

# **Outsmarting Smart Cards**

**Gerhard de Koning Gans**

Copyright © Gerhard de Koning Gans, 2013  
ISBN: 978-94-6191-675-4  
IPA dissertation series: 2013-05

Typeset using L<sup>A</sup>T<sub>E</sub>X



The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics)



The graphical art of this work, except for the comic on page 25, is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/>



The remaining part of this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Netherlands License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/3.0/nl/>

# Outsmarting Smart Cards

PROEFSCHRIFT

ter verkrijging van de graad van doctor  
aan de Radboud Universiteit Nijmegen  
op gezag van de rector magnificus prof. mr. S.C.J.J. Kortmann,  
volgens besluit van het college van decanen  
in het openbaar te verdedigen op donderdag 11 april 2013  
om 15:30 uur precies

door

Gerrit Theodoor de Koning Gans

geboren op 1 juni 1983  
te Zwolle

**Promotor:**

Prof. dr. B.P.F. Jacobs

**Copromotor:**

Dr. F.D. Garcia

**Manuscriptcommissie:**

Prof. dr. G. Avoine

Prof. dr. ir. H.J. Bos

Dr. J.C. Hernandez-Castro

Dr. J.H. Hoepman

Prof. dr. E.R. Verheul

Université Catholique de Louvain

Vrije Universiteit Amsterdam

University of Kent



In memory of  
*Johannes Pap*



---

## Acknowledgements

I would like to thank the many people that helped me directly or indirectly in the completion of my thesis. First of all, my gratitude goes to my direct supervisor Flavio Garcia for his guidance and pleasant assistance in my research. I would also like to thank my promotor Bart Jacobs. Bart has always been willing to share his clear view on a wide range of security related subjects. It encouraged me to develop my own critical view on new developments and applications in our information society, which helps me to this day. I thank Bart and Flavio for proofreading the early versions of this thesis and providing me with constructive comments. Special thanks also go to the members of the reading committee, Gildas Avoine, Herbert Bos, Julio Cesar Hernandez-Castro, Jaap-Henk Hoepman and Eric Verheul, who helped improving this thesis a lot by their valuable and professional comments.

Thanks to all the members of the “Mifare team” with whom I had the privilege to work on some exciting research. I am convinced that the dedicated and close collaboration between the team members has been one of the cornerstones of this success. We have defeated the Germans.

I am grateful that the Digital Security group as a whole, with its dynamic composition of members, was a nice environment for discussion and collaboration. Let me mention some people in particular. Ken Madlener, a good friend and colleague who means a lot to me. In our countless number of afternoon breaks we kept analyzing the (academic) world and motivated each other to keep on going. Another special friend and colleague I would like to thank here is Roel Verdult. Thank you for your close collaboration and friendship.

Thanks go to my co-authors. Besides Flavio, Bart, Jaap-Henk, Eric and Roel it also has been a pleasure to work with Arjan Blom, Milosch Meriac, Ruben Muijers, Erik Poll, Peter van Rossum, Joeri de Ruiter and Ronny Wichers Schreur.

The past four years will remain in my thoughts as a period of new experiences, meeting inspirational people and a period in which I could develop myself in my profession. I would like to thank all the people who supported me in many ways over the years. It is impossible to name every person, but do know that I appreciated the support of many more people than mentioned here. I am really thankful for the opportunities that I have been given to increase my knowledge and skills.

For the entire duration of my study I felt unconditionally supported by my family and close friends, without whom all of this would not have been possible.

Gerhard de Koning Gans  
Arnhem, February 2013



---

# Contents

<b>Acknowledgements</b>	<b>vi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Smart cards and RFID	3
1.2 Cryptography	5
1.2.1 Basic building blocks	6
1.3 Security protocols	10
1.3.1 Security goals	10
1.3.2 Formal verification and testing	11
1.4 Attack scenarios	12
1.5 Outline and results	18
<b>2 Tools for eavesdropping and analysis</b>	<b>23</b>
2.1 Communication protocols	25
2.1.1 The physical layer	26
2.1.2 Encoding techniques	27
2.1.3 Modulation techniques	29
2.2 Proxmark III	31
2.2.1 Hardware board	32
2.2.2 FPGA implementation	34
2.2.3 Demodulation	38
2.2.4 Other RFID research tools	38
2.3 SmartLogic	40
2.3.1 ISO/IEC 7816	41
2.3.2 SmartLogic setup	41
2.3.3 Hardware	42
2.3.4 Software	44
2.3.5 Other smart card tools	45
2.4 Conclusion	46

<b>3</b>	<b>Case study: Smart cards in practice</b>	<b>49</b>
3.1	An active man-in-the-middle attack on EMV . . . . .	50
3.1.1	The EMV protocol . . . . .	51
3.1.2	The attack . . . . .	52
3.1.3	Using the SmartLogic . . . . .	52
3.1.4	EMV attack implementation . . . . .	54
3.2	Security tokens for internet banking . . . . .	56
3.2.1	The e.dentifier2 . . . . .	56
3.2.2	Attack on the USB-connected mode . . . . .	57
3.3	Distance relaying . . . . .	59
3.4	Smart card emulation . . . . .	60
3.5	Concurrent SIM card sharing . . . . .	62
<b>4</b>	<b>Dismantling Mifare Classic</b>	<b>65</b>
4.1	Research context and related work . . . . .	66
4.2	Mifare Classic . . . . .	70
4.2.1	Communication layer . . . . .	71
4.2.2	Memory layout . . . . .	71
4.2.3	Commands . . . . .	73
4.2.4	Anticollision and authentication . . . . .	73
4.3	Weak pseudo-random number generator . . . . .	74
4.4	Recovering the command codes . . . . .	75
4.4.1	Keystream recovery . . . . .	77
4.4.2	Reading sector zero . . . . .	80
4.4.3	Reading higher sectors . . . . .	81
4.4.4	Command codes . . . . .	81
4.5	Recovering the cryptographic system . . . . .	83
4.5.1	Authentication protocol . . . . .	83
4.5.2	CRYPTO1 cipher . . . . .	85
4.6	Weaknesses and exploits . . . . .	89
4.6.1	LFSR state recovery . . . . .	89
4.6.2	LFSR rollback . . . . .	91
4.6.3	Odd inputs to the filter function . . . . .	92
4.7	Attacking Mifare Classic . . . . .	94
4.7.1	Attack one . . . . .	94
4.7.2	Attack two . . . . .	95
4.7.3	Multiple-sector authentication . . . . .	95
4.7.4	Improved attacks . . . . .	96
4.8	Conclusion . . . . .	97

<b>5</b>	<b>Dismantling iClass and iClass Elite</b>	<b>99</b>
5.1	Research context and related work . . . . .	100
5.2	iClass . . . . .	103
5.2.1	Functionality . . . . .	104
5.2.2	Authentication protocol . . . . .	106
5.3	iClass Standard . . . . .	106
5.3.1	Black box reverse engineering . . . . .	107
5.3.2	The function <i>hash0</i> . . . . .	114
5.3.3	Weaknesses in iClass Standard key diversification . . . . .	115
5.3.4	Attacking iClass Standard key diversification . . . . .	118
5.4	The iClass cipher . . . . .	119
5.4.1	Firmware reverse engineering . . . . .	119
5.4.2	The cipher . . . . .	121
5.5	Weakness in iClass . . . . .	123
5.5.1	Weak keys . . . . .	123
5.5.2	XOR key update weakness . . . . .	123
5.5.3	Privilege escalation . . . . .	124
5.5.4	Lower card key entropy . . . . .	124
5.5.5	Key recovery attack on iClass Standard . . . . .	124
5.6	iClass Elite . . . . .	125
5.6.1	Key diversification on iClass Elite . . . . .	126
5.6.2	Weaknesses in iClass Elite key diversification . . . . .	128
5.6.3	Key recovery attack on iClass Elite . . . . .	128
5.7	Conclusion . . . . .	130
<b>6</b>	<b>A synchronizable forward-private low-cost RFID protocol</b>	<b>133</b>
6.1	RFID next to barcodes . . . . .	135
6.2	Forward privacy . . . . .	136
6.3	The desynchronization problem . . . . .	137
6.3.1	Barcode analogy . . . . .	139
6.4	System model . . . . .	140
6.5	Security definitions . . . . .	141
6.6	Protocol description . . . . .	144
6.6.1	Second channel . . . . .	144
6.6.2	Tag and reader state . . . . .	145
6.6.3	Success, failure and synchronization run . . . . .	146
6.6.4	Precomputation and state resolution . . . . .	148
6.7	Security analysis . . . . .	150
6.8	Conclusion . . . . .	152
	<b>Bibliography</b>	<b>155</b>
	<b>Index</b>	<b>169</b>

<b>Abbreviations</b>	<b>173</b>
<b>Samenvatting (Dutch summary)</b>	<b>177</b>
<b>Curriculum vitae</b>	<b>181</b>

## Chapter 1

---

# Introduction

*“We can’t solve problems by using the same kind of thinking we used when we created them.”*

Albert Einstein

Computers become more and more part of our everyday life. They have evolved from big mainframes, occupying complete rooms, to tiny chips that have the size of a grain of sand. Also, their communication capabilities have improved and evolved from the use of slow mechanical carriers, i.e., punch cards, to much faster electronic interfaces. There are many examples of the use of computers in our everyday life and often we are not even aware of their presence. Think for example of all the controlling mechanisms that are present in modern cars. In 1977, the first Engine Control Unit (ECU) was introduced by General Motors [Cha09]. It was a single function that controlled the electronic spark timing of the car engine. Nowadays, cars have numerous functions implemented by computer software. These functions range from essential car controlling mechanisms to user entertainment systems. At first sight, the seamless integration of this software in on-board computers delivers quite some convenience and contributes to our safety in many ways. However, we are also exposed to new threats of abuse and disruption that, once they manifest themselves, might have a high impact on our everyday life. Think for example of a failing airbag system during a car crash. This relates directly to the physical security of a person. Or, think of a massive car recall by Toyota in 2005. The car manufacturer was recalling 160.000 Prius hybrid cars that contained a software bug [Gar05]. It appeared that the gasoline engine could suddenly stall. The error was corrected by updating the car software. These incidents were not deliberately abused, but vulnerabilities may also lead to attacks where unauthorized people take control over the car [CMK<sup>+</sup>11]. These kind of security problems are increasingly seen in all kinds of automated systems. We are long past the point where most of our communications were by regular mail. All kinds of information systems are connected to the Internet, e.g., medical systems, Internet banking, etc. The increasing interconnectivity of these systems result in a more and more complex security problem.

Although computer security has received increasingly more media attention over the last few years, it is in many situations still an afterthought. It is not introduced in an early stage of system design. Also, security reviews, security monitoring and maintenance do not always get the priority they deserve. As a result, vulnerabilities are present in the design and implementation of systems. They can get easily abused

while having a severe impact on society. Fortunately, a gentle shift in security thinking can be observed and the awareness of security risks has increased. Of course, it is important to understand that security will always be about finding a balance. A balance between the acceptable risk and the acceptable costs of security measures. The International Organization for Standardization has published ISO/IEC 27005 that gives guidelines to help organizations in managing these information security risks [ISO08]. It is often said that ‘100% security can never be obtained’. However, this saying can never be used as an argument to lower the bar for security measures. In the end, there is really a big difference in breaking good or bad security mechanisms, and this difference will mainly reflect in the cost and effort for an attacker to break them.

It is clear that security mechanisms are an important aspect in the development of systems. At the same time, it is beyond the scope of this thesis to cover all different areas in which digital security plays an important role. In this thesis we focus on the security of smart cards, e.g., bank cards, access cards and public transport tickets. There exist two types of smart cards. On the one hand, we have contact-based cards that can be recognized by their eight contact points on the card, on the other hand, we have contactless smart cards that communicate wirelessly and do not have any physical contact points at all. Contactless (or wireless) cards are often referred to as Radio Frequency Identification (RFID) cards. We want to have a thorough understanding of the security mechanisms that are implemented on smart cards. A very powerful strategy to reach a high level of understanding of these mechanisms is the use of formal methods. This technique allows us to model the security requirements and the system implementation, and makes it possible to check whether the former is achieved by the latter. It has been shown that security really benefits from formal methods by the discovery of flaws that went unnoticed for years [NS78, Low96]. Also, it has been shown that when “proper” cryptographic primitives are used, formal security proofs have an associated security proof in the traditional computational security model [AR02, MW04a, Gar08, GHPvR05, GvR08, GvR06a, GvR06b, GGvR08], which means that an adversary can break the protocol only with negligible probability.

Although the development of models and techniques for formal verification have made a valuable contribution to the secure design of systems, still many things can go wrong during the implementation or maintenance of a system. Furthermore, many system designs are kept secret by its manufacturer and are not checked with formal methods. In this thesis, we want to investigate the security level of deployed systems for which no exact implementation or formal model is known. The errors that we find will, in the first place, help to raise the awareness of developers, manufacturers, administrators and eventually society. Secondly, we are better off when security holes are detected before criminals abuse them. Simply put, it will not help when a system design is fully proven to be correct, while at the same time, we have no clue about its actual implementation. This especially applies to so-called proprietary systems. Proprietary systems are systems whose internal workings are kept

secret by the manufacturer. In the third place, it once again opens up the discussion around open design, which means that literally everybody can review the internal workings of a system, increasing the trust in its security. Eventually, this contributes to the move to better security designs and implementations.

This chapter will first continue with a short introduction to smart cards and RFID. Then, cryptography is briefly introduced followed by its application in security protocols. Additionally, we describe some major attack scenarios on these security protocols. We conclude with an overview of the remaining chapters.

## 1.1 Smart cards and RFID

A smart card typically holds a tiny chip or Integrated Circuit (IC) that has an input-output interface, some memory and a logic unit to execute arithmetic operations. Compared to today's personal computers, smart cards have limited memory, energy consumption and computing power. One might roughly compare the capabilities of today's smart cards to the first personal home computers. Not to say that smart cards are a recent phenomenon, on the contrary, the first cards equipped with magnetic stripes stem from the early seventies [RE10]. Around 1984 several banks around the world started to use these magnetic stripe cards and introduced them in the form of debit cards. This can generally be seen as the first steps towards the use of smart cards. Carrying data on a magnetic stripe is not a very 'smart' task, but it allowed the banks to introduce some smarter tasks like automatic card processing and card holder verification by using a Personal Identification Number (PIN). Still, these type of cards can not make calculations or interact, and therefore they do not qualify for the adjective 'smart'.

Currently, the use of magnetic stripe cards is declining and many banks have decided to move on to the contact-based smart card, which is actually a plastic card that contains an embedded chip. Figure 1.1 shows a cross-sectional view of a contact-based smart card. In this thesis, when we refer to smart cards, we do not refer to the magnetic stripe cards but to the chip-based cards as shown in Figure 1.1. This figure shows that we do not actually see the chip itself. What we typically see on the surface of a contact-based smart card are its metal contact points, together forming a small square on the card like depicted in the upper left corner of the figure. The chip is situated below these contact points. Since a smart card has the capabilities of a tiny computer, it is very flexible and suitable for many applications. Apart from their use as debit cards, contact-based smart cards are also used as access cards, identity cards, e-health cards, insurance cards, SIM cards<sup>1</sup>, etc. Nowadays, high-end smart cards support computationally intensive operations [NXP10], like public-key operations based on RSA or elliptic curves.

Apart from contact-based there are also contactless smart cards, better known as RFID cards. Radio Frequency Identification is a technique that already has been used

---

<sup>1</sup>Subscriber Identity Module that is used in mobile phones.

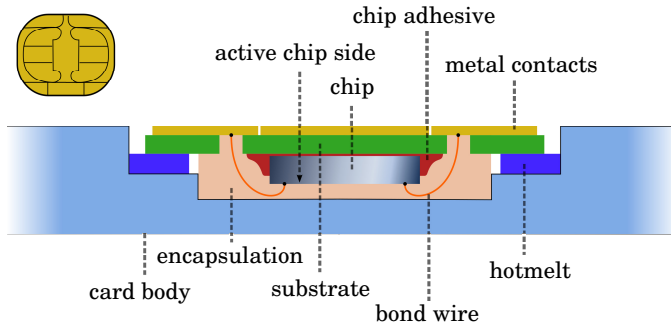


Figure 1.1: A cross-sectional view of a contact-based smart card

during World War II to identify friend or foe airplanes. The British army did send radio signals from the ground on which British airplanes would respond by generating a radio signal back. Nowadays, RFID is deployed on a much larger scale with much smaller technology, but the principle remains the same. An RFID system consists of a *reader* and a *transponder* (a.k.a. tag). The reader is equipped with an antenna that generates an electromagnetic field at a specific frequency. The transponder also contains an antenna and a small IC. The transponder receives the digital messages that are modulated on the reader antenna and modulates its answers using a sub-carrier. The reader continuously sends out a signal to identify transponders that reside within its proximity. A transponder replies with its unique identifier when it receives a reader request. This allows the reader to identify which transponder is in its proximity. The transponder might be incorporated into a label or smart card, we then often speak of an *RFID label* or *RFID card*, respectively. There is a huge diversity in RFID systems and applications, ranging from low frequency radio systems (< 300 kHz) to ultra-high frequency radio systems (> 3000 MHz) and from Electronic Article Surveillance (EAS) to high-end smart cards which are fully programmable. The focus of this thesis will be on two kinds of RFID systems. First, on the systems that fall within the high frequency range and operate at 13.56 MHz. This range is used for contactless smart cards and typically allows these cards to be accessed by a reader at a maximum distance of 10 centimeters. An example of such a contactless card is shown in Figure 1.2a, the chip is usually smaller than the one depicted in this figure. Later on, we will focus on the privacy problems that occur in Electronic Product Code (EPC) type of systems that make use of low-cost RFID labels (860-960 MHz), which allows readers to communicate with the labels at a distance of a few meters. An example of an EPC-like tag is shown in Figure 1.2b. Apart from different frequencies that are used, there is also a difference between *active* and *passive* RFID systems. In active RFID systems, transponders have their own power source, i.e., a battery. Passive RFID transponders, on the contrary, are powered by the electromagnetic field of the reader. This means that the antenna is used for both power supply and communication, and will be explained in more detail in the next chapter.





Figure 1.2: Examples of RFID devices

The many different applications of smart cards and RFID have led to the design of at least as many *communication protocols*. In the first place, these protocols are needed to ensure several useful properties, e.g., that every participant of the system gets its turn to send messages, that accidental errors in the communication can be detected, that protocol participants can be identified, and so forth. Such protocols might try to achieve various design goals, e.g., *robustness*, *reliability* and *availability*. Another very important use of protocols is to fulfill different types of security goals. Typical design goals from a security perspective are *confidentiality*, *integrity* and *authenticity*. In other words, how do we keep our communication between two or more parties confidential? How do we detect whether the data that we receive is not corrupted by an attacker? How do we know that the data is authentic, originating from the source that we expected?

## 1.2 Cryptography

Secure communication over (semi-)trusted or untrusted channels is an ancient problem. It has existed for centuries in an ever changing context. The basic problem, however, remains the same: how do we keep a message confidential and make sure that only the intended recipient is able to read it when we send it over an insecure channel. The solutions to this problem, and its study in general, is called *cryptography*. The word cryptography is derived from a composition of two ancient Greek words. The first part  $\kappa\rho\upsilon\pi\tau\acute{o}\varsigma$  (*crypto*) means *concealing* or *hiding*. The second part stems from the Greek word  $\gamma\rho\acute{\alpha}\phi\omega$  (*graphy*), where its meaning comes close to *writing*. In short, cryptography means *secret writing*. It is the art of hiding a message by representing it in a different and disordered way. Apart from confidentiality, cryptography also delivers building blocks that can be used to achieve integrity and authenticity. These are separate properties and they do not imply each other as we will see in the next example on the Caesar cipher. This cipher only provides confidentiality and no integrity. The famous Caesar cipher serves as an early example of the use of cryptographic techniques. It consists of a simple technique that involves shifting every single letter of the alphabet a number of positions to the right or left.

Using a right shift of three letters results in the following scheme.

Original:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Replace by:	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Using this encoding scheme, the secret message 'RETREAT' becomes the encoded message 'UHWUHDW'. The message that needs to remain confidential is usually called the *plaintext* and the encoding of this message is called the *ciphertext*. Then, the mechanism that is used to transform a plaintext into a ciphertext is called a *cryptographic algorithm* or *cipher*. The transformation from plaintext to ciphertext is called *encryption* and the way back from ciphertext to plaintext is called *decryption*. The information that defines how a cryptographic algorithm transforms a plaintext into a ciphertext and vice versa is called the *key*, e.g., in our small example of the Caesar cipher the key is the aforementioned table, which can be reconstructed by remembering a right shift of three letters.

A cryptographic algorithm that uses the same key for both encryption and decryption is referred to as applying *symmetric-key cryptography*. Conversely, the phrase *asymmetric-key cryptography* refers to systems where the encryption and decryption of messages requires two different keys, a private one and a public one, where the private key cannot be deduced from the public key.

The study of cryptographic algorithms is called *cryptanalysis* and contributes to the important condition that ciphers should be well-studied before they are actually used. If it turns out that a cipher is broken, e.g., an attacker is able to decrypt messages without knowledge of the key, the cipher obviously fails and should not be used. This peer reviewing of ciphers is needed to gain confidence in its security and follows the famous principle of Kerckhoffs from 1883 [Ker83]. This principle states that it is important to only rely on the secrecy of the key for security, not on the secrecy of the cryptographic algorithm itself. So, the cryptographic algorithm does not become useless when the secret key gets compromised by an attacker. In that case, replacing the secret key would be enough to reestablish a secure system.

### 1.2.1 Basic building blocks

Let us review some important building blocks in cryptography, namely *stream ciphers*, *block ciphers* and *public-key cryptography*.

#### Stream ciphers

In order to obtain confidentiality, the most secure way of encrypting data would be to XOR the data with a truly random data source. This random source serves as a non-repeating key that has the same length as the original plaintext and covers the original plaintext message. Since the key is of the same length as the message, and is combined symbol by symbol with the plaintext, it is usually called the *keystream*. In practice, the plaintext and keystream are combined at bit-level in order to obtain the

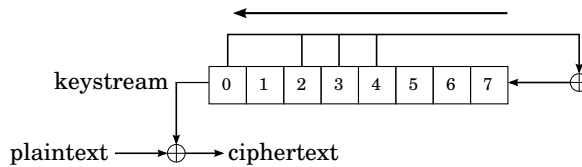
ciphertext. For this reason it is called a *stream cipher*. The exclusive or (XOR) operator used here is denoted by  $\oplus$ . Now we give the truth table of the XOR operation and an example encryption.

$a$	$b$	$a \oplus b$	
0	0	0	plaintext
0	1	1	keystream
1	0	1	ciphertext
1	1	0	

01010010010001100100100101000100...	
11101011101010101011011100010110...	$\oplus$
1011100111101100111111001010010...	

When we have a truly random keystream, it is impossible to break this encryption scheme. This was shown by Claude Shannon in his *Communication Theory of Secrecy Systems* in 1949 [Sha49]. However, in practice it is not very convenient to use such lengthy keys and, even trickier, how do we get such a long key secretly to the side of the receiver? One could think of solutions that use the contents of a phonebook page as key material. The communicating parties have to agree upon an offset page to start, which is obviously secret. A more general solution to this problem of communicating long keys is to use a replacement for the random keystream, a pseudo-random number generator. A Linear Feedback Shift Register (LFSR) is an example of such a mechanism as it generates a pseudo-random (i.e., not fully random) bitstring from a fixed-length internal state.



**Figure 1.3:** Linear Feedback Shift Register (LFSR)

A simple 8-bit LFSR is shown in Figure 1.3. At every update, all the bits are shifted one position to the left. As a result, the leftmost bit at position 0 falls out and is the output bit of the LFSR. In Figure 1.3 the output bit of the LFSR is used as keystream bit. Then, the new rightmost bit at position 7 becomes the XOR of the bits located at the LFSR taps, i.e., positions 0, 2, 3 and 4. Note that by convention we start numbering with 0. LFSRs have often been used in chip designs for smart cards since this technique is relatively cheap and efficient. We emphasize that LFSRs are mostly used as a building block for ciphers and do not represent a complete cipher as our example might suggest.

## Block ciphers

Another symmetric cipher is the *block cipher*. As its name says, a block cipher operates on blocks of data, instead of single bits. The blocks are usually of a predetermined size, e.g., 64 or 128-bit data blocks. The plaintext length should be a multiple

of the block size. It is padded with some predetermined data when this is not the case. In general, block ciphers do consume more power and need a larger chip surface compared to stream ciphers.

The block cipher Lucifer was developed by IBM in the 1970s and is based on an idea of Horst Feistel [Fei73]. Lucifer formed the basis for the block cipher DES [FIP99]. The basic structure of the Lucifer and DES algorithm is also known as Feistel network. An example of a 3-round Feistel network is shown in Figure 1.4. The input to the network is split into a left ( $L$ ) and right ( $R$ ) half. Every round, the right half  $R$  is input to a round function  $f_i$ . The output of  $f_i$  is combined with the left half  $L$  and forms the new right half, while  $R$  is swapped to the left. It is because of this structure that one can define any possible function  $f_i$  and is still able to reverse the operations, which is needed for decryption. In DES the round functions are defined by round keys of 48 bits. These round keys are derived from the 56-bit main key. Furthermore, the DES algorithm is based on a 16-round Feistel network. Feistel networks have been well-studied over the years and this research [Pat04, Pat92, NR99, Lub96, Knu08, Pat98] has resulted in theoretical security bounds. However, these bounds mean nothing when a small key length is used. The key length of single DES is only 56 bits. Nowadays, single DES keys can be brute forced, which means that an attacker simply tries every possible key, within a couple of days. For this reason Triple DES was introduced in 1998, which is based on applying three times single DES. In 2001, the National Institute of Standards and Technology (NIST) published another well-known block cipher which is called the Advanced Encryption Standard (AES). This block cipher has been designed by Vincent Rijmen and Joan Daemen [DR02].

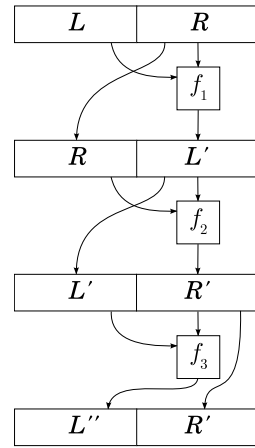


Figure 1.4: Feistel

## Public-key cryptography

Finally, we have asymmetric encryption schemes, or *public-key cryptography*. A big advantage of these schemes is that they are based on mathematically hard problems. In this sense, breaking the cryptographic algorithm comes down to solving the mathematical problem which is believed to be hard. Compared to both, the block cipher and the stream cipher schemes, this is a much more expensive technique in terms of computational complexity. This reflects in the costs of smart cards that support public-key cryptography. Development of public-key cryptography started in the seventies and the first famous paper was a solution for key exchange over an untrusted channel by Diffie and Hellman in 1976 [DH76]. In this paper, Diffie and Hellman also presented the concept of a public-key cryptosystem. Motivated by this work [Rob03], Rivest, Shamir and Adleman invented an asymmetric encryption scheme called RSA in 1978 [RSA78].

