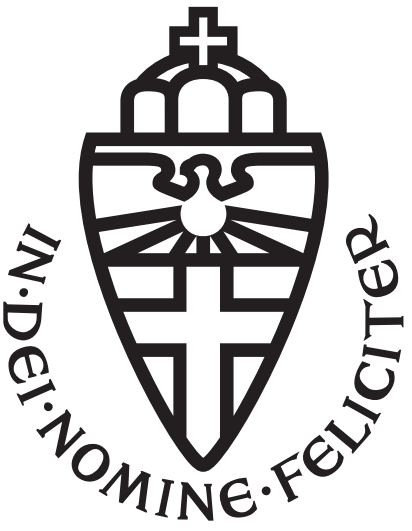
# Analysis of the MIFARE Classic used in the OV-Chipkaart project

Gerhard de Koning Gans
Radboud University Nijmegen




*Supervisors*
Jaap-Henk Hoepman
Flavio D. Garcia

Thesis 584
Gerhard de Koning Gans
Version 1.00
June 2008

**Abstract**

The MIFARE Classic is the most widely used contactless smart card in the market. Its design and implementation details are kept secret by its manufacturer. We investigate the MIFARE Classic because this card should become the new ticket, called the OV-Chipkaart, in the Dutch public transport system.

This thesis studies the architecture of the card and the communication protocol between card and reader. At the start of this research, there was no information available on the MIFARE Classic protocol nor the implementation of the OV-Chipkaart. To perform this research we used the Proxmark, a device that allows us to eavesdrop on the communication between the reader and the card.

Our contributions are as follows. First, an ISO14443-A firmware implementation for the Proxmark that enables eavesdropping on the MIFARE Classic, among other card types. Secondly, we present an overview of the commands and responses of the protocol. Furthermore, we develop a method to read data from the MIFARE Classic card without knowledge of the secret key. Due to a weakness in the pseudo-random generator, we are able to recover the keystream generated by the CRYPTO1 stream cipher. We exploit the malleability of the stream cipher to read *all* memory blocks of the first sector of the card. Moreover, we are able to read *any* sector of the memory of the card, provided that we know *one* memory block within this sector. Finally, and perhaps more damaging, the same holds for *modifying* memory blocks.

# Preface

In the past year I have had the privilege to perform research on an interesting topic. A topic where I could combine theory and practice. It was my supervisor Jaap-Henk Hoepman who suggested the OV-Chipkaart as subject of investigation. How does this card actually work? How does it communicate and, most important of all, is it secure enough?

One important prerequisite at the start of this research was the possibility to look into the communication of the OV-Chipkaart. At that point I ran into the work of Roel Verdult who was already working on a device which should be able to do this. Apparently, it was not that easy. Appropriate hardware became one of the main problems. For months we worked in relative silence on this matter. November 14th, Roel and I had some little success in Rotterdam. Finally we got our hardware working.

It remained quiet until December 2007 when I received an e-mail from Jaap-Henk. He pointed me to a presentation of two German researchers who had found some major weaknesses in the MIFARE Classic chip. Well, this might sound totally different from our research and it would not have gained much attention if the OV-Chipkaart did not actually use the MIFARE Classic chip. Dutch media took this very seriously, and soon many questions rose about the security of the OV-Chipkaart. Then on January 14th, Roel Verdult showed that it was possible to clone a disposable card. Meanwhile, I had already developed an attack that allowed to read and modify memory contents without knowing the secret key.

From that time on it remained busy. One reason was the press attention generated on this topic. Another reason, we had already an interesting attack and decided to write a paper on these findings. When this paper, the contents of which is featured in this thesis, was finished, we got some interesting new developments. More people got interested in the topic. At that point, it seemed that with some effort we could get complete insight in the MIFARE Classic security. This thesis stops at that point and is not about that joint research. I enjoyed the fact that I could participate in a group of enthuastic people who joined Roel and me in some feverish investigations of the detailed workings of the chip. This resulted in the dismantling of the MIFARE Classic. Which is, as said before, described elsewhere.

## Acknowledgements

Hereby I would like to thank a lot of people. First, I would like to thank Jaap-Henk Hoepman, he made me enthusiastic for the subject and took good care of the research direction. Although circumstances prevented him from joining the latest research I would like to stress that it was his idea to look into the OV-Chipkaart. Flavio Garcia, who was already working in this field and an excellent second supervisor. Roel Verdult, who almost thought of terminating his work on the hardware and thought of starting another research. Happily he refrained from this idea. We spent both many hours on the hardware of this research and succeeded. Then, I would like to thank the Kali brothers, as I call Vinesh and Ravindra sometimes, for their enduring support from the beginning.

Furthermore, I would like to thank people who joined us later on in further research on this topic. Ruben Muijrers, a student who just bumped into this, has given an outstanding contribution to the research. The same counts for Ronny Wichers Schreur. He was hard to stop in his attempts to improve the attack. Peter van Rossum, his mathematical skill combined with the area of computer security was very effective and I am convinced this saved us many hours. Wouter Teepe, yes the guy who handled the media, and yes that also means he had to handle the difficult questions. He did very well on that job. Last but not least, I would like to thank Bart Jacobs for steering this sometimes chaotic group and keeping us motivated.

There were many others involved. I hope you understand it is difficult to name every person involved in this project. And also remember that important people are not always listed.

Gerhard de Koning Gans, June 2008

# Contents

# 1 Introduction

In this thesis the focus is on Radio Frequency Identification (RFID) technology. RFID is a technology that is used in many different applications and purposes. Even though many people see RFID as a 'new technology', it has been in use since World War II for military purposes. It was used to distinguish an allied plane from an enemy plane [Fin03]. Nowadays, the technology has developed to tiny *digital labels* that can be read out from varying distances[1].

## 1.1 RFID

In short, RFID is a wireless technique to identify objects. The concept of wireless communication has been adopted in many applications. Some examples are radio, cell phones and the Global Positioning System (GPS). In case of RFID, there is communication between a reader (also known as Proximity Coupling Device (PCD)) and a transponder[2] (also known as tag). Transponders are available in different types, there are active and passive transponders. Where the former have their own power source, like a battery, the latter use the power they receive from the magnetic field of the reader. The advantage of a self-powered transponder is that it is able to communicate over bigger distances. The advantage of a passive transponder is that it has no battery, which can unload, and its production can be really cheap.

RFID has become a pervasive technology as it has been adopted in a wide range of applications. It is used as a replacement of existing 'analog' products like barcodes. It is also used in addition to existing identification methods like access cards, identity cards, passports and electronic labels. The technology also introduces complete new applications like RFID powder. RFID powder has the size of 0,05mm by 0,05mm and was announced by Hitachi on February 13th, 2007[3]. Due to its tiny size it is possible to hide it in a sheet of paper.

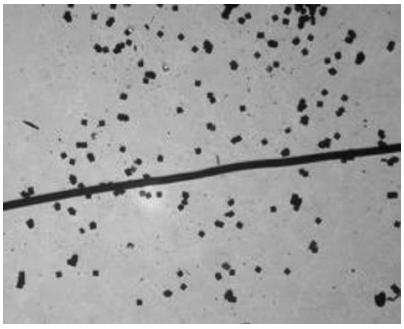


Figure 1: RFID powder next to a human hair

The combination of smartcards and a contactless interface (RFID) results in 'smart' contactless cards. These kind of cards are used in access control, electronic purse, electronic ticketing and many other applications. In this thesis we focus on these contactless cards. In the most simple applications a contactless card just sends its Unique Identifier (UID). This is a passive contactless card that has no computational power and it is just suitable for identification. There also exist cards that contain a piece of memory that can be written (is re-programmable) and can return the content of its memory on request of a reader. An example is the MIFARE Ultralight card which has 512 bits of memory. But also more

[1] distance depends on the physical characteristics
[2] in this thesis most of the time we refer to it as a (contactless) card)
[3] http://www.hitachi.com/New/cnews/2007.html

advanced cards which have some computational power are available in the market. These allow encryption like DES and AES, but also proprietary ciphers are used. Think of the Sony FeliCa and the MIFARE Classic from NXP. Being this MIFARE Classic that is our subject of investigation. MIFARE Classic is part of the MIFARE product family of NXP Semiconductors. The MIFARE family consists of the following cards: Ultralight, Classic, DESFire and SmartMX. The MIFARE Classic card is available with 1 KB and 4 KB of memory. Currently it is the most used contactless card worldwide[4].

## 1.2 Technology in Action

Contactless smartcards like the MIFARE Classic are used for example in access control and public transport ticketing. In London, the Oyster card, which is a MIFARE Classic card, is used in public transport ticketing. Besides the ticketing system of London there are many other systems that use the MIFARE Classic chip. In the Netherlands, the OV-Chipkaart should become the new ticketing standard for public transport. It is the first project where such a system is set up nationwide.

## 1.3 Today's Use

Although almost everyone uses RFID in everyday life, most people are not aware of it. In most cases the technology is out of sight and hidden in applications. Most of the time it is not even recognized and brings the ease of use it was expected to bring. In this section we give some examples of applications that use RFID technology.

**Animal Identification** In Animal Identification a tiny chip is implanted just below the skin of an animal. This way it can easily be identified. Additionally, medical information like vaccinations can be linked to this animal. The ISO 11784/85 standard is used in Animal Identification. The information like vaccination data needs to be looked up in a database. ISO 14223 defines how to store this data on the implanted chip. This means that there is no need to access a database, because all information is contained on the chip itself.

**Car Keys** Almost every new car comes with a remote control to open the car remotely. In the early days the transmitted signal was just an identification number. The car checked for the correct number and opened the door if its number and the one of the transponder (car key) matched. Nowadays, a popular system for car keys is KeeLoq. KeeLoq is a technology from Microchip that uses RFID technology. It uses a cryptographic algorithm to prevent eavesdroppers from copying the car key and getting unauthorized access to the car. The operating distance of the KeeLoq system is about 100 meters. The security of KeeLoq has been broken. After cryptanalysis of the algorithm by Bogdanov [Bog07] many attacks followed. The latest ones are very serious and use side-channel information to recover the key of the controller. Even the master key can be recovered.

[4] the MIFARE family represents 85% of the contactless smartcard market, `http://www.nxp.com`

**Access Control** Many buildings use RFID for Access Control. A contactless card is used for identification and authentication. As such a card or token can be stolen it is a good practice to add extra security layers. One could, for example, double-check the authentication by using information about a person, think of biometric information like a fingerprint or iris scan. Another check could be done on what someone knows, like a secret code.

Figure 2: University entrance

**Public Transport** RFID has also found its way into Public Transport systems. In Hong Kong a contactless card system was introduced in 1997[5]. In London a contactless card for the metro was issued in 2003[6]. There are many examples of other cities around the world that use contactless cards in their transport system. The first system where the contactless card should become a national traveling card was the OV-Chipkaart in the Netherlands. Some reasons that make RFID an attractive solution are:

- Travel information can be stored on the label or card.
- Low transaction times.
- No physical contact needed between reader and card. This prevents wear and tear damage.
- Better pricing, the traveler pays for the traveled distance instead of the crossed zones.

Some general disadvantages of RFID which also count for use in public transport are:

- There are some privacy issues as most cards respond with a fixed unique identifier. This way the traveler can be identified not only by the transport system but also can be traced by an attacker equipped with a simple reader. This is a violation of the location privacy of a traveler.
- Communication can be easily eavesdropped without being recognized by the traveler.
- Relay attacks are possible by eavesdropping on the communication. In a relay attack, one attacker is near the victim and communicates the signal of the card of the victim to another attacker positioned at the gate. These kind of attacks are shown to be feasible by Hancke and Kasper [Han05, Kas06]. A possible countermeasure for these kind of attacks is to use distance bounding protocols [HK].

Despite of these disadvantages, many transport systems use contactless cards. A very popular card is the MIFARE Classic, which has been already in use for

[5] `http://en.wikipedia.org/wiki/Octopus_card`
[6] `http://en.wikipedia.org/wiki/Oyster_card`

years. Just as in the system of London the Dutch public transport, represented by TransLink Systems (TLS), decided to use the MIFARE Classic in their implementation. The card has been used in many systems around the world and has never shown any failures on its security. Unfortunately, after some serious analysis of the card it turned out that the MIFARE Classic is broken and can be easily compromised. This means that many systems around the world need to migrate to better protected cards within a little time frame.

**Ticketing** The FIFA World Cup tickets of 2006 in Germany are an example of RFID use in ticketing. Timo Kasper showed [Kas06] that he could perform a successful replay attack on these tickets. Of course the tickets were secured twofold. First, the tickets physical appearance needed to be correct and second the data stored on the MIFARE Ultralight chip also needed to be correct.

**Labels** RFID is also used as replacement for or in addition to barcodes. Barcodes make it possible to quickly scan a serial number of a product and obtain or store information about it. This is especially useful in logistics. For the barcodes to be successfully scanned it is important that the barcode, and therefore the product, is correctly positioned with respect to the reader. The use of RFID removes this constraint. It does only require the label to be within a certain range. Depending on the strength of the signal of the reader, this range might be from several centimeters to several meters. This allows applications where a truck drives through a gate and all products that leave the factory are scanned[7].

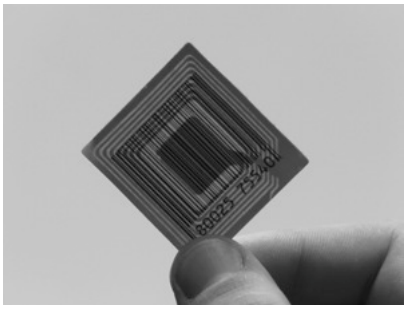


Figure 3: Barcode and RFID label

The barcode label in Figure 3 is extended with an RFID chip which is visible when we keep it against the sunlight. The RFID chip is visible in the middle and the antenna windings at the border of the label.

