test data is not sufficient to apply advanced statistical methods. For the measurements 22 different nonce recovery processes have been used. In Figure 5.3 we can see the distribution of times needed to recover a nonce. We can definitely say, that nonce recovery is accelerated by using multiple threads. Using all possible cores of our test system, we can do this process within an average time of 83 seconds. The acceleration is illustrated in Figure 5.2.

Furthermore, we have conducted the same measurements on a sophisticated server with 32 threads. This enabled us to recover a nonce in less than 20 seconds.

Threads	Average Time In [s]	Standard Deviation In [s]
1	440.18	258.97
2	232.31	133.77
3	170.42	97.94
4	132.63	71.08
5	113.28	70.59
6	101.45	57.36
7	86.67	55.04
8	83.45	45.10
32	18.59	11.09

Table 5.4: Average values and standard deviations of our tests.

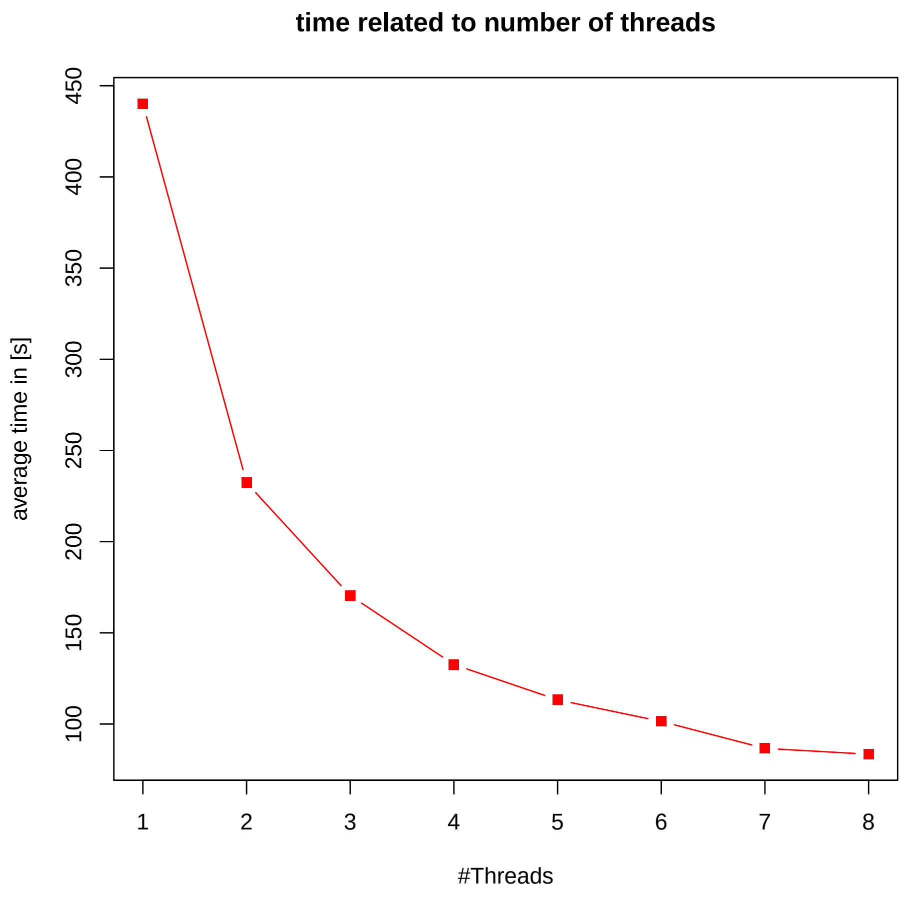


Figure 5.2: Average nonce recovery times for multiple threads.