## Cryptographic hashes

Many security mechanisms make use of *cryptographic hash functions*. A hash function  $h$  takes as input a binary string of arbitrary length and outputs a fixed-length binary string:

$$h: \{0, 1\}^* \rightarrow \{0, 1\}^n, m \mapsto h(m).$$

This output message is often called *hash value* or *message digest*. A hash function reduces larger messages to relatively small messages. Input messages that lead to a particular hash value are called *pre-images*. Because of the limited size of the output domain there exist multiple pre-images for a given hash value. A *cryptographic hash function*, following the informal definition of [DK02], should meet the following requirements.

*Pre-image resistance* Given an output value  $y$  it must be infeasible to find a pre-image  $x$  such that  $h(x) = y$ .

*Second pre-image resistance* Given an output value  $h(x)$  and its corresponding input value  $x$  it is infeasible to find another pre-image  $x'$  ( $x \neq x'$ ) such that  $h(x) = h(x')$ .

*Collision resistance* It is infeasible to find two pre-images  $x$  and  $x'$  ( $x \neq x'$ ) such that  $h(x) = h(x')$ .

To ensure these properties, flipping a single bit of the input message should result in flipping many bits (on average half of them) of the hash value. This is called the *avalanche effect* and makes the hash function a suitable method to detect message changes and look after its integrity. An example of its use is found in digital signing. Signing a digital message is conducted by signing its hash. This is much more efficient than signing the often larger original message. The properties described above guarantee that the signed hash value only relates to the original input, i.e., an attacker is not able to find another message that leads to the same hash value and thus to the same signature.

## Message Authentication Codes

A Message Authentication Code (MAC) is a security mechanism that checks the integrity of a message as well as its authenticity. The input of a MAC algorithm, following the definition of [DK02], is a symmetric key  $k$  and a message  $m$  of arbitrary length. It outputs a fixed-length message  $m'$ , usually called the MAC. Now, everyone that knows the secret key  $k$  can check the authenticity of a message  $m$  that is accompanied by a MAC  $m'$  by checking  $MAC(k, m) \stackrel{?}{=} m'$ . Note, that this is also true the other way around, everyone that knows the secret key  $k$  can create the MAC for any message they want.

## Key diversification

When a security system makes use of symmetric keys and consist of many entities, like tickets in a public transport system, key management becomes a hard problem. Maintaining a list of all different keys that are used throughout the system is not a very practical solution. Especially, when this list contains millions of keys. Also, it is undesirable to use just one key for all entities in a system. Obviously, the system security gets compromised when this key leaks out.

In these situations *key diversification* is a standard solution. In key diversification the secret key for every entity is derived from a *master key*  $K$  and some unique information, e.g., a Unique Identifier (UID), that identifies this entity. The assignment of a key  $k_{id}$  for every entity is as follows

$$k_{id} = \text{Diversify}(K, id).$$

Diversification functions often use some symmetric-key algorithm, like 3DES or AES, or some cryptographic hash function like SHA-256. It is important that the function that is used satisfies the *pre-image resistance* property. In a public transport ticketing system, for instance, the different entities are the cards. Every card gets assigned a key  $k_{id}$  based on its UID. The master key is only stored in a Secure Application Module (SAM) that is integrated into an RFID gate. In this situation, breaking one card key does not imply that the other cards in the system are compromised. The attacker only knows the key  $k_{id}$  for one card. Because of the pre-image resistance property it is computationally infeasible to get back to the master key  $K$  only by using his knowledge of  $k_{id}$ .

## 1.3 Security protocols

Cryptographic algorithms can be used to secure our communications and data storage. Security protocols arise from the need for communication over an untrusted channel. For instance, two parties want to exchange confidential information while at the same time they do not trust their communication channel. In other words, they cannot be sure that a third party is not eavesdropping their communication. Another example is to achieve anonymity where security protocols are used to retain the anonymity of the participants. In RFID this kind of protocols are useful to prevent adversaries from tracking specific RFID labels, and thus from following goods or individuals that carry these labels.

### 1.3.1 Security goals

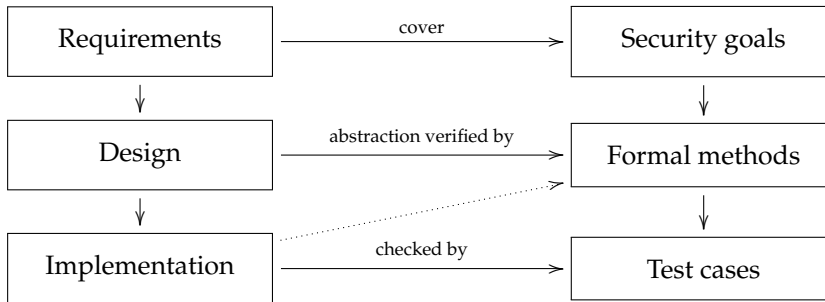
A security protocol always needs to fulfill some security goals. The following list covers the objectives that are regularly pursued in protocol designs. This list is partly based on definitions from [DK02] and it does not pretend to be complete. It starts

with the well-known CIA core triple, which stands for *confidentiality*, *integrity* and *availability*.

1. *Confidentiality*. It should be possible to keep a message confidential. Only the intended recipient should be able to decrypt and read the message.
2. *Integrity*. The receiver of a message should be able to check its integrity. Accidental or deliberate changes of the message should be detectable and it should not be possible to (partly) substitute a message.
3. *Availability*. It should be possible to use a functionality of a system. A Denial-of-Service (DOS) attack is about bringing down the availability of a system.
4. *Authenticity*. It should be possible to verify the sender's identity of a message. When two parties start communicating they should be able to verify each other's identities.
5. *Non-repudiation*. It should not be possible for the sender of a message to deny that he constructed this message later on.
6. *Privacy*. It should be possible for an individual to control what information is collected about him or her. Furthermore, it should be clear who keeps this information, who uses this information and for what purpose. For instance in RFID, where the Unique Identifier (UID) is often a fixed number that does not change over time. This undermines the location privacy of the individual carrying the RFID label.

### 1.3.2 Formal verification and testing

Designing a security protocol is an error-prone task. First, it is important to identify several design criteria that should be met by the protocol. A security protocol should protect against someone or something, usually called the *attacker* or *adversary*. How powerful is this attacker? What kind of attacks should the protocol resist? Over the years, many mistakes have been found in protocols that were first considered to be secure. A way to improve protocol design and to prevent design errors is to be very precise in the goals, in the definition of the protocol, its environment and the capabilities of an attacker. Formal methods provide a framework in which such a precise formulation of the protocol, the security goals and the attacker can be given. After formalization, the second step is to verify whether the security goals hold. This can be done by hand, semi-automatically or fully automatic [Low96, AG97, Bla01]. Computers are very suitable for this automated task. Formal methods have been applied in protocol analysis for quite some time now [BS80]. Furthermore, they have shown successful in finding protocol errors that have gone unnoticed for nearly two decades [NS78, Low96]. Of course, the actual model should correctly resemble the specification, otherwise it is still possible that attacks remain unnoticed.



The diagram above is an attempt to identify the different approaches in the design of security protocols. The *requirements* should at least cover the *security goals* but can also add additional constraints like a runtime, or a specific target platform for the implementation. The security goals as well as the design itself might be input to a formalization of the protocol that can be checked for errors. Sometimes, even the implementation can be checked by formal methods when they are not too complex. A lot of research has been done in this area and many models have been proposed. As long as the protocols do not become too complex it is feasible to provide them with correctness proofs.

The *implementation* should be strongly based on the abstract design, but needs some unavoidable refinements that might introduce errors at the implementation level of the protocol. To find out if such errors exist, lots of *test cases* are defined and checked against the actual implementation. The test cases can be defined by hand or derived from the formalized model. In security research, the focus is mainly on the design and verification of security protocols. Despite the myriad of protocol mistakes that were discovered in the literature by these methods, there still exists a gap with the industry. Manufacturers often choose to keep the cryptographic algorithms and protocol designs secret. These confidential algorithms, also known as proprietary algorithms, are in most cases not formalized at all. In these situations, the only way to find out how a system works is to find out how it is implemented and test it. From this level, we can work up to the original design and verify if there exist any problems, be it in the application, protocol or in the cryptographic algorithm.

## 1.4 Attack scenarios

This section discusses the most well known attacks on smart card protocols. Some attacks are more specific to contactless smart cards, like for example in tracking attacks that allow an attacker to follow specific identities without physical interference. The protocol notation used throughout this thesis follows the simple example depicted in Figure 1.5. We make use of a message chart diagram where all entities that partici-



pate in a protocol run are represented by vertical lines. The protocol steps are drawn from top to bottom in chronological order and are represented using labeled arrows. In Figure 1.5 we have two entities  $A$  and  $B$  (Alice and Bob). In this example protocol,  $A$  sends a message  $m$  to  $B$  and  $B$  replies with a different message  $m'$  to  $A$ . When an attacker is involved in the protocol she is denoted by  $E$  (Eve).

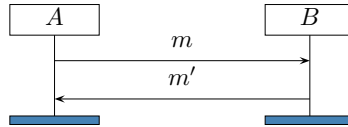


Figure 1.5: Protocol

In the following text we introduce different attack scenarios. Of course, these scenarios may be combined but for clarity reasons we explain them separately. The first scenario is the *Man-in-the-Middle attack* where an attacker  $E$  fully controls the communication between two parties  $A$  and  $B$ . Then, the *relay attack* is a similar situation where the attacker does only forward the communication over greater distances between two parties  $A$  and  $B$ . Furthermore, in a *replay attack* the attacker simply sends previously recorded messages. Then, we explain the *reflection attack* where the symmetry property that some protocols might have is exploited by reflecting challenges. This way, the honest sender answers its own challenge and this answer can be used by the attacker for authentication. It differs from a simple replay attack in the sense that fresh challenges are used in the protocol. We finish with the *side channel attack* and *fault injection attack* which are attacks that go beyond the logical level and try to carefully listen to or influence the protocol on a physical level by looking at the power, radio waves, etc.

### Man-in-the-Middle attack

The *Man-in-the-Middle attack* is a very well-known attack scenario where the attacker takes position between the participating entities in the protocol. There is a passive variant where the attacker does not affect the data and just sits on the communication line. This variant is often called a *passive Man-in-the-Middle attack* or *eavesdropping attack*. When attacker  $E$  is actually modifying, blocking or injecting messages and thereby influencing the process, we speak of an *active Man-in-the-Middle attack*. Figure 1.6 illustrates an active Man-in-the-Middle attack since the message  $m'$  is replaced with  $m''$  by attacker  $E$ .

### Relay attack

In a *relay attack* the attacker relays all communication between two entities without them noticing. The communication can be relayed over larger distances, fooling the entities as if they are situated close to each other. This type of attack has been given

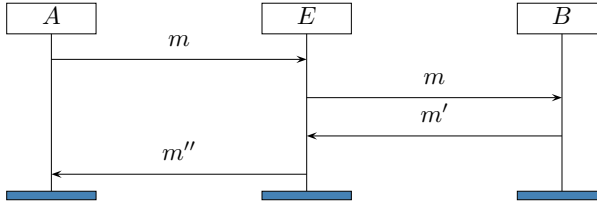


Figure 1.6: Man-in-the-Middle attack

the name Mafia Fraud in a paper by Desmedt et al. [DGB88] that discusses this type of attacks. The attacker controls two interfaces  $E_1$  and  $E_2$ , see Figure 1.7. These can be seen as two collaborating corrupted entities. These interfaces  $E_1$  and  $E_2$  can interconnect two honest entities  $A$  and  $B$  over any other existing medium. For example, consider  $A$  an RFID reader and  $B$  an RFID card that communicate over the 13.56 MHz radio band. An attacker may install  $E_1$  in front of the reader and let  $E_2$  speak with the card, then it is possible to relay all communication between  $E_1$  and  $E_2$  over, for instance, the GSM network. Of course, this switch to a different communication channel might introduce time delays. This is used in an early proposal by Beth and Desmedth [BD91] to measure the Round-Trip Time (RTT) of a protocol message. The idea behind this is to bind the maximal response time to the speed of light. Protocols that implement these measurements are also known as *distance bounding protocols*. These protocols use a rapid protocol phase [BC94, HK05, DM07] to eliminate relay attacks. The rapid phase in a protocol should precede the regular protocol messages and operate on a low physical level. During this rapid phase, a challenge is sent to the entity that needs to prove its physical presence. Apart from sending the correct answer to this challenge, this answer should also arrive within a very short time frame. This very strong time restriction is feasible by implementing the checks on a low level, eliminating as much overhead as possible.

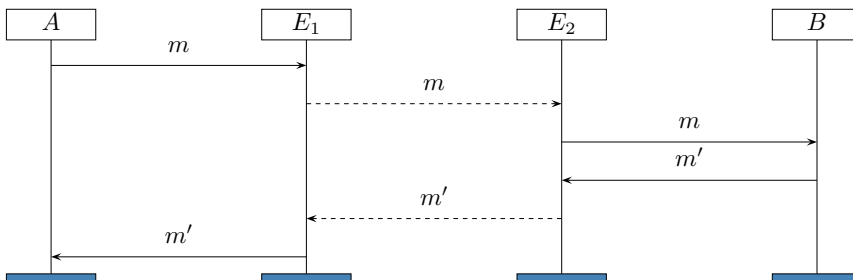


Figure 1.7: Relay attack



### GARAGE SALE: THE REPLAY ATTACK IN PRACTICE

The parking garage of the Huygens building of the university in Nijmegen uses RFID technology to automatically control the entrance gate. The system uses a simple identification protocol that involves a card telling its UID to the RFID gate. After a simple lookup in a database the system knows whether this card is bound to a parking subscription. If so, the gate opens and lets the car through. We have demonstrated that the replay attack, as we have explained in Figure 1.9, can be mounted on this system. However, good care has to be taken since the system tracks whether a car is inside or outside the garage. When we execute the replay attack on the gate with identity  $A$ , the system registers that the car that belongs to identity  $A$  is inside the garage. Now, the person with the genuine card can not enter the garage any longer since the system believes that  $A$  is already inside. The parking garage was one of the first systems we investigated during our research on RFID systems.



Figure 1.8: Huygens building

### Replay attack

The *replay attack* is an attack where a message  $m$  is intercepted and sent again by an attacker in order to, for example, impersonate an entity. In Figure 1.9a, the attacker  $E$  pretends to be  $B$  and retrieves message  $m$  from  $A$ . Then, in Figure 1.9b, the attacker sends the earlier captured message  $m$  to  $B$ , thereby impersonating  $A$ . The message  $m$  could in this case be a unique identifying number.

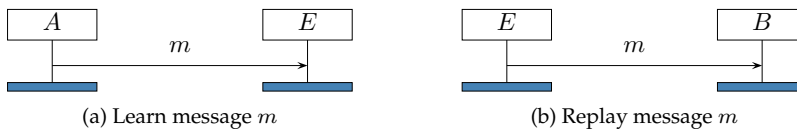
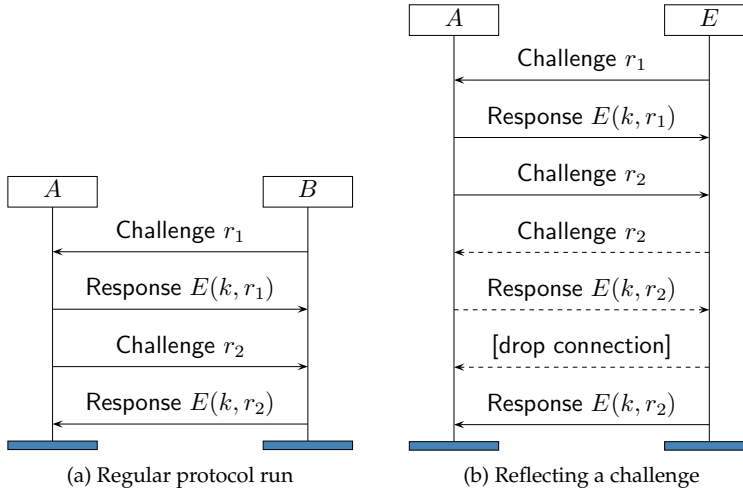


Figure 1.9: Replay attack

### Reflection attack

A *reflection attack* can be used to attack a system that uses a symmetric authentication protocol. This means that two entities  $A$  and  $B$  authenticate one another using the same method. When a protocol is prone to a reflection attack, an attacker  $E$  is able to authenticate to  $A$  by letting  $A$  answer its own challenge, see Figure 1.10b. A protocol that is subject to a reflection attack is depicted in Figure 1.10a. In order to attack this protocol, the attacker reflects the challenge  $r_1$  from  $A$  back to  $A$  in a second

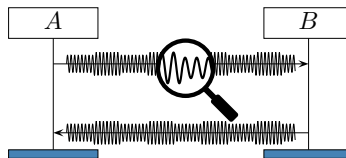


**Figure 1.10:** Reflection attack

session (marked by the dotted arrows in Figure 1.10b). Then,  $E$  uses the response to  $r_2$  in the second session to answer the challenge  $r_2$  of the first session and this way authenticates itself to  $A$ . In this example the answer is the challenge itself encrypted with the secret key  $k$ . Note that  $E$  does not learn  $k$  but just reflects the answer  $E(k, r_2)$  to  $A$ . In order to prevent reflection attacks it is important to include the identity of the authenticating entity in the answer to the challenge. Other ways to eliminate the reflection attack is not to use a symmetric key or not to use a symmetric protocol.

### Side channel attack

The *side channel attack* is a very powerful attack method that can be used to learn, for example, the secret key that is used during communication. Side channel analysis can be based on the fact that every calculation step that is made by an electronic device results in some small electrical interferences. The subject covers a wide range of different techniques, like watching the power consumption, electromagnetic radiation, variations in temperature, etc. Contactless smart cards make use of passive RFID technology, which means that they are powered by the electric field which is generated by the reader. Here, side channel analysis can be used to measure the



**Figure 1.11:** Side channel attack

impedance of the signal in the field in order to measure the power that the card uses. Figure 1.11 illustrates this side channel setup. When two entities  $A$  and  $B$  are communicating, the attacker is not directly involved in this communication, but still tries to learn the secret key by looking at side channel information, hence the phrase ‘side channel attack’.

### Fault injection attack

The *fault injection attack* is a method that actively tries to affect the process. This can be done in a random way, like shooting with a laser beam at random locations of a chip, hoping for a crucial process modification such that certain security mechanisms are circumvented. Or this can be done the opposite way, such as a introducing very precise timed power or clock glitches, which means that the attacker induces slight variations in the supply voltage or clock period. When the power is removed at a crucial moment during the process, it might cause a chip to malfunction or, more conveniently, prevent the chip from a updating crucial data. Figure 1.12 illustrates

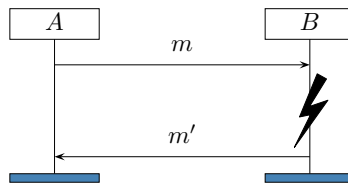


Figure 1.12: Fault injection attack

that the point of attack in this case would be on the entity itself, e.g., the chip of a contactless smart card. In the literature there are several models of fault injection attacks. These models assume a certain power of the attacker, i.e., what changes the attacker can introduce during the process and how precise (localized) these changes can be. The feasibility of an attack strongly depends on the model that is used. For example, several attacks in the literature assume the capability of flipping specific bits, which in most cases is a very hard, if not infeasible, task. For instance, in current 22 nm nano technology, more than 4000 of these transistors fit across the width of a human hair. Furthermore, chips do often consist of multiple layers, which makes it harder to reach certain areas of the circuit.

### Tracking attack

The *tracking attack* is a more specific problem to contactless smart cards. Since nearly all RFID protocols start with the card telling its Unique Identifier (UID), it is a simple task for an attacker to just query any card that is around. Now, when an attacker controls several readers at different physical locations, this will allow the attacker to track the movements of one specific card or RFID label and thus compromises the privacy of the carrier of this card or label. Figure 1.13 illustrates how an attacker

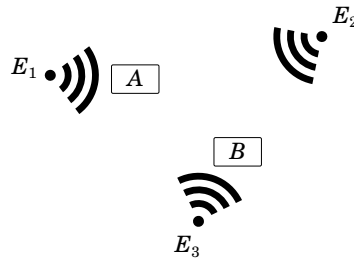


Figure 1.13: Tracking attack

could track RFID cards *A* and *B* by requesting their UID at three different physical locations. A solution to this problem might be to use random UIDs. This means that upon every new request the card sends a different randomized UID. This is feasible if the card supports cryptographic algorithms that allow identification in a randomized way, since the main task of legitimate readers is still to identify the card. These solutions exist in practice, but the problem remains for the much smaller RFID labels that are attached to products. They do not support powerful cryptographic algorithms, however, they still need a way to preserve the privacy of its holder and prevent tracking attacks.

## 1.5 Outline and results

The focus of this thesis is mainly on security and privacy of contact-based and contactless smart cards. We introduce concrete tools that enable us to look at the lowest communication levels of smart card protocol implementations. Using these tools we investigated concrete examples of bad security practice in widely used systems. The most prominent example is the Mifare Classic card. The Mifare Classic was first sold in 1994 [Smi94] and is being sold to date. It was only in 2008 that the card appeared to have serious security problems. The weaknesses that were found back then allow attackers to break the security of Mifare Classic cards within seconds or minutes, depending on the system configuration. It is alarming to see how many system integrators blindly put their trust in this product, purely based on the claim of the manufacturer that it is ‘field-proven technology’. Unfortunately, the vulnerabilities of the Mifare Classic are not an isolated example. Another example of this kind is the iClass technology.

Further topics that will be addressed in this thesis are the use of smart cards in banking applications, the problem of key diversification for smart cards and the privacy problem in RFID. In short, all topics are related to smart card security and privacy. The contribution and structure of this thesis is as follows.

**Chapter 2** introduces the two generic hardware tools, the Proxmark III and the SmartLogic. These tools play a fundamental role in the research that is described in this thesis. The Proxmark and the SmartLogic provide access and control to the lowest levels of the protocol stack for contactless and contact-based smart cards, respectively. Both tools assisted the research that is described in Chapter 3, 4 and 5.

This chapter is partly based on two papers. The first paper is *“The SmartLogic Tool: Analysing and Testing Smart Card Protocols”* [dKGdR12] by Joeri de Ruiter and the author. The second paper is *“A Toolbox for RFID Protocol Analysis”* [VdKGG12] by Roel Verdult, Flavio Garcia and the author.

#### *Contribution*

My contribution in this chapter is the development of the SmartLogic tool for smart card protocol research and the Proxmark firmware for RFID protocol research. Both tools provide support for all kinds of attack scenarios like the Man-in-the-Middle, relay, reflection and replay attacks. With the SmartLogic, I provided an implementation of a smart card test tool which allows ‘field research’ on deployed smart card protocols. This implementation aims to provide a straightforward way to implement new test scenarios using Java. All contributions are made available in the open source domain.

**Chapter 3** concentrates on contact-based smart cards, especially in payment systems. It describes several use cases that make use of the SmartLogic. The introduction of new chip-based payment protocols deserves a closer look at how these protocols are actually implemented. This chapter includes a use case on the Europay, Mastercard and VISA (EMV) standard, which is a definition for chip-based payment protocols. Many countries, including the Netherlands, recently switched to chip-based payment systems. Furthermore, this chapter addresses a problem that was found in an Internet banking security token and shows that smart card protocols can be relayed over 10.000 km.

This chapter is based on two papers. The first paper is *“The SmartLogic Tool: Analysing and Testing Smart Card Protocols”* [dKGdR12] by Joeri de Ruiter and the author. The second paper is *“Designed to Fail: A USB-Connected Reader for Online Banking”* [BdKGP<sup>+</sup>12] by Arjan Blom, Erik Poll, Joeri de Ruiter, Roel Verdult and the author.

#### *Contribution*

My contribution in this chapter is the implementation of the several test setups on different smart card applications like bank cards and SIM cards for mobile phones.

**Chapter 4** unveils the secrets of the Mifare Classic card. The Mifare Classic is a contactless smart card (RFID) that is widely used. This chapter starts with a description of the reverse engineering process, then it continues with the very first practical attack on a Mifare Classic card. This attack bypasses the cryptographic algorithm without knowledge about its internal operation. The second part of this chapter describes the revealing research on the Mifare Classic that resulted in the full disclosure of the cryptographic algorithm itself. Additionally, two early cryptographic attacks are presented that exploit several cryptographic weaknesses. To conclude, an overview is given of the attacks that are known to date.

This chapter is based on two papers. The first paper is "*A Practical Attack on the Mifare Classic*" [dKGHG08] by Jaap-Henk Hoepman, Flavio Garcia and the author. The second paper is "*Dismantling Mifare Classic*" [GdKGM<sup>+</sup>08] by Flavio Garcia, Ruben Muijers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, Bart Jacobs and the author.

#### *Contribution*

My contribution in this chapter is in the first place the facilitation of the hardware infrastructure. I developed the firmware for ISO/IEC 14443-A support on the Proxmark and this way I did provide the equipment that was essential in reverse engineering and attacking the Mifare Classic. In preparation of this reverse engineering I did recover already a lot of information like command codes and protocol behavior that was useful in this process. Furthermore, I actively participated in the development of the attacks that are described in this chapter.

**Chapter 5** describes major errors in the design and implementation of the security mechanisms of iClass. With more than 300 million cards sold, HID iClass is one of the most popular contactless smart cards on the market. It is widely used for access control, secure login and payment systems. The cipher and key diversification algorithms are proprietary and little information about them was publicly available until recently. This chapter reviews the key diversification mechanism, which handles the derivation of card specific keys from one master key. The second part is on the cryptographic system and its implementation errors. Some of these errors are clearly introduced while attempting to fix other problems. It demonstrates that instead of fixing a broken algorithm, it is better to start over again and design a completely new algorithm. Several design mistakes add up and allow several attacks that recover the master key. The attack times range from seconds to a couple of hours.



Chapter 5 is based on two papers. The first paper is “*Exposing iClass Key Diversification*” [GdKGV11] by Flavio Garcia, Roel Verdult and the author. The second paper is “*Dismantling iClass and iClass Elite*” [GdKGV12] by Flavio Garcia, Roel Verdult, Milosch Meriac and the author.

*Contribution*

My contribution in this chapter is the reverse engineering of the key diversification function of iClass Standard. To facilitate this research, I implemented iClass support on the Proxmark for eavesdropping and emulating card and reader. Furthermore, I contributed to the cryptanalysis of the iClass proprietary cipher and implemented the attacks on both the iClass Standard and the iClass Elite system.