**Electronic Passport** Electronic passports are basically just passports, but with an embedded RFID chip. Again an application where RFID is used in addition to an existing product. The electronic passport is developed to store biometric data. This application uses a heavier, and therefore more expensive, RFID chip. For sake of privacy the RFID chip uses a random UID every time it gets selected. To prevent e-passports from being read by just anyone, an e-passport uses a Basic Access Control (BAC) mechanism. This requires access to printed information inside the passport. Weaknesses of this method are shown in [HHJ+06, AKQ]. But even before authentication e-passports already leak information about the nationality of a person [RMP07].

[7] `http://www.qualitydigest.com/may07/articles/04_article.shtml`

## 1.4 Outline of this Thesis

This thesis is based on and contains parts of the paper 'A Practical Attack on the MIFARE Classic' (forthcoming [dKGHG08]). Section 2 discusses the research goals that we wanted to achieve. Then, Section 3 discusses the communication technique that is used in the specific RFID system (MIFARE Classic) we want to analyze. In Section 4 the currently available open-source tools are described which can be used to analyze RFID communication. In the end, we choose the Proxmark III of Jonathan Westhues, which needed some software extension to meet our needs. This software extension is discussed in Section 5 and is a general solution that can be used in any further research on ISO 14443-A protocols. Then Section 6 discusses the protocol and the found weaknesses. A practical attack is deployed on these weaknesses. Also, results of concurrent research is addressed. Although, it turned out to be a lost race for the MIFARE Classic, it is our responsibility to give recommendations on safer usage of the card. We conclude with the conclusions and recommendations in Section 7. Finally, further research is discussed in Section 8.

# 2 Research

## 2.1 Problem Definition

RFID does not solely bring ease, it also introduces new security risks. It is a wireless application, which means the data is transferred over the air. This introduces possibilities for eavesdroppers to eavesdrop on the communication while the chance of being detected is low. In this thesis the protocol of the OV-Chipkaart project and therefore MIFARE Classic is subject of investigation. The MIFARE Ultralight and the MIFARE Classic 4k cards are used in the OV-Chipkaart project. The goal of this research project is to perform an analysis of the OV-Chipkaart protocol and get insight in the communication.

The communication in MIFARE Classic applications is encrypted and hides the protocol messages. NXP does not give any information on the protocol used by MIFARE 4k and MIFARE Ultralight applications. NXP has developed several ASICs[8] that are able to handle the protocol. Specifications on those ASICs talk about the interface of the chip. The actual processing of the RF communication is done by this ASIC.

In the Netherlands the company TransLink Systems (TLS) is implementing the first travel card system that will operate nationwide called the OV-Chipkaart. The project runs since 2002 and is, to some extend, comparable to the Octopus card of Hong Kong. The Octopus card was introduced in Hong Kong in 1997 to collect fares in the public transport system. Nowadays it is used for many more applications in micro payments[9]. Participants in the project are the Dutch government together with the carriers NS, Connexxion, RET, HTM and GVB which serve 80 % of the public transport in the Netherlands.

**Problem Definition**

1. How do tag and reader communicate?
2. Are there privacy problems?
3. How secure is it?

**Research Goal**

1. Reverse engineering the MIFARE protocol.
2. Reverse engineering an application specific protocol like the OV-Chipkaart.

## 2.2 Related work

There have been several studies on RFID that focus on the contactless interface of RFID cards and the limitations on the complexity of their design.

[8] Application Specific Integrated Circuit (ASIC)
[9] `http://en.wikipedia.org/wiki/Octopus_card`

**Relay attack** A contactless interface means that relay attacks are possible which is shown by Gerhard Hancke in [Han05] and Timo Kasper in [Kas06]. A possible solution to prevent relay attacks is to make good restrictions on the relay time using distance bounding protocols [HK].

**Replay attack** Another possible attack is the replay attack. This means that the attacker replays an earlier recorded message. This is obvious a problem for RFID tags that just send out an unique identification number like the VeriChip. Jonathan Westhues describes on his website[10] how to clone a VeriChip. He also demonstrated that some Government buildings use access cards that just send out a number. Replaying this signal was enough to open the door[11].
Timo Kasper also carried out a replay attack in [Kas06].

**Disposable RFID cards** Sometimes RFID is used in a bad way. Siekerman and van der Schee found a flaw in the disposable card of the Dutch Public Transport system [SvdS07]. There was a wrong use of the locking functionality of the card. The card has lock bits which prevent writing to the card memory when set. It was possible to lock the lock bits which means that the system could not lock portions of to the card memory anymore. The system did not check for this situation. This is an example of a bad implementation.

**Clone attacks** The disposable card (MIFARE Ultralight) does not use any encryption. The command set is known and the functionality and memory is limited. Roel Verdult managed to program a cloning device (Ghost) in such a way that it was recognized as a disposable card [Ver08]. The 'advantage' of such a device is that a memory dump can be placed back when needed. Since the back-end system did not check for any duplicates in the transport system this attack was not recognized.

**Reverse engineering** Karsten Nohl and Henryk Plötz have partially reverse engineered the MIFARE Classic cryptographic algorithm [NP07]. The MIFARE Classic card is in use since 1994 which is quit a long time. Nohl and Plötz managed to recover the cipher by removing layers from the chip and taking pictures of the result with a 500x optical microscope. After determination of 70 different logical gates the rest was detected using image processing techniques.

## 2.3 Our contribution

We used a Proxmark III[12] to analyze MIFARE cards and mount an attack. To do so, we have implemented the ISO 14443-A functionality on the Proxmark, since only ISO 14443-B was implemented at that time. We programmed both processing and generation of reader-to-tag and tag-to-reader communication at physical and higher levels of the protocol. The source code of the firmware is available in the public domain[13]. Concurrently, and independently from Nohl and Plötz results, we also noticed a weakness in the pseudo-random generator.

[10] `http://www.cq.cx/verichip.pl`
[11] `http://www.youtube.com/watch?v=4jpRFgDPWVA`
[12] `http://cq.cx/proxmark3.pl`
[13] `http://www.proxmark.org`

Our contribution is threefold: First and foremost, using the weakness of the pseudo-random generator, and given access to a particular MIFARE card, we are able to recover the keystream generated by the CRYPTO1 stream cipher, without knowing the encryption key. Secondly, we describe in detail the communication between tag and reader. Finally, we exploit the malleability of the stream cipher to read *all* memory blocks of the first sector (sector zero) of the card (without having access to the secret key). In general, we are able to read *any* sector of the memory of the card, provided that we know *one* memory block within this sector. After eavesdropping a transaction, we are always able to read the first 6 bytes of every block in that sector, and in most cases also the last 6 bytes. This leaves only 4 non revealed bytes in those blocks.

We would like to stress that we notified NXP of our findings before publishing our results. Moreover, we gave them the opportunity to discuss with us how to publish our results without damaging their (and their customers) immediate interests. They did not take advantage of this offer.

## 2.4 Background information

This section gives a short overview on developments in MIFARE Classic security to give some information on the context of this research.
For many years the MIFARE Classic card has been a cheap solution for applications where security was needed at low cost. With the announcement that CRYPTO1 was revealed in December 2007 by Nohl and Plötz [NP07] a lot of new developments on the security of the chip followed. At that time the attack deployed in this thesis was already in an advanced state [dKGHG08]. Before this announcement, in November 2007, the MIFARE Ultralight clone attack of Verdult [Ver08] was already a fact. The attack on the Dutch public transport system, with a cloned disposable card, hit the news on January 14th. The information that was disclosed by Nohl and Plötz was a very useful input for the RFID research team of the Digital Security Group in Nijmegen. The Usenix submission [NESP08] gives no information about the $f$-function (filter function). The filter function hides the internal state of the Linear Feedback Shift Register (LFSR). The Digital Security Group in Nijmegen recovered the missing crucial information and revealed the complete CRYPTO1 stream cipher together with the authentication and initialization protocols. At the same time an attack was developed that did not need a full brute-force to recover the card keys. Then, on March 10th, things went fast. Nohl developed a theoretical attack [Noh08] that uses output bits of the reader which leaks information. This attack relies on the fact that the sequence of the reader random is known and the current position of the generator in this sequence is known. To reveal this it is required to know at least one key. March 12th, the DS Group of Nijmegen demonstrates [Dig08] an implementation of a developed attack that uses a precomputed table of about half a gigabyte in size and $2^{16}$ recorded authentication trials of a reader. The announced dismantling of the algorithm has become true. After the practical demonstration the attack of Nijmegen improved due to cryptanalysis on the CRYPTO1 algorithm. Several weaknesses in the design make an attack possible that does not require any precomputation or brute-force. The current attack speed[14] is 12 keys per second. Another method is to use more general algebraic

[14] May 2008

attacks that are known for streamciphers. This algebraic method is announced by Nicolas T. Courtois et al [CNO08].
To conclude this developments so far, the algorithm is recovered and it turned out to contain serious weaknesses. The attack speed of 12 keys per second allows a realtime attack scenario.

# 3 The Mifare Classic

Contactless smartcards are used in many applications nowadays. Contactless cards are based on *radio frequency identification* technology (RFID) [Fin03]. In 1995 NXP, Philips at that time, introduced MIFARE[15]. Some target applications of MIFARE are public transportation, access control and event ticketing. The MIFARE Classic [NXP07b] card is a member of the MIFARE product family and is compliant with ISO 14443 type A up to part 3. Part 4 defines the high-level protocol and the implementation of NXP differs from the standard. Section 3.1 discusses the different parts.

## 3.1 Communication Layer

The communication layer of the MIFARE Classic card is based on the ISO 14443 standard [ISO01]. This ISO standard defines the communication for identification cards, contactless integrated circuit(s) cards and proximity cards. The standard consists of four parts.

**Part 1** Physical characteristics

**Part 2** Radio frequency interface power and signal interface

**Part 3** Initialization and anticollision

**Part 4** Transmission protocol

Part 1 describes the physical characteristics and circumstances under which the card should be able to operate.
Part 2 defines the communication between the reader and card and vice versa. The data can be encoded and modulated in two ways, type A and type B. MIFARE Classic uses type A which defines *Amplitude Shift Keying* (ASK) for reader to card communication. To encode data bits the reader stops to generate a carrier for about $2\mu$s with certain intervals. This corresponds with 100% ASK because there is no amplitude at all in this period. The card to reader communication for type A is done by load modulation. The card will add a subcarrier or not, *On-off Keying* (OOK), to encode data bits. For more detailed information about the communication on RFID we refer to the "RFID Handbook" by Klaus Finkenzeller [Fin03].
Part 3 describes the initialization and anticollision protocol. The *anticollision* is needed to select a particular card when more cards are present within the reading range of the reader. After a successful initialization and anticollision the card is in an active state and ready to receive a command. This state is the starting point for part 4 of the standard and also the point where MIFARE Classic differs from the ISO standard.
The MIFARE Classic data sheets [NXP07b] do not mention any commands that could be send on this level nor does it specify answers from the card or the length of the messages. The data sheets does define though the structure of the memory of the card and how to organize it, which is explained in Section 3.2. The modulation of commands is done by the MIFARE Classic reader chip. Knowledge about the actual modulation is therefore not needed. Note that the *PC to reader* interface *is* defined and provides commands and codes.

[15] `http://www.nxp.com`

## 3.2 Logical Structure

A MIFARE Classic card is in principle a memory card with few extra functionalities. The memory is divided in data blocks of 16 bytes. Those data blocks are grouped into sectors. The MIFARE Classic 1k card has 16 sectors of 4 data blocks each. The first 32 sectors of a MIFARE Classic 4k card consists of 4 data blocks and the remaining 8 sectors consist of 16 data blocks. Every last data block of a sector is called *sector trailer*. A schematic of the memory of a MIFARE Classic 4k card is shown in Figure 4.

Note that block 0 of sector 0 contains special data. The first 4 data bytes contain the unique identifier of the card (UID) followed by its 1-byte *bit count check* (BCC). The bit count check is calculated by successively XOR-ing the separate UID bytes. The remaining bytes are used to store manufacturer data. This data block is set and immediately locked by the manufacturer so its contents cannot longer be modified. The reader needs to authenticate for a sector

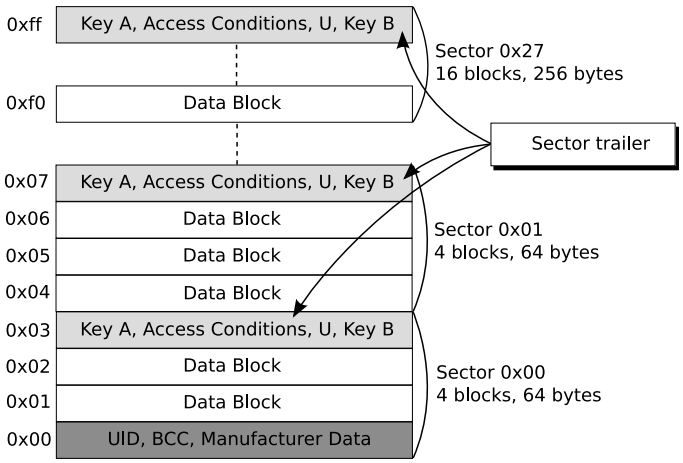



Figure 4: MIFARE Classic 4k Memory

before any memory operations are allowed. The sector trailer contains the secret keys *A* and *B* which are used for authentication. The *access conditions* define which operations are available for this sector. Depending on which key is used for authentication and the access conditions for this key, different restrictions apply to the memory operations.
The sector trailer has special access conditions. Key A cannot be read by a reader. In some configurations key B is readable. In that case the memory is just used for data storage and key B cannot be used as a key for authentication. Besides the access conditions (AC) and keys, there is one data byte (U) remaining which has no defined purpose. A schematic of the sector trailer is shown in Figure 5a. A data block is used to store arbitrary data or can be configured as a *value block*. When used as a value block a signed 4-byte value is stored twice non-inverted and once inverted. Inverted here means that every bit of the value is XOR-ed with 1. This 4 bytes are stored from the least significant byte on the left to the most significant byte on the right.

The four remaining bytes are used to store a 1-byte block address that can be used as a pointer. The address is stored twice non-inverted and twice inverted. Besides this specific format the access conditions should be configured such that the specific value block commands are allowed for this block.