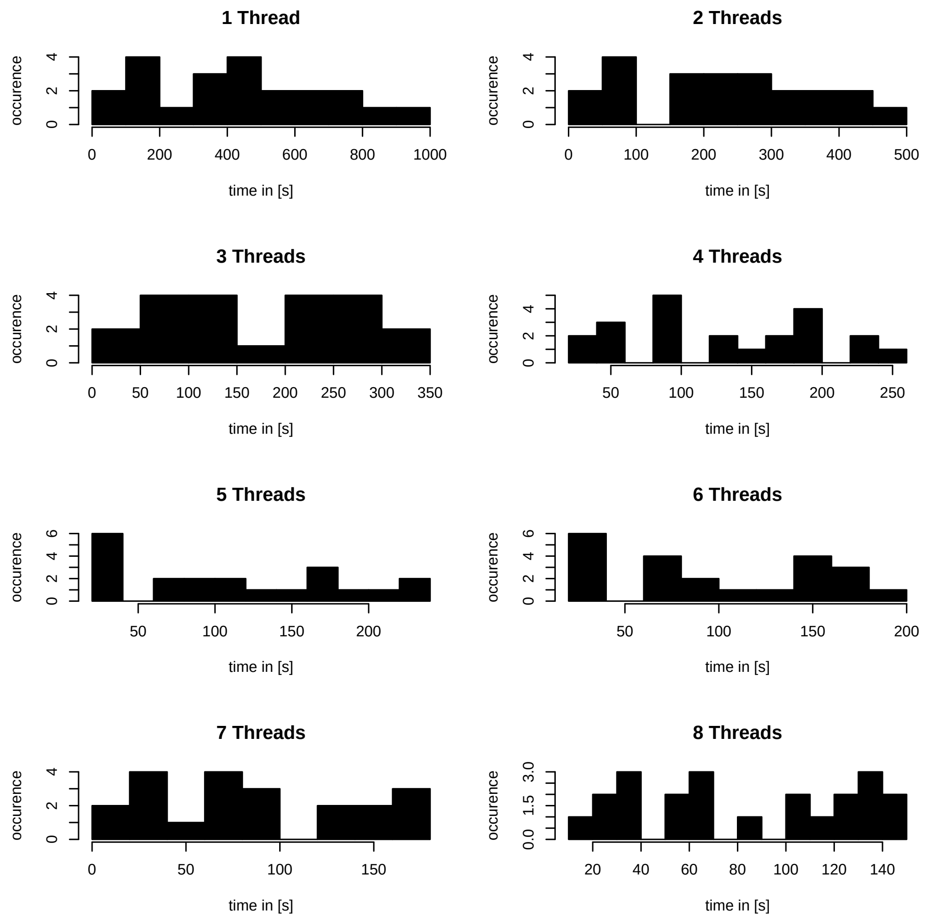


Figure 5.3: Histograms for the times measured to recover a nonce.

6 Conclusion

In summary, we have shown that vulnerable RFID infrastructures still exist. Built upon ProxmarkIII a tool, DISCOmfort, has been developed to analyze such systems. Using the software, the user performs passive eavesdropping attacks without disrupting the system. With the underlying hardware platform we are able to detect and compromise Mifare Classic devices. A graphical user interface thereby supports the user to gain information about used data blocks and helps to find further weak points.

We have applied our technology and found serious security failures. This involves big companies as well as universities. A case study has been performed at TU Kaiserslautern to evaluate the Mensa Card and copying system. It turned out that a possible attacker has an easy job to do financial damage while staying unrecognized.

Furthermore, countermeasures have been proposed which could improve RFID security even with insecure devices like Mifare Classic. However, this should not be a general solution.

Moreover, we have seen that DISCOmfort is not only effective, but also efficient. We have conducted several experiments to test its performance. It has been shown that even on a general-purpose PC it is possible to decipher Mifare Classic traces within minutes. Using more sophisticated hardware we can even recover nonces in the magnitude of seconds.

However, DISCOmfort is fast, but several approaches could be made to improve performance. It has been mentioned in [1] that nonces strongly depend on timing. One could apply this theory in our software to gain even faster recovery results. Further investigations may also be applied concerning upcoming technologies. The Mifare DESFire system may have insuperable barriers concerning cryptography. But as we have seen, it also depends on the administrator to use these features making the resulting system really secure. It would be interesting to have a closer look at this and other technology.

Bibliography

[1] Flavio D. Garcia, Peter Rossum, Roel Verdult, and Ronny Wichers Schreur. Wirelessly pickpocketing a mifare classic card. In *In IEEE Symposium on Security and Privacy 2009*, pages 3–15. IEEE, 2009.

[2] Wikipedia – The Free Encyclopedia. Mifare. Online version – February 10th, 2011. `http://de.wikipedia.org/wiki/Mifare`.

[3] Ari Juels. Rfid security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, 2006.

[4] Melanie Rieback, Georgi Gaydadjiev, Bruno Crispo, Rutger Hofman, and Andrew Tanenbaum. A platform for rfid security and privacy administration. In *Proc. USENIX/SAGE Large Installation System Administration conference*, pages 89–102, Washington DC, USA, December 2006. `http://www.rfidguardian.org/papers/lisa.06.pdf`.