**Chapter 6** addresses the problem of anonymity in the specific case where only light-weight, symmetric-key cryptographic primitives can be used. For instance, in low-cost RFID labels that are used in consumer product identification. These RFID labels often are still combined with barcodes. This chapter proposes a protocol that exploits this to achieve untraceability without being subject to desynchronization attacks. Barcodes and RFID labels are bound to coexist for quite some time which makes this a plausible approach. It reduces the workload at the reader and the back-office and guarantees resynchronization at any point in time when an RFID label gets desynchronized with the back-office. Concretely, this authentication protocol achieves correctness, forward-privacy under mild additional assumptions and synchronization in the random oracle model.

This chapter is based on the paper “*Towards a Practical Solution to the RFID Desynchronization Problem*” [dKGG10] by Flavio Garcia and the author.

*Contribution*

My contribution in this chapter is the design of a low-cost RFID protocol that is desynchronization resistant. In order to reach this goal without the use of public-key cryptography, I introduced a new concept called *second channel*. This notion resembles the fact that an adversary does not ‘see’ the input from a source other than the wireless link. Furthermore, I provided the proof sketches that follow the random oracle model.



## Chapter 2

---

# Tools for eavesdropping and analysis

*"We shape our tools. And then our tools shape us."*

Marshall McLuhan

Assessing the security of deployed systems is a complex task. Security aspects can be viewed at different levels of abstraction. These levels range from a very high abstract level of protocol analysis, e.g., using formal methods, to a very technical and detailed inspection of hardware implementations. This latter form of research is facilitated by hardware inspection tools. In this chapter we introduce and explain the functionalities of two tools, the Proxmark III and the SmartLogic. These tools have been used in several studies on smart card protocols [HJSW06, dKG08, dKGG08, GdKGM<sup>+</sup>08, GvRVWS09, Tan09, Cho10a, VK11, GdKGV11, GdKGV12, PN12, dKGdR12, ABV12, GdKGV12, VdKGG12]. The Proxmark can be used in research on Radio Frequency Identification (RFID) applications and the SmartLogic can be used in research on contact-based smart card applications. Both tools benefit from a flexible design by implementing a software based protocol stack. In practice, this means that protocol implementations can be analyzed at the lowest levels of the protocol stack. This protocol stack is often mapped to the well known theoretical Open Systems Interconnection (OSI) model [Zim80], as shown in Figure 2.1. The OSI model is used as a reference model to explain how the different protocol layers are implemented. As Nie et al. [NGEF99] point out, it depends on the system requirements how and where these OSI protocol layers are implemented. Communication speed and flexibility considerations play an important role in the choice which layers are implemented in hardware and which ones in software. Nie et al. distinguish three implementation layers called the host layer, adaptation layer and the media layer. The host layer implements the end functionalities for the user, like card identification or initiating a particular replay of earlier recorded data. Then, the adaptation layer prepares the data for transmission and provides the host layer only with application-relevant data, like the result of a replay session or the result of a card identification command. Lastly, the media layer is responsible for reliable physical transmission of the data. Where these three layers begin and end, with respect to the OSI model, depends on the requirements of a particular protocol implementation. Flexibility and control at bit level are very important requirements for the Proxmark and the SmartLogic. Figure 2.1 illustrates which protocol layers are implemented on which components of the Proxmark and the SmartLogic. Clearly, the OSI model is more of a reference model for protocol definitions because in practice its layer func-

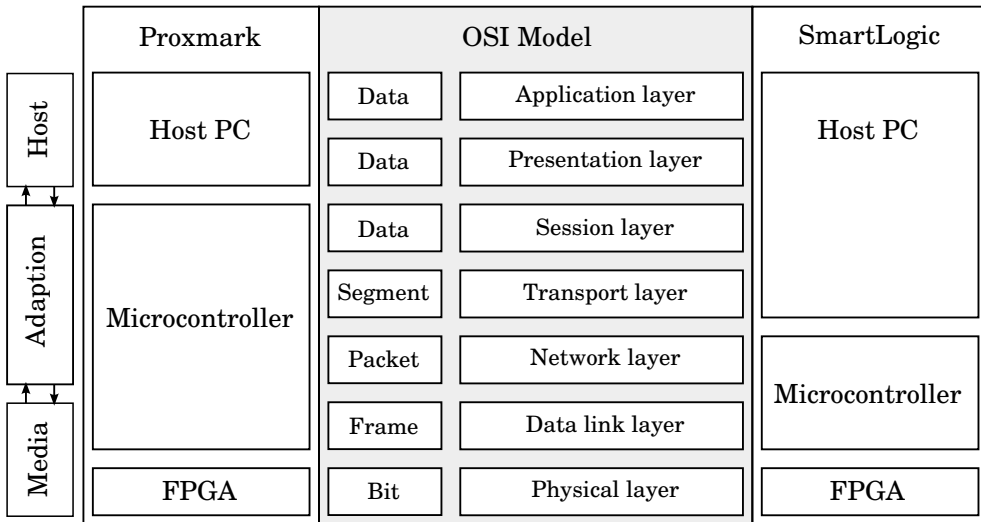


Figure 2.1: OSI Model

functionalities do often mingle. Up to the lowest physical layer, which is implemented on a Field Programmable Gate Array (FPGA), all protocol layers of both tools are defined in software. This opens up access to the lowest levels of protocol implementations in a flexible manner. When we speak of the lowest levels of communication, we mean the three lowest layers that are described in the OSI model. Namely, the network layer, data link layer and physical layer.

This chapter is organized as follows. First, Section 2.1 starts with a short introduction on signal modulation and encoding techniques used in RFID and smart card communication. Then, Section 2.2 continues with the Proxmark III. The Proxmark is an RFID research tool designed by Jonathan Westhues [Wes12]. There are three versions of the Proxmark whereas only the software and hardware design of the last one, the Proxmark III, has been made public. The first version was used to demonstrate the ability to execute a replay attack on simple ID-only RFID systems [HJSW06]. The Proxmark III is more advanced compared to its earlier versions and has attracted a large group of developers and users over the last few years. Its hardware and software are open source which leads to many improvements and new functionalities. The core functionalities of the Proxmark are eavesdropping, man-in-the-middle and emulation of both readers and transponders. Also, timing and brute force attacks are feasible with the Proxmark III. In a timing attack the attacker tries to break a system by monitoring the time a system needs to process a certain input. The attacker tries to obtain a valid key from this information. In a brute force attack the attacker tries to break a system by simply trying every possible key. The Proxmark is very suitable for this task since it can autonomously execute this attack without the overhead of communicating back to a host computer.

Finally, Section 2.3 describes the SmartLogic which is a contact-based smart card research tool. It is implemented on a small FPGA evaluation board. This board is manufactured by ZTEX [Zie]. There exist several ZTEX boards at a price range from tens to several hundreds of euros. Furthermore, the necessary firm- and software are open source, like for the Proxmark. The SmartLogic gives complete control over the smart card communication channel for eavesdropping, man-in-the-middle attacks, relaying and card emulation. These functionalities are essential for smart card protocol research and testing.

## 2.1 Communication protocols

In smart card technology we have contact-based smart cards that physically make contact with a reader and contactless smart cards that use radio waves to communicate with a reader. There also exist dual-interface cards that support both contactless and contact-based communication. Like in human conversations some basic rules are needed in order to have a successful conversation. First, it is important that only one person speaks at a time. When someone starts speaking while another person is already speaking it will jam the conversation and consequently the message is lost. Similarly, it is important that a reader and card do not transmit simultaneously. Other conversational problems might arise when one of the parties mumbles, speaks very fast or does not articulate very well. Likewise, in RFID, the signal strength should be strong enough, the reader and card should agree on some predefined communication speed and complete messages, including start and stop bits, should be transmitted. To let communication function correctly in electronic systems like smart cards we need communication protocols. These protocols give a very precise definition of the commands that can be used, the communication speed, the maximum distance between the card and reader, the signal strength, etc. We speak in this context about smart card protocols and in case of RFID technology we refer to it as RFID protocols.



Another analogy between human language and communication protocols is that both know many variations. In technology, these variations mainly stem from the many different manufacturers that all introduce their own standards. Although attempts have been made to limit the number of different specifications there still exist many variations, especially in RFID. The following sections discuss parts of the ISO/IEC standards that are used in most popular RFID systems and the standard that is used for contact-based smart cards. We explain how to get from their specification to an implementation that runs on the Proxmark or SmartLogic.

There is a great variety of frequencies that can be used in RFID. The most used frequencies are in the Low Frequency (LF) band at 125-134 kHz and in the High Frequency (HF) band at 13.56 MHz. These frequencies are popular in access control and ticketing systems. Then, there is the Ultra High Frequency (UHF) frequency that is used by RFID tags that operate on the 860-960 MHz range. This ultra high frequencies are more used in supply chain management and implement the Electronic Product Code (EPC) protocol<sup>1</sup>. The Proxmark, which we discuss in this chapter, is capable of communicating on both 125-134 kHz and 13.56 MHz by two built-in antenna circuits. This is beneficial for the flexibility of the device since these two frequencies are most used in RFID cards. In this thesis we will especially look at the lowest protocol layers of cards that use the 13.56 MHz frequency. The relevant standards are ISO/IEC 15693 and ISO/IEC 14443. For contact-based smart cards the relevant standard is ISO/IEC 7816.

Despite the existence of standards, manufacturers design products that deviate from the standards, while claiming to be compliant. In practice this means that there exist RFID systems that behave according to the standards apart from some subtle differences. Since there are RFID systems [NXP07] that do not follow the standards, we need to find ways to assess the security level of these systems. These systems use protocols that are less accessible because of their proprietary implementation. One could argue that this inaccessibility is a feature because it hampers attackers. However, poor accessibility will only be a temporary obstruction. This does not give any security guarantee in itself. In essence, a device like the Proxmark is indispensable to investigate these (proprietary) RFID protocols. The following sections explain more about the ISO/IEC standards, protocol layers, signal modulation and encoding techniques.

### 2.1.1 The physical layer

The physical layer is the lowest layer of communication in the OSI model. This layer covers the actual bits and bytes that are eventually sent over the line or over the air. Contact-based smart cards are directly connected to the reader interface by their metal contact points. For this reason, the communication can be done using digital pulse modulation. The message is represented digitally and therefore it is not needed to use the analog domain as it is used for contactless smart cards. The digital

---

<sup>1</sup>The EPC Gen 2 Standard.

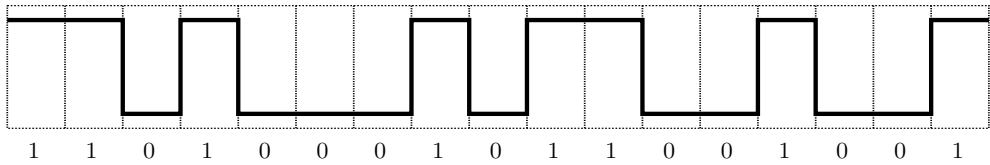


Figure 2.3: Non-return-to-zero encoding

messages are represented by high and low voltages. We will first briefly visit the input-output interface of contact-based smart cards and then elaborate more on the communication interface of contactless smart cards.

The single communication line that is used (half-duplex channel) for contact-based smart cards makes use of a pull-up resistor [ISO07a] (see  $R_1$  in Figure 2.2) to keep the signal level high when both the reader and card do not communicate (i.e., keep the switch  $S_1$  open). The voltage level floats to a high voltage when no party is communicating. The party that communicates connects the line to ground in order to send a '0' and releases it again to send a '1'. The resistor prevents that this shortcuts the circuit. Since the line is half-duplex, only one party can communicate at a time. For contact-based smart cards it is important to decide on the rules of communication such as clock speed, permitted voltage levels, etc. The actual bits that are transmitted over the line are encoded using non-return-to-zero encoding as shown in Figure 2.3. In this encoding a high voltage level is used to indicate a '1' and a low voltage level is used to indicate a '0'.

There are many more encoding and modulation variants in use for RFID systems, see Section 2.1.2 and 2.1.3 for more details. A contactless smart card is not physically connected to the logic circuit of the reader as a contact-based smart card is. An RFID reader creates an electro-magnetic field by generating a 13.56 MHz carrier wave. This carrier provides power to a card and is used for communication at the same time. The reader-to-card communication is in most cases achieved by interrupting the carrier wave for very short periods (a couple of milliseconds). A contactless card that is near the reader gets powered by induction. At the lowest level of the communication, the sender has to transform the information to an analog signal. This analog signal is then captured at the receiver side and transformed back to the original digital message that was initially sent. The main steps of this transformation consist of encoding and modulation. It is important that these steps can be reversed by first demodulating and then decoding the signal.

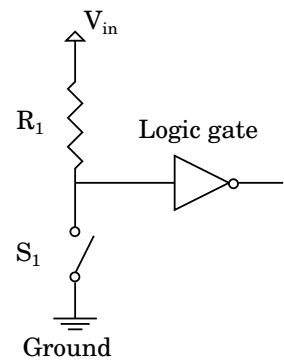
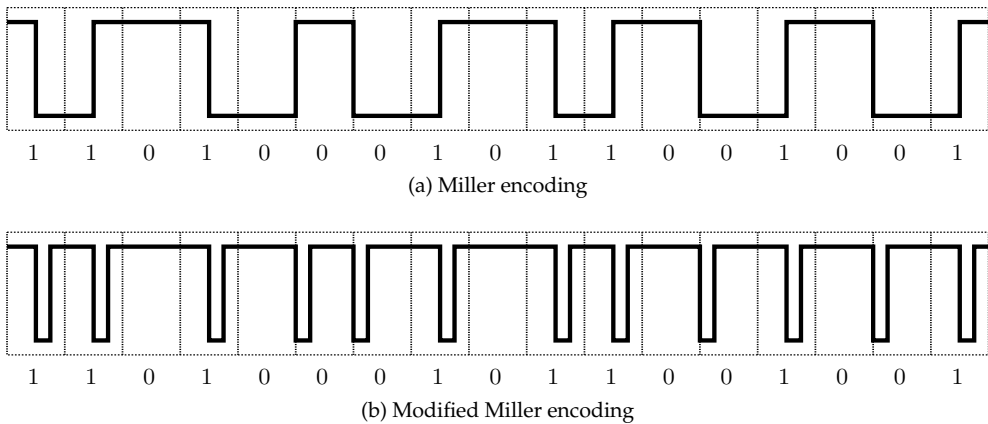


Figure 2.2: Pull-up

## 2.1.2 Encoding techniques

There are different encoding techniques and the ones that we discuss here are not solely used for RFID but are also applied in other fields of radio communication.



**Figure 2.4:** Encoding techniques

Sometimes the reader-to-card encoding is done differently than the card-to-reader encoding. For instance, in ISO/IEC 14443-A, Modified Miller encoding is used for reader-to-card communication and Manchester encoding is used for card-to-reader communication. This standard is widely used in many RFID systems, e.g., the Mifare Ultralight chip from NXP follows this standard up to the third level.

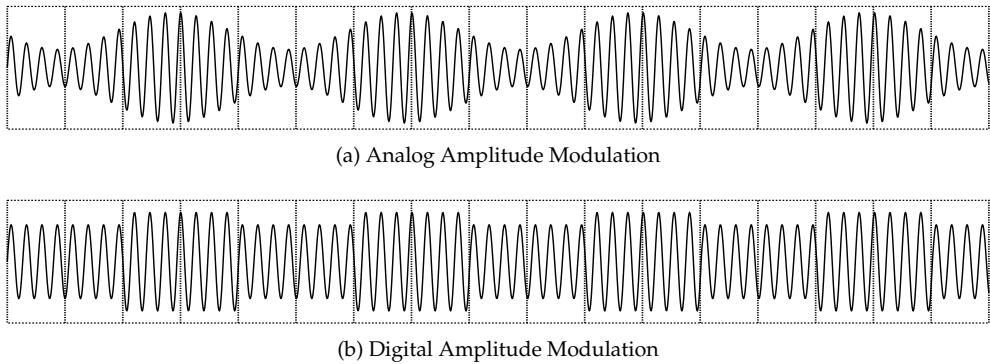
This section discusses the frequently used encoding techniques Modified Miller and Manchester. The former is used for reader-to-card encoding and the latter is used for card-to-reader encoding of messages in contactless smart cards that follow the standard ISO/IEC 14443A.

### **(Modified) Miller encoding**

As its name suggests, Modified Miller is a slightly adapted version of regular Miller encoding. In Miller encoding, bits are encoded by making transitions between two states of communication. In case of ISO/IEC 14443-A, this means that the reader differentiates over two amplitude levels. The transitions between these levels, a high and low level, are used to encode data bits. A fixed Elementary Time Unit (ETU) is used to indicate the bit length or bit period. The offset of the transition, w.r.t. the ETU timing grid, tells whether a binary one or zero is encoded. A '0' is encoded by no transition at all. Except when a '0' follows another '0', then a transition is made at the start of the ETU. A '1' is encoded by a transition halfway an ETU. An example of this encoding technique is shown in Figure 2.4a.

Recall that the card is powered by an electro-magnetic field of the reader and the generated carrier wave is also used to transmit data to the card. ISO/IEC 14443A uses 100% Amplitude Shift Keying (ASK) which means that the carrier drops out completely for some small period of time. During this drop, the Integrated Circuit (IC) does not get power from the reader and a capacitor is used to keep it running. In regular Miller, the encoded signal is as long high as it is low. This makes it unsuitable





**Figure 2.5:** Amplitude Modulation (AM)

to use with 100% ASK since the card would lose its power during the long periods where no carrier wave is generated. Modified Miller solves this issue by keeping the encoded signal high as much as possible. State changes in Modified Miller are encoded by short drops where regular Miller uses transitions. By default, the signal level is high. A '0' is encoded by a continuous high signal except for the case of consecutive zeros. Subsequent zeros are encoded by a drop right at the start of an ETU, see Figure 2.4b. A '1' is always encoded by a short drop halfway an ETU. As a result the encoded signal is high most of the time, which is useful since the card is also powered by this signal.

### Manchester encoding

The card also needs to communicate back to the reader. The encoding technique that is used for this in ISO/IEC 14443-A is Manchester encoding, which is applied using On-Off Keying (OOK). 100% ASK comes down to a subcarrier that is switched on or off and therefore it is equivalent to OOK. This on-off keying of the subcarrier is also known as load modulation. A 847.5 kHz subcarrier is used in card-to-reader communication. The encoding itself is straightforward as a '0' is encoded by load modulation during the first half and a '1' is encoded by load modulation during the second half of an ETU.

### 2.1.3 Modulation techniques

There are many modulation techniques. This section discusses the most common ones. Modulation is the technique of embedding a signal into a carrier wave. This signal can either be discrete or continuous. Modulation of a continuous signal (see Fig. 2.5a), or so-called analog modulation, is for instance used in radio broadcasting where the analog audio signals are modulated using Frequency Modulation (FM) or Amplitude Modulation (AM). In RFID discrete (or digital) modulation is used. The signal that is being modulated consists only of zeros and ones. For amplitude

modulation, this results in a transmission signal that has only two amplitude levels (Fig. 2.5b), unlike the continuous change of the amplitude level in analog modulation (Fig. 2.5a). All these modulation techniques use the characteristics of a waveform. The frequency of a wave is the number of periods that occur within one second. One period corresponds to one cycle of the wave. The frequency of a wave is expressed in Hertz, e.g., 1 Hz = 1 cycle/second. The amplitude of a wave is the deviation from its average value. The bigger the amplitude, the more energy the wave carries. The phase of a wave is the initial angle at the origin of a sinusoidal function. Changing the phase of a wave can be seen as shifting the wave in time. A carrier wave  $c$  can be described by the sinusoidal function  $c = A \cdot \sin(\omega t + \phi)$  where  $A$  is the amplitude,  $\omega$  is the frequency and  $\phi$  is the phase. These three parameters change the characteristics of the wave. Also, differentiations in an observed wave can be seen as changing  $A$ ,  $\omega$  and  $\phi$  values. When the sender is able to introduce these changes, and the receiver is able to detect these changes, it is possible to communicate information. Altogether, we have three basic ways to influence the radio communication. First, changing the amplitude  $A$ , this is known as Amplitude Shift Keying (ASK). Then, changing the frequency of the signal, which is known as Frequency Shift Keying (FSK). And finally, changing the phase of the signal, this is known as Phase Shift Keying (PSK).

There are different communication modes. *Simplex* communication is used in radio broadcasting where only one party is sending and many others are listening. In RFID a *half-duplex* channel is used between the reader and the card. In half-duplex only one of the parties can send at a time. As opposed to *full-duplex*, where both parties can send at the same time, e.g., communication within a telephone network.

### Amplitude Shift Keying

ASK uses changes in the amplitude to modulate a digital or analog signal into the carrier wave, see Figure 2.5b. The most simple form of ASK is On-Off Keying, also known as 100% ASK, and it basically comes down to switching the carrier wave on and off completely.

### Frequency Shift Keying

In FSK the frequency of the carrier wave changes over time in order to communicate information. In its most simple form it is called 2-FSK since two frequencies are used. There are several variants of FSK that use more frequencies which allows to communicate more than one bit of information during one ETU. The more complex variant 16-FSK uses 16 different frequencies where each frequency represents 4 bits of information. The effect of changing the frequency of the carrier wave and an example of 2-FSK is shown in Figure 2.6.

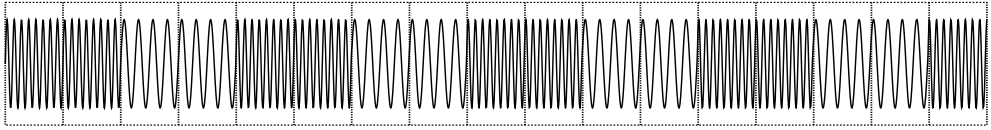


Figure 2.6: Frequency Shift Keying (FSK)

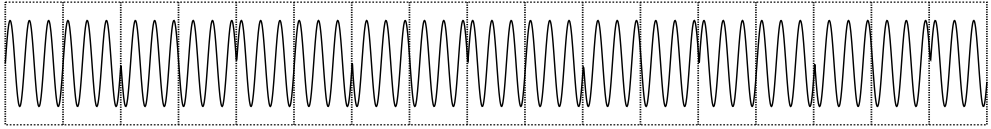


Figure 2.7: Phase Shift Keying (PSK)

### Phase Shift Keying

In PSK the phase of the carrier wave changes over time. In line with ASK and FSK the different phases encode information. A very simple variant of PSK is Binary Phase Shift Keying (BPSK). In this form of phase shift keying the signal only switches between two phases to encode information. An example of BPSK is shown by Figure 2.7. BPSK is used in the ISO/IEC 14443-B standard for card-to-reader communication. Here the initial phase  $\phi = 0$  and represents a logic '1'. A phase change of  $180^\circ$  indicates a transition of this logical value. Phase changes are positioned at the edges of an ETU.

## 2.2 Proxmark III

The Proxmark III, henceforth called the Proxmark, is a flexible RFID research tool that can emulate both an RFID reader and transponder, or can eavesdrop a conversation between a reader and transponder. In RFID communication the reader has a different role than the card in the sense that it also needs to provide power to the card. Also modulation, encoding and transmission speeds might differ for the reader and card in one system. The philosophy behind the Proxmark design is to do as much as possible in software. Roughly, this comes down to two antenna circuits that connect to an Analog-to-Digital Converter (ADC). The digital output of this signal is then routed through an FPGA which is responsible for the signal demodulation. The FPGA samples the digital signal and passes the sampled data on to a microcontroller. The microcontroller decodes the received samples in order to recover the digital message.



## HISTORY OF THE PROXMARK

The first version of the Proxmark was created by Jonathan Westhues in 2003. He described the first version as *a nice toy*, it was not even called Proxmark at that time. He started designing an improved second version. This second version should do signal demodulation in software opposed to the demodulation that was done in hardware with the first version. In 2005, the Proxmark II was ready and Jonathan noted that it was not possible to buy a similar device on the market. Given the fact that a tool like this is essential for experiments with RFID transponders, Jonathan announced to release the full design and software of the Proxmark at some point. Two years later, in May 2007, this resulted in the public release of the Proxmark III. In July 2007, we decided to order a Proxmark to use in a research project on the Mifare Classic that is described in Chapter 4. The first challenge was to implement the signal processing support for ISO/IEC 14443-A communication, see Section 2.2.2 below.



**Figure 2.8:** The Proxmark I

Source: J. Westhues, <http://cq.cx>

### 2.2.1 Hardware board

The Proxmark III consists of a four-layer circuit board which contains all the necessary components to do real-time signal processing of 13.56 MHz and 125-134 kHz signals. Figure 2.9 shows the Proxmark and its most important components. These components are briefly visited in the following sections.

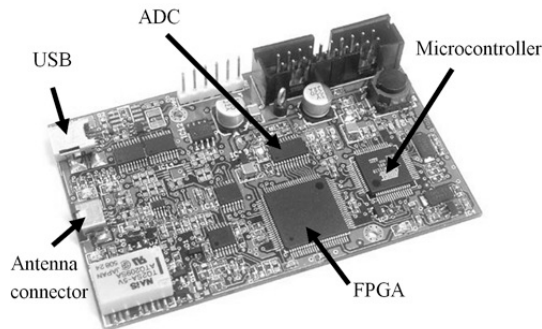
#### Antenna Circuit

The Proxmark has two separate on-board antenna circuits to cover a wide range of RFID applications. One antenna circuit is designed for the 13.56 MHz frequency and the second circuit serves the 125-134 kHz frequency range. These two circuits cover the high and low frequency RFID applications, respectively. The analog signal from the antenna circuit is led through an ADC that generates the digitized signal for the FPGA.

#### Field Programmable Gate Array

The FPGA is one of the main components of the Proxmark. An FPGA is a programmable IC that can be flashed with a logic circuit. The task of the FPGA is to sample data that it receives from the ADC and to send the sampled data to the Microcontroller. An implementation on the FPGA is much faster than a microcontroller implementation.

This speed advantage of the FPGA allows us to do several operations at every clock cycle of the 13.56 MHz wave. These operations might differ depending on modulation, encoding schemes, bit rates etc. The different requirements on the sampling of the incoming ADC data result in a need for different implementations. The FPGA solves this problem of varying requirements since it is reprogrammable. An FPGA implementation basically comes down to a description of a hardware circuit. The language that is used for this description is a hardware description language like Verilog or VHDL. The Proxmark project uses Verilog which was, like VHDL, originally designed to describe and simulate hardware designs. Later, its use got extended to the synthesis of hardware designs.