## 3.3 Commands

The command set of MIFARE Classic is small. Most commands are related to a data block and require the reader to be authenticated for its containing sector. The access conditions are checked every time a command is executed to determine whether it is allowed or not. A block of data might be configured to be read only. Another example of a restriction might be a value block which can only be decremented.

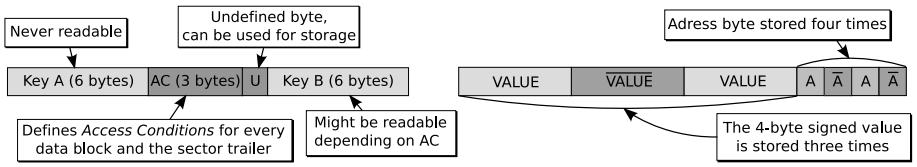
**Read and Write** The read and write commands read or write one data block. This is either a data block or a value block. The write command can be used to format a data block as value block or just store arbitrary data.

**Decrement, Increment, Restore and Transfer** These commands are only allowed on data blocks that are formatted as value blocks. The increment and decrement commands will increment or decrement a value block with a given value and place the result in a memory register. The restore command loads a value into the memory register without any change. Finally the memory register is transferred in the same block or transferred to another block by the transfer command.

## 3.4 Security Features

The MIFARE Classic card has some built-in security features. The communication is encrypted by the proprietary stream cipher CRYPTO1.

**Keys** The 48-bit keys used for authentication are stored in the sector trailer of each sector (see section 3.2). MIFARE Classic uses symmetric keys. Every sector can have to keys. At least a key A is defined and Sector key A can never be read. If key B is configured to be used for authentication, this key cannot be read either.




(a) Sector Trailer (b) Value Block

Figure 5: Block contents

### 3.4.1 Authentication Protocol

According to the MIFARE documentation [NXP07b], MIFARE Classic makes use of a mutual three pass authentication protocol that is based on ISO 9798-2. However, it turned out that this is not completely true [**?**]. In this paper we only use the first initial nonce that is send by the card. So the exact authentication protocol used does not matter. The reader sends a request for sector authentication and the card will respond with a 32-bit nonce $N_C$. Then, the reader sends back an 8-byte answer to that nonce which also contains a reader random $N_R$. This answer is the first encrypted message after the start of the authentication procedure. Finally, the card sends a 4-byte response. As far as our attack is concerned this description captures all the necessary information.

## 3.5 Mifare Higher Level Protocol

To find out what the MIFARE Classic communication looks like we made traces of transactions between MIFARE readers and cards. In this way, we gathered many traces which gave us some insights on the high-level protocol of MIFARE Classic. In this section we explain a trace we recorded as an example, which is shown in Figure 6. This trace contains every part of a transaction. We will refer

| ETU | SEQ | sender | | |
|---|---|---|---|---|
| 0 : | 01 : | PCD | 26 | Anticollision |
| 64 : | 02 : | TAG | 04 00 | |
| 12097 : | 03 : | PCD | 93 20 | |
| 64 : | 04 : | TAG | 2a 69 8d 43 8d | |
| 16305 : | 05 : | PCD | 93 70 2a 69 8d 43 8d 52 55 | |
| 64 : | 06 : | TAG | 08 b6 dd | |
| 16504 : | 07 : | PCD | 60 04 d1 3d | Authentication |
| 112 : | 08 : | TAG | 3b ae 03 2d | |
| 6952 : | 09 : | PCD | c4! 94 a1 d2 6e! 96 86! 42 | |
| 64 : | 10 : | TAG | 84 66! 05! 9e! | |
| 396196 : | 11 : | PCD | a0 61! d3! e3 | Increment & Transfer |
| 208 : | 12 : | TAG | 0d | |
| 8442 : | 13 : | PCD | 26 42 ea 1d f1! 68! | |
| 5120 : | 14 : | PCD | 8d! ca cd ea | |
| 2816 : | 15 : | TAG | 06! | |
| 1349238 : | 16 : | PCD | 2a 2b 17 97 | Read |
| 72 : | 17 : | TAG | 49! 09! 3b! 4e! 9e! 5e b0 06 d0! | |
| | | | 07! 1a! 4a! b4! 5c b0! 4f c8! a4! | |

Figure 6: Trace of a card with default keys, recorded by the Proxmark III

to the sequence number (SEQ) of the messages we discuss. The messages from the reader are shown as PCD (Proximity Coupling Device) messages and from the card as TAG messages. The time between messages is shown in Elementary Time Units (ETU). One ETU is a quarter of the bit period, which equals 1.18$\mu$s. The messages are represented in hexadecimal notation. If the parity bit (which is not explicitly shown in the trace) of a byte is incorrect, this is shown by an exclamation mark. We will discuss only the most significant messages.

**Anticollision** The reader starts the SELECT procedure. The reader sends `93 20` (#3), on which the card will respond with its unique identifier (#4). The

reader sends `93 70` followed by the UID and two CRC bytes (#5) to select the card.

**Authentication** The card is in the active state and ready to handle any higher layer commands. In Section 3.4.1 we discussed the authentication protocol. In Figure 6, messages #7 to #10 correspond to authentication.
The authentication request of the reader is `60 04 d1 3d` (#07). The first byte `60` stands for an authentication request with key A. For authentication with key B, the first byte must be `61`. The second byte indicates that the reader wants to authenticate for block 4. Note that block 4 is part of sector 1 and therefore this is an authentication request for sector 1. The last two bytes are CRC bytes.

**Encrypted Communication** After this successful authentication the card is ready to handle commands for sector 1. The structure of the commands can be recognized clearly. Since we control the MIFARE Classic reader we knew which commands were send. Message #11 to #15 show how an increment is performed. The increment is immediately followed by a read command (#16 and #17).

The MIFARE Classic commands of the higher level protocol consist of 4 bytes of the form `XX YY ZZ ZZ`. The first byte `XX` indicates the command type. The second byte `YY` indicates the memory address on which the command should be executed. A command is not always related to a specific memory address. The halt command (`50 00 57 cd`) illustrates this. The last two bytes `ZZ ZZ` are CRC bytes.

# 4 Hardware

An RFID system consists of a transponder (card) and a reader [Fin03]. The reader contains a radio frequency module, a control unit and a coupling element to the card. The card contains a coupling element and a microchip. The control unit of a MIFARE Classic enabled reader is typically a NXP microchip (e.g. RC500, RC632) with a closed design. This microchip communicates with the application software and executes commands from it. Note that the actual modulation of commands is done by this microchip and not by the application software. The design of the microchip of the card is closed and so is the communication protocol between card and reader. We want to evaluate the security properties of the MIFARE system. Therefore we need hardware to eavesdrop a transaction (Figure 7). It should also be possible to act like a MIFARE reader to communicate with the card.

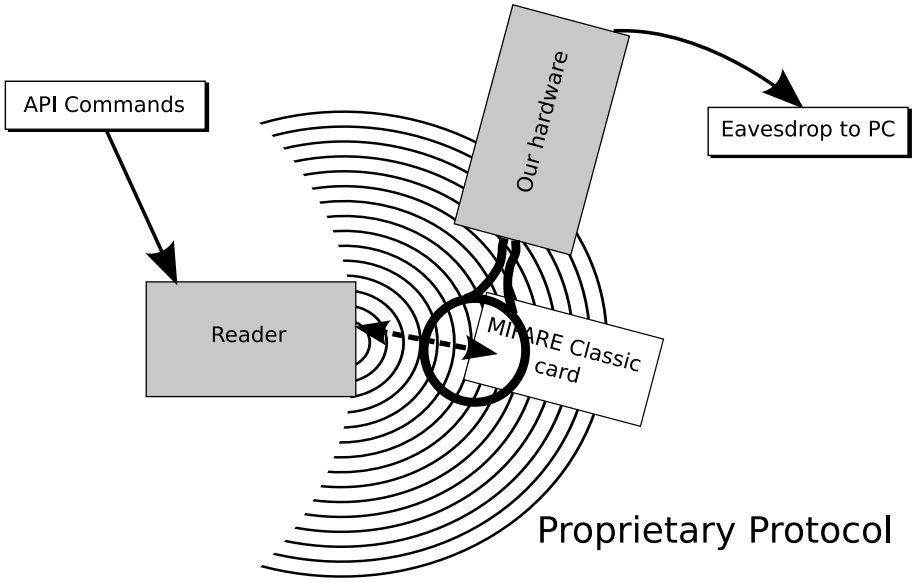



Figure 7: Experimental Setup

**Available Hardware** For many years there were no open source tools available on RFID technology. The RFID readers of the manufacturers allow a limited set of commands. It was not possible to see the RF communication with low-cost tools. Therefore, it was not possible to perform protocol analysis at the lowest level. Recently there have been developments on open source devices. There is a project called the OpenPCD project. Furthermore, there is the Proxmark III that is available in the public domain. The University of Nijmegen developed the Ghost. There are also other projects where hand-build devices are used such as the Mole [Han05] and the Fake Tag [Kas06].

## 4.1 Ghost

As the name of the device suggests, the Ghost is a device which is capable to act as an RFID card. The hardware was developed by Peter Dolron of the University of Nijmegen. At the start of this research Roel Verdult was still busy with the firmware of the device. It was not ready yet for use in protocol analysis.

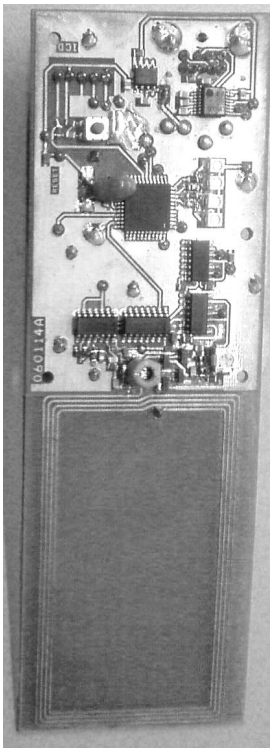


Figure 8: Ghost

On January 14th, it was this device that impersonated a disposable card of the Dutch public transport system [Ver08]. This was a MIFARE Ultralight card that was used on a test location in Rotterdam. No encryption is involved with this card, so all card content is readable. Since the memory organization and the few commands are explained in the NXP product specification [NXP07a], it was possible to implement this functionality on the Ghost. The manufacturer guarantees that every card has its own unique identifier. This identifier cannot be changed. With the Ghost this identifier can be easily spoofed. This means that a reader can not completely relay on the identifier to authenticate a card. And even worse, the memory content of the Ghost can be brought back to earlier states, time after time. So the locking mechanism [NXP07a], which was meant to prevent abuse, unfortunately fails.

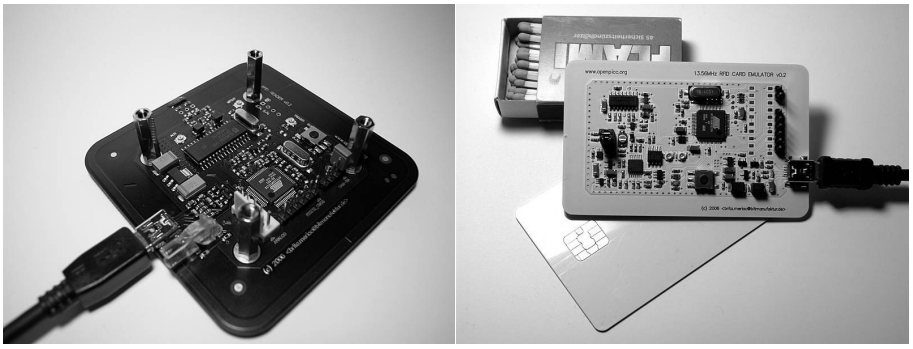
The lack of authentication mechanisms on the MIFARE Ultralight card makes it possible to read out a card. A complete memory dump can be made at any time. Even if the card is blocked, all memory is still readable. The attacker 'steals' the card from another traveler without any constraints except that he is close enough to read the card. There are some countermeasures possible which are discussed in Section 7. But in case of this disposable card there is little that can help. Card blocking might be a solution, but one of little comfort to the customer.

## 4.2 OpenPCD and OpenPICC

The OpenPCD project was started by Harald Welte[16] to develop a reader that was able to do more than following the fixed possibilities of the manufacturer. It has an on-board MIFARE enabled controller (RC632), and at the same time

[16] `http://www.openpcd.org`

allows to by-pass this chip to modulate any arbitrary message.
Next to the OpenPCD there is also the OpenPICC. This is a card emulator



(a) OpenPCD (b) OpenPICC

Figure 9: OpenPCD

which has the ability to sniff communication. We used the OpenPICC at the start of this research in order to gather samples from the MIFARE Classic readers in the building. It was not possible to use the OpenPICC as an emulator. At the time of this writing the OpenPICC is still under development.

We switched to another project because the hardware was not fully operational and we wanted to sniff the communication in both directions.

## 4.3 Proxmark III

The Proxmark III is a device developed by Jonathan Westhues. Its design is very useful for RFID testing and research. All the needed information about the hardware has been made public in May 2007. The firmware has also been made public under a General Public License[17]. Although it had no support for ISO 14443-A its design allows to implement this in the firmware. In this section we will discuss the separate components of the Proxmark and their contribution to a flexible design. It is possible to adjust the Digital Signal Processing to support a specific protocol.
This device supports both low frequency (125 kHz-134 kHz) and high frequency (13.56 MHz) signal processing. This is achieved by implementing two parallel antenna circuits that can be used independently. Both circuits are connected to a 4-pin Hirose connector which functions as an interface to an external loop antenna. For the purpose of acting like a PCD or reader it is possible to drive the antenna coils with the appropriate frequency. This is not needed when the Proxmark is used for sniffing or when it emulates a card. In that case the field is generated by a reader.
The signal from the antenna is routed through a Field Programmable Gate Array (FPGA). This FPGA relays the signal to the microcontroller and can be used to perform some filtering operations before relaying. The software

[17]Hardware design and software is publicly available at `http://cq.cx/proxmark3.pl`

implementation allows the Proxmark to eavesdrop communication between an RFID tag and a reader, to emulate a tag and to emulate a reader.