[5] G. P. Hancke. Eavesdropping attacks on high-frequency rfid tokens. In *4th Workshop on RFID Security*, pages 100–113, 2008.

[6] Thomas Finke and Harald Kelter. Radio Frequency Identification – Abhörmöglichkeiten der Kommunikation zwischen Lesegerät und Transponder am Beispiel eines ISO14443-Systems, September 2004. Whitepaper ISO 14443; BSI – Bundesamt für Sicherheit in der Informationstechnik `http://www.bsi.bund.de/fachthem/rfid/Abh_RFID.pdf` – geprüft: 13. Februar 2009.

[7] Gerhard P. Hancke. Practical attacks on proximity identification systems (short paper). In *IEEE Symposium on Security and Privacy*, pages 328–333. IEEE Computer Society, 2006.

[8] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for rfid systems using the aes algorithm. In *CHES'04*, pages 357–370, 2004.

[9] Yossef Oren and Martin Feldhofer. A low-resource public-key identification scheme for rfid tags and sensor nodes. In *WISEC'09*, pages 59–68, 2009.

[10] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Cambridge, MA, USA, 1979.

[11] Boris Danev, Thomas S. Heydt-Benjamin, and Srdjan Čapkun. Physical-layer identification of rfid devices. In *Proceedings of the 18th conference on USENIX*

security symposium, SSYM'09, pages 199–214, Berkeley, CA, USA, 2009. USENIX Association.

[12] Stephen C. Bono, Matthew Green, Adam Stubblefield, Ari Juels, Aviel D. Rubin, and Michael Szydlo. Security analysis of a cryptographically-enabled rfid device. In *Proceedings of the 14th conference on USENIX Security Symposium - Volume 14*, pages 1–1, Berkeley, CA, USA, 2005. USENIX Association.

[13] Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz. Reverse-engineering a cryptographic rfid tag. In *Proceedings of the 17th conference on Security symposium*, pages 185–193, Berkeley, CA, USA, 2008. USENIX Association.

[14] Gerhard Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A practical attack on the mifare classic. In *Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications*, CARDIS '08, pages 267–282, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] Flavio Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Schreur, and Bart Jacobs. Dismantling MIFARE classic. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, chapter 7, pages 97–114. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[16] Wikipedia – The Free Encyclopedia. Rfid. Online version – February 10th, 2011. `http://de.wikipedia.org/wiki/RFID`.

[17] Wikipedia – The Free Encyclopedia. Amplitude-shift keying. Online version – February 10th, 2011. `http://en.wikipedia.org/wiki/Amplitude_Shift_Keying`.

[18] Wikipedia – The Free Encyclopedia. Frequency-shift keying. Online version – February 10th, 2011. `http://en.wikipedia.org/wiki/Frequency-shift_keying`.

[19] Wikipedia – The Free Encyclopedia. Phase-shift keying. Online version – February 10th, 2011. `http://en.wikipedia.org/wiki/Phase-shift_keying`.

[20] Wikipedia – The Free Encyclopedia. Non-return-to-zero. Online version – February 10th, 2011. `http://en.wikipedia.org/wiki/Non-return-to-zero`.

[21] Wikipedia – The Free Encyclopedia. Manchester encoding. Online version – February 10th, 2011. `http://en.wikipedia.org/wiki/Manchester_encoding`.

[22] ISO 14443:2008. *Part 2: Radio frquency power and signal interface – Identification cards – Contactless integrated circuit(s) cards - Proximity cards.* ISO, Geneva, Switzerland.

[23] ISO 14443:2008. *Part 4: Physical characteristics – Identification cards – Contactless integrated circuit(s) cards - Proximity cards.* ISO, Geneva, Switzerland.

[24] ISO 14443:2008. *Part 3: Initialization and anticollision – Identification cards – Contactless integrated circuit(s) cards - Proximity cards*. ISO, Geneva, Switzerland.

[25] ISO 14443:2008. *Part 4: Transmission protocol – Identification cards – Contactless integrated circuit(s) cards - Proximity cards*. ISO, Geneva, Switzerland.

[26] NXP Semiconductors. *MF1S5009 Public Data Sheet*, July 2010.

[27] Henryk Plötz. Mifare Classic – Eine Analyse der Implementierung. Diploma thesis, Institut für Informatik, Humboldt-Universität zu Berlin, 2008.

[28] Intel Corporation. Intel core i7-920 processsor specifications. Online version – February 15th, 2011. `http://ark.intel.com/Product.aspx?id=37147`.

List of Figures

List of Tables