**Figure 2.9:** The Proxmark III

### Microcontroller

The microcontroller, an ARM processor, is used for the transport layer implementation of RFID protocols. Its major task is the signal encoding and decoding of reader and card messages. The Manchester encoding, for instance, is implemented on the microcontroller. Also, direct interactions and protocol steps are programmed in the microcontroller. For example, card emulation needs real-time decisions on how to proceed with the protocol. The conditions and different protocol branches are programmed in the microcontroller. This could be done on the PC side in the future. The current bottleneck is the communication speed between the Proxmark and the host PC. It uses the default Human Interface Device (HID) interface that is used for interfacing devices like the keyboard. The HID interface is too slow to define the protocol steps at the PC side. Especially the anticollision phase of the protocol imposes strict time constraints.

### 2.2.2 FPGA implementation

In this section we discuss the FPGA implementation of the ISO/IEC 14443-A standard in the Proxmark [dKG08]. We will discuss the `hi_iso14443a` module, which is the ISO/IEC 14443-A Verilog implementation for the Proxmark. In short, the FPGA is used for the modulation and demodulation part of the Digital Signal Processing (DSP). It samples at a clock speed of 13.56 MHz and sends the bits that describe the transitions of the modulated signal to the microcontroller. These bits are sent at a rate of 8 samples per period (847.5 kbps) for the reader-to-card signal. The FPGA code defines several modules that all make use of the same interfaces. A multiplexer switches between the FPGA modules that cover the high (13.56 MHz) and low (125-134 kHz) frequency processing. This separation of functionalities is needed because different circuitry is used for high and low frequencies. Every module runs with different parameters that are specific for sending, receiving and eavesdropping. The following modules (operating modes) are implemented for ISO/IEC 14443-A.

**Mode 0** *Eavesdropping*

Sample reader-to-card as well as card-to-reader communication. The sampling speed is 1.7 Mbps.

**Mode 1** *Card simulation (listening)*

Sample reader communication at 847.5 kbps.

**Mode 2** *Card simulation (transmitting)*

Modulate card-to-reader communication and generate a 847.5 kHz subcarrier.

**Mode 3** *Reader simulation (listening)*

Sample card-to-reader communication at 847.5 kbps and generate a carrier wave.

**Mode 4** *Reader simulation (transmitting)*

Modulate reader-to-card communication and generate a 13.56 MHz carrier wave.

The Proxmark switches seamlessly between these operating modes. This is especially important when switching between transmitting and listening in the reader simulation modes (mode 3 and 4). If the Proxmark stops generating a field for a short moment this will result in the card losing its power and thus its current state.

#### Analog-to-Digital Converter

The Analog-to-Digital Converter (ADC) transfers the analog input voltage of the antenna to a digital representation. The Proxmark uses an 8-bit ADC<sup>2</sup> which divides the analog input into 256 output steps. Two reference voltages on input pins `REFB`

---

<sup>2</sup>TLC5540 from Texas Instruments.

and REFT correspond to the bottom and top input of the input range, respectively. This input range is divided in equally sized digital steps.

The analog input signal is connected to the ANALOG-IN pin of the ADC. The digitized output signal is put onto eight parallel output lines (D1-D8) that connect the ADC and FPGA. These lines are clearly visible on the Proxmark circuit board and are shown as bold lines in Figure 2.10.

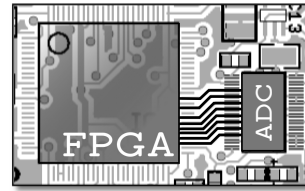


Figure 2.10: ADC-FPGA wiring

### FPGA interfaces

We will now discuss the most important input and output interfaces that are defined on the FPGA. A 13.56 MHz crystal is used for the high frequency FPGA modes. Every period of the carrier wave, the FPGA evaluates the digital output of the ADC and processes this information into samples that are at some point sent out to the microcontroller. Furthermore, the FPGA has two outputs (`pwr_lo` and `pwr_hi`) that are used to generate a low frequency (125-134 kHz) or high frequency (13.56 MHz) field. Apart from the carrier signal, the FPGA also controls four outputs that determine the level of load modulation on the signal. This is used for transponder emulation. Listing 2.1 gives a quick overview of the signals on the most important output pins for the different FPGA modes.

Listing 2.1: Overview of the interfaces

<code>mod_type</code>	<code>ssp_clk</code>	<code>pwr_hi</code>	<code>pwr_oe4</code>
3'b000 (0)	<code>clk2</code>	0	0
3'b001 (1)	<code>clk3</code>	0	0
3'b010 (2)	<code>clk3</code>	0	<code>clk3 &amp; mod_sig_coil</code>
3'b011 (3)	<code>clk3</code>	<code>clk1 &amp; mod_sig_coil</code>	0
3'b100 (4)	<code>clk3</code>	<code>clk1</code>	0

Where `clk1` = 13.56 MHz, `clk2` = 1.695 MHz, `clk3` = 847.5 KHz

### Decoding and sampling reader-to-card communication

Decoding reader communication is easier compared to decoding card communication, since the reader controls the field and can easily introduce signal changes. A card can only induce small changes in the reader field. The ISO/IEC 14443-A standard defines a default bit rate of  $f_c/128 \approx 106$  kbit/s with  $f_c = 13.56$  MHz. This means that one bit is transmitted every 128 clock cycles of the carrier wave. Recall from Section 2.1.2 that the reader-to-card communication for ISO/IEC 14443-A is encoded by Modified Miller. In this encoding scheme the signal is high at default.

Information is encoded by dropping the reader field for very short periods at certain time intervals. These drops last  $\frac{1}{4} \cdot 128 = 32$  clock cycles and occur at most once per bit period of 128 cycles. Following the Nyquist theorem [Nyq24, Nyq28] we sample at twice the frequency of our encoded signal. This theorem is about the transformation of a continuous signal into a discrete signal and vice versa. The discrete signal describes the continuous signal of a given frequency and needs to consist of samples that are taken at least two times of every sine wave of this frequency. This way it is always possible to reconstruct the continuous signal and accordingly this signal can be preserved in a discrete form. In our case, this means that we let the FPGA sample the reader-to-card communication every 16 cycles. These samples are sent in binary format to the microcontroller. The following bit string is an example of this communication from the FPGA to the microcontroller.

$$\dots \overbrace{1111} \text{SOE} \overbrace{00111111} \text{0} \overbrace{00111111} \text{0} \overbrace{11110011} \text{1} \overbrace{11110011} \text{1} \overbrace{11111111} \text{0} \\ \overbrace{00111111} \text{0} \overbrace{11110011} \text{1} \overbrace{11111111} \text{0} \overbrace{0011111111111111} \text{EOE} 1111 \dots$$

This bit string encodes the `REQA` command<sup>3</sup> which is sent out by the reader continuously when it is polling for new cards. It consists of 7 bits (a so-called *short frame*) and is like all reader frames enclosed by a Start-of-Frame (SOF) and an End-of-Frame (EOF).

### Decoding and sampling card-to-reader communication

In case of eavesdropping or reader simulation the Proxmark demodulates card signals. In order to do this, a more refined method is needed compared to the reader decoding. The main reason for this is that the card is powered by the field of the reader. It communicates to the reader by generating a subcarrier. This subcarrier affects the power level that is received by the Proxmark antenna. Although this induces changes in the power level, the changes are far more subtle and thus harder to detect. It is for this reason that reader communication can be eavesdropped at much larger distances than card communication. In the literature [WSRE04, Avo05], this is known as a forward (reader-to-card) and backward (card-to-reader) communication channel. In some contexts this property is used to achieve location-privacy by using the backward channel as a secure channel that cannot be eavesdropped by an attacker from a certain distance. Despite the weaker signal of the card the default bit rate is also 106 kbit/s. The communication is Manchester encoded as described in Section 2.1.2. During communication, half the bit periods have a modulated subcarrier, and half the bit periods do not contain any modulation. These periods last 64 cycles of the carrier wave and are twice as long as the signal drops in reader-to-card communication. However, the power variations are very small and the positioning of the card in the reader field highly influences the signal impedance. Therefore,

<sup>3</sup>The Request Type A command is used to select a card that (partly) supports ISO/IEC 14443-A.



a standard static threshold technique is not suitable in this situation. We use an *adaptive progressive thresholding* technique instead. It is a real-time problem where, at every period of the 13.56 MHz wave, the FPGA has to decide whether a transition took place or not. It is not feasible to reconsider earlier points in time. The thresholding technique that we use is adaptive since a variable threshold value is used at different points in time. Furthermore, it is progressive since it scans the digital signal for extremes and adjusts the thresholds according to these extremes.


The FPGA implementation averages the ADC output over 16 periods of the carrier wave. When the difference between two consecutive averages exceeds a threshold value it is counted as a transition. After every evaluation, the threshold value is updated to the difference between the last two averages. This corrects automatically for a too sensitive threshold value. On the other hand, a too insensitive threshold value is prevented by automatically resetting it to its initial value after a period without transitions. Concretely, it is impossible to have more than one bit period (8 samples) without transitions according to the ISO/IEC 14443-A standard. In our FPGA code we take some additional margin into account and reset the threshold after 16 identical samples. The following bit string that represents transitions in the subcarrier is an example of what is obtained by the FPGA and sent to the microcontroller.

... 0000 11110000 00001111 11110000 00001111 11110000 11111111 1100 ...  
                   SOF                  0                  1                  0                  1                  EOF

This bit string encodes the acknowledgement (ACK) of a Mifare Classic card to some reader command. It consists of 4 bits and is like all card frames enclosed by a SOF and an EOF.

PROXMARK COMMUNITY

Since its public release in 2007 a small community of RFID enthusiasts has further developed and extended the Proxmark functionality. After the release of the ISO/IEC 14443-A implementation and all the attention around the broken Mifare Classic, the demand for the Proxmark increased dramatically. Many people tried to reproduce the attacks that were found and started contributing to the Proxmark project. The Proxmark.org platform was launched to facilitate the enormous amount of requests and questions of RFID enthusiasts around this subject.



Source: <http://www.proxmark.org>

### 2.2.3 Demodulation

The modulation and demodulation of messages is done in the microcontroller. Basically, the microcontroller is the central control unit of the Proxmark and controls the FPGA modes, the communication with the FPGA and the host PC, and stores the traces in its EEPROM. Opposed to the FPGA code, the microcontroller code is more subject to change because of its central role. New functionalities, like the implementation of algorithms such as CRYPTO1 are mainly implemented in the microcontroller. The host software on the PC is compatible with different operating systems and provides a command-line interface to control the Proxmark. For further detailed reference on the hardware and software developments we refer the reader to the Proxmark community [VdKG09] and the Proxmark repository [Repa].

### 2.2.4 Other RFID research tools

The list of available open source RFID research tools is limited. Apart from the Proxmark, there are a few other hardware tools that got reasonable attention and have some overlap with the Proxmark functionality. There is the OpenPCD project that started back in 2005. Then, there is the RFID Guardian project that was mainly led by Melanie Rieback from the Vrije Universiteit Amsterdam. At the same time, the Radboud University developed a simple, inexpensive but very effective RFID emulator called the Ghost.

#### The OpenPCD project

In 2005, Harald Welte and Milosch Meriac started developing a device that could passively eavesdrop RFID communication. This project can be seen as the forerunner of the OpenPCD project. The OpenPCD device, as shown in Figure 2.11, started to provide open access to the lowest levels of RFID communication. The focus was on a free toolchain support for RFID protocol verification. The hardware of the OpenPCD contains an RC632 reader chip from NXP. This chip does all the signal processing and also contains the Mifare Classic circuitry. Furthermore, it supports ISO/IEC 14443, ISO/IEC 15693 and ICODE. However, this version of the OpenPCD was not able to send arbitrary bits over the air. Like the Proxmark, this project intends to give complete control over the communication channel. Currently, the second version of the OpenPCD, which is not dependent on a NXP chip, is being developed.

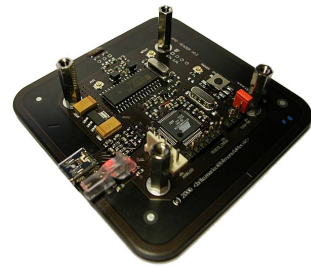


Figure 2.11: The OpenPCD

## RFID Guardian

This project started in 2004. Both, the TU Delft and the Vrije Universiteit Amsterdam were involved in this project. Many people contributed to the development of the RFID Guardian. The project was sponsored by *Stichting NLnet*, a Dutch foundation that stimulates network research and development. The last funding period ended in 2010 and since then no new sponsors have been attracted.

Because of the large amount of expertise it was possible to design and build a highly modular tool with many interfaces. Moreover, the RFID Guardian has some advanced functionalities like

- a TFT touch screen to control the RFID Guardian;
- several interfaces like ethernet and bluetooth;
- surround sound.

At first sight, one would not expect these functionalities to be part of an RFID security tool. They are mainly there because the tool was initially not intended for security testing. Originally, the RFID Guardian was meant to be a compact, portable, electronic device that could eventually be integrated into a PDA or cell phone [RCT05]. Its main responsibilities would be auditing, key management, access control and authentication. Later, the scope of the RFID Guardian was extended to security testing [Rie08].

## The Ghost

The Ghost, depicted in Figure 2.13, was developed by the Radboud University Nijmegen and is based on a PIC microcontroller. Roel Verdult developed the firmware and host software for this device. The Ghost can be programmed with an arbitrary Unique Identifier (UID) in order to emulate an RFID card or tag. The Ghost was used to demonstrate a proof of concept cloning attack of the Mifare Ultralight that was used as a disposable train ticket in the Netherlands [Ver08a, Ver08b]. The genuine ticket was only valid for two rides. The Mifare Ultralight did not use any cryptographic protection and was generally a piece of memory that could be written. So-called lock bits were used to indicate the card as used. Since the Ghost could emulate the memory, it could also reset the memory to a previous state, resulting in free traveling.



Figure 2.12: RFID Guardian

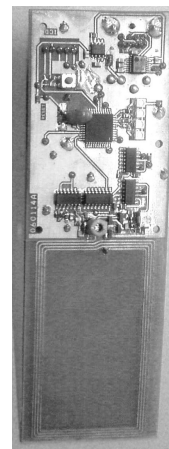


Figure 2.13: The Ghost

## 2.3 SmartLogic

The SmartLogic is a tool for contact-based smart card research which implements the ISO/IEC 7816 standard. The main focus of the SmartLogic is to provide a flexible setup. Part of its flexibility is achieved by its client-server architecture. The server controls a standard smart card reader with a genuine smart card or it emulates a smart card. The client, equipped with the SmartLogic hardware, connects to the server to get access to either this genuine smart card or an emulated one. The SmartLogic is placed between a card and reader in order to get control over their communication, see Figure 2.14. Any request from a smart card reader gets for-

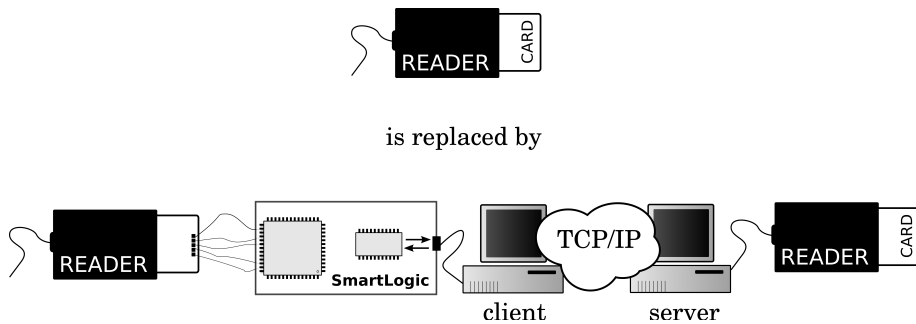


Figure 2.14: Basic SmartLogic setup

warded by the client to the server. The server controls the request, i.e., can forward, block or modify the request. The server constructs a response that is just based on an eventual response from the genuine smart card or a self-created response. Again, the server can modify or block this response. The server might even execute intermediate requests to the smart card before a final response is sent to the client. The client just communicates the server's response to the smart card reader. This architecture allows to set up relay attacks over the Internet and allows to share one smart card between different clients, at different locations. The main components of the SmartLogic hardware are an FPGA and an USB microcontroller. No knowledge about the hardware is required in order to implement the different attack scenarios, i.e. no microcontroller or FPGA programming is needed since the lower level ISO/IEC 7816 implementation is already implemented.

In the remainder of this chapter we will discuss the different components of the SmartLogic. First, a short introduction to the ISO/IEC 7816 standard is given. This standard specifies the physical characteristics and the different protocol levels of contact-based smart cards. The implementation of the SmartLogic hardware follows the ISO/IEC 7816 protocol specification. Then, the SmartLogic setup and hardware components are visited. Followed by a description of the software and functionalities of the SmartLogic. Finally, we compare the SmartLogic to other smart card research tools.

### 2.3.1 ISO/IEC 7816

The ISO/IEC 7816 standard defines *identification cards* as Integrated Circuit Cards (ICCs) with contacts. This section presents a quick overview of the standard. For a more detailed description we refer to [ISO07a, ISO07b]. ISO/IEC 7816 was introduced in 1998 and defines circuit cards (or smart cards) on different levels. We will mainly focus on Part 3 [ISO07a] where the smart card contact interface is explained. This part mainly covers the physical layer of the protocol. Besides normative references, electrical characteristics and some basic card operation procedures, the ISO/IEC 7816 standard defines the transmission protocols T=0 and T=1.

#### Answer to Reset

The *Answer to Reset* (ATR) is always sent after a signal on the reset pin (one of the eight contact points) on the card. The ATR conveys information about the supported protocols and possible configurations. Protocols are referred to by T= $x$  where  $x$  is the transmission protocol number. There are 15 possible transmission protocols, but only T=0 and T=1 are well known.

#### T=0 and T=1 protocols

The SmartLogic supports both the T=0 and the T=1 protocol. The T=0 protocol is widely used in smart cards and implements half-duplex transmission of bytes. It is configured in a master-slave setting where the reader is master and the card is slave. A reader command consists of five bytes, also known as command APDU<sup>4</sup>. This is defined in more detail in ISO/IEC 7816-3 [ISO07a].

CLA	INS	P1	P2	P3
-----	-----	----	----	----

Here CLA is the class byte which indicates the class of the command. Then, INS is the instruction byte followed by two parameter bytes P1 and P2. Finally, P3 is the length byte which indicates the length of an optional data message. Depending on the command, a data message might be sent by either the reader or the card. The card indicates that it is ready to send or receive data by repeating the instruction byte of the corresponding command APDU. The T=1 protocol specifies a half-duplex transmission of blocks. It is very similar to T=0 and its main difference is that commands and data messages are wrapped into blocks. These blocks contain header information and a final check byte.

### 2.3.2 SmartLogic setup

A typical SmartLogic setup is shown in Figure 2.15. Here, the ZTEX board (3) is the hardware that emulates the smart card communication down to the physical protocol layer. This board is directly connected to a smart card interface board (2) that can

<sup>4</sup>Application Protocol Data Unit

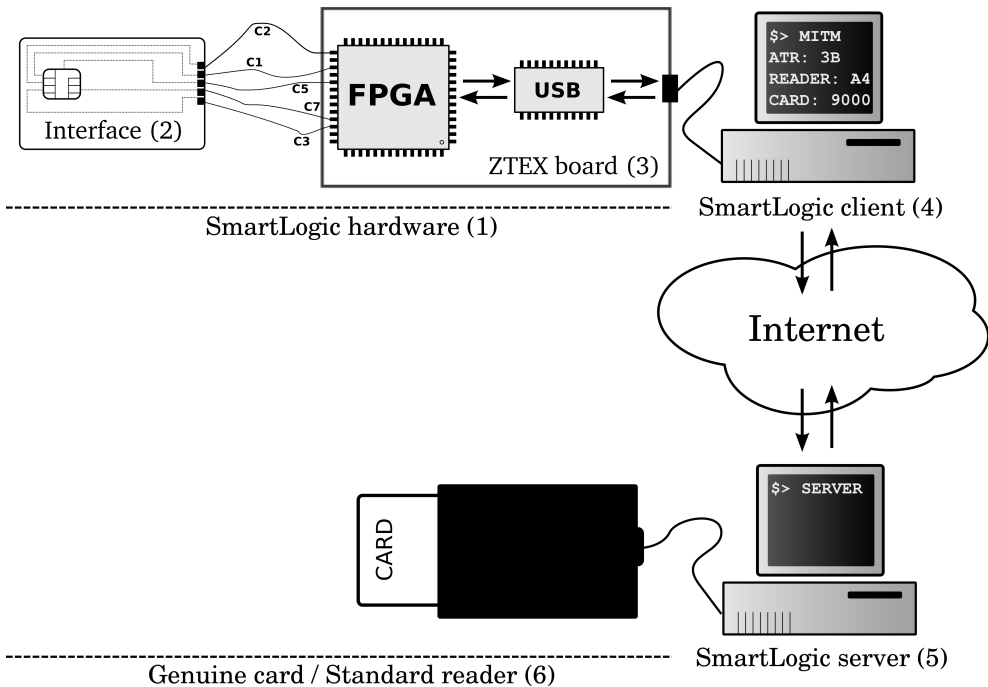


Figure 2.15: SmartLogic setup

be inserted into a genuine card terminal. A PC connects to the SmartLogic hardware (1) by its SmartLogic client software (4) and relays the smart card communication to the SmartLogic server (5). The SmartLogic server runs a standard smart card reader with a genuine smart card (6) and controls all communication to it. Note that it is possible to run the SmartLogic client and Server on the same host machine.

There are some major advantages of the SmartLogic setup. First, the application layer of the protocol is implemented in the host software of the SmartLogic client and Server. All protocol logic on this level is written in Java. The low-level parts like the FPGA firmware and the microcontroller firmware remain unmodified, i.e., there is no need to flash or re-program the hardware. Another advantage is that the SmartLogic server is able to emulate a card or may function as a proxy that caches smart card communication. Finally, this architecture allows to share one smart card at several different physical locations at the same time. This is useful in research on location based systems.

### 2.3.3 Hardware

The general purpose hardware (see Fig. 2.15) that forms the basis of the tool is an FPGA evaluation board from ZTEX [Zie]. The ZTEX board is a small programmable device that is equipped with an FPGA and USB microcontroller chip. There are

different versions of the ZTEX board available<sup>5</sup>. This section describes the main hardware components which is a combination of a smart card interface board and the ZTEX board.

### Smart card interface board

The smart card interface board contains the smart card connectors on the physical locations as described in the ISO/IEC 7816 standard. The connectors of the smart card circuit board are depicted in Figure 2.16. The power (VCC) and ground (GND) connectors are not connected but only use the circuitry and ground from the ZTEX board. Connectors C4, C6 and C8 are not used and the remaining three connectors, reset (RST), clock (CLK) and input-output (I/O), are connected to the ZTEX board.

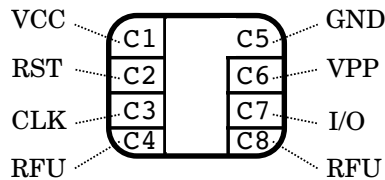


Figure 2.16: Smart card interface

### ZTEX board

The main component of the ZTEX board is an FPGA. The hardware design is described in the hardware description language VHDL. The input-output line is a half-duplex channel which makes use of a pull-up resistor.

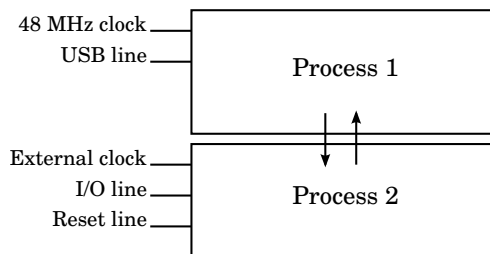


Figure 2.17: FPGA processes

The FPGA is responsible for the communication on bit level. Since it uses the clock of the genuine card terminal it is able to find the correct bitrate by counting the clock cycles. By default one bit period is 372 clock cycles according to the ISO/IEC 7816. Concretely, the FPGA runs two processes, see Figure 2.17. The first process

<sup>5</sup>We tested the SmartLogic with ZTEX USB-FPGA-Module 1.2 and 1.11c.

controls the input and output buffers that contain raw bits for sending and receiving. This process runs at the main clock frequency of 48 MHz. The second process uses the clock of the external genuine card terminal. The speed of this process varies but is typically a couple of megahertz and it makes sure that the bits are communicated at the right speed on the I/O line (C7). The communication between the FPGA and the SmartLogic client is controlled by the USB chip.

### 2.3.4 Software

The software for the SmartLogic consists of a client and server implementation in Java. The SmartLogic client controls the ZTEX board and establishes a connection with the SmartLogic server. In its turn, the SmartLogic server runs a standard smart card reader and controls all communication to a genuine smart card. Alternatively, the SmartLogic server can be programmed such that it emulates a smart card.

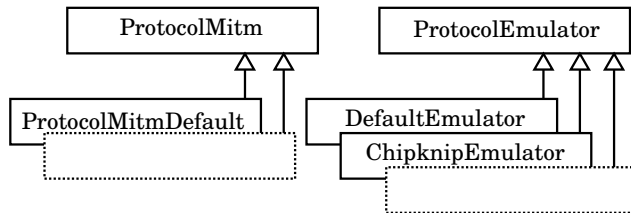


Figure 2.18: Man-in-the-middle and emulator Java classes

Figure 2.18 shows two abstract classes *ProtocolMitm* and *ProtocolEmulator* that are used in the SmartLogic server. These classes can be extended for a specific application like the man-in-the-middle attack on the EMV protocol that is presented in Chapter 3. There are some example implementations available [Repb] like *ProtocolMitmDefault* that intercepts and displays plaintext Personal Identification Number (PIN) codes in an alert box. In order to set up new man-in-the-middle or emulation setups, only an extension on either the *ProtocolMitm* or *ProtocolEmulator* class has to be written.

### Smart card sharing

One of the features of the SmartLogic server is that it can accept multiple connections. Depending on the smart card application that is being shared, some additional checks are needed to keep track of the different states per client. For example, the SIM application contains a directory structure. First, the directory is chosen by a *select* command. Only then *read* or *write* commands can be issued. The server needs to keep track of which client wants to access which directory.



### Set Answer-to-Reset

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

### Speed and baudrate detection

It is not necessary to set the baudrate for the SmartLogic since it connects directly to the clock of the genuine card terminal. Other tools, such as the RebelSim, need first to be configured with a baudrate since the baudrate varies for different smart card readers. By using the internal clock of the ZTEX board, which runs on 48 MHz, and comparing its difference in cycles with the clock of the genuine card terminal, it is possible to approximate and communicate the clock speed of the terminal to the PC.