Despite the basic hardware support for these operations the actual processing of the digitized signal and (de)modulation needs to be programmed for each specific application. The physical layer of the MIFARE Classic card is implemented according to the ISO 14443 type A standard [ISO01]. We had to implement the ISO14443-A functionality since it was not yet implemented. This means we had to program both processing and generation of reader-to-tag and tag-to-reader communication in the physical layer and higher level protocol. To meet the requirements of a replay attack we added the functions 'hi14asnoop' to make traces, 'hi14areader' to act like a reader and 'hi14asim' to simulate a card. We added the possibility to send 'wrong' parity bits. This was necessary because we needed to be able to act like a real MIFARE Classic reader during encrypted communication.

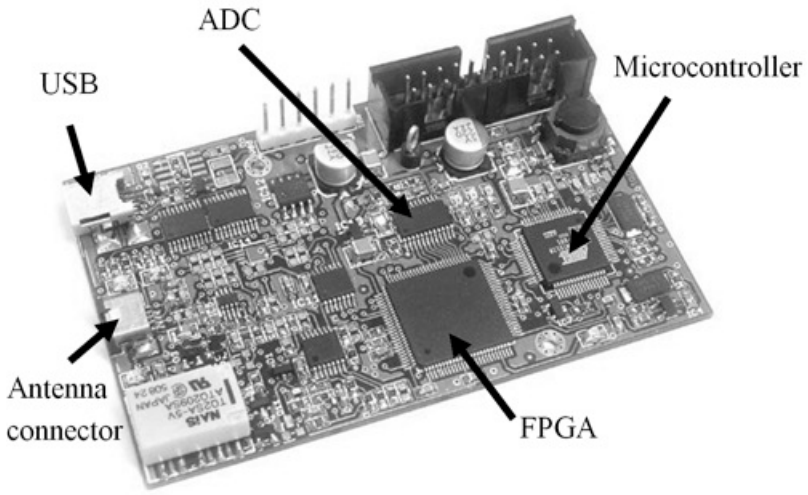



Figure 10: The Proxmark III

**Analog to Digital Converter** The analog signal that comes from the antenna circuit is fed into an 8-bit Analog to Digital Converter (ADC). This delivers 8 output bits in parralel which represent the current voltage retrieved from the field.

**Field Programmable Gate Array** The 8 output pins from the ADC are connected to 8 pins of the Field Programmable Gate Array (FPGA). An FPGA has a great advantage over a normal microcontroller in the sense that it emulates hardware. A hardware description can be compiled and flashed into an FPGA. Because basic arithmetic functions can be performed fast and in parallel by an FPGA it is faster than an implementation on a normal microcontroller. Only a real hardware implementation would be faster but this lacks the flexibility of an FPGA.

**Microcontroller** The microcontroller is responsible for the protocol part. It receives the digital encoded signals from the FPGA and decodes them. The decoded signals can just be copied to a buffer in the EEPROM memory. Additionally, an answer to the received message can be send by encoding a reply and communicating this to the FPGA.

# 5 Software

In this section we discuss the developed software to support ISO 14443A on the Proxmark. The Proxmark is able to operate in three modes. These modes are the *sniffing mode*, the *card emulation mode* and the *reader mode*. A few requirements need to be fulfilled to implement these functions. First, we need an underlying physical layer which takes care of the Digital Signal Processing (DSP), this is implemented in the FPGA. Next, the modes of operation should be implemented as functions on the microcontroller. Finally, a client should be able to call the functions and display the results. This leads to a rough division of the software into three parts:

- Client software, calls the functions implemented on the Proxmark, responsible for representation of the results. The client can be seen as the application layer in the communication. It makes use of the underlying protocol to receive information about an RFID card.
- Microcontroller software, implements the different modes of operation. This is done by defining which protocol messages should be sent in which format and in what order. This can be seen as the transport layer of the communication.
- FPGA software, is responsible for the DSP and therefore responsible for the physical layer of the communication.

Figure 11 shows the different components of the Proxmark application and their different responsibilities. The processing and generation of the protocol messages is partly done by the FPGA and partly by the Microcontroller (ARM). The FPGA mainly does the edge detection and communicates to the ARM whether the signal was high or low. The ARM then tries to decode the retrieved bit stream using Manchester or Modified Miller. For generation of messages the ARM will send a bit stream to the FPGA that represents the Manchester or Modified Miller encoded message. The FPGA will modulate according to this bit stream. The design choice to split this DSP in two parts was mainly because of the limited capacity of the FPGA. It has not enough space to do flash a design that does the signal processing and message decoding/encoding at the same time. The next subsections are about the way different communication modes are implemented in each different component.

## 5.1 Client

The client application is written by Jonathan Westhues [Wes] and connects to the Proxmark via the standard HID protocol. The operating system on the ARM does not represent a proper real-time operating system in the sense that it still polls for things like USB packets. This way it is not possible to stream the retrieved samples on real-time to the PC. So when the ARM retrieves a command from the client it runs this command and stores any useful messages in its memory buffer. After the command finishes the client should send a new

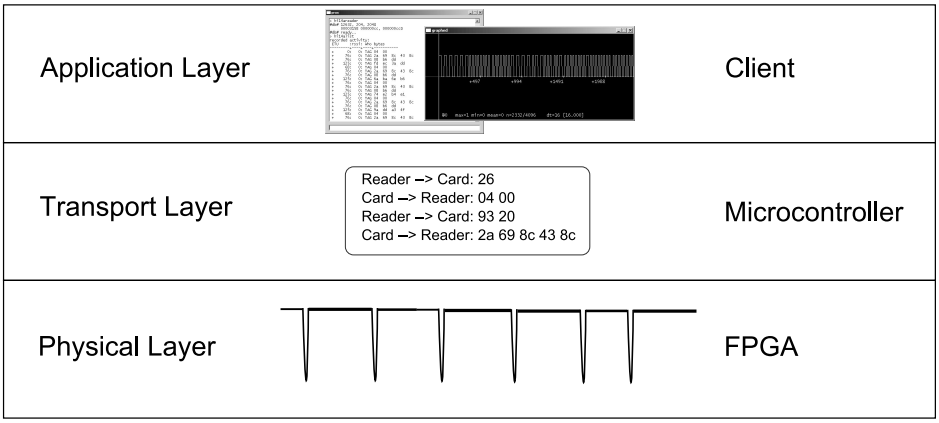



Figure 11: Proxmark Communication Layers

command to capture the data from the ARM buffer.
The Windows client has been extended with four more commands:

- **hi14asim**
  Simulate an ISO 14443-A tag. In the code I simulate the anticollision of a Mifare Classic 4K card. But this can be changed by programming different response messages on the Proxmark.

- **hi14areader**
  Act like a reader. The Proxmark generates a field and uses 100% ASK and Modified Miller encoding to communicate with a card or tag. The answers of the card are stored in a buffer (BigBuf) on the Proxmark and can be downloaded by hi14alist.

- **hi14asnoop**
  Sniff the communication between a reader and a tag. The communication from both directions is captured and stored in a buffer (BigBuf) on the Proxmark and can be downloaded by hi14alist.

- **hi14alist**
  With this command any data captured by hi14areader or hi14asnoop can be downloaded and displayed in the Windows client.

**Client: Sniffing** The command **hi14asnoop** starts sniffing the communication between a reader and card until the buffer (BigBuf) is full. In Figure 12 the result of the sniffing is captured by executing **hi14alist**. The result shows the repeated anticollision loop. This indicates that the reader is just polling if the card is still there. This anticollision loop walks through the following states:

- Reader: `26` ⟶ Card: `04 00`
  The reader sends an REQA (Request for type A) message on which the card responded with its type.

- Reader: `93 20` ⟶ Card: `2a 69 8c 43 8c`
  The reader starts the anticollision to select a card and requests its UID. The card responded with its UID. The last byte is the so-called BCC[18] and is the result of XOR-ing the first four UID-bytes.

- Reader: `93 70 2a 69 8c 43 8c` ⟶ Card: `08 b6 dd`
  The reader selects the desired card by sending `93 70` followed by the UID of the card. If the card is successfully selected it will response with a SAK[19] which means that the card is now ready to handle commands of the higher layer protocol. The SAK consists of one byte (`08`) that indicates the card type and is followed by two CRC bytes.

**Client: Card to Reader Communication** The command **hi14asim** sets the Proxmark in emulation mode. Until the button is pressed the device will respond as programmed in the firmware. We programmed the Proxmark to act like a MIFARE Classic 4k card. The reader (Omnikey 5121) runs the anticollision and gets convinced that it communicates with an MIFARE Standard 4k[20] card and shows this in the client application of the reader. In Figure 13 the Proxmark client and the third-party client are shown in one screenshot.

**Client: Reader to Card Communication** The command **hi14areader** lets the Proxmark act like a reader. This means that the Proxmark drives the antenna coils to generate a field. The field is removed with certain intervals of $2\mu$s (as explained in Section 5.3) to communicate with a card. The Proxmark switches between sending and listening. In Figure 14 only the anticollision phase is executed and only the messages from the card (TAG) are stored.

## 5.2 Microcontroller

The microcontroller (ARM) implements the transport layer. First it decodes the samples received from the FPGA. These samples are stored in a Direct Memory Access (DMA) buffer. The samples are binary sequences that represent whether the signal was high or low. The software on the ARM tries to decode these samples. When the Proxmark is in *sniffing mode* this is done for both the Manchester and Modified Miller at the same time. Whenever one of the decoding procedures returns a valid message, this message is stored in another buffer (**BigBuf**) and both decoding procedures are set to an unsynced state. The **BigBuf** is limited to the available memory on the ARM. In our research we reserved about 2 KB of memory for the traces (Besides the traces the buffer also stores some temporary data that is needed in the processing).
When the BigBuf buffer is full the function normally returns. A new function call from the client is needed to download the BigBuf contents to the computer. The BigBuf is especially useful for protocol investigation. Every single message is stored in this buffer. When a card is emulated or when the Proxmark is used as a reader the BigBuf can be used to store status messages or protocol exceptions.

[18] bit count check
[19] Selection Acknowledged
[20] 'Classic' and 'Standard' are both used for the same product

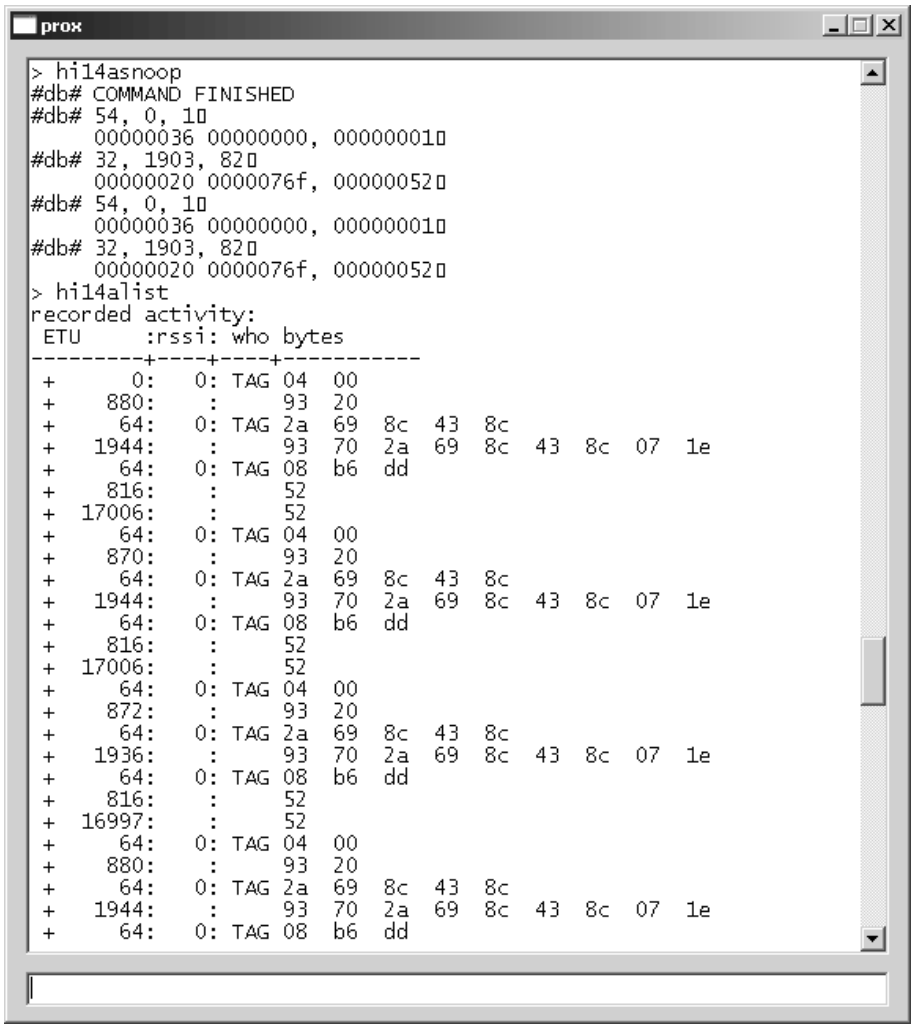



Figure 12: hi14asnoop

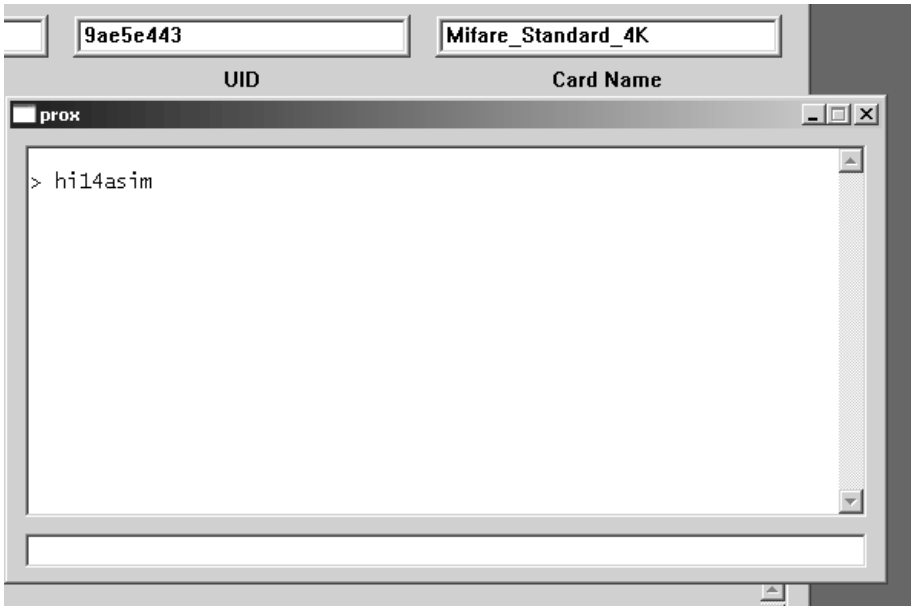



Figure 13: hi14asim

## 5.3 FPGA

The FPGA can be seen as dynamic hardware. It is possible to make a hardware design and flash it into the memory of the FPGA. This gives some major advantages:

- 'Hardware' errors can be corrected, the FPGA can be flashed with a new hardware design.
- Although not as fast as a real hardware implementation, an FPGA is faster than its equivalent on a microprocessor. That is, it is specialized for one job.