#### 2.3.5 Other smart card tools

Here we describe other smart card tools that share a subset of functionalities with the SmartLogic. Some of the tools are not publicly available, while others are limited in their functionalities or are targeted at specific card types like Subscriber Identity Modules (SIMs). Figure 2.19 gives an overview of non-commercial smart card research tools. The first column indicates whether the tool is publicly available. By publicly available we mean that the hardware and software are open source or available for sale. By *eavesdropping* we mean that it can passively overhear the smart card communication. Support for an active man-in-the-middle setup is indicated by the column *active MitM*. The column *baudrate detection* refers to the ability of the tool to automatically detect the data transfer rate. This is essential for a correct interpretation and manipulation of the intercepted data. Some tools explicitly need to be configured at the right speed as they cannot detect this automatically. Furthermore, *distance relaying* means that the terminal and card are not required to be at the same physical location, e.g. the communication is relayed over the Internet. Finally, a tool supports *sharing* when it is possible to use one smart card simultaneously at multiple locations.

The *RebelSim APDU Scanner* [Reb12] can be used to passively eavesdrop the communication between a smart card and a reader. As its name suggests the main focus of the RebelSim is on SIM cards, presumably to analyse and undo SIM locking. It provides SIM interfaces as depicted in Figure 2.20. The communication is intercepted with a Universal Asynchronous Receiver Transmitter (UART) chip and can be read out using standard terminal software. A drawback of the RebelSim is that the baudrate needs to be set beforehand in order to capture the communication.

The *Osmocom SIMtrace* [OSM12] is a tool from the OsmocomBB project. This project aims to produce an open source GSM baseband software implementation.

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

Figure 2.19: Smart card research tools

The Osmocom SIMtrace tool is used within this project to eavesdrop on communication between a SIM card and a mobile phone. It uses the SIM connector from the RebelSim (see Fig. 2.20).

Examples of active man-in-the-middle tools are the *Leon Device* [Leo12], developed at the University of Michigan, and the *Season3* [SEA12]. As far as we know no hardware design or software for the Leon Device has been made public. The Season3 can be controlled over a serial connection where the baudrate needs to be pre-configured.

Lastly, the *Smart Card Detective* (SCD) [Cho10b] is a more recent tool that supports active man-in-the-middle attacks. The SCD has been developed by Choudary as a hand-held EMV interceptor. The resulting traces can be stored in EEPROM which is read out over a USB connection. Although the SCD was designed specifically for EMV protocols, it might be used for other protocols as well. However, this requires modification of the firmware. Both the hardware design and the software for the SCD are publicly available.

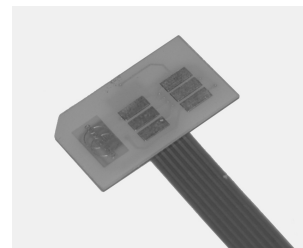


Figure 2.20: SIM interface

## 2.4 Conclusion

In this chapter we have presented two hardware tools, the Proxmark and the SmartLogic. Both tools are generic and highly flexible smart card research tools for which the hardware and software are open source and publicly available.

The Proxmark covers the contactless smart cards, while the SmartLogic covers the contact-based smart cards. Both tools provide access to and control of the protocol stack down to the lowest layers. Their flexibility mainly stems from the fact that both hardware implementations make use of an FPGA. This allows to add new

communication standards and protocols.

The purpose of these tools is to provide a platform for protocol verification on the implementation level of a system. In the next chapter we show that the SmartLogic can be used to execute common attacks like active man-in-the-middle and relaying. Other attack scenarios are emulation, passive eavesdropping and timing attacks. The Proxmark played a crucial role in the research on the Mifare Classic and iClass cards that is described in Chapter 4 and 5.



## Chapter 3

---

### Case study: Smart cards in practice

*“It is so characteristic, that just when the mechanics of reproduction are so vastly improved, there are fewer and fewer people who know how the music should be played.”*

Ludwig Wittgenstein

Reproducing attacks and testing protocol implementations is an important functionality of the SmartLogic. This chapter describes some use cases that demonstrate its different functionalities. The first use case is about a security review on an EMV payment terminal. EMV stands for Europay, Mastercard and VISA (EMV), the companies that initially started to work on the EMV specifications in 1994. These specifications define the use of Integrated Circuit Cards (ICCs) in payment systems. Ultimately, chip based payment systems should replace the magnetic stripe systems and provide better fraud protection. In so-called skimming attacks criminals copy the card information and harvest the corresponding Personal Identification Number (PIN). Since magnetic stripe cards are rather easy to copy, comparable to copying audio cassettes, and the ‘return on investment’ is high, skimming vastly increased over the last few years. Currently, many payment systems move from magnetic stripe to ICC based systems. Official EMVCo figures<sup>1</sup> of the last quarter of 2011 show that 1.5 billion EMV cards are deployed worldwide. Chip based cards provide better protection against fraud and provide several cardholder verification methods. Also, transactions can be signed by the card, which makes it most suitable for off-line transactions. If an EMV payment system is correctly employed this should bring protection against counterfeit cards since the card information is signed by the card issuer. Furthermore, when properly used, it protects against PIN harvesting since the PIN is sent encrypted to the card. Also, skimmed data cannot be reused by criminals since every transaction involves some random data and transaction counters. Important to mention here is that the EMV specification leaves room for different implementation choices. Interoperability is an important aspect here, but the deployed systems are no exact copies of each other. In other words, an attack on an EMV implementation in one system does not automatically mean that the attack can be reproduced on another EMV system. Several attacks have been published [MDAB10,BBLF11] that underline the exceptional care that is required in these implementations. In the first use case we reproduce the attack of Barisani [BBLF11] where the PIN code is sent as plaintext instead of encrypted.

---

<sup>1</sup><http://www.emvco.com>

The second use case is on an extended version of EMV called EMV-CAP. The abbreviation CAP stands for Chip Authentication Program and is used in internet banking applications. In this use case we use the SmartLogic to eavesdrop and replay messages between an internet banking token of a large Dutch bank and a bank card. In this use case we used the SmartLogic to keep some card messages in the protocol constant to check different operating modes of the internet banking token. It confirmed a flaw in the internet token.

The third use case consists of relaying smart card communication over large distances. For this use case we used the Chipknip, a Dutch electronic cash scheme where credit is stored on the card itself. The system is expected to be phased out during the coming years<sup>2</sup>, partly because of the introduction of the EMV chips and contactless payment schemes. However, it is still used in parking, catering and vending machines. Using the SmartLogic, we relay the communication of a payment at a vending machine over 20 km. So, the physical card was 20 km away, while the vending machine was convinced it was communicating directly with the card. In this setup we mimic a skimming attack where direct access to a card is immediately exploited over a great distance. In a relay attack like we demonstrate here the actual protocol is not attacked but only the distance between the participants is increased. This setup demonstrates the feasibility of replay attacks and underlines the need for distance-bounding protocols. Elaborating on this use case we show that a distance of 10.000 km can be bridged.

We conclude with two smaller use cases that do not reveal that much but merely demonstrate the capabilities of the tool. In the third use case we use the SmartLogic to emulate a Chipknip. First, we eavesdrop and learn parts of the Chipknip protocol. Then, we emulate a Chipknip using this eavesdropped data and replay parts of the payment protocol. Finally, the last use case demonstrates that multiple clients can be connected to and share one smart card. We use this setup to share one Subscriber Identity Module (SIM) between two mobile phones. This is realized by two hardware interfaces that connect over TCP/IP to a server which regulates access to the original SIM.

### 3.1 An active man-in-the-middle attack on EMV

In this section we discuss a man-in-the-middle attack on an EMV protocol. The attack was presented by Barisani et al. [BBLF11] using dedicated hardware. Usually these kind of attacks are not reproduced because it is a time consuming and tedious job. The SmartLogic lowers the effort needed to mount such an attack. By writing an extended class (`ProtocolMitmEMV`) in Java, we were able to reproduce the attack.

---

<sup>2</sup><http://www.chipknip.nl>

### 3.1.1 The EMV protocol

EMV is a specification for electronic payment systems using smart cards [EMV08a, EMV08b, EMV08c, EMV08d]. The initiative for EMV was taken by Europay, MasterCard and Visa in the 1990s. Currently the EMV standard is maintained by EMVCo, a company jointly owned by MasterCard, Visa, American Express, and JCB. An EMV session consists of four phases:

1. *initialisation*
2. *data authentication* (optional)
3. *cardholder verification* (optional)
4. the actual *transaction*

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

The *data authentication* phase is used for authentication of data that is stored on the smart card. It includes a signature over static data, e.g., the account number and expiry date of the card. The optional, dynamic part of the authentication (DDA) is achieved by either running a challenge-response protocol or retrieving a signature over the transaction data. If the verification of the signature on the static data fails, the dynamic part is not performed and the data authentication is aborted.

There are several methods for *cardholder verification* of which only two involve the card in the verification process: off-line encrypted PIN and off-line plaintext PIN. Off-line plaintext PIN is supported by terminals to be compatible with smart cards that do not support asymmetric cryptographic operations. In the initialisation phase the card provides a list with supported Cardholder Verification Methods (CVM List) in order of preference and with possible additional conditions. The data of this CVM list is stored on the card and is in all Dutch banking cards that we have seen part of the static data authentication (SDA). For example, the preferred cardholder verification method of Dutch banking cards is to use off-line encrypted PIN instead of off-line plaintext PIN.

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

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

### 3.1.2 The attack

Recently, a method was shown by Barisani et al. [BBLF11] to force a rollback from encrypted PIN to plaintext PIN using a man-in-the-middle attack. Their method makes use of the fact that after a failed data authentication, the transaction might still continue and be performed online depending on the Action Codes. In the attack, the CVM List is modified such that off-line plaintext PIN is the preferred method for cardholder verification. The Issuer Action Codes are modified such that in case of a failed data authentication the transaction is not aborted but performed online. Since modifying the CVM List and Issuer Action Codes might result in a failed data authentication, it depends on the Terminal Action Codes whether the transaction is then aborted or performed online. If the transaction is not aborted, the terminal continues with the data it received from the card, including the modified CVM List, as it cannot tell which data was modified. This results in the PIN code being sent to the card in plaintext, which can then be intercepted by a man-in-the-middle.

### 3.1.3 Using the SmartLogic

To test whether the forced fall-back could also be applied in the Netherlands we used the SmartLogic to intercept and modify the communication between a Point-of-Sale (POS) terminal and a Dutch banking card. The Dutch banking card supported the challenge-response mechanism as data authentication method. After the presentation by Barisani et al. [BBLF11], the Dutch banks rolled out a fix to their terminals to prevent the attack. In the initialisation phase we modified the CVM List and Issuer Action Codes, both of which are included in the static data that is retrieved during the data authentication phase.

The Action Codes are modified such that on failed data authentication the transaction is performed online. The rest of the communication is passed on unchanged. In Listing 3.1, part of an original transaction is compared with a modified one. The modified data in the READ RECORD command is indicated in bold. The two CVM bytes are changed from

42	<b>Enciphered PIN verified online</b>
01	<b>If unattended cash</b>

to

01	<b>Plaintext PIN verification performed by ICC</b>
00	<b>Always</b>

which tells the terminal that the card can only do plaintext PIN. The first byte of *Action Code - Online* is changed to FF in order to trigger an online transaction in case of errors. After modifying the data, the challenge-response part of the data authentication was no longer performed.



**Listing 3.1: EMV payment: card authentication and PIN verification**

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

This is as expected, as data authentication already fails when verifying the signature over the static data. The transaction was denied by the bank. There are several reasons why the transaction could be denied. In the challenge-response part of DDA, the card's nonce (amongst other data) is sent encrypted to the terminal. This nonce is then sent back to the card in the transaction phase. This might indicate that this nonce is used in the MAC that is generated by the card to sign the message. Since DDA is not performed completely, the terminal does not know the card's nonce and thus cannot send this to the bank. If the nonce is used in the MAC, the bank will not be able to verify this. This might be a reason why the transaction is denied. A solution to this might be to retrieve the nonce from the card by performing the challenge-response mechanism by the SmartLogic. This nonce could then be provided in one of the messages to the terminal. For this to work we would however need the public key of the certification authority (in our case MasterCard), which

we did not have access to. Another possible reason might be that the transaction is denied because data authentication failed.

Although the transaction was denied by the back-end, the modification of the data still resulted in a plaintext PIN code transmission (see Listing 3.1) to the card, as opposed to the claim of the Dutch banks. According to them, the terminal we encountered was one of the few terminals that had not been patched yet [PRI11]. Although earlier it was claimed that all everyday use terminals were fixed [Ess11], we demonstrated that this was not true.

### 3.1.4 EMV attack implementation

The implementation listed in Listing 3.2 verifies the described attack on a Dutch EMV payment terminal. The extended class `ProtocolMitmEMV` sets up a man-in-the-middle using the `SmartLogic` as described before. The class implements the `getResponse` function (line 16) for all reader requests. By default this function forwards all communication without any modification. The function has access to a genuine card and can make requests to it. It may drop or modify the request from the reader, or make intermediate requests before it constructs its final response. Furthermore, this function can modify the answer before it is returned. In case of our attack the card information needs to be modified such that the terminal believes that the card only supports plaintext PIN. To do this, all reader requests are handled normally except for the `READ RECORD`. We check for this command by its byte representation `0x00B2010C8A` (line 36) after we have checked that there is a response at all (line 29). Then, a simple replacement is made for the bytes that need to be modified (lines 42-44). This attack implementation resulted in the traces that are listed in Listing 3.1. Eventually, the terminal did send the PIN in plaintext to the card, which was of course intercepted and stored by our man-in-the-middle setup.

Listing 3.2: EMV attack implementation of plaintext PIN interception

```

1 import javax.smartcardio.Card;
2 import javax.smartcardio.CommandAPDU;
3 import javax.smartcardio.ResponseAPDU;
4 import net.sourceforge.scuba.smartcards.CardService;
5 import net.sourceforge.scuba.smartcards.CardServiceException;
6
7 class ProtocolMitmEMV extends ProtocolMitm {
8
9     public ProtocolMitmEMV() {
10         this.setActivated(true);
11     }
12
13     public void reset() {
14     }
15
16     public byte[] getResponse(CardService card, byte[] readerMessage) {
17         byte[] empty = {};
18         byte[] reply;
19         CommandAPDU command;
20         ResponseAPDU response;
21
22         reply = empty;
23
24         try {
25             command = new CommandAPDU(readerMessage);
26             response = card.transmit(command);
27             reply = response.getBytes();
28
29             if (readerMessage.length == 5 && reply.length > 0) {
30                 byte CLA = readerMessage[0];
31                 byte INS = readerMessage[1];
32                 byte P1 = readerMessage[2];
33                 byte P2 = readerMessage[3];
34                 byte P3 = readerMessage[4];
35
36                 if ( CLA == (byte) 0x00 &&
37                     INS == (byte) 0xB2 &&
38                     P1 == (byte) 0x01 &&
39                     P2 == (byte) 0x0C &&
40                     P3 == (byte) 0x8A) {
41
42                     reply[46] = (byte) 0x01;
43                     reply[47] = (byte) 0x00;
44                     reply[75] = (byte) 0xFF;
45                 }
46             }
47         } catch (Exception e) {
48             reply = empty;
49         }
50
51         return reply;
52     }
53 }

```

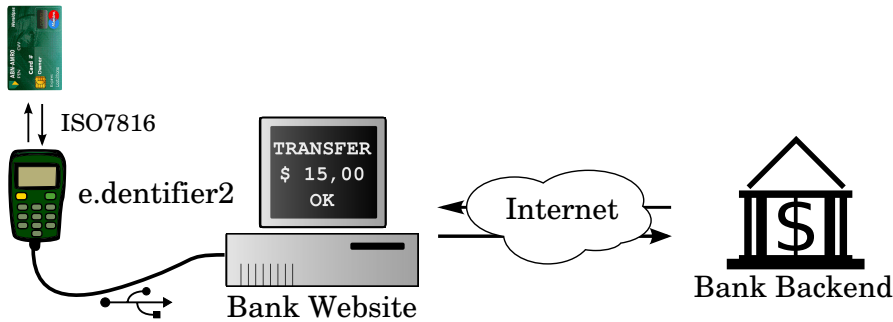


Figure 3.1: Setup for internet banking with the e.dentifier2

## 3.2 Security tokens for internet banking

In this use case we used the SmartLogic to study internet banking tokens. The token-based internet banking systems that we have studied use a variant of EMV-CAP, a proprietary standard of Mastercard that is widely used for internet banking.

In EMV-CAP, a regular EMV transaction is started but cancelled in the last step. In the course of an EMV-CAP transaction a smart card generates two so-called *Application Cryptograms* (ACs) as proof of authorisation. The first cryptogram, an *ARQC* (Authorization Request Cryptogram), is used as authorisation of some online banking transaction. The second cryptogram, an *AAC* (Application Authentication Cryptogram), just serves to cleanly terminate the transaction.

### 3.2.1 The e.dentifier2

One of the devices we investigated is called the e.dentifier2. It is a smart card reader with a display and keyboard. The display can contain up to 68 characters. The keyboard provides numeric keys, an OK, and a Cancel button. The e.dentifier2 can be used for online banking with or without USB connection. We found an attack on the connected mode.

To use the connected mode, customers have to install a special driver (only available for Windows and MacOS). The web-browser then interacts with this driver via JavaScript and a browser plugin. The browser plugin checks whether it is connected to the right backend of the bank (a webserver) using SSL, see Figure 3.1. If this is not the case, the plugin will not function.

In connected mode, an internet banking session starts with the reader reading the bank account number and the card number from the smart card and supplying it to the web-browser. So, the user does not have to type this in, making the system more user-friendly. To log in, the reader first prompts the user for his PIN code. It then displays a message saying that the user is about to log in and asks the user to confirm this by pressing OK (see Fig. 3.2a). To confirm a bank transfer, or a set of bank transfers, the reader will again prompt the user for his PIN code. It then

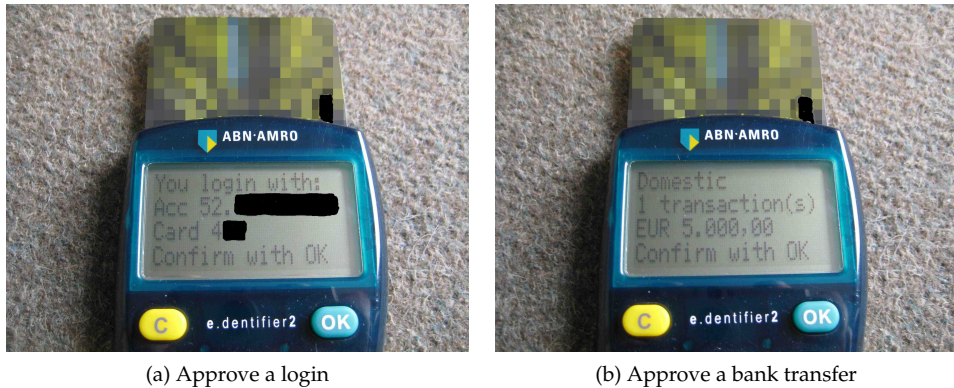


Figure 3.2: The USB-connected e.dentifier2

displays a message giving *the number of transfers* the user is about to approve and *the total amount of money* involved, and asks the user to approve this by pressing OK (see Fig. 3.2b). The additional security of the connected mode over the unconnected mode here is that *what-you-see-is-what-you-sign* (WYSIWYS), even when the browser or PC is controlled by malware.

### 3.2.2 Attack on the USB-connected mode

As can be seen in Figure 3.3, the reader sends a message to the host PC indicating the user pressed OK. After this the host PC sends a command to generate the cryptograms to the reader. This seems strange, as the reader would be expected to generate the cryptograms automatically after OK has been pressed. The driver on the host PC should not play a role in this.

This weakness can be exploited: by sending the request over the USB line to generate the cryptograms *without* waiting for the user to press OK, the cryptograms are generated and the reader returns the response over the USB line, without the user getting a chance to approve or cancel the transaction. To make matters worse, a side-effect of giving this command is that the display is cleared, so the transaction details only appear on the display for less than a second. We demonstrated this attack in an actual internet banking session.

This means that an attacker controlling an infected PC can let the user sign messages that the user did *not* approve, thus defeating one of the key objectives of *what-you-see-is-what-you-sign* (WYSIWYS). The user still has to enter his PIN, but this is entered at the start of a transaction, and after this no more interaction is needed from the user to sign malicious transactions.

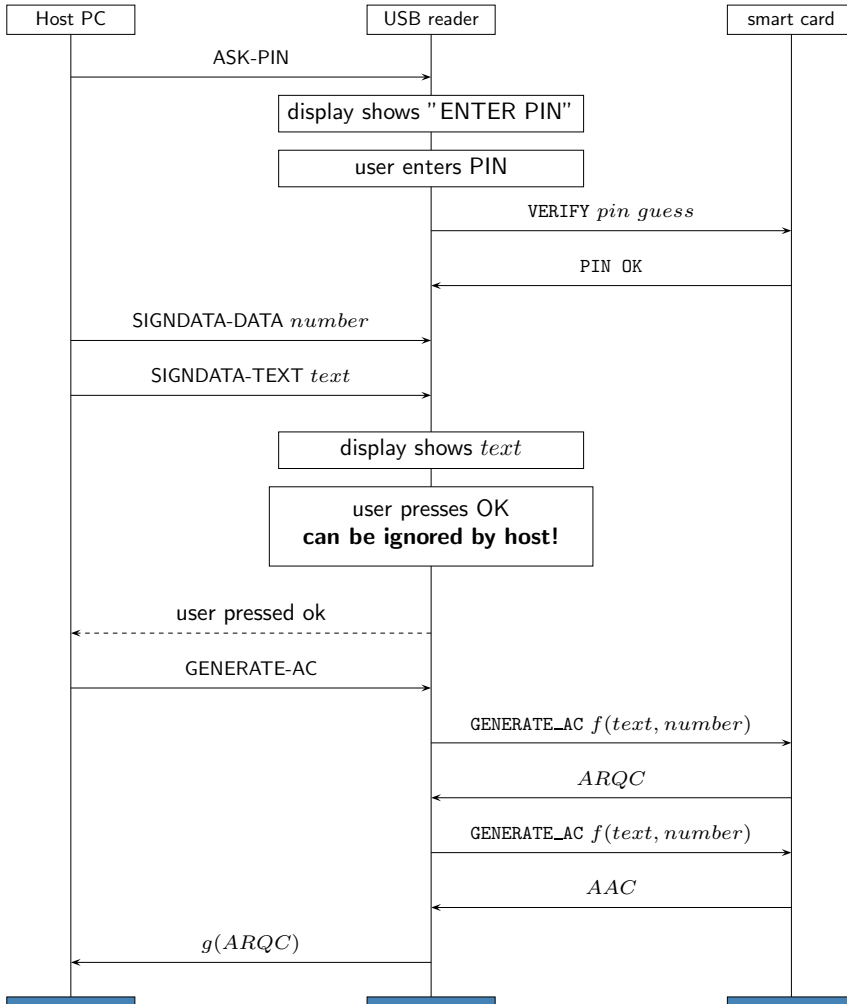


Figure 3.3: Attack on the WYSIWYS protocol of the e.dentifier2

We used the SmartLogic to monitor the behavior between the e.dentifier2 and the smart card. We observed that the protocol between the reader (e.dentifier2) and the smart card (see Fig. 3.3) was identical in both the cases, when the OK message was forced by the client or when the OK message was pressed by the user. The dashed arrow in Figure 3.3 indicates that this message can be ignored. When the host just continues and requests the cryptogram, the reader will continue as if the user did press OK. The functions  $f$  and  $g$  are unknown and could very well be some hash functions, we were not able to figure out how the output is computed from the input. Knowledge about  $f$  and  $g$  is no prerequisite in the aforementioned attack. We refer the interested reader to [BdKGP<sup>+</sup>12] for more detailed information on this attack.

### 3.3 Distance relaying

In a relay attack all communication between two parties is forwarded by a man-in-the-middle to a different physical location than originally intended. In a relay attack it is not needed to have any knowledge about the protocol itself apart from how the information is communicated. Even when the most powerful cryptographic solutions are implemented, when no proper countermeasures against distance relaying are taken it is possible to fool the participants about their mutual distance. For instance, somebody pays by using a banking card in shop A, not knowing that at the same time a bill is being paid for an attacker in shop B using the same banking card. An interesting question is to what extent the distance can be increased between a genuine card and the terminal. For this experiment we used the Chipknip, a Dutch smart card payment scheme. Comparable systems are, for example, the Geldkarte from Germany and Proton from Belgium. These systems are designed for micro payments and function like an electronic wallet. No PIN is needed to perform payments using a Chipknip card. The only transaction that involves a PIN is when money is transferred from a regular bank account to the Chipknip card. It is possible to buy anonymous Chipknip cards that are not linked to a bank account; these cards cannot be charged.

Listing 3.3: Relay over 20 km between Arnhem and Nijmegen

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

In a practical setup, we were able to relay a payment between two Dutch cities that are situated about 20 km apart. In Arnhem we installed a SmartLogic server with a genuine Chipknip smart card and in Nijmegen we used a SmartLogic client to buy candy from a vending machine. The client was connected to the Internet through a wireless connection and the server was directly connected through an ADSL connection. Round-trip times of the messages in the relayed protocol run are depicted in Listing 3.3.

Listing 3.4: Maximum measured delay times

Terminal/Reader	Provided clock speed	Waiting Time (WT)	Allowed delay
VASCO DIGIPASS 810*	1.05 MHz	3410 ms	3560 ms
e.dentifier2*	2.00 MHz	1790 ms	1910 ms
Ingenico 5300	4.91 MHz	730 ms	1100 ms
Chipknip Charging Terminal	3.69 MHz	970 ms	1200 ms
Chipknip Payment Terminal	4.92 MHz	730 ms	500 ms

\* These are Dutch e-banking devices.

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

Listing 3.4 shows the results of the observed maximal delay times. The second column shows the measured clock speed that the reader provides to the card. We measured the maximal response delay that was still allowed by the reader. This means that we delayed the responses up to the point where a protocol run was still successful. The results for the default configuration show that it is possible to run protocols over huge distances. For instance, we measured a round-trip time to an IP address in Osaka (Japan) of about 270 ms which is at a distance of approximately 10.000 km from our testing location. Note that the Chipknip terminal allows a delay that is almost twice as long.

To conclude, the waiting time is not a boundary when it comes to relay attacks. If good connection speeds are available at both the location of the client and the server, a relay attack can be mounted from one side of the world to the other. Several solutions to prevent relay attacks have been proposed and are known as distance bounding protocols [BC94, HK05, DM07, KAK<sup>+</sup>09, Han11]. Practical attacks like demonstrated in this section show the need for such protocol implementations.

### 3.4 Smart card emulation

Another functionality of the SmartLogic is to emulate a smart card. In this use case we show how this can be used to investigate an existing proprietary unpublished protocol like the Chipknip. In emulation mode the protocol messages are generated by the host PC. For this use case we focus on the two payment traces between the candy machine and a genuine smart card of Listing 3.5. Note that some of the instruction bytes (INS) look familiar and are likely to be derived from the generic Inter-sector Electronic Purse standard [BSI00]. However, there is no public document that describes this protocol.



Listing 3.5: Chipknip payment

Time	Sender	Payment Run 1	Payment Run 2	Info
+0335	READER.0	E1 B4 00 01 05	E1 B4 00 01 05	Balance: 20,00 euro 19,15 euro
	CARD	B4 <b>00 07 D0</b> 09 78 90 00	B4 <b>00 07 7B</b> 09 78 90 00	
!9999	READER.0	BC A4 00 00 02	BC A4 00 00 02	
	CARD	A4	A4	
+0027	READER.0	29 01	29 01	Transaction: Nr. 3 Nr. 4
	CARD	90 00	90 00	
+0280	READER.0	BC B0 00 1F 02	BC B0 00 1F 02	
	CARD	B0 01 01 90 00	B0 01 01 90 00	
+0406	READER.0	E1 50 02 00 0F	E1 50 02 00 0F	Cryptogram $c_1$
	CARD	50 01 00 20 62 1D 09 78 00 11 11 30 02 00 <b>00 03</b> 90 00	50 01 00 20 62 1D 09 78 00 11 11 30 02 00 <b>00 04</b> 90 00	
+0515	READER.0	E1 5A 00 00 08	E1 5A 00 00 08	
	CARD	5A	5A	
+0033	READER.0	<b>D1 39 C1 0F BB 9F 50 09</b>	<b>D8 FA 4B F4 9F DE 78 9D</b>	Deduct: 0,85 euro 0,85 euro
	CARD	90 00	90 00	
+0292	READER.0	E1 54 01 00 1B	E1 54 01 00 1B	
	CARD	54	54	
+0045	READER.0	00 00 55 09 78 21 04 63 58 01 00 06 <b>BD FD 16 B3</b> <b>E9 2B 57 1D 05 04 E0 67</b> <b>AA 63 D9</b>	00 00 55 09 78 21 04 63 58 01 00 06 <b>BE AC BE 54</b> <b>01 42 15 12 9C 04 E0 67</b> <b>FF 63 DA</b>	Cryptogram $c_2$
	CARD	90 00	90 00	
+0449	READER.0	E1 C0 00 00 0A	E1 C0 00 00 0A	
	CARD	C0 <b>BB 9B A7 FB 09 C2 0B EA</b> 04 80 90 00	C0 <b>A7 EA 6A E8 71 E1 24 E0</b> 04 80 90 00	
+0414	READER.0	E1 5A 01 00 0A	E1 5A 01 00 0A	Cryptogram $c_3$
	CARD	5A <b>D8 FA 4B F4 9F DE 78 9D</b> 04 80 90 00	5A <b>01 44 3F 74 CB 40 32 FC</b> 04 80 90 00	
+0120	READER.0	E1 5A 01 00 0A	E1 5A 01 00 0A	
	CARD	5A <b>D8 FA 4B F4 9F DE 78 9D</b> 04 80 90 00	5A <b>01 44 3F 74 CB 40 32 FC</b> 04 80 90 00	
+0591	READER.0	E1 56 00 00 0A	E1 56 00 00 0A	Cryptogram $c_4$
	CARD	56 <b>16 76 6F 7C 2C 0F 0E F7</b> 04 80 90 00	56 <b>59 A9 4F 28 3B A6 EF A2</b> 04 80 90 00	
!9999	READER.0	RESET	RESET	

Listing 3.5 shows that apart from some initialization messages a payment consists of the following steps.

- The card sends its current balance.
- The card sends general card info including the transaction counter.
- The machine calculates a cryptogram  $c_1$  based on this information and sends it to the card.
- The machine sends a deduct command of 0,85 euro containing a cryptogram to prevent data tampering.
- The card sends three cryptograms  $c_2$ ,  $c_3$  and  $c_4$  in reply.

One would expect that every cryptogram in a protocol run is different. However, we see in Listing 3.5 that  $c_3$  occurs twice in one run. Since there is no difference in the command APDU nor the response  $c_3$ , the only thing that can be concluded from this is that it is redundant communication. Another observation is that this same cryptogram  $c_3$  of the first run reoccurs in the second run as cryptogram  $c_1$ . Now, in order to investigate whether this cryptogram is generated based on data provided by the card or just stored at the terminal we used the SmartLogic to replay the payment protocol several times. The adjusted balance and transaction counter did not influence the value of  $c_1$  that was sent by the terminal. We found that the terminal stores the  $c_3$  of the latest successful run and resends it in the next payment run as  $c_1$  regardless the card that is used.

### 3.5 Concurrent SIM card sharing

Our last use case describes SIM card sharing. Mobile equipment that communicates over a GSM network is identified by Subscriber Identity Modules or so-called SIM cards. A SIM card authenticates a subscriber identity to the network. SIM cards are smart cards that follow the ISO/IEC 7816 standard and follow the same low-level protocol as, for example, banking cards.

A SIM card is an important component of a mobile phone. In most cases the SIM is installed behind the phone battery. This makes it harder to accidentally remove or swap SIM cards while the phone is operational. The SmartLogic allows a setup where multiple clients (i.e. multiple phones) connect to one SIM card (Fig. 3.4). We tested the SIM sharing setup with two Nokia 3310 phones and tried to make phone calls and send/receive text messages.

Listing 3.6 shows a trace of two phones that communicate using the same SIM card. A SIM card typically has a file system that is organised in a directory structure. The GSM standard [GSM95] describes where certain information is stored and leaves room for application dependent files. Most of the requests that the phone makes to the SIM card can be cached and replayed later. Only specific requests always need

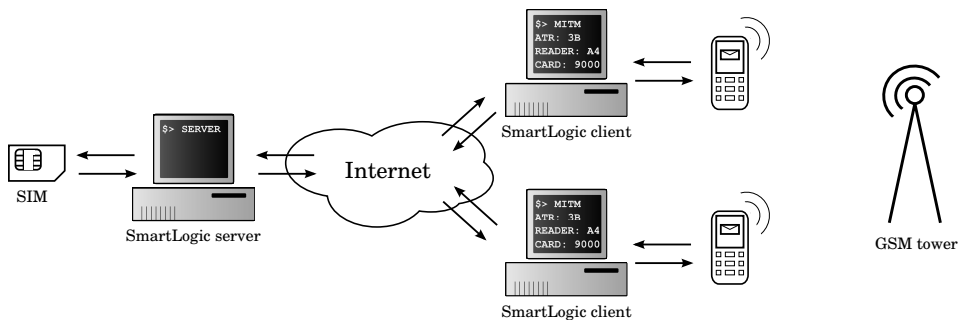


Figure 3.4: SIM card sharing

to be forwarded to the SIM card, e.g. an authentication request. In our experiment we connected two mobile phones to the same SIM card. In effect, the SIM card was cloned. Then, we did send three text messages to the phone number that was bound to the SIM card within a few minutes. During this experiment both phones made several authentication attempts. From the three text messages that were sent in this example, only the first one (Test 01) arrived at PHONE.0. After authentication of PHONE.1 the two remaining messages were received by PHONE.1. The trace from Listing 3.6 shows that the phone which authenticated last to the GSM tower received the text message.

The same behavior was experienced when making phone calls. Usually, a session key *KD* is valid for a couple of hours and a mobile phone does not need to authenticate that often. However, when PHONE.0 is making a phone call and PHONE.1 tries to do the same, a new authentication request is sent to the SIM card by PHONE.1. The reply of this request is used by PHONE.1 to prove that it is legitimate. As a result, the operator dismisses PHONE.0 with a spoken ‘technical problem’ notice. To conclude, this experiment showed that the last phone that authenticates is registered on the network. More interesting future experiments could be conducted to combine the distance relaying and smart card sharing. For instance, to run two mobile phones that share one SIM card and are physically located in two different countries. The question would then be whether the same observations can be made or if this sharing is allowed because of network segmentation.

Listing 3.6: SIM sharing trace

Event	Phone	Message
574	PHONE.0	Authenticate
580	PHONE.0	Authenticate
899	PHONE.0	Authenticate
905	PHONE.0	Authenticate
1107	PHONE.0	Authenticate
1113	PHONE.0	Authenticate
1169	PHONE.0	PHONENR: +3161267**** DATE: 03-06-11 TIME: 13:56:36 GMT: +08 SMS: Test 01
1297	PHONE.0	Authenticate
1652	PHONE.0	Authenticate
2070	PHONE.1	Authenticate
4264	PHONE.1	PHONENR: +3161267**** DATE: 03-06-11 TIME: 14:01:39 GMT: +08 SMS: Test 02
4287	PHONE.1	PHONENR: +3161267**** DATE: 03-06-11 TIME: 14:05:31 GMT: +08 SMS: Test 03
9215	PHONE.1	Authenticate
9285	PHONE.0	Authenticate



## Chapter 4

---

# Dismantling Mifare Classic

*“Il faut qu’il n’exige pas le secret, et qu’il puisse sans inconvénient tomber entre les mains de l’ennemi”*

Auguste Kerckhoffs (1883)

Over the last few years, more and more systems adopted RFID and contactless smart cards as replacement for barcodes, magnetic stripe cards and paper tickets for a wide variety of applications. Contactless smart cards consist of a small piece of memory that can be accessed wirelessly, but unlike RFID tags, they also have some computing capabilities. Most of these cards implement some sort of simple symmetric-key cryptography, making them suitable for applications that require authentication.

A number of large-scale applications make use of contactless smart cards. For example, they are used for payment in several public transport systems like the Oyster card<sup>1</sup> in London and the OV-chipkaart<sup>2</sup> in The Netherlands, among others. Many countries have already incorporated a contactless smart card in their electronic passports [HHJ<sup>+</sup>06]. Many office buildings and even secured facilities like airports and military bases use contactless smart cards for access control.

There is a huge variety of cards on the market. They differ in size, casing, memory, and computing power. They also differ in the security features they provide. A well known and widely used product family is Mifare. This product family is manufactured by NXP Semiconductors (formerly Philips Semiconductors), and currently consists of Ultralight, Ultralight C, Classic, Plus, DESFire and DESFire EV1 (See Figure 4.1 later in this chapter for more detail on their storage and cryptographic capabilities). Throughout this chapter we focus on the Mifare Classic tag. This tag provides mutual authentication and data secrecy by means of the so called CRYPTO1 cipher. This is a stream cipher using a 48-bit secret key. The cipher is proprietary technology of NXP who kept its design secret.

This chapter describes the research that we carried out on the Mifare Classic [dKGGH08, GdKGM<sup>+</sup>08]. The Mifare Classic is a contactless smart card which is used extensively in access control for office buildings, payment systems for public transport, and other applications. The contribution of this chapter is twofold.

---

<sup>1</sup><http://oyster.tfl.gov.uk>

<sup>2</sup><http://www.ov-chipkaart.nl>

First, we study the architecture of the card and the communication protocol between card and reader. Without knowing the cryptographic algorithm it is already possible to give a practical, low-cost, attack that recovers secret information from the memory of the card. This attack exploits the weak pseudo-random number generator of the Mifare Classic. As a consequence, we are able to recover the keystream generated by the CRYPTO1 stream cipher. This recovery of the stream cipher can be exploited to read *all* memory blocks of the first sector of the card. Moreover, it allows us to read *any* sector of the memory of the card, provided that we know *one* memory block within this sector. The same holds for *modifying* memory blocks.

Second, we use these findings to further reverse engineer the security mechanisms of the Mifare Classic chip: the authentication protocol, the symmetric-key cipher, and the initialization mechanism. The command codes and error messages that we recovered are of great use in this phase. We describe several security vulnerabilities and exploit these vulnerabilities with two cryptographic attacks. In both attacks we are able to retrieve the secret key from a genuine reader. The most serious attack recovers the secret key from just one or two authentication attempts with a genuine reader in less than a second on ordinary hardware and without any pre-computation. Using the same methods, an attacker can also eavesdrop the communication between a tag and a reader, and decrypt the whole trace, even if it involves multiple authentications. This enables an attacker to clone a card or to restore a card to its previous state.

## 4.1 Research context and related work

In *June 2006*, the Radboud University started developing a Radio Frequency Identification (RFID) research tool called the Ghost. The project suffered many setbacks and it was not until November 2007 that the Ghost was fully operational. Because of the slow progress that was made with the Ghost, the author decided in *July 2007* to start working on a parallel project using the Proxmark, an RFID research tool by Jonathan Westhues. A few months earlier, the hardware design and software of the Proxmark were published online. Our goal was to investigate the Mifare Classic cards that operates on 13.56 MHz, a frequency supported by the Proxmark. Although the Proxmark was clearly capable of processing 13.56 MHz signals, the specific implementation for ISO/IEC 14443-A was still missing.

It took until *November 2007* to get the ISO/IEC 14443-A implementation ready and working. From this moment on we could start investigating the proprietary protocol in full detail. The results of this work [dKGHG08] are presented in Section 4.3 and 4.4. At the same time, German researchers Karsten Nohl and Henryk Plötz were also looking at the Mifare Classic. They used a different approach where they looked at the Mifare chip layer by layer. They recovered the internal wirings of the Integrated Circuit (IC) using an optical microscope. Each layer was optically magnified 500x. It would have been too time consuming to analyze the complete IC, but by

looking at the highly interconnected areas it was possible to single out the area that contained the cryptographic algorithm. The pictures of these parts were then automatically analyzed using image processing techniques. This allowed Nohl and Plötz to recover the cryptographic algorithm CRYPTO1 that was hidden in the chip. In *December 2007* they partly presented their results [NP07] at the 24th Chaos Communication Congress (CCC) in Berlin, Germany. This presentation and the later Usenix paper [NESP08] did not include the complete CRYPTO1 cipher. The filter function and the cipher initialization were left out. However, it was enough to boost our own research using both the Ghost and the Proxmark. We were able to recover the full authentication protocol and CRYPTO1 algorithm by a non-invasive approach, i.e., by only communicating with Mifare Classic cards and readers. The Proxmark and the Ghost were of great value in this research since the communication could be fully controlled up to bit level.

The impact of the research on the Mifare Classic chip was huge since the chip was actually the most widely used contactless chip. According to NXP, more than 1 billion Mifare cards were sold and about 200 million Mifare Classic tags were in use around the world, covering about 85% of the contactless smart card market. All kinds of contactless smart cards are used in large-scale applications like in public transport or access control systems. However, it was the Mifare Classic that was used in the Oyster card in London and in many other public transport systems. Also, the OV-chipkaart in The Netherlands was based on the Mifare Classic technology. There are many more examples of Mifare Classic based projects but especially the OV-chipkaart project became subject of a political debate in the Netherlands. When the news about the supposed security weaknesses of the Mifare Classic came out, the OV-chipkaart project was still in a testing phase and only used in the Rotterdam area. It immediately generated a lot of media attention. The Dutch research organization TNO was asked to investigate the security risks. They published a report [TNO08] in *February 2008*. Their main conclusions were that fraud was unlikely and advanced equipment was needed to mount an attack. Based on this, they predicted a 2-year transition period in which the technology would still suffice.

Soon after, it became clear that the severeness of these problems was underestimated. Only one month later, in *March 2008*, we did not only recover the algorithm, but we also mounted two attacks. One of these attacks recovered the secret key from just one or two authentication attempts with a genuine reader, in less than a second. This could all be done on ordinary hardware and without any pre-computation. Because of the sensitivity of these matters we informed the Dutch government and the manufacturer, and gave a press conference [WSvRG<sup>+</sup>08]. We did not immediately publish the results of our findings, in line with the principles of responsible disclosure. Eventually, the results were planned to be published, much to the displeasure of the manufacturer NXP. On the one hand, it was understandable that they were not happy with the scientific publication, but on the other hand, it was our responsibility to inform the society by providing an in-depth understanding of the problems at hand, in order to prevent a false sense of security.

Royal Holloway University of London (RHUL) was asked to start a counter expertise review on the security problems of the Mifare Classic in the context of the Dutch OV-chipkaart project. In *April 2008*, the RHUL concluded [RHU08], in contrast to the TNO report, that fraud was much more likely to happen on the short run, and thus the Mifare Classic cards should be replaced in a much shorter time frame. Also, they advised to design a new, open and modular transport ticketing system that is more “future-proof”.

In *June 2008*, NXP filed an injunction against the Radboud University in order to stop publication of the cipher description. In *July 2008*, NXP lost this lawsuit<sup>3</sup> and the Radboud University Nijmegen was allowed to publish their scientific article based on article 10 of the European Convention on Human Rights, the freedom of expression. The university warned the stakeholders at an early stage, as it is appropriate in a responsible disclosure procedure. The court ruled that the damage to NXP’s customers is not a result of the publication, but of apparent deficiencies in the Mifare Classic design.

Finally, the results of the Mifare Classic research were published [GdKGM<sup>+</sup>08] at the 13th European Symposium on Research in Computer Security (ESORICS) in *October 2008*. The results of this research are described in Sections 4.5 to 4.7.

Of course, the developments around the Mifare Classic did not end here. It was inevitable that attacks on the Mifare Classic technology could only improve over time. This happened in *November 2008* when the Radboud University Nijmegen developed a card-only attack [GvRVWS09] that got published at the IEEE Symposium on Security and Privacy in *May 2009*. In this attack, it was no longer needed to eavesdrop a communication session with a genuine reader. Possession of a Mifare Classic card alone was enough to recover its keys. Later, in *December 2008*, it became possible to run this card-only attack using cheap off-the-shelf hardware worth around 30 euros. Nicolas Courtois invented a faster card-only attack [Cou09] that was presented at the International Conference on Security and Cryptography (SECRYPT) in *July 2009*.

To conclude, Nohl and Plötz have partly reverse engineered the Mifare Classic tag earlier [NP07], although they did not reveal all details of their findings. Their research took a very different, hardware oriented, approach. Their presentation has been of great stimulus in our discovery process. Our approach, however, was radically different as our reverse engineering was based on the study of the communication behavior of tags and readers. Furthermore, the recovery of the authentication protocol, the cryptanalysis, and the attacks presented in this chapter are novel. A more detailed account of these events may be found in [GJ12]

---

<sup>3</sup><http://www.rechtspraak.nl/ljn.asp?ljn=BD7578>



## Chapter outline

This chapter continues with an introduction to the Mifare Classic in Section 4.2 and to the core problem that was found in its weak pseudo-random number generator in Section 4.3. Then, the remainder of this chapter consists of two parts.

The first part, Section 4.4, describes how the weakness of the pseudo-random number generator can be exploited without knowledge of the cryptographic algorithm. Given access to a particular Mifare card, we are able to recover the keystream generated by the CRYPTO1 stream cipher, without knowing the secret key. Furthermore, 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 unrevealed bytes in those blocks.

The second part, Section 4.5 to 4.7, describes the reverse engineering of the cryptographic algorithm. Apart from the earlier attacks on the Mifare Classic chip that circumvent the cryptographic algorithm, we reverse engineered the proprietary cryptographic algorithm and authentication protocol. In this part, we describe the process of reverse engineering the algorithm. This was done by recording and studying traces from communication between tags and readers. We also unveil several vulnerabilities in the design and implementation of the Mifare Classic chip. This results in two attacks that recover one or more secret keys from a Mifare reader.

The first attack exploits a vulnerability in the way the cipher is initialized to split the 48 bit search space in a  $n$ -bit online search space and  $(48 - n)$ -bit offline search space. To mount this attack, the attacker needs to gather a modest amount of data from a genuine reader. Once this data has been gathered, recovering the secret key is as efficient as a lookup operation on a table. Therefore, it is much more efficient than an exhaustive search over the whole 48-bit key space.

The second and more efficient attack uses a cryptographic weakness of the Mifare CRYPTO1 cipher allowing us to recover the internal state of the cipher given a small part of the keystream. To mount this attack, one only needs one or two authentication attempts from a reader to recover the secret key within one second, on ordinary hardware. This attack does not require any pre-computation and only needs about 8 MB of memory to be executed.

When an attacker eavesdrops communication between a tag and a reader, the same methods enable us to recover all keys used in the trace and decrypt it. This gives us sufficient information to read a card, clone it, or restore it to a previous state. We have successfully executed these attacks against real systems, including the London Oyster card and the Dutch OV-chipkaart.

## 4.2 Mifare Classic

Originally, the Mifare Classic [NXP07] was developed by an Austrian company called Mikron who introduced the Mifare chip as a solution for Automatic Fare Collection (AFC). The name ‘Mifare’ stems from the contraction of Mikron and fare. In 1994, early orders of the Mifare system came from Oslo, Norway. Here, system integrator Scanpoint ordered 3,000 reader modules and 600,000 cards for an AFC project that had been started back in 1992 [Smi93]. Halfway 1995, 1 million cards and 4,000 reader modules had been manufactured. In the same year, the Netherlands-based company NXP Semiconductors (Philips Semiconductors at that time) took a 100% share in the Austrian company Mikron [Smi95]. The Mifare Classic became the core of a larger family of cards which is shown in Figure 4.1.

	Mifare Ultralight	Mifare Ultralight C	Mifare Classic	Mifare Plus	Mifare DESFire (MF3ICD40)	Mifare DESFire EV1
Introduced in	2003	2008	1994	2008	2002	2006
Memory bytes	64	192	320 1024 4096	2048 4096	4096	2048 4096 8192
Cryptography	-	3DES	CRYPTO1	AES	DES 3DES	DES 3DES AES
UID bytes	7	7	4*	4/7	7	4/7
Common Criteria	-	-	-	EAL4+	-	EAL4+

\* From May 2010 onwards, this card is also available with a 7-byte UID.

**Figure 4.1:** Mifare product family

The Mifare Classic was very attractive for use in transportation, access control and event ticketing because of its low cost. The security mechanisms of the card were advertised as being field-proven [NXP02], which means more or less that “no attacks have been reported so far”. The Mifare share started growing. The Korean Bus Fare System ordered 13,000 bus readers and 4 million Mifare cards in 1996 [Smi96]. In 1997, the French Post Office ‘La Poste’ announced that they would start using Mifare cards for access control to their 500,000 buildings, they estimated to use 5 million cards in a period of 5 years [Smi97]. This way, the Mifare market share kept growing. Eventually, in 2004, it was announced that the Netherlands would receive the first full contactless transport system [Smi04], meaning that all public transport would be accessible by only one card, the Mifare Classic. Ten years after its introduction, the Mifare Classic was seen as the major candidate for AFC systems. The first estimates suggested that 12 million Mifare Classic cards were to be deployed in the Netherlands by 2006. This project, later known as the Dutch OV-chipkaart, and countless other projects illustrate the popularity that the Mifare Classic gained over the years.

### 4.2.1 Communication layer

The communication layer of the Mifare Classic is based on ISO/IEC 14443 [ISO01]. This standard defines the communication for identification cards, contactless integrated circuit(s) cards and proximity cards. The Mifare Classic is partially compliant with ISO/IEC 14443. The fourth part of this standard, ISO/IEC 14443-4, defines the high-level protocol. At this level the implementation of NXP differs from the standard. The standard consists of four parts.

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 the card and vice versa. The data can be encoded and modulated in two ways, type A and type B. Mifare Classic uses type A.

Part 3 describes the initialization and anticollision protocol. The *anticollision* is needed in order to select a particular card when multiple 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 commands.

Part 4 defines how commands are sent. This is the point where Mifare Classic differs from the ISO/IEC 14443 standard, using a proprietary protocol. The Mifare Classic starts with an authentication, after that, all communication is encrypted. On every eight bits a parity bit is computed to detect transmission errors. In the Mifare Classic protocol this parity bit is also encrypted which means that integrity checks are only possible in the application layer.

### 4.2.2 Memory layout

The Mifare Classic card consists essentially of an EEPROM memory chip<sup>4</sup> with secure wireless communication functionality. Basic operations like read, write, increment and decrement can be performed. The memory of the card is divided into sectors. Each sector is divided into blocks of 4 or 16 bytes each. A Mifare Classic 1K card is divided into 16 sectors of 4 data blocks each. The larger Mifare Classic 4K card has 32 sectors of 4 data blocks while the remaining eight sectors are 16 data blocks in size. The last block of each sector is called the sector trailer and stores two secret keys (key A and B) and *access conditions* (AC) corresponding to that sector. To perform an operation on a specific block, the reader must first authenticate for the corresponding sector. The access conditions of that sector determine whether key A or B provides the required authorization. The access conditions define the allowed operation for this sector. Key A is never readable and key B can be configured as either readable or non-readable. In the latter case, the memory is used for data storage and key B cannot be used as an authentication key. Besides the access conditions

---

<sup>4</sup>Electrically Erasable Programmable Read-Only Memory (EEPROM)

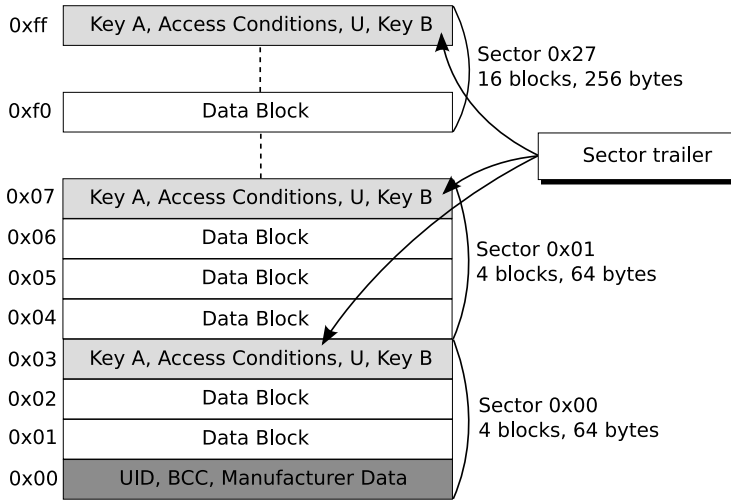


Figure 4.2: Memory structure of the Mifare Classic 4K.

and keys, there is one undefined data byte (U) which has no predefined purpose. A schematic of the sector trailer is shown in Figure 4.3a. Figure 4.2 shows a schematic of the memory structure of the Mifare Classic card. Note that the first block (block 0x00) of the first sector contains special read-only data. The first four bytes contain the Unique Identifier (UID) of the card followed by its 1-byte Bit Count Check (BCC). The BCC is calculated by successively XOR-ing all UID bytes. The remaining bytes are used to store manufacturer data.