The FPGA has two main tasks. The first task is to demodulate the signal received from the ADC and relay this as a digital encoded signal to the ARM. Depending on the task this might be the demodulation of a 100% Amplitude Shift Keying (ASK) signal from the reader or the load modulation of a card. The encoding schemes used to communicate the signal to the ARM are Modified Miller for the reader and Manchester encoding for the card signal. These encoding schemes are explained further on in Section 5.3.2.
The second task is to modulate an encoded signal that is received from the ARM into the field of the antenna. This can be both the encoding of reader messages or card messages. For reader messages the FPGA generates a electromagnetic field on `pwr_hi` and drops the amplitude for short periods. A complete overview of the IO pins of the FPGA is shown in Table 1.

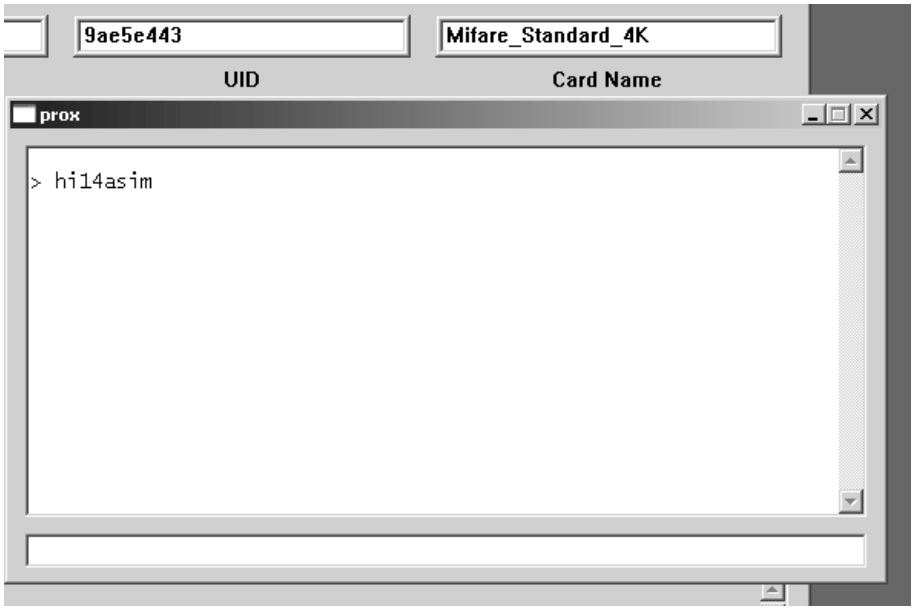



Figure 14: hi14areader

### 5.3.1 Verilog

The hardware design that can be flashed into the FPGA is written in Verilog. Verilog is a hardware description language which allows to describe a hardware design in a C-style syntax.

### 5.3.2 FPGA Modes

The FPGA module can operate in different modes to allow reader and card modulation and deliver useful samples to the microcontroller. The different modes of operation are shown by Table 2. For eavesdropping we use `SNIFFER` which delivers samples of the communication of both directions. To emulate a card we alternately use `TAGSIM_LISTEN` for retrieving and `TAGSIM_MOD` for sending messages. For reader emulation the alternating modes are `READER_LISTEN` for retrieving and `READER_MOD` for sending messages.

**FPGA: Sniffing** When the FPGA is in sniffing mode it tries to demodulate both the reader signal and the card signal simultaneously. The samples are communicated over `ssp_din` to the ARM.

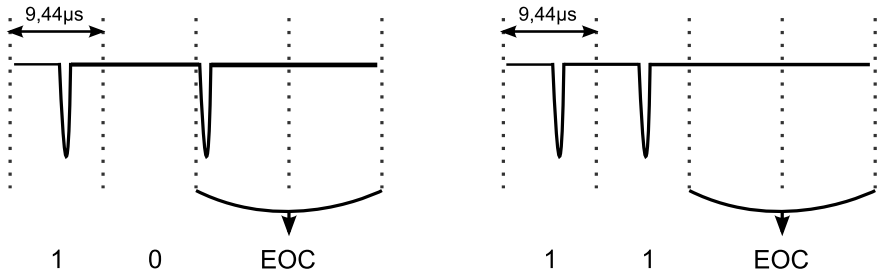
**FPGA: Card to Reader Communication** The Frame Delay Time (FDT) defines when a card is expected to answer. The FDT is given in periods of the carrier wave[21]. Remember that one bit period in the communication between reader and card takes 128 periods of the carrier wave. It is important, especially in the anticollision phase, that a card answer is calibrated to the timing

[21] 13.56MHz in case of ISO 14443

| hi_iso14443a.v | | |
|---|---|---|
| Input | | |
| pck0 | P36 | not used |
| ck_1356meg | P91 | assigned to adc_clk to get the speed of incoming ADC samples |
| ck_1356megb | P93 | same source as ck_1356meg |
| [7 : 0] adc_d | ... | Range of pins that come from the ADC |
| ssp_dout | P34 | communication from ARM |
| cross_hi | P88 | not used (not connected in FPGA design) |
| cross_lo | P87 | low frequency application |
| [2 : 0] mod_type | - | no physical connector |
| reset | - | no physical connector |
| Output | | |
| pwr_lo | P81 | to generate a low frequency field |
| pwr_hi | P80 | to generate a high frequency field |
| pwr_oe1 | P82 | to modulate a subcarrier |
| pwr_oe2 | P83 | only for low frequency |
| pwr_oe3 | P84 | always off |
| pwr_oe4 | P86 | to modulate a subcarrier |
| adc_clk | P46 | the clock of 13.56 MHz |
| ssp_frame | P31 | frame clock of communication to/from ARM |
| ssp_din | P32 | communication to arm |
| ssp_clk | P71 | bit clock for communication to/from ARM |
| dbg | P22 | debug pin, any arbitrary signal can be asigned |

Table 1: I/O of ISO 14443 type A module

grid of the reader. The reader generates the field and therefore determines the clock of the card. The reference point for the card is the last dip in the reader communication. If the last bit of the reader message is a zero, the modulation will look like Figure 15a. If the last bit is a one the modulation will look like Figure 15b.




(a) Last bit is zero (b) Last bit is one

Figure 15: Last bit in reader communication

The last two bit periods construct the End of Communication. The End of Communication (EOC) terminates the data frame of the reader. Depending on the last bit in communication (zero or one) the EOC contains a dip or not. Notice that the difference between the two dips is half a bit period. This is 64

| Modes of operation | | |
|---|---|---|
| **Binary** | **Mode** | **Generates field** |
| **000'b** | **FPGA_HF_ISO14443A_SNIFFER** | **No** |
| Reports samples to ARM that contain both reader → card modulation and card → reader modulation. | | |
| **001'b** | **FPGA_HF_ISO14443A_TAGSIM_LISTEN** | **No** |
| Reports samples to ARM of reader → card modulation. | | |
| **010'b** | **FPGA_HF_ISO14443A_TAGSIM_MOD** | **No** |
| Modulates the signal from the ARM into the reader field. | | |
| **011'b** | **FPGA_HF_ISO14443A_READER_LISTEN** | **Yes** |
| Reports samples to ARM of card → reader modulation. | | |
| **100'b** | **FPGA_HF_ISO14443A_READER_MOD** | **Yes** |
| Modulates the signal from the ARM into the field. | | |

Table 2: Modes of operation

periods of the carrier wave and the FDT needs to be corrected for this.

$$\texttt{FDT} = 128n + 64i + 20 \tag{1}$$

The FDT is defined in Equation 1 where $n$ is the number of periods and $i$ is the last bit in reader communication, $i \in \{0, 1\}$. In the anticollision phase it is required that all cards within the field will react at exactly the same time. This allows the reader to detect collisions. Therefore $n = 9$ in the anticollision phase. For all other reader messages $n \geqslant 9$. This means that an answer of the card should always be aligned to the bit period of the reader[22].

When the Proxmark emulates a card, the FPGA mode is set to `TAGSIM_LISTEN` when listening and `TAGSIM_MOD` when modulating an answer. To align the answer of the card to the bit period the last dip in the reader communication needs to be detected. The FPGA resets the value of `fdt_counter` after a dip detection in the `TAGSIM_LISTEN` mode. When `fdt_counter` reaches a certain value[23] the `fdt_elapsed` register is set to one.
The microcontroller sends its answer that has to be modulated into the field to the FPGA. This answer cannot be modulated into the field when $n < 9$. Therefore the FPGA buffers the received bits from the ARM into a 48-bit buffer (See Figure 16). The pointer `ptr` points to the first binary one that enters the buffer. Every 1,14$\mu$s the bits in the buffer shift one position to the left. The `ptr` keeps pointing to the first binary one that was received. The new value entering the buffer comes from `ssp_dout` which is a pin (See Table 1) directly connected to the ARM.
The `ssp_clock` makes sure that every 1,14$\mu$s the ARM sends another encoding bit. The modulated signal `mod_signal` depends on the contents of the FDT buffer and whether the minimal FDT elapsed. This is illustrated in Figure 16 by an AND-gate.
In the anticollision phase it is important for the emulation to be quick and answer on $n = 9$. However, in practice some readers still accept anticollision

[22] every 128 cycles of the carrier
[23] close to 1172 which is the FDT for $n = 9$ and $i = 0$

messages with $n > 9$. The software on the ARM is quick enough to modulate answers on $n = 9$ for the anticollision phase. When more complex computations are needed the ARM is not able to react on $n = 9$. This means that the FDT buffer remains empty (contains only zeros). When the value of `fdt_elapsed` turns to one when the minimal FDT has expired the first binary one (encoding bit) from the ARM will be modulated into the field immediately. We do not want this because the signal needs to be aligned with the bit grid of the reader (a bitperiod starts every 8 encoding bits). So when the buffer is still empty at $n = 9$ the value of `ptr` starts cycling from 7 to 0[24] and gets fixed as soon as a binary one enters the buffer. This way every answer of the ARM will be correctly aligned by the FPGA to the bit grid of the reader (no matter how much time the ARM needs to prepare its answer).

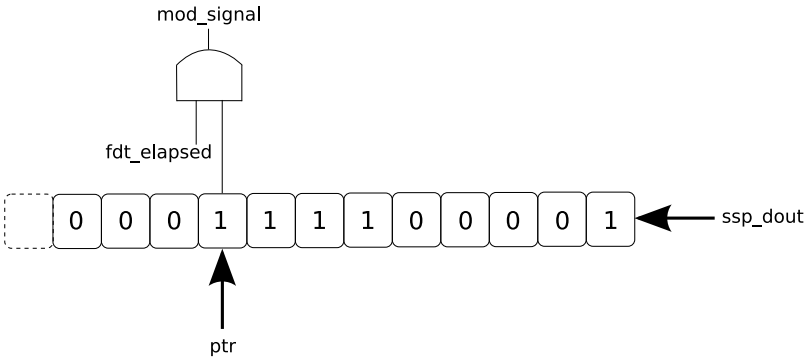



Figure 16: FDT Buffer

**Manchester** The signal from card to reader is Manchester encoded. The bit period is split into a first half and second half. A binary one is encoded by a high first and a low second half. A zero is encoded the opposite way with a low first and a high second half. The first few bits of the Manchester signal in Figure 17 are decoded as 100100000'b. The Manchester encoding is com-

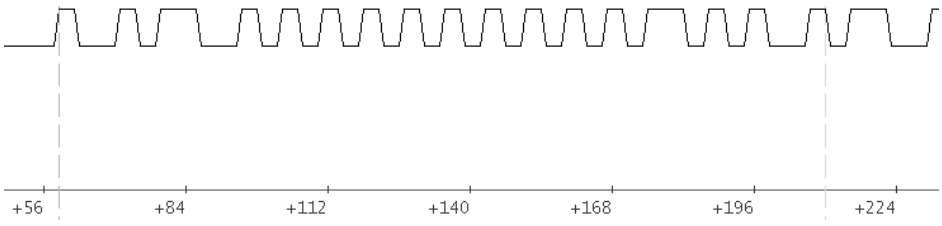



Figure 17: Manchester Encoded Signal

pletely done by the ARM. The ARM prepares a bitstring that is communicated to the FPGA over `ssp_dout`. One bit period is represented by 8 bits. The FPGA has no more responsibility than the load modulation according to this

[24] bit 0 is the rightmost bit in the buffer

bit string. This is done by modulating a subcarrier in the field. This subcarrier is $f_{subc} = f_c/16$ where $f_c = 135600$kHz and thus the subcarrier frequency is 847.5kHz. On every negative edge of the `adc_clk` the `negedge_cnt` register gets incremented. Multiplying or dividing a binary number by 2 means that the bits in the register shift to the left or right respectively. A division of $f_c$ by 16 equals $2^4$, this means we can use the 4th bit of `negedge_cnt` to modulate $f_{subc}$. The `negedge_cnt` register gets incremented on every negative edge of

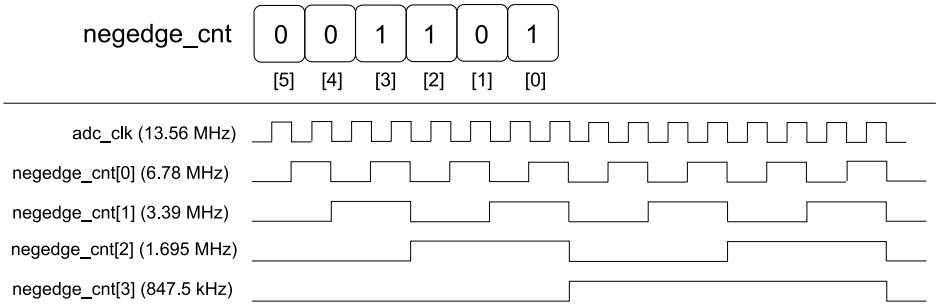



Figure 18: How to generate a 847.5kHz clock

the `adc_clk` which is running on 13.56MHz. So the `negedge_cnt[0]` bit has a clock of 13.56MHz divided by two already. In Figure 18 the generated signals for the first four bits of `negedge_cnt` are shown. `negedge_cnt[3]` delivers the requested 857.5kHz signal.

```
assign modulating_carrier = (mod_sig_coil &
       negedge_cnt[3] & (mod_type == 3'b010));
```

The `modulating_carrier` on its turn is assigned to `pwr_oe4`[25]. The `mod_sig_coil` relays the Manchester encoded signal. The `mod_sig_coil` value is related to `ssp_dout` by the `fdt_buffer` (See Figure 16).

**FPGA: Reader to Card Communication** The FPGA is able to relay the bitstream received on `ssp_dout` to `pwr_hi`. The modulation technique used is 100% ASK. The used encoding scheme is Modified Miller which will be explained later on in this section.

```
assign pwr_hi = (ck_1356megb &
       (((mod_type == 3'b100) & ~mod_sig_coil) ||
       (mod_type == 3'b011)));
```

Here `ck_1356megb` is a 13.56MHz clock that is assigned to `pwr_hi`. It will only be assigned when the FPGA is in reader mode (3'b100) and there is no dip ($\sim$`mod_sig_coil`) or the FPGA is in listening for card mode (3'b011). The `mod_sig_coil` value is related to `ssp_dout` by the `fdt_buffer` (See Figure 16).

**Modified Miller** The signal from reader to card is encoded by the Modified Miller encoding scheme. This scheme basicly consists of three ways to

[25] `pwr_oe4` is connected to the antenna

represent a bit in a given bitperiod. The electromagnetic field is removed completely for 2,28$\mu$s. The card uses the power from the field to operate but is able to come over this short drop because of a capacitor. In the ISO14443 [ISO01] standard the three different bitperiod representations are denoted X, Y and Z. The first bitperiod in Figure 19 is of type X, the second of type Y and the third of type Z. In short, to encode a one just drop the signal after half a bitperiod and to encode a zero do not drop the signal at all unless the elapsed time after the last drop is a bitperiod or more. If the latter is the case then drop at the beginning of the bitperiod. Just like the Manchester encoded signal also the

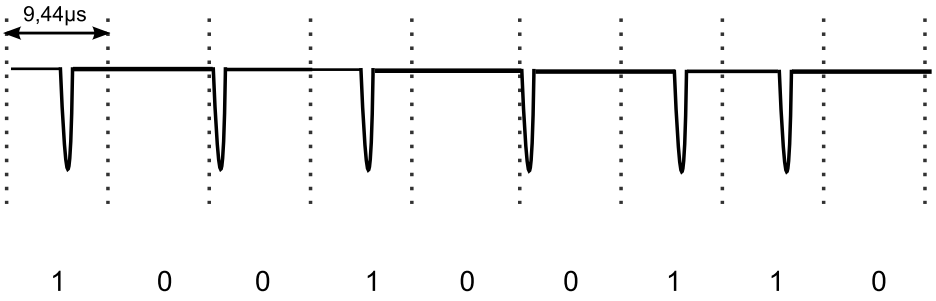



Figure 19: Modified Miller Encoded Signal

Modified Miller message is bitstring encoded by the ARM. The preparation of the desired message is completely done by the ARM and the FPGA just routes the incoming signal from the ARM in the right time interval and uses it to drive the antenna coil or not.

# 6 Case studies

Nohl and Plötz [NP07] discovered that the pseudo-random generator, used to generate the nonces in the authentication, is weak. During our experiments, independently, we also discovered this weakness of the pseudo-random generator by requesting many nonces from the card, at arbitrary times. This experiment showed that a 'random' nonce repeats a few times per hour. This is just by chance because Nohl and Plötz discovered that the nonce is generated by an *Linear Feedback Shift Register* (LFSR) which shifts every 9.44$\mu$s. This is exactly one bit period in the communication. Therefore a random nonce could theoretically reappear after 0.618s, if the card is queried at exactly the right time.

Without knowing the cryptographic algorithm, only an online brute force attack can be mounted, trying all possible keys in an actual authentication run between a reader and a card. Because of the communication delay, this would take 5ms for each attempt. An exhaustive key search would then take 16,289,061 days, which equals about 44,627 years.
When the cryptographic algorithm is known, an off-line brute force attack can be mounted using a few eavesdropped traces of an authentication run. Nohl and Plötz state that with dedicated hardware of around $17.000 this would take about 1 hour. For this attack to work, some known plaintext is required. Our analysis provides this plaintext.

It is however possible to attack the MIFARE Classic in another way, that does not require recovering the key. This attack, that we describe here, uses the weakness of the pseudo-random generator to recover the keystream.

## 6.1 Attacks on MIFARE

There are several attacks possible on MIFARE Classic. The first practical attack was carried out by us in [dKGHG08] and was the topic of this master studies, so will be explained in detail further on. This attack recovers used keystream in a transaction between a reader and a card. Due to a weak pseudo-random generator the same keystream can be used twice. For this attack the secret key remains unknown and is not needed. Also the algorithm used can be unknown. Other attacks have the secret key as target and try to recover the key. One way to do this is a brute-force attack. Until now a brute-force attack could only be performed on-line against a card because the CRYPTO1 algorithm was secret. The algorithm is recovered partially by reverse engineering of the hardware chip [NP07] and the rest could be revealed by protocol analysis [Dig08, NESP08]. An off-line attack is now one of the possibilities but still takes a lot of time (one week per key) or is expensive ($17.000) [NP07]. Cryptanalysis showed that there are weaknesses in the design of the algorithm which make it possible to recover 12 keys per second [GdKGM$^+$08]. Also algebraic attacks are possible. In this section we visit the different attacks known so far.

### 6.1.1 Keystream Recovery Attack

In this section we develop a method to recover the keystream that is used in an earlier recorded transaction between a reader and a card. As a result the keystream of the communication will be recovered. For this attack we need to be in possession of the card. The following reasons make this attack interesting:

1. Our attack provides the known plaintext necessary to mount a brute force attack on the key.
2. Using our attack we recovered details about the byte commands.
3. Using the recovered key stream we can *read* card contents without knowing the key.
4. Using the recovered key stream we can also *modify* the contents of the card without knowing the key.

**Keystream Recovery** To recover the keystream we exploit the weakness on the pseudo-random generator (Figure 20). We also use the fact that this random nonce in combination with only one valid response of the reader determines the continuation of the keystream. This allows us to replace for example the address byte of a read command as shown on the bottom of the diagram in Figure 20. For this attack we need complete control over the reader (Proxmark) and access to a (genuine) card. The attack consists of the following steps:

1. Eavesdrop the communication between a reader and a card. This can be for example be in an access control system or public transport system.
2. Start a new communication with the same card, but now using the Proxmark. Make sure that the card will use the same keystream as in the recorded communication. This is possible because the card repeats the same nonce in reasonable time, and we completely control the reader.
3. Modify the plaintext, such that the card receives a command for which we know plaintext in the response (e.g., by changing the block number in a read command).
4. For each segment of known plaintext, compute the corresponding keystream segment.
5. Use this keystream to partially decrypt the trace obtained in 1.
6. Try recovering more keystream bits by shifting commands.

The plaintext $P_1$ in the communication is XOR-ed bitwise with a keystream $K$ which gives the encrypted data $C_1$.

It should not be possible to control the initialization of the stream cipher in a way that it generates the same key stream again. Because of the weak pseudo-random generator it is possible to retrieve the same card challenge over and over again. When it is possible to use the same keystream on a different plaintext $P_2$ and either $P_1$ or $P_2$ is known, then both $P_1$ and $P_2$ are revealed.

$$\left.\begin{array}{l} P_1 \oplus K = C_1 \\ P_2 \oplus K = C_2 \end{array}\right\} C_1 \oplus C_2 \Rightarrow P_1 \oplus P_2 \oplus K \oplus K \Rightarrow P_1 \oplus P_2 \qquad (2)$$

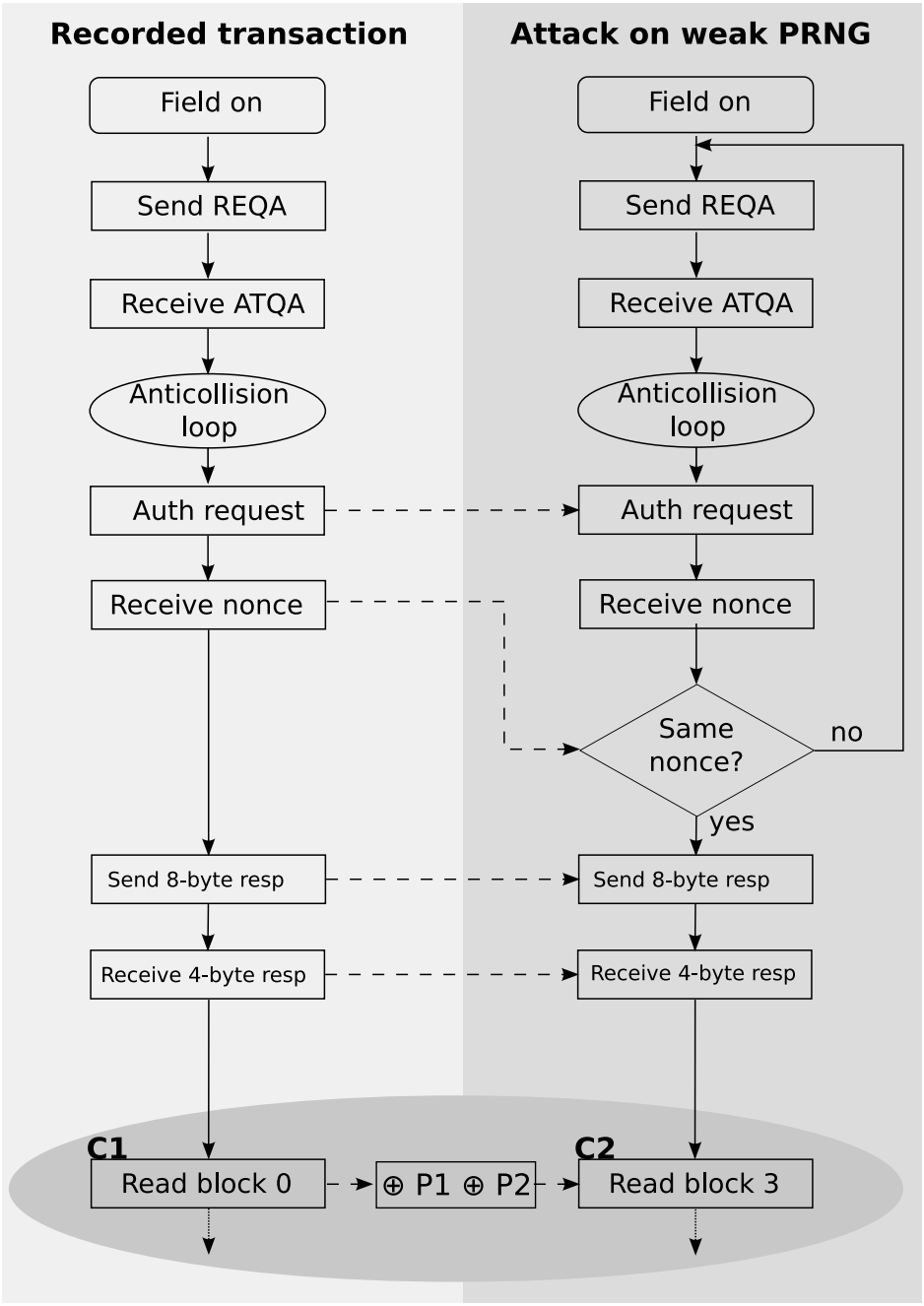



Figure 20: Exploit for the pseudo-random generator

The weak pseudo-random generator makes it possible to replay an earlier recorded transaction. We can flip ciphertext bits to try to modify the first command such that it gives another result. Another result gives us another plain text. The attack is based on this principle. Mention that you have to do it this way because you do not know the command codes. Example values for Equation 2 are shown in Figure 21. These show which steps should be taken in order to change the address byte like in the diagram shown in Figure 20.