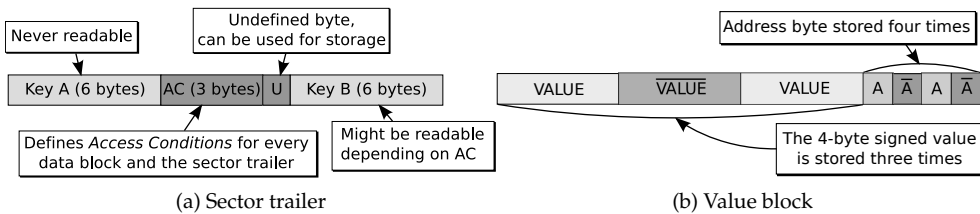


Figure 4.3: Block contents

A data block can be used in two different ways. It is either possible to store data in it or to configure it as a *value block*, which is depicted in Figure 4.3b. A value block is a special block that stores a 4-byte value. This value is stored three times, twice the value itself is stored and once its bitwise complement. The Least Significant Bit (LSB) of this value is stored as the leftmost byte. Finally, the four remaining bytes of the value block are used to store a 1-byte block address (A) that can be used as a pointer.

### 4.2.3 Commands

The Mifare Classic command set is relatively small. In this section we discuss the commands at application level. These commands can be run from several demo applications that come with readers that support Mifare Classic. For instance, the application `ContactlessDemoVC.exe` can be used with an Omnikey 5321 reader and is part of the CardMan Synchronous API SDK<sup>5</sup>. Section 4.4 explains how we recovered the commands at the data link layer. In other words, how the command frames are sent over the air. Most commands are related to a data block operation and require the reader to be authorized for this specific operation. Recall that authentication for a specific sector is required first before any command can be executed on blocks in that sector. After authentication, the authorization is checked at every command execution against the configured access conditions of the sector.

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

**Decrement, Increment, Restore and Transfer** The decrement, increment, restore and transfer commands are related to value block operations. The *increment* and *decrement* commands increment or decrement a value block with a given value and store the result in a temporary register  $T$ . The *restore* command similarly loads a value into the temporary register  $T$  without changing it. Finally, the value in  $T$  is stored back into its originating block or it is transferred to another block by the *transfer* command.

Listing 4.1: Authentication trace

Step	Sender	Hex	Abstract	
01	Reader	26	req type A	} Anticollision
02	Card	04 00	answer req	
03	Reader	93 20	select	
04	Card	c2 a8 2d f4 b3	UID, BCC	
05	Reader	93 70 c2 a8 2d f4 b3 ba a3	select(UID)	
06	Card	08 b6 dd	Mifare 1K	
07	Reader	60 30 76 4a	auth(block 0x30)	} Authentication
08	Card	42 97 c0 a4	$n_T$	
09	Reader	7d db 9b 83 67 eb 5d 83	$n_R \oplus ks_1, a_R \oplus ks_2$	
10	Card	8b d4 10 08	$a_T \oplus ks_3$	

### 4.2.4 Anticollision and authentication

Before the reader can send any command to the card it needs to select the card by its uid. This is done by a so-called anticollision protocol that ensures that every card

<sup>5</sup>The API SDK v1.1.1.4 can be downloaded for free at <http://www.hidglobal.com>

in the proximity of the reader can be addressed uniquely. When the reader selects a specific uid, the corresponding card is ready to handle commands, see Listing 4.1. Whenever a reader needs to authenticate for a memory block on the card, it starts the authentication protocol. After a successful mutual authentication, the card and reader are both convinced that they share the same secret key. This shared secret key is 48 bits long and it is stored in the sector trailer as either key A or key B. There are two authentication methods, one for authentication using key A, and one for authentication using key B. An authentication command takes a block number as parameter. This is the block number of the memory block that the reader wants to access. We captured an example anticollision and authentication in Listing 4.1. When the card receives an authentication command, it picks a challenge nonce  $n_T$  and sends it to the reader in the clear. Then, the reader sends its own challenge nonce  $n_R$  together with the answer  $a_R$  to the challenge of the card. The card finishes the authentication phase by replying  $a_T$  to the challenge of the reader. Starting with  $n_R$ , all communication is encrypted. This means that  $n_R$ ,  $a_R$ , and  $a_T$  are XOR-ed with the keystream fragments  $ks_1$ ,  $ks_2$ ,  $ks_3$ .

### 4.3 Weak pseudo-random number generator

During our experiments, independently from Nohl and Plötz [NP07, NESP08], we also noted the weakness of the pseudo-random number generator of the card by requesting many card nonces. We were able to request about 600,000 nonces every hour. Within one hour, a nonce reappeared at least four times. A Linear Feedback Shift Register (LFSR), as we introduced in Section 1.2.1, is used to generate the nonces [NESP08]. This LFSR shifts every  $9.44 \mu\text{s}$  which corresponds to exactly one bit period at the communication speed. Therefore, a random nonce generated by an LFSR with 16 bits of entropy will, theoretically, reappear after 0.618s. To observe this, the card should be queried at exactly the right moment. The pseudo-random number generator in the card is fully deterministic. Therefore, the nonce it generates only depends on the time between power up and the start of communication [NP07]. Since we control the reader, we control this timing and therefore can get the same card nonce every time. With the Proxmark operating as a tag, we can choose custom challenge nonces and uids. Furthermore, by fixing  $n_T$  (and uid) and repeatedly authenticating, we found out that the reader produces the same sequence of nonces every time after it is restarted. Unlike in the tag, the state of the pseudo-random number generator in the reader does not update every clock tick but only when it is invoked to produce a reader nonce.

The pseudo-random number generator in the tag that is used to generate  $n_T$  is a 16-bit LFSR [NP07] with generating polynomial

$$x^{16} + x^{14} + x^{13} + x^{11} + 1 \quad (4.1)$$


Since nonces are 32 bits long and the LFSR has a 16-bit state, the first half of  $n_T$  determines the second half. This means that given a 32 bit value, we can tell if it is

a proper tag nonce, i.e., if it could be generated by this LFSR. To be precise, a 32-bit value  $n_0n_1 \dots n_{31}$  is a proper tag nonce if and only if

$$n_i \oplus n_{i+2} \oplus n_{i+3} \oplus n_{i+5} \oplus n_{i+16} = 0 \quad \text{for all } i \in \{0, 1, \dots, 15\}.$$

This is actually a check on 16 output bits that once formed the internal state of the LFSR. When these bits are really a sequence produced by the 16-bit LFSR then the 17th bit has to be the result of XOR-ing the bits at tap positions. Then, XOR-ing the 17th bit with the computed output results in 0. This is tested for every possible position in the 32-bit output.

## WIKI LINEAR FEEDBACK SHIFT REGISTER



**WIKIPEDIA**  
The Free Encyclopedia

The arrangement of taps for feedback in an LFSR can be expressed in more formal arithmetic as a polynomial. The coefficients of the polynomial must be 1's or 0's. This is called the feedback polynomial or characteristic polynomial. If the taps are at the 16th, 14th, 13th and 11th bits (as shown), the feedback polynomial is

$$x^{16} + x^{14} + x^{13} + x^{11} + 1.$$

The 'one' in the polynomial does not correspond to a tap — it corresponds to the input to the first bit. The powers of the terms represent the tapped bits, counting from the left. The first and last bits are .

At the 24th CCC in Berlin, Germany, Karsten Nohl gave a presentation on the Mifare Classic security [NP07]. The generating LFSR for the card random nonces has a 16-bit state. During his talk, Karsten pointed the audience at the funny fact that the polynomial that was used in the random number generator of the Mifare Classic card was the one used in an example about LFSRs on Wikipedia.

*Source: 24th Chaos Communication Congress (2007), <http://en.wikipedia.org>*

## 4.4 Recovering the command codes

In the first practical attack on the Mifare Classic we only use the first initial nonce  $n_T$  that is sent by the card. The reader sends a request for sector authentication and the card will respond with a 32-bit nonce  $n_T$ . 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. In the keystream recovery attack this is all the information that we use.

To find out what the Mifare Classic communication looks like, we eavesdropped several transactions between Mifare readers and cards. This way, we gathered many traces that provided some insight in the high-level protocol of Mifare Classic. In this section we discuss one of these traces. One trace is shown in Listing 4.2 and contains every part of a transaction. We refer to the sequence number of the messages in our discussion. The messages are represented in hexadecimal notation. If the parity

bit of a byte is incorrect<sup>6</sup>, this is shown by an exclamation mark after the byte, see Listing 4.2. Since the parity bits are also encrypted, it is easy to identify the encrypted parts of the communication. These parts will contain “parity errors”. Now, we will only discuss the most significant messages.

**Anticollision** The reader starts the SELECT procedure. The reader sends 0x9320 (Message 3), on which the card will respond by sending its UID (Message 4). The reader sends 0x9370 followed by the UID and two CRC bytes (Message 5) to select the card.

**Authentication** The card is activated and ready to handle any higher layer commands. In Listing 4.2, messages 7 to 10 correspond to the authentication protocol. The authentication request of the reader is 0x6004d13d (Message 7). The first byte 0x60 stands for an authentication request with key A. For authentication with key B, the first byte must be 0x61. 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 know which commands are sent. Message 11 to 15 show how an *increment* is sent over the air. The *increment* is immediately followed by a *read* command (Message 16 and 17).

Listing 4.2: Authentication and transaction trace

Step	Sender	Hex	Abstract	
01	Reader	26	req type A	} Anticollision
02	Card	04 00	answer req	
03	Reader	93 20	select	
04	Card	2a 69 8d 43 8d	UID, BCC	
05	Reader	93 70 2a 69 8d 43 8d 52 55	select(UID)	
06	Card	08 b6 dd	Mifare 1K	
07	Reader	60 04 d1 3d	auth(block 0x04)	} Authentication
08	Card	3b ae 03 2d	$n_T$	
09	Reader	c4!94 a1 d2 6e!96 86!42	$n_R \oplus ks_1, a_R \oplus ks_2$	
10	Card	84 66!05!9e!	$a_T \oplus ks_3$	
11	Reader	a0 61!d3!e3	increment blocknr	} Increment
12	Card	0d	4-bit ACK	
13	Reader	26 42 ea 1d f1!68!	4-byte value	
14	Reader	8d!ca cd ea	transfer to blocknr	
15	Card	06!	4-bit ACK	
16	Reader	2a 2b 17 97	read blocknr	} Read
17	Card	49!09!3b!4e!9e!5e b0 06 d0! 07!1a!4a!b4!5c b0!4f c8!a4!	16-byte value	

<sup>6</sup>Encrypted parity bits might show up as a parity error in the message.



### 4.4.1 Keystream recovery

The stream cipher that is used by the Mifare Classic generates a keystream  $KS$ . In order to encrypt the communication between the card and reader the plaintext message  $P$  is XOR-ed bitwise with  $KS$ . This XOR operation results in the ciphertext  $C$ . We write this as

$$P \oplus KS = C.$$

In our analysis, we use the weakness of the pseudo-random number generator on the card to recover the keystream, and thus the plaintext. We used this to recover byte commands that were unknown before and to read parts of the Mifare Classic memory. Furthermore, it allowed us to describe the Mifare Classic protocol at the data link layer which is also used later in the more advanced attacks. In this attack, we do not need to know the CRYPTO1 algorithm itself. We only use the fact that it is a stream cipher which encrypts bitwise.

We propose a method that exploits the weak pseudo-random number generator to recover the keystream of an earlier recorded transaction. Concretely, we record two traces that use the same card nonce but differ in their commands. We make sure that for one of the traces we know which plaintext was sent. As a result the other plaintext of the second trace is also recovered. For this attack we need to be in possession of the card. The following reasons make this attack interesting:

1. Using this attack we can recover details about the byte commands.
2. Using the recovered keystream we can *read* card memory contents without knowing the key.
3. Using the recovered keystream we can also *modify* the contents of the card without knowing the key.
4. This attack provides known plaintext that can be used to mount a brute force attack on the key. Later in this chapter this advantage becomes obsolete as we will find much bigger problems in the Mifare Classic design.

#### Keystream recovery procedure

One recorded random card nonce  $n_T$  combined with only one recorded valid response of the reader determines the remaining keystream for a specific card since there are no other varying variables, e.g., the card UID is fixed. For our attack, we need complete control over the lowest level of communication (Proxmark). The Proxmark is programmed to be able to send arbitrary parity bits and to control the message timing. Furthermore, we need to be in possession of a (genuine) card. An adversary  $A$  proceeds as follows:

1. The adversary  $A$  eavesdrops the communication between a reader and a card and stores this trace  $t$ . This can be for example in an access control system or public transport system.

2. Then,  $A$  makes sure that in a new communication the card will use the same keystream as in the recorded communication. This is possible because the card repeats the same nonce within reasonable time, and  $A$  controls the timing.
3.  $A$  modifies the plaintext of the command that was sent in  $t$ , such that the card creates a response for which we know the plaintext (e.g., by changing the block number in a read command).
4. Now,  $A$  computes the corresponding keystream segment for each segment of known plaintext.
5.  $A$  uses this keystream to partially decrypt the trace  $t$  obtained in step 1.
6.  $A$  may recover more keystream by alternating commands of different lengths.

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

$$\left. \begin{array}{l} P_1 \oplus KS = C_1 \\ P_2 \oplus KS = C_2 \end{array} \right\} \Rightarrow C_1 \oplus C_2 = P_1 \oplus P_2 \oplus KS \oplus KS = P_1 \oplus P_2 \quad (4.2)$$

The weak pseudo-random number generator makes it possible to replay an earlier recorded transaction. We modify the first command by flipping ciphertext bits such that the command execution gives another result. This other result gives us another ciphertext while the keystream itself does not change. In other words, we can enforce different plaintexts to be encrypted using the same keystream. The attack is based on this principle.

### Keystream mapping

The data is encrypted bitwise. When the reader sends or receives a message, the keystream is shifted by the length of this message on both the reader and card side. This is needed to stay synchronized and to use the same keystream bits to encrypt and decrypt. The stream cipher does not use any feedback mechanism. Despite that, when we try to reveal the contents of a message sequence using a known keystream of an earlier trace, we find an anomaly. First, we record an *increment* followed by a *transfer* command. Then, we use this trace to apply our attack and change 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 is known and therefore we recover 198 keystream bits.

Listing 4.3: Recovering keystream and commands

Step	Sender	Hex (ciphertext)	Hex (plaintext)	Abstract
01	Reader	4c 88 31 bc!	c1 04 f6 8b	increment block 4
02	Card	0a!	0a	4-bit ACK
03	Reader	e2 79!2a!14 35!6f!	01 00 00 00 bb 4a	value 0x00000001
04	Reader	04!81 2d!1e!	b0 04 ea 62	transfer to block 4
05	Card	0c!	0a	4-bit ACK

When we use this keystream to map it on the original trace of the *increment* (Listing 4.3), it turns out that the keystream is not in sync after the first command. This has to do with the short 4-bit acknowledgement (ACK) of the card (Message 2) that is not followed by a parity bit. In our trace of the *read* command we are halfway the first response byte at this point. After 4 more bits we arrive at the parity bit in the trace of the *read* command. However, in our *increment* trace we are then half way the next command byte. To correct this we need to throw away the keystream bit in the *read* command trace that encrypts the parity bit. The general solution to this problem is to encrypt the parity bit with the next keystream bit and use this same keystream bit to decrypt the next data bit.

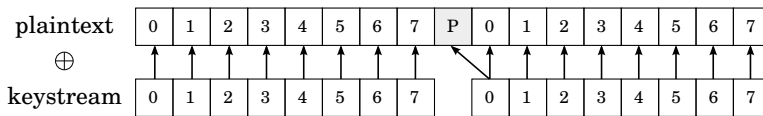


Figure 4.4: Encryption of parity bits.

From this we can conclude that parity bits are encrypted with keystream bits that are also used to encrypt databits. After every 8 bits a parity bit follows. It turns out that the parity is not computed over the ciphertext, at the lowest level of the protocol, but over the plaintext. Figure 4.4 illustrates the mapping of the keystream bits to the plaintext. In general, this leaks one bit of information about the plaintext for every byte sent.

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 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, as is shown in Figure 4.4.

### Authentication replay

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

After we recorded an authentication between card and reader, we do not modify the memory. This ensures that the memory of the card remains unaltered and therefore it will return the same plaintext. Now we will act like a Mifare reader and try to initiate the same authentication. In short, an adversary  $A$  proceeds as follows:

1. First,  $A$  records a trace  $t$  of a successful authentication between a genuine card and reader.
2. Then,  $A$  starts sending authentication requests (Message 1) until she gets a nonce that is equal to the one (Message 2) in the original trace  $t$ .
3. Now,  $A$  continues by sending the recorded response (Message 3) to this nonce. It consists of a valid response to the challenge nonce and a challenge from the reader.
4. At this point  $A$  retrieves the response (Message 4) to the challenge from the card and compares it with the one received in  $t$ . This response should be exactly the same as the recorded one.
5. Finally,  $A$  is now able to resend the same command (Message 5) or send a modified command.

#### 4.4.2 Reading sector zero

We will now show that it is possible to read sector 0 from a card without knowing the key. We only need to eavesdrop one transaction that reads sector 0 between a genuine Mifare reader and card. Every Mifare Classic card has some known memory contents. The data sheet published by NXP [NXP07] provides 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 whether key B is readable or not. The access conditions can be recovered by using the known 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 4.5. This means that the keystream can be recovered using the known

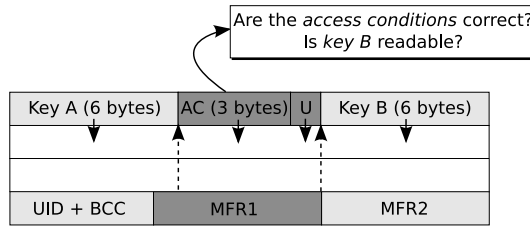


Figure 4.5: Recovering sector zero

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 often to be  $0x00$  or  $0x69$ . The access conditions tell us whether key B is readable or not. In many cases key B is not readable, for instance as in the OV-chipkaart<sup>7</sup> that is used in the Dutch public transport system. The first 5 bytes of the manufacturer data (MFR1 in Figure 4.5) 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 all data blocks in sector 0 were revealed as well.

### 4.4.3 Reading higher sectors

In the higher sectors of the Mifare Classic card we do not have the advantage of the stored manufacturer data. We basically have the sector trailer and some unknown data blocks. Because of key A we can always recover the first 6 keystream bytes. Key B is in most cases not readable and therefore will give 6 extra keystream bytes. This still leaves us with a gap of 4 bytes (AC and U). Although it is harder to achieve, there is a potential threat for these sectors to get completely compromised as well.

### 4.4.4 Command codes

At the time this research was performed, we were not aware that the command codes could be found in example firmware of NXP<sup>8</sup>. 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 reader commands consist of a command byte, parameter byte and two CRC bytes. We made several attempts to uncover the command by modifying the ciphertext of

<sup>7</sup>Mifare Classic 4k card.

<sup>8</sup><http://www.nxp.com/files/markets/identification/download/MC081380.zip>

<b>Authentication</b>			
READER	CARD	READER	CARD
60 YY*	Using KeyA 4-byte nonce	8-byte response	4-byte response
61 YY*	Using KeyB 4-byte nonce	8-byte response	4-byte response
<b>Data</b>			
READER	CARD	READER	
30 YY*	Read 16 data bytes*		
A0 YY*	Write ACK / NACK	16 data bytes*	
<b>Value blocks</b>			
READER	CARD	READER	READER
C0 YY*	Decrement ACK / NACK	4-byte value*	Transfer
C1 YY*	Increment ACK / NACK	4-byte value*	Transfer
C2 YY*	Restore ACK / NACK	4-byte value*	Transfer
B0 YY*	Transfer ACK / NACK		
<b>Other</b>			
READER			
50 00*	Halt		
<div style="border: 1px solid black; padding: 5px; width: fit-content; margin-left: auto;">           YY = block address            * = Followed by two CRC bytes         </div>			
<b>Card responses (ACK / NACK)</b>			
A (1010)	ACK		
4 (0100)	NACK, not allowed		
5 (0101)	NACK, transmission error		

Figure 4.6: Mifare Classic command set

this command. First, we guess the command. We XOR this guess with the ciphertext which gives us the keystream. Then, we check if this is indeed the correct keystream by XOR-ing 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 equal to that known response. This method revealed the commands shown in Figure 4.6.

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. On a *write* command the card returns a 4-bit ACK, which 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  $0xYYYYYYYYZZZZ$ , where  $0xYY$  are the value bytes and  $0xZZ$  the CRC bytes. Finally, a *transfer* command is sent to transfer the result of one of the previous commands to a memory block. The card

responds with an ACK when everything went well and it responds with a NACK in case of a failure or authentication problem. The 4-bit ACK is  $0xa$ . The card sends  $0x4$  when a command is not allowed. When the card detects a transmission error it responds with  $0x5$ . The card does not respond at all if the command is of the wrong length. The session gets aborted on any mistake or disallowed command.

## 4.5 Recovering the cryptographic system

We have recovered the byte commands and know how to circumvent the cryptographic system of the Mifare Classic card. In the coming sections we explain how we used this knowledge, the tools (Proxmark and Ghost) and the partial information of [NP07, NESP08] to fully recover the cryptographic system. In Mifare, there is a difference between the way bytes are represented in most tools and the way they are being sent over the air. Throughout this chapter, we represent bytes with their most significant bit on the left. However, the least significant bit is transmitted first over the air (compliant with ISO/IEC 14443). This is the same order in which the bits are input to the cryptographic functions. In other words,  $0x0a0b0c$  is transmitted and processed as input  $0x50d030$ . Finally, we number bits (in keys, nonces, and cipher states) from left to right, starting with 0. For data that is transmitted, this means that lower numbered bits are transmitted before higher numbered bits.

### 4.5.1 Authentication protocol

The recovered authentication protocol is shown in Figure 4.7. First, note that the Proxmark can send arbitrary values as nonces and is not restricted to sending proper

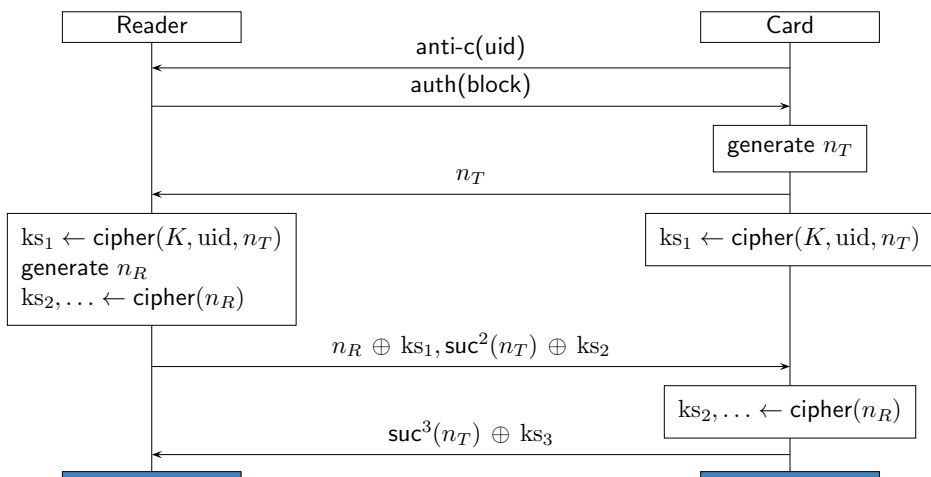


Figure 4.7: Authentication protocol

tag nonces. The generated keystream is denoted by  $ks_1, ks_2, ks_3, \dots$  where every  $ks_i$  represents 32 bits of keystream. Experimenting with authentication sessions with various uids and tag nonces, we noticed that when  $n_T \oplus uid$  and  $n_R$  remain constant, then the ciphertext of the encrypted reader nonce ( $n_R \oplus ks_1$ ) also remains constant. The answers  $a_T$  and  $a_R$ , however, have different ciphertexts in the two sessions. For example, in Listing 4.1 the uid is  $0xc2a82df4$  and  $n_T$  is  $0x4297c0a4$ , therefore  $n_T \oplus uid$  is  $0x803fed50$ .

$$\begin{array}{rclcl}
 uid & = & 0xc2a82df4 & & uid' & = & 0x1dfbe033 \\
 n_T & = & \frac{0x4297c0a4 \oplus}{0x803fed50} & \iff & n_T' & = & \frac{0x9dc40d63 \oplus}{0x803fed50} \\
 n_R \oplus ks_1 & = & 0x7ddb9b83 & \iff & n_R \oplus ks_1 & = & 0x7ddb9b83 \\
 a_R \oplus ks_2 & = & 0x67eb5d83 & & a_R' \oplus ks_2 & = & 0x4295c446 \\
 a_T \oplus ks_3 & = & 0x8bd41008 & & a_T' \oplus ks_3 & = & 0xeb3ef7da
 \end{array}$$

If we instead take  $uid'$  to be  $0x1dfbe033$  and  $n_T'$  to be  $0x9dc40d63$ , then  $n_T' \oplus uid'$  still equals  $0x803fed50$ . In both cases, the encrypted reader nonce  $n_R \oplus ks_1$  is  $0x7ddb9b83$ . However,  $a_R \oplus ks_2$  is  $0x67eb5d83$  and  $a_T \oplus ks_3$  is  $0x8bd41008$ , while  $a_R' \oplus ks_2$  and  $a_T' \oplus ks_3$  have the value  $0x4295c446$  and  $0xeb3ef7da$ , respectively.

This suggests that the keystream in both runs is the same and it also suggests that  $a_T$  and  $a_R$  depend on  $n_T$ . By XOR-ing both answers  $a_R \oplus ks_2$  and  $a_R' \oplus ks_2$  together we get  $a_R \oplus a_R'$ . We noticed that  $a_R \oplus a_R'$  is a proper tag nonce. Because the set of proper tag nonces is a linear subspace of  $\mathbb{F}_2^{32}$ , where  $\mathbb{F}_2$  is the field of two elements, the XOR of proper tag nonces is also a proper tag nonce. This suggests that  $a_R$  and  $a_R'$  are also proper tag nonces.

Given a 32 bit nonce  $n_T$  generated by the LFSR, one can compute the successor  $\text{suc}(n_T)$  consisting of the next 32 generated bits. At this stage we could verify that  $a_R \oplus a_R' = \text{suc}^2(n_T \oplus n_T') = \text{suc}^2(n_T) \oplus \text{suc}^2(n_T')$  which suggests that  $a_R = \text{suc}^2(n_T)$  and  $a_R' = \text{suc}^2(n_T')$ . Similarly for the answer from the tag we could verify that  $a_T = \text{suc}^3(n_T)$  and  $a_T' = \text{suc}^3(n_T')$ .

Summarizing, the authentication protocol can be described as follows (see Figure 4.7). After the nonce  $n_T$  is sent by the tag, both tag and reader initialize the cipher with the shared key  $K$ , the uid, and the nonce  $n_T$ . The reader then picks its challenge nonce  $n_R$  and sends it encrypted with the first part of the keystream  $ks_1$ . Then it updates the cipher state with  $n_R$ . The reader authenticates by sending  $\text{suc}^2(n_T)$  encrypted, i.e.,  $\text{suc}^2(n_T) \oplus ks_2$ . At this point the tag is able to update the cipher state in the same way and verify the authenticity of the reader. The remainder of the keystream  $ks_3, ks_4 \dots$  is now determined and from now on all communication is encrypted, i.e., XOR-ed with the keystream. The tag finishes the authentication protocol by sending  $\text{suc}^3(n_T) \oplus ks_3$ . Now the reader is able to verify the authenticity of the tag.