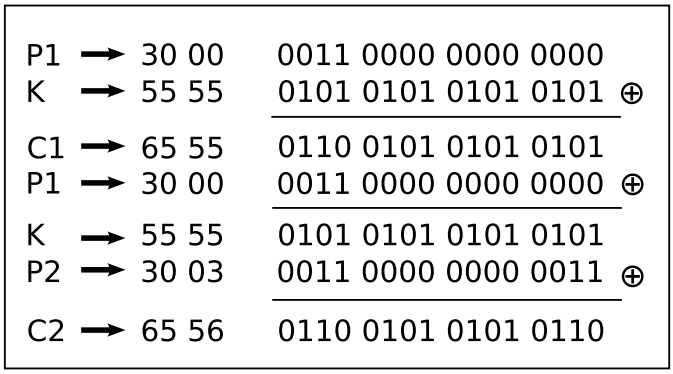



Figure 21: Example plaintext modification

**Known plaintext** When a sector trailer is read the card will return logical '0's instead of key A because key A is not readable. If key B is not readable the card returns logical '0's there as well. See Figure 4 for the location of key A and key B.

We made sure that the trace we recorded did only contain read commands. The first command of a transaction executed after an authentication can be derived by the structure of the trace. Each command takes a block address as parameter. This decreases the possible plaintexts for those first four bytes to 4 or 16 depending on the size of the sector.

Because the first command in the trace is known the keystream for those positions is also known. Obviously this allows to send another command than recorded in the original trace. When the first command is transformed into a 'read sector trailer' it is certain that the first 6 bytes in the response are logical '0's.

A logical step would now be to replay the same authentication again and try to execute a command that will return only an ACK or NACK[26]. Because this would shift the keystream 40 bits[27]. There will be enough known keystream left to construct a 'read sector trailer' command. Because we know (a part of) the plaintext in the response of the card this will recover more keystream bits.

[26] ACK = Acknowledged, NACK = Not Acknowledged

[27] 4-byte command, 4 times 8 bits + 4 parity bits and a 4-bit response makes it a 40-bit shift

This attempt does not work when we send commands that result in a NACK. The protocol aborts when any incorrectness is detected. This might be in case of access violation or errors in the communication. Messages that are too short, too long or have wrong parity bits are not accepted. Also unknown commands are rejected. Because the card halts after such messages we can not send a read command and recover more keystream bits.

**Keystream Mapping** The data is encrypted bitwise. When the reader sends or receives a message, the keystream is shifted the number of bits in this message on both the reader and card side. This is needed to stay synchronized and use the same keystream bits to encrypt and decrypt. The stream cipher does not use any feedback mechanism. Despite that, when we tried to reveal the contents of a message sequence using a known keystream of an earlier trace, something went wrong. We recorded an *increment* followed by a *transfer* command. We used this trace to apply our attack and changed the first command to a *read* command which consists of 4 command bytes and delivers 18 response bytes. Together with the parity bits this makes it a 198 bit stream. The plaintext was known and therefore we recovered 198 keystream bits.

When we used this keystream to map it on the original trace of the *increment* (Figure 22), it turned out that the keystream was not in phase after the first command. The reason was the short 4-bit answer of the card that is not followed by a parity bit. In our original trace we are now half way the first response byte. This means that after 4 more bits we arrive at the parity bit in the original trace. However, in our new trace we are then half way the next command byte. To correct this we needed to throw away the keystream bit that was originally used to encrypt the parity bit.
But what to do when we need to decrypt a parity bit in the new situation and we are half way a byte with respect to the first trace? The solution is to encrypt the parity bit with the next bit from the recovered keystream and use this same keystream bit to decrypt the next data bit.
From this we can conclude that parity bits are encrypted with keystream bits that are also used to encrypt databits.

| | INCREMENT | ACK | VALUE | TRANSFER | ACK |
|---|---|---|---|---|---|
| Plaintext | c1 04 f6 8b | 0a | 01 00 00 00 bb 4a | b0 04 ea 62 | 0a |
| Ciphertext | 4c 88 31 bc! | 0a! | e2 79!2a!14 35!6f! | 04!81 2d!1e! | 0c! |

Figure 22: Recovering the Keystream and Commands

The following method successfully maps the keystream on another message sequence as we described above.
Take the recovered keystream and strip all the keystream bits from it that were at parity bit positions. The remaining keystream can be used to encrypt new messages. Every time a parity bit needs to be encrypted, use the next keystream bit without shifting the keystream, in all other cases use the next keystream bit and shift the keystream.
So parity bits are always encrypted with the next available keystream bit, but the keystream is not shifted in that case.

**Authentication Replay** As shown in Section 3.1 the high-level protocol starts always with an authentication. This can be an authentication with key A or B (see table 24). So after a successful anticollision phase (part 3 of ISO 14443 [ISO01]) the only option will be to send an authentication request `60` (for key A) or `61` (for key B)[28]. In response to this authentication request the card will send a challenge nonce $N_C$. In Section **??** it is shown that the card repeats $N_C$ within reasonable time. Because no other varying information is used in the authentication like timestamps this enables a replay of an authentication.
To replay an authentication we first need a trace of a successful authentication between a genuine MIFARE reader and card. An example of an authentication followed by one read command is shown below.

```
1  PCD   60  03  6e  49
2  TAG   e0  92  93  98
3  PCD   ad  e7  96! 48! 20! 22  df  93
4  TAG   bf  06  91! 82
5  PCD   b5! 05! 47  3f
6  TAG   3f  14! 4f  e9! 86  38! 96! 85 3e!
         f3  e3! 3d! eb! 2b! a2  d4  dd 76!
```

After this recorded authentication between card and reader, we make sure that the memory of the card is not modified. This ensures that when the memory of the card is read it will return the same plaintext. Now we will act like a MIFARE reader and try to initiate the same authentication. In short:

1. We recorded a trace of a successful authentication between a genuine card and reader.
2. We send authentication requests (#1) until we get a nonce that is equal to the one (#2) in the original trace.
3. We send the recorded response (#3) to this nonce. It consists of a valid response to the challenge nonce and challenge from the reader.
4. We retrieve the response (#4) to the challenge from the card.
5. Now we are at the point we where we could resend the same command (#5) or attempt to modify it.

After step 4 the card is in a state where we have successfully authenticated for (in this case) sector 0 (block 3). Now it expects a command for this sector. If we send the same command we recorded earlier, we get the same encrypted response as in the original trace. Therefore the keystream is the same.

**Reading a Sector** We will show that it is possible to read sector 0 from a card without knowing the key. We only need one transaction between a genuine MIFARE reader and card. Every MIFARE Classic card has some known memory contents. The product information published by NXP [NXP07b] gives this information.
When a sector trailer is read the card will return logical '0's instead of key A because key A is not readable. If key B is not readable the card also returns logical '0's. It depends on the access conditions if key B is readable or not. The

[28] Except from sending a HALT command

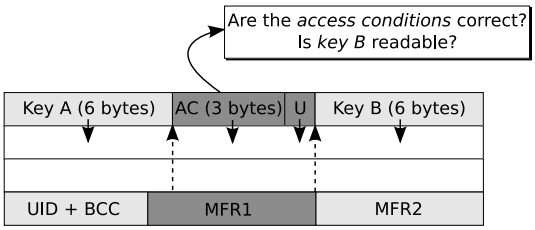



Figure 23: Recovering Sector 0

access conditions can be recovered by using the manufacturer data. Block 0 contains the UID and BCC followed by the manufacturer data. The UID and BCC cover 5 bytes and are known. The remaining 11 bytes are covered by the manufacturer data. Some investigation on different cards (MIFARE Classic 1k and 4k) revealed that the first 5 bytes of the manufacturer data almost never change. These bytes (MFR1) cover the positions of the access conditions (AC) and the unknown byte U, as shown in Figure 23. This means that the keystream can be recovered using the known MFR1 bytes by reading block 0 and block 3 (sector trailer) subsequently. Remember that the access conditions are stored twice in 3 bytes. Once inverted and once non-inverted. This way it is easy to detect if we indeed revealed the access conditions. The unknown byte U can be in any state when the card leaves the manufacturer but appears to be often `00` or `69`.

The access conditions tell us whether key B is readable or not. In many cases key B is not readable. In the Netherlands the MIFARE Classic 4k card is used in the public transport system. The first 5 bytes of the manufacturer data (MFR1 in Figure 23) recovered the access conditions for sector 0. Because the access conditions for the sector trailer define key B as not readable, we know the plaintext is zeros. Hence the whole sector trailer was revealed and therefore the contents of the whole sector 0 were revealed as well.

### 6.1.2 Bruteforce Attack

If the security of a cryptosystem only relies on the secrecy of its keys this means that knowledge of the used algorithm should not compromise a secret message that was encoded by this algorithm. Back in the 19th century Kerckhoff stated that a crypto system must be able to fall into the hands of the enemy without any inconvenience [Ker83]. Experience has shown that this is a good design principle when building secure systems. A system is well designed if the only practical attack is to try every possible key to retrieve the secret. This is called a bruteforce attack. In December 2007, Karsten Nohl and Henryk Plötz announced the complete recovery of the CRYPTO1 algorithm [NP07].
Until that time a brute force attack on a MIFARE Classic card was possible but not really feasible. The cards logic contains the algorithm and therefore can tell whether a key is correct or not. Although the keys are only 48-bit in size which is obviously too small [MBW96], this is compensated by the delay that is introduced by the communication in the authentication procedure. Knowledge of the CRYPTO1 streamcipher enables an off-line brute force attack. This eliminates the delay that is caused by the communication with the card in an on-line attack

where every attempt takes about 6 milliseconds. An on-line attack on one card takes more than 44 thousand years to try all $2^{48}$ possible combinations. This seems safe but the main reason why this takes so long is because of the secrecy of the algorithm. This is also called security by obscurity. Now the algorithm is revealed the key remains the only secret in the MIFARE Classic encryption. The key size of 48-bit is too small to prevent a successful brute force attack within reasonable time. Back in 1995 it was already strongly discouraged to implement symmetric encryption systems that use 56-bit keys [MBW96]. Nohl and Plötz stated in December 2007 [NP07] that depending on the amount of money available a brute force implementation recovers the sector keys within a week, day or even within one hour.

### 6.1.3 Key Recovery using Cryptanalysis

Less than half a year later other attacks were demonstrated by the Digital Security Group of the Radboud University Nijmegen [GdKGM+08] and Courtois et al [CNO08]. These attacks show that it is possible to retrieve keys much faster than in a bruteforce attack due to several weaknesses in the design of the algorithm. In both proposed attacks it is now a matter of seconds on a normal laptop.

## 6.2 Proprietary Commands

At the time this research was performed, we were not aware that the command codes, which we revealed with our attack, could already be found in example firmware of NXP[29]. Note that the firmware refers to the command codes sent from PC to reader. Our research shows that (perhaps obviously) these are the same command codes sent from reader to card.

We used a card in transport configuration with default keys and empty data blocks to reveal the encrypted commands used in the high-level protocol. All the commands send by the reader consist of a command byte, parameter byte and two CRC bytes. We made several attempts to reveal the command by modifying the ciphertext of this command. The way to do this is to assume we actually know the command. With this 'knowledge' we XOR the ciphertext which gives us the keystream. To check if this is indeed the correct keystream, we XOR it with a new command for which we know the response. If we guessed the initial command right the response of the card will be that known response. This method revealed the commands shown in Figure 24.

Now, one could try to replay the same authentication again and try to execute a command that returns an ACK or NACK in order to recover more keystream. Because an ACK or NACK is only 4 bits in size, it leaves some spare bits for which we know the keystream. We can use these bits to execute another command for which we now know the plaintext. This delivers more known keystream as a result, and this method can be applied repeatedly. However, this approach does only work if a *decrement*, *increment* or *transfer* is allowed. These are the commands that return an ACK and therefore are in total shorter than the *read*. We can only send valid commands because otherwise the protocol aborts. The *read* command returns 16 data bytes and 2 CRC bytes.

[29] `http://www.nxp.com/files/markets/identification/download/MC081380.zip`

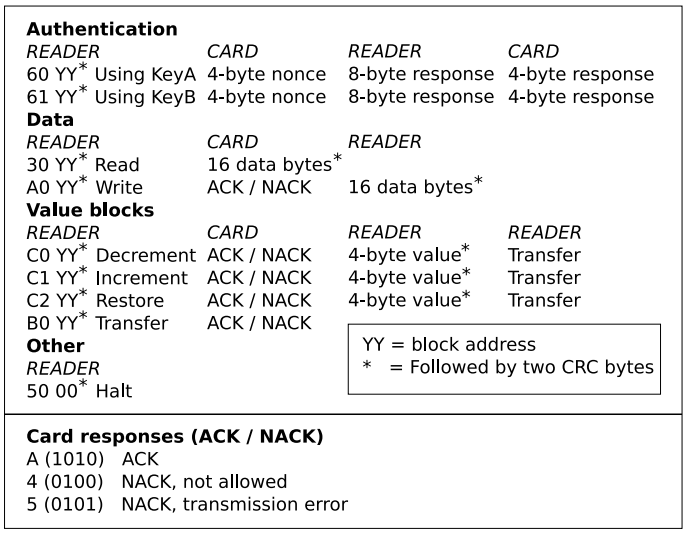



Figure 24: Command set of MIFARE Classic

On a *write* command the card returns a 4-bit ACK, this indicates that the card is ready to receive 16 data bytes followed by 2 CRC bytes.
The *decrement*, *increment* and *restore* commands all follow the same procedure. The card indicates that it is expecting a value from the reader by sending a 4-bit ACK response. This value is 4 bytes and is followed by 2 CRC bytes. For the *restore* this value is send but not used. The value is send as `YY YY YY YY ZZ ZZ`, where `YY` are the value bytes and `ZZ` the CRC bytes.
Finally, a *transfer* command is send to transfer the result of one of the previous commands to a memory block. The card response is an ACK if it went well. Otherwise it responds with a NACK.

The 4-bit ACK is `0xa`. When a command is not allowed the card sends `0x4`. When a transmission error is detected the card sends `0x5`. The card does not even give a response at all if the command is of the wrong length. The protocol aborts on every mistake or disallowed command.

# 7 Conclusions & Recommendations

We have implemented a successful attack to recover the keystream of an earlier recorded transaction between a genuine MIFARE Classic reader and card.

We used a MIFARE Classic reader in combination with a 'blank' card with default keys to recover the byte commands that are used in the proprietary protocol. Knowing the byte commands and a sufficiently long keystream allowed us to perform any operation as if we were in possession of the secret key.

We managed to read *all* memory blocks of the sector zero of the card, without having access to the secret key. In general, we were able to read *any* sector of the memory of the card, provided that we know *one* memory block within this sector. Moreover, after recording a valid transaction on any sector, we were able to read the first 6 bytes of any block in that sector and also the last 6 bytes if key B is read only. Similarly, we are able to *modify* the information stored in a particular sector.

## 7.1 Observations

**Weak pseudo-random generator** It is known that good pseudo-random generators are needed for cryptographic protocols to provide secure communication. By empirical research we found that the nonces generated by the MIFARE Classic card reappeared very often. This was the main reason why our attack developed in [dKGHG08] was successful.

**Keystream recovery is possible** The keystream recovery compromises MIFARE Classic in three ways. First and foremost, using the weakness of the pseudo-random generator, and given access to a particular MIFARE Classic card, the keystream generated by the CRYPTO1 streamcipher can be recovered, without knowing the key used. Secondly, the exact details of the communication between reader and card can be revealed. The command codes and structure that so far were unknown are revealed. Although the command codes can be found in firmware code that controls the MIFARE PCD chip, this research confirms that they are actually sent by the MIFARE PCD chip. And last, the malleability of the streamcipher is used to read *all* memory blocks of the first sector (sector 0) of the card (without having access to the secret key). This gives us the same possibilities as if we were in possession of the secret key. So, *modification* is also possible with this recovered keystream.

**Consequences** First of all, all data stored on the card (except the keys themselves) should no longer be considered secret. In particular, if the MIFARE Classic card is used to store personal information (like name, date of birth, or travel information), this constitutes a direct privacy risk. The security risk is relatively low because in general the security is guaranteed by the secrecy of the keys. Note that in particular we are not able to clone cards, because the secret keys remain secret.

Secondly, the integrity and authenticity of the data stored on the card can no longer be relied on. This is quite a severe security risk. This is particularly worrying in applications where the card is used to store a certain value, like loyalty points or, even worse, some form of digital currency. The loyalty level or

the value stored in the electronic purse could easily be increased (or decreased, in a denial-of-service type of attack).

Thirdly, knowledge of the plaintext (or the keystream) is a necessary condition to perform brute force (or other more sophisticated) attacks to recover the secret key. This allowed us to develop a very efficient attack to recover arbitrary sector keys of a MIFARE Classic card [dKGHG08].

## 7.2 Recommendations

**Migrate to a more advanced card** The main recommendation to solve the problems with the MIFARE Classic is unfortunately to migrate to another card type. The contactless smartcard market has developed more advanced cards with cryptographic schemes like DES, AES and even public-key cryptography over the years. For the implementation of new systems the MIFARE Classic is often chosen for its low price on one hand, and because it is thought to provide a reasonable level of security on the other. With respect to the latest developments, MIFARE Classic is not a serious candidate anymore. It is not said that, because it is possible, every system using MIFARE Classic is immediately under attack. The question is what the award will be if an attacker breaks into the system. Now the MIFARE Classic does not deliver the protection for which it was once sold it is needed to migrate to another product. Which product this should be depends again on the balance between level of security and price.

**Make sure abuse is detectable** Because migration takes a lot of time and should be prepared carefully it is useful to reduce the chance of abuse by some countermeasures.
While it has become relative easy to copy a MIFARE Classic card, it is not said to be impossible to detect this kind of attack. We have made an implementation of the CRYPTO1 algorithm on the Proxmark III which emulates a card completely. This means that there are two 'cards' in the system that are completely equal to the system. The back office should log all transactions. In most systems this is already the case. Detection is then done by finding contradictions in these logs. It is not possible for person $A$ to enter building $B1$ and building $B2$ at the same time. On the other hand it *is* possible for person $A$ to enter building $B1$ and after some time enter building $B2$. Then it is hard to detect if a cloned card was used.

A mechanism that detects the use of duplicate cards in any case is easy to achieve. The encryption provided by the card is broken, so store data encrypted with an encryption scheme like AES. When an attacker reads the data, he still can not tamper with it as long as he is not in possession of the right AES key. This way it is possible to store a counter on the card that gets incremented in every transaction by the reader. The card holder nor a possible attacker is able to increment this counter by one and store the new encrypted value because they do not posses the AES key. The latest value of this counter is stored in the back office. Every new transaction the counter is compared to the one in the back office and if they match nothing is wrong. If the value in the back office appears to be a successor of the value on the card this means that another card (duplicate or original) caused a transaction and increased the counter.

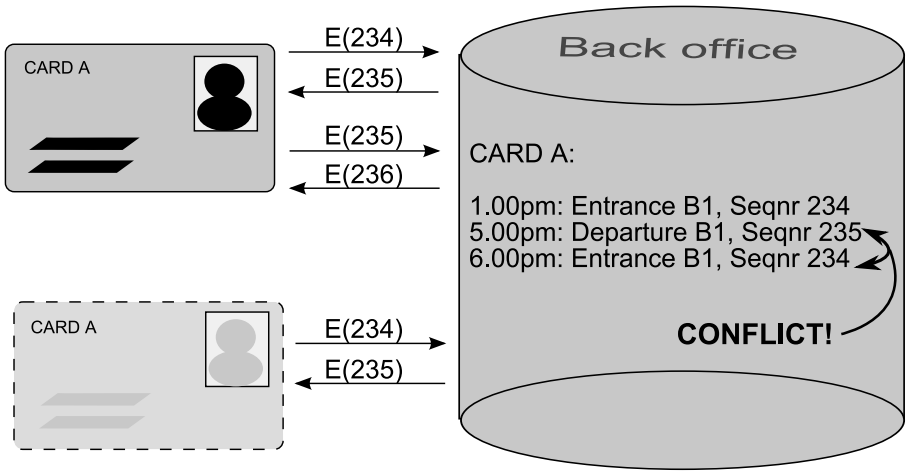



Figure 25: Detection of duplicate cards

This way it is possible to detect if there have been two or more duplicate cards used. However, it does not tell which card is the duplicate and which one the original. A card can be blocked immediately if a duplicate is detected. In larger systems, like public transport, this might be infeasible because of the delay that is introduced by communicating everything back to the back office. Then a card (and its duplicate) can be blocked afterwards and will be caught later on.

**Additional checks** A system does not only consist of RFID cards entering a building. In case of access control it is good practice to double check the entrance of a building by security personal, camera's etc. There are also systems that use MIFARE Classic in combination with a code which the user needs to enter[30]. This are all methods that decrease the level of success for an attacker.

**Use feedback in algorithm** In any stream cipher it is wise to use some kind of feedback of the encrypted plaintext. This prevents malleability. Modifications in the messages are detected because the continuation of the keystream is different which results in a corrupt message. This is done in the DESFire.

**Security by Obscurity** Security by obscurity means that a security system is kept secret by its manufacturer. There is an analogy between how a door lock works and how an encryption algorithm works. If a system is well designed and based on strong security principles there is no need to keep the algorithm (or the lock design) secret. The only secret is the key. The MIFARE Classic card is a typical example where the system is kept secret. Once it has been revealed more people can have a critical look on its design. It is known for a long time that this is a bad way of securing a system [Ker83]. This case of the MIFARE Classic card adds more evidence to this principle.

[30] `http://webwereld.nl/articles/49360/shell--easypay-immuun-voor-mifare-lek.html`

# 8 Further research

The problems concerning the MIFARE Classic are not only due to a weak design of the cryptographic algorithm. It is an old chip in comparison to similar products in the market. Back in 1994, when MIFARE Classic was introduced, the chip complexity was significant lower than any developed chips nowadays. It is a challenge to find methods and design protocols that provide sufficient security, privacy and anonymity given the constraints of an RFID system.

The results on the MIFARE Classic card might indicate that similar products could bear the same problems. One could think of the Sony FeliCa card which has comparable functionality. Sony claims on its website that it uses 'industry standard' security algorithms. The communication speed is 212 kbps, twice as fast as the 106 kbps of MIFARE Classic, and is based on other modulation schemes. More advanced cards of the MIFARE family like the DESFire and the SmartMX seem to have better protection against the attack we developed. Basically, the initialization of the protocols uses a random with a higher entropy than the MIFARE Classic. Additionally, they use some sort of feedback that involves the plaintext. This makes it harder to recover the keystream. Of course, also these cards have to deal with limiting constraints, but at least the used encryption scheme is public. Which of course should not be confused with the way it is implemented. Furthermore, there are much more RFID systems which are relatively cheap and provide proprietary security like the MIFARE Classic. Research on randomness of the used pseudo-random generators and the protocol should point out whether the products actually deliver the claimed security or not. The development of new research tools like the OpenPCD, OpenPICC, Proxmark, Ghost, Mole and many others show that the technology becomes cheaper over the years. Inherently, the number of people that look into the used protocols and security mechanisms will grow. Therefore, it is needed to detect design flaws in early stages to prevent abuse.

The MIFARE product family is also capable of communication speeds faster than the 106 kbps we have implemented in the Proxmark. It would be very helpful for additional research to make the Proxmark compatible with these higher speeds.

Nowadays it is possible to buy contactless cards that implement encryption schemes like AES. So the question is not any more if this is possible at all. However, for authentication and initialization one needs good pseudo-random generators. It is challenging to design a pseudo-random number generator that is heavily restricted by its environment (timing and power from reader).

The focus in the field of RFID is more on security instead of privacy. While one of the main results of the many RFID tags used today is that goods become traceable. First, because logistic companies want so, but also in cases when products leave the logistic chain and the customer is not aware of the RFID tag. This is only one example and many others are available. To provide privacy it is of course possible to use a random identifier, perform an authentication to the tag and then retrieve the real product ID under encryption. But these

implementations are too expensive in systems that trace goods by millions of tags. Easier concepts are presented that introduce tags which send a random number concatenated with the hash of that random and their identifier (or secret). This gives strong privacy, but lacks performance. For a worst-case look-up the system has to compute the hashes for all tags in the system. This increases linearly. There are solutions to this by using trees, but these sacrifice some privacy [NE]. Then the question is if it possible to maintain strong privacy on low-cost RFID tags.

# References


[AKQ] Gildas Avoine, Kassem Kalach, and Jean-Jacques Quisquater. ePassport: Securing International Contacts with Contactless Chips. to appear.

[Bog07] A. Bogdanov. Cryptanalysis of the KeeLoq block cipher. Technical report, Cryptology ePrint Archive, Report 2007/055, February 16, 2007.

[CNO08] Nicolas T. Courtois, Karsten Nohl, and Sean O'Neil. Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards. Cryptology ePrint Archive, Report 2008/166, 2008. `http://eprint.iacr.org/`.

[Dig08] Digital Security Group, Radboud University Nijmegen. Dismantling contactless smartcards. Press release, March 2008.

[dKGHG08] G. de Koning Gans, J.H. Hoepman, and F.D. Garcia. A Practical Attack on the MIFARE Classic. *Arxiv preprint arXiv:0803.2285*, 2008.

[Fin03] Klaus Finkenzeller. *RFID Handbook*. John Wiley and Sons, 2nd edition, 2003.

[GdKGM+08] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Dismantling Mifare Classic. Forthcoming, 2008.

[Han05] G.P. Hancke. A practical relay attack on ISO 14443 proximity cards. *Technical report, University of Cambridge Computer Laboratory*, 2005.

[HHJ+06] J.-H. Hoepman, E. Hubbers, B. Jacobs, M. Oostdijk, and R. Wichers Schreur. Crossing Borders: Security and Privacy Issues of the European e-Passport. In Hiroshi Yoshiura, Kouichi Sakurai, Kai Rannenberg, Yuko Murayama, and Shinichi Kawamura, editors, *Advances in Information and Computer Security. International Workshop on Security (IWSEC 2006)*, volume 4266 of *Lecture Notes in Computer Science*, pages 152–167. Springer Verlag, 2006.

[HK] G.P. Hancke and M.G. Kuhn. An RFID distance bounding protocol. *Conference on Security and Privacy for Emerging Areas in Communication Networks–SecureComm 2005*.

[ISO01] ISO/IEC 14443. *Identification cards - Contactless integrated circuit(s) cards - Proximity cards*, 2001.

[Kas06] Timo Kasper. Embedded Security Analysis of RFID Devices. Master's thesis, Ruhr-University Bochum, 2006.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX, 1983. pp. 5–38, Jan. 1883, and pp. 161–191, Feb. 1883.

[MBW96] R. Rivest B. Schneier T. Shimomura E. Thompson M. Blaze, W. Diffie and M. Wiener. Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security. 1996.

[NE] K. Nohl and D. Evans. Quantifying Information Leakage in Tree-Based Hash Protocols. *Proceedings of the Conference on Information and Communications Security*, pages 228–237.

[NESP08] Karsten Nohl, David Evans, Starbug, and Henryk Plötz. Reverse-Engineering a Cryptographic RFID Tag. 2008. USENIX Security Symposium. San Jose, CA. 31 July 2008.

[Noh08] Karsten Nohl. Cryptanalysis of Crypto-1. 2008. Published on March 10th.

[NP07] Karsten Nohl and Henryk Plötz. Mifare, Little Security, Despite Obscurity. Presentation on the 24th Congress of the Chaos Computer Club in Berlin, December 2007.

[NXP07a] NXP Semiconductors. *Functional specification contactless single-trip ticket IC*, April 2007.

[NXP07b] NXP Semiconductors. *MIFARE Standard 4kByte Card IC functional specification*, February 2007.

[RMP07] H. Richter, W. Mostowski, and E. Poll. Fingerprinting Passports. 2007.

[SvdS07] P. Siekerman and M. van der Schee. Security Evaluation of the disposable OV-chipkaart v1.6. Master's thesis, University of Amsterdam, 2007.

[Ver08] Roel Verdult. Proof of concept, cloning the OV-Chip card. Technical report, Radboud University Nijmegen, 2008.

[Wes] J. Westhues. Hacking the prox card. *RFID: Applications, Security, and Privacy*, pages 291–300.