### Known plaintext

From the description of the authentication protocol it is easy to see that parts of the keystream can be recovered. Having seen  $n_T$  and  $\text{suc}^2(n_T) \oplus \text{ks}_2$ , one can recover  $\text{ks}_2$  (i.e., 32 bits of keystream) by computing  $\text{suc}^2(n_T)$  and XOR-ing as shown below.

$$\begin{array}{rcl}
 \text{suc}^2(n_T) \oplus \text{ks}_2 & = & 0x67eb5d83 \\
 \text{suc}^2(n_T) & = & \underline{0x90f14a44 \oplus} \\
 \text{ks}_2 & = & 0xf71a17c7 \\
 \\
 \text{suc}^3(n_T) \oplus \text{ks}_3 & = & 0x8bd41008 \\
 \text{suc}^3(n_T) & = & \underline{0xf367ce41 \oplus} \\
 \text{ks}_3 & = & 0x78b3de49
 \end{array}
 \qquad
 \begin{array}{rcl}
 \text{suc}^2(n_{T'}) \oplus \text{ks}_2 & = & 0x4295c446 \\
 \text{suc}^2(n_{T'}) & = & \underline{0xb58fd381 \oplus} \\
 \text{ks}_2 & = & 0xf71a17c7 \\
 \\
 \text{suc}^3(n_{T'}) \oplus \text{ks}_3 & = & 0xeb3ef7da \\
 \text{suc}^3(n_{T'}) & = & \underline{0x938d2993 \oplus} \\
 \text{ks}_3 & = & 0x78b3de49
 \end{array}$$

Moreover, experiments show that if the card does not send the last message during the authentication protocol, then most readers will time out and send a `halt` command. Since communication is encrypted, it actually sends `halt`  $\oplus$   $\text{ks}_3$ . Knowing the byte code of the `halt` command (`0x500057cd` [ISO01]) we can recover  $\text{ks}_3$ .

Some readers do not send a `halt` command but instead continue as if authentication succeeded. This typically means that they send an encrypted `read` command. As the byte code of the `read` command is also known (see Section 4.4) this also enables us to recover  $\text{ks}_3$  by guessing the block number. An example of recovering  $\text{ks}_3$  by the known plaintext of a `halt` or `read` is shown below.

$$\begin{array}{rcl}
 \text{halt} \oplus \text{ks}_3 & = & 0x28b38984 \\
 \text{halt} & = & \underline{0x500057cd \oplus} \\
 \text{ks}_3 & = & 0x78b3de49
 \end{array}
 \qquad
 \begin{array}{rcl}
 \text{read} \oplus \text{ks}_3 & = & 0x48b3dce1 \\
 \text{read} & = & \underline{0x300002a8 \oplus} \\
 \text{ks}_3 & = & 0x78b3de49
 \end{array}$$

It is important to note that one can obtain such an authentication session (or rather, a partial authentication session, as the Proxmark never authenticates itself) from a reader (and hence  $\text{ks}_2$ ,  $\text{ks}_3$ ) without knowing the secret key and, in fact, without a genuine tag.

If an attacker does have access to both tag and reader and is able to eavesdrop a successful (complete) authentication session, then both  $\text{ks}_2$  and  $\text{ks}_3$  can be recovered from the answers  $\text{suc}^2(n_T) \oplus \text{ks}_2$  and  $\text{suc}^3(n_T) \oplus \text{ks}_3$ . This works even if the reader does not send `halt` or `read` after timeout.

### 4.5.2 CRYPTO1 cipher

The core of the CRYPTO1 cipher is a 48-bit linear feedback shift register (LFSR) with generating polynomial

$$g(x) = x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^9 + x^7 + x^6 + x^5 + 1. \quad (4.3)$$

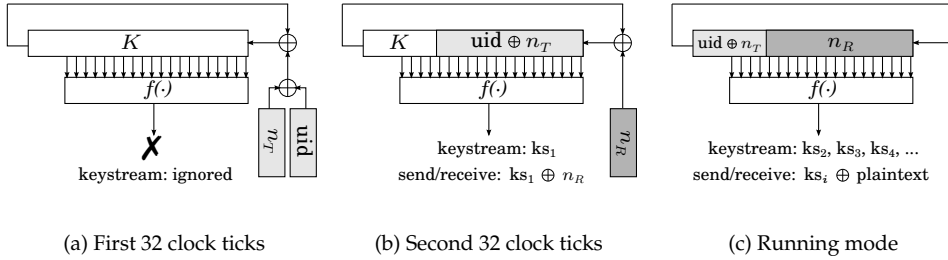


Figure 4.8: Cipher initialization

This polynomial was recovered by examining the chip with a microscope [NESP08]. At every clock tick the register is shifted one bit to the left. The leftmost bit is discarded and the feedback bit is computed according to  $g(x)$ . Additionally, the LFSR has an input bit that is XOR-ed with the feedback bit and then fed into the LFSR on the right. To be precise, if the state of the LFSR at time  $t$  is  $r_t r_{t+1} \dots r_{t+47}$  and the input bit is  $b$ , then its state at time  $t + 1$  is  $r_{t+1} r_{t+2} \dots r_{t+48}$ , where

$$r_{t+48} = r_t \oplus r_{t+5} \oplus r_{t+9} \oplus r_{t+10} \oplus r_{t+12} \oplus r_{t+14} \oplus r_{t+15} \oplus r_{t+17} \oplus r_{t+19} \oplus r_{t+24} \oplus r_{t+27} \oplus r_{t+29} \oplus r_{t+35} \oplus r_{t+39} \oplus r_{t+41} \oplus r_{t+42} \oplus r_{t+43} \oplus b. \quad (4.4)$$

The input bit  $b$  is only used during the first 64 clock ticks of the initialization (See Fig. 4.8a and 4.8b). The output bit is used to encrypt the plaintext stream. It is computed by a filter function  $f$  which takes multiple input bits from the LFSR state. Which bits actually are used as input for  $f$  was not revealed in [NP07]. Note that the general structure of CRYPTO1 in Figure 4.9b is very similar to that of the Hitag2 cipher in Figure 4.9a. Hitag2 is a low frequency tag from NXP and the description of the cipher used in Hitag2 is available on the Internet<sup>9</sup>. We used this to make educated guesses about the details of the initialization of the cipher and about the details of the filter function  $f$ . Both, the recovery of the cipher initialization and the filter function are discussed below.

### Initialization

The LFSR is initialized during the authentication protocol. As before, we experimented running several authentication sessions with varying parameters. Since we could reset the reader every time, we forced it to use the same reader nonce  $n_R$  in each authentication. We noticed that when  $n_T \oplus \text{uid}$  is kept constant, then the encrypted reader nonce, although unknown, also remains constant. This suggests that  $n_T \oplus \text{uid}$  is fed into the LFSR first. Moreover, experiments showed that, if special care is taken with the feedback bits, it is possible to modify  $n_T \oplus \text{uid}$  and the secret key  $K$  in such a way that the ciphertext after authentication also remains constant.

<sup>9</sup><http://cryptolib.com/ciphers/hitag2/>

Concretely, we verified that if

$$n_T \oplus \text{uid} \oplus K \oplus \text{'feedback bits'}$$

remains constant, then the keystream generated after authentication is constant as well. Here the 'feedback bits' are computed according to  $g(x)$ . This suggests that the secret key  $K$  is the initial state of the LFSR. This also suggests that the keystream feedback loop from the output back to the LFSR present in the Hitag2 cipher is not present in CRYPTO1, which greatly simplified the analysis.

Proceeding to the next step in the authentication protocol, the reader nonce  $n_R$  is fed into the LFSR as well (Fig. 4.8b). Note that earlier bits of  $n_R$  already affect the encryption of the later bits of  $n_R$ . At this point, the initialization is complete and the input bit of the LFSR is no longer used. Figure 4.8 shows the initialization diagram for both reader and tag. The only difference is that the reader generates  $n_R$  and then computes and sends  $n_R \oplus \text{ks}_1$ , while the tag receives  $n_R \oplus \text{ks}_1$  and then computes  $n_R$ .

Note that we can, by selecting an appropriate key  $K$ , uid, and tag nonce  $n_T$ , totally control the internal state of the LFSR just before feeding in the reader nonce. In practice, if we want to observe the behavior of the LFSR starting in state  $\alpha$ , we often set the key of the reader to 0, let the Proxmark select a uid of 0 and compute which  $n_T$  we should let the Proxmark send to reach the state  $\alpha$ . Now, because  $n_T$  is only 32 bits long and  $\alpha$  is 48 bits long, this does not seem to allow us to control the leftmost 16 bits of  $\alpha$ : they will always be 0. In practice, however, many readers accept and process tag nonces of arbitrary length. So by sending an appropriate 48-bit tag nonce  $n_T$ , we can fully control the state of the LFSR just before the reader nonce. This will be very useful in the next section, where we describe how we recovered the filter function  $f$ .

### Filter function

The first time the filter function  $f$  is used, is when the first bit of the reader nonce,  $n_{R,0}$ , is transmitted. At this point, we fully control the state  $\alpha$  of the LFSR by setting the uid, the key, and the tag nonce. As before, we use the Proxmark to send a uid of 0, use the key 0 on the reader, and use 48-bit tag nonces to set the LFSR state. So, for values  $\alpha$  of our choice, we can observe  $n_{R,0} \oplus f(\alpha)$ , since that is what is being sent by the reader. Because we power up the reader every time, the generated reader nonce is also every time the same. Therefore, even though we do not know  $n_{R,0}$ , it is a constant.

The first task is now to determine which bits of the LFSR are inputs to the filter function  $f$ . For this, we pick a random state  $\alpha$  and observe  $n_{R,0} \oplus f(\alpha)$ . We then vary a single bit in  $\alpha$ , say the  $i$ th, giving state  $\alpha'$ , and observe  $n_{R,0} \oplus f(\alpha')$ . If  $f(\alpha) \neq f(\alpha')$ , then the  $i$ th bit must be input to  $f$ . If  $f(\alpha) = f(\alpha')$ , then we can draw no conclusion about the  $i$ th bit, but if this happens for many choices of  $\alpha$ , it is likely that the  $i$ th bit is not an input to  $f$ .

Listing 4.4: Nearly equal LFSR states

Sender	Hex	Hex	Abstract
Reader	26	26	req type A
Proxmark	04 00	04 00	answer req
Reader	93 20	93 20	select
Proxmark	00 00 00 00 00	00 00 00 00 00	uid,bcc
Reader	93 70 00 00 00 00 00 9c d9	93 70 00 00 00 00 00 9c d9	select(uid)
Proxmark	08 b6 dd	08 b6 dd	Mifare 1k
Reader	60 00 f5 7b	60 00 f5 7b	auth(block 0)
Proxmark	6d c4 13 ab d0 f3	6d c4 13 ab d0 73	$n_T$
Reader	df 19 d5 7a e5 81 ce cb	5e ef 51 1e 5e fb a6 21	$n_R \oplus ks_1$ , $suc^2(n_T) \oplus ks_2$

Listing 4.4 shows an example. The key in the reader (for block 0) is set to 0 and the Proxmark sends a uid of 0. On the left hand side, the Proxmark sends the tag nonce  $0x6dc413abd0f3$  and on the right hand side it sends the tag nonce  $0x6dc413abd073$ . This leads, respectively, to LFSR states of  $0xb05d53bfbdb10$  and  $0xb05d53bfbdb11$ . These states differ only in the rightmost bit, i.e., bit 47. On the left hand side, the first bit of the encrypted reader nonce is 1 and on the right hand side it is 0 (recall the byte-swapping convention used in traces). Hence, bit 47 must be an input to the filter function  $f$ .

This way, we were able to see that the bits 9, 11, ..., 45, 47 are input to the filter function  $f$ . Based on the similarity with the Hitag2, shown in Figure 4.9a, we guessed that there are 5 “first layer circuits” each taking four inputs, respectively, 9, 11, 13, 15 for the left-most circuit up to 41, 43, 45, 47 for the right-most circuit. The five results from these circuit are then, we guessed, input into a “second layer circuit”, producing a keystream bit. (See Figure 4.9b for the structure of CRYPTO1). Note that in the Hitag2, all these circuits are “balanced”, in the sense that for half the possible (16 or 32) inputs they give a 0 and for half the possible inputs they give a 1.

Listing 4.5: First bit of encrypted reader nonce

LFSR \ XX	55	54	51	50	45	44	41	40	15	14	11	10	05	04	01	00
0xb05d53bfbdbXX	0	0	0	0	1	1	0	1	1	1	0	1	0	0	1	1
0xfbb57bbc7fXX	1	1	1	1	0	0	1	0	0	0	1	0	1	1	0	0
0xe2fd86e299XX	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

To verify our guess and to determine  $f$ , we again take a random state  $\alpha$  of the LFSR. We then vary 4 (guessed) inputs to a first layer circuit in all 16 ways possible, giving states  $\alpha_0, \alpha_1, \dots, \alpha_{15}$  and observe  $r_0 \oplus f(\alpha_0), \dots, r_0 \oplus f(\alpha_{15})$ . If our guess was correct, we expect these to be 16 zeros, 16 ones, or 8 zeros and 8 ones: either the 16 non-varying inputs are such that the 4 varying inputs do not influence the keystream bit (in which case we get all zeros or all ones), or we get a “balanced” result as in the Hitag2. In the first two cases, we try again; in the latter case, we have found the component (up to a NOT, but that is irrelevant). Listing 4.5 shows examples of LFSRs that vary the inputs to a first layer circuit.

It turned out that our guess was correct. There are two different circuits used in the first layer. Two circuits in the first layer compute  $f_a(y_0, y_1, y_2, y_3)$  represented by

the boolean table  $0 \times 1c9b$  and the other three compute  $f_b(y_0, y_1, y_2, y_3)$  represented by the boolean table  $0 \times 344f$ . It is represented as follows, from left to right the bits of  $0 \times 1c9b$  are the corresponding values of  $f_a(0, 0, 0, 0), f_a(0, 0, 0, 1), \dots, f_a(1, 1, 1, 1)$  and similarly for  $f_b$  (and  $f_c$  below). These five output bits are input into the circuit in the second layer. By trying 32 states that produce all 32 possible outputs for the first layer, we build a table for the circuits in the second layer. It computes  $f_c(y_0, y_1, y_2, y_3, y_4)$  represented by the boolean table  $0 \times 4457c3b3$ . In this way we recovered the filter function  $f$ . See Figure 4.9b. Note that  $f_a$  and  $f_b$  here are negated and swapped when compared to [GdKGM<sup>+</sup>08] and  $f_c$  is swapped accordingly.

## 4.6 Weaknesses and exploits

This section describes four design flaws of the Mifare Classic. These flaws allow us to recover the secret key from a genuine Mifare reader with two different attacks. For the first attack, the core of which is described in Section 4.6.1, we first have to gather a modest amount of data from the reader. Together with a precomputed table this can be used to invert the filter function  $f$  and then, with an LFSR rollback technique described in Section 4.6.2, we can recover the secret key. The second attack is described in Section 4.6.3, here we can directly invert the filter function  $f$  in under one second on ordinary hardware without the need for any precomputed tables. The same LFSR rollback technique then also recovers the secret key. Finally, recall the reuse of keystream bits to encrypt parity bits in Section 4.4.1. This is used in the attack from Section 4.7.3.

### 4.6.1 LFSR state recovery

The tag nonce directly manipulates the internal state of the LFSR. This enables us to recover the state of the LFSR, given a segment of keystream.

First, we build a table consisting of tuples  $(\text{lfsr}, \text{ks})$  where  $\text{lfsr}$  runs over all LFSR states of the form  $0 \times 000\text{WWWWWWWWWW}$  and  $\text{ks}$  are the first 64 bits of keystream they generate. This one time computation can be performed on an ordinary computer and can be reused for any reader/key. This produces a table of  $2^{36}$  rows.

Now we focus on a specific reader that we want to attack. For each 12 bit number  $0 \times \text{XXXX}$ , we start an authentication session using the same  $\text{uid}$ . We set the challenge nonce of the tag to  $n_T = 0 \times 0000\text{XXX}0$ . After the reader answers with  $n_R \oplus \text{ks}_1, \text{suc}^2(n_T) \oplus \text{ks}_2$  we do not reply. Then most readers send  $\text{halt} \oplus \text{ks}_3$ . Since we know  $\text{suc}^2(n_T)$  and  $\text{halt}$  we can recover  $\text{ks}_2, \text{ks}_3$ . There is exactly one value for  $0 \times \text{XXXX}$  that produces an LFSR state of the form  $0 \times \text{YYYYYYYY}000\text{Y}$  after feeding in  $n_T = 0 \times 0000\text{XXX}0$ . While feeding in the reader nonce  $n_R$ , the zeros in the LFSR are shifted to the left, producing an LFSR state of the form  $0 \times 000\text{YZZZZZZZZ}$ . Since we have all LFSR states of this form in our table, we can recover it by searching for  $\text{ks}_2, \text{ks}_3$ .

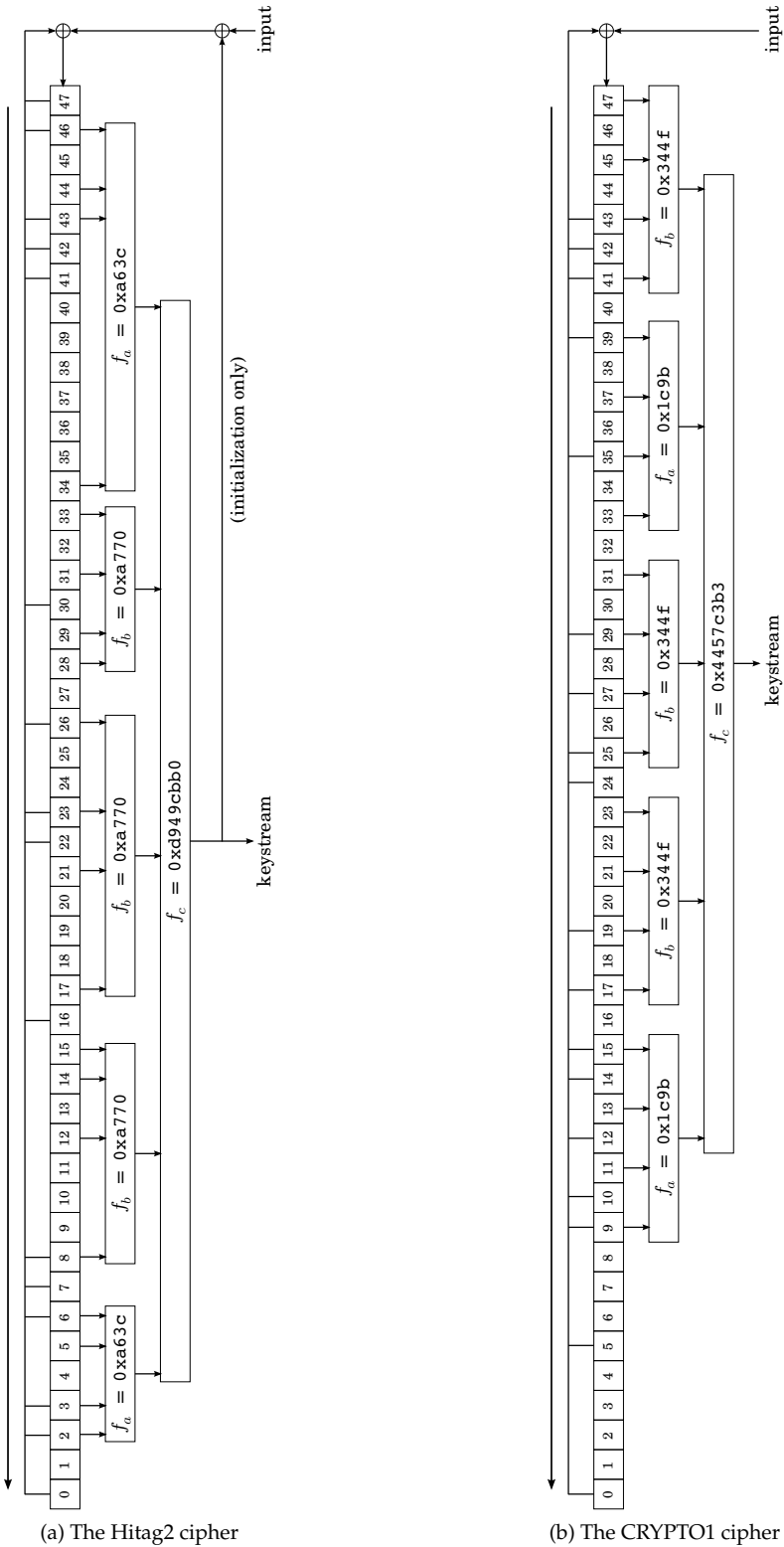


Figure 4.9: The CRYPTO1 cipher structure compared to the Hitag2 cipher structure.

Typically, only for a single value of  $0xXXX$  we do get a hit in our table, because the size of the keystream is 64 bits and the size of the LFSR is only 48 bits. In Section 4.6.2 we show how we can use the LFSR state that we find in the table, together with  $n_T$  and  $n_R \oplus ks_1$ , to obtain the secret key.

In the above description it is possible to trade off between the size of the lookup table and the number of authentication sessions needed. In the above setup, the size of the table is approximately one terabyte and the number of required authentication sessions is 4096. For instance, by varying 13 instead of 12 bits of the tag nonce we halve the size of the table at the cost of doubling the number of required sessions.

Note that even if the reader does not respond in case of time out, we can still use this technique to recover the LFSR state. In that case, for each  $0xXXX$ , we search only for the corresponding  $ks_2$  in the table. Since there are  $2^{48-12}$  entries in the table, and  $ks_2$  is 32 bits long, we get on average  $2^4$  matches. Since we are considering  $2^{12}$  possible values of  $0xXXX$ , we get a total of approximately  $2^{16}$  possible LFSR states. Each of these LFSR states gives us, using Section 4.6.2, a candidate key. With a single other partial authentication session, i.e., one up to and including the answer from the reader, we can then check which of those keys is the correct one.

## 4.6.2 LFSR rollback

Given the state  $r_k r_{k+1} \dots r_{k+47}$  of the LFSR at a certain time  $k$  (and the input bit, if any), one can use the relation (4.4) to compute the previous state  $r_{k-1} r_k \dots r_{k+46}$ .

Now suppose that we somehow learned the state of the LFSR right after the reader nonce has been fed in, for instance using the approach from the previous section, and that we have eavesdropped the encrypted reader nonce. Because we do not know the plaintext reader nonce, we cannot immediately roll back the LFSR to the state before feeding in the reader nonce. However, the input to the filter function  $f$  does not include the leftmost bit of the LFSR. This weakness does enable us to recover this state (and the plaintext reader nonce) anyway.

To do so we shift the LFSR to the right. The rightmost bit falls out and we set the leftmost bit to an arbitrary value  $r$ . Then we compute the function  $f$  and we get one bit of keystream that was used to encrypt the last bit  $n_{R,31}$  of the reader nonce. Note that the leftmost bit of the LFSR is not an input to the function  $f$ , and therefore our choice of  $r$  is irrelevant. Using the encrypted reader nonce we recover  $n_{R,31}$ . Computing the feedback of the LFSR we can now set the bit  $r$  to the correct value, i.e., so that the LFSR is in the state prior to feeding  $n_{R,31}$ . Repeating this procedure 31 times more, we recover the state of the LFSR before the reader nonce was fed in.

Since the tag nonce and uid are sent as plaintext, we also recover the LFSR state before feeding in  $n_T \oplus uid$ . Note that this LFSR state is the secret key!

### 4.6.3 Odd inputs to the filter function

The inputs to the filter function  $f$  are only on odd-numbered bits of the LFSR. The fact that they are so evenly placed can be exploited. Given a part of keystream, we can generate those relevant bits of the LFSR state that give the even bits of the keystream and those relevant bits of the LFSR state that give the odd bits of the keystream separately. By splitting the feedback in two parts as well, we can combine those even and odd parts efficiently and recover exactly those states of the LFSR that produce a given keystream. This may be understood as “inverting” the filter function  $f$ .

Let  $b_0b_1 \dots b_{n-1}$  be  $n$  consecutive bits of keystream. For simplicity of the presentation we assume that  $n$  is even; in practice  $n$  is either 32 or 64. Our goal is to recover all states of the LFSR that produce this keystream. To be precise, we will search for all sequences  $\bar{r} = r_0r_1 \dots r_{46+n}$  of bits such that

$$\begin{aligned} r_k \oplus r_{k+5} \oplus r_{k+9} \oplus r_{k+10} \oplus r_{k+12} \oplus r_{k+14} \oplus r_{k+15} \oplus r_{k+17} \\ \oplus r_{k+19} \oplus r_{k+24} \oplus r_{k+25} \oplus r_{k+27} \oplus r_{k+29} \oplus r_{k+35} \oplus r_{k+39} \oplus r_{k+41} \\ \oplus r_{k+42} \oplus r_{k+43} \oplus r_{k+48} = 0, \text{ for all } k \in \{0, \dots, n-2\}, \end{aligned} \quad (4.5)$$

and such that

$$f(r_k \dots r_{k+47}) = b_k, \text{ for all } k \in \{0, \dots, n-1\}. \quad (4.6)$$

Condition (4.5) says that  $\bar{r}$  is generated by the LFSR, i.e., that  $r_0r_1 \dots r_{47}, r_1r_2 \dots r_{48}, \dots$  are successive states of the LFSR; Condition (4.6) says that it generates the required keystream. Since  $f$  only depends on 20 bits of the LFSR, we will overload notation and write  $f(r_{k+9}, r_{k+11}, \dots, r_{k+45}, r_{k+47})$  for  $f(r_k \dots r_{k+47})$ . Note that when  $n$  is larger than 48, there is typically only one sequence satisfying (4.5) and (4.6), otherwise there are on average  $2^{48-n}$  such sequences.

During our attack we build two tables of approximately  $2^{19}$  elements. These tables contain respectively the even numbered bits and the odd numbered bits of the LFSR sequences that produce the evenly and oddly numbered bits of the required keystream.

We proceed as follows. Looking at the first bit of the keystream,  $b_0$ , we generate all sequences of 20 bits  $s_0s_1 \dots s_{19}$  such that  $f(s_0, s_1, \dots, s_{19}) = b_0$ . The structure of  $f$  guarantees that there are exactly  $2^{19}$  of these sequences. Note that the sequences  $\bar{r}$  of the LFSR that we are looking for must have one of these sequences as its bits  $r_9, r_{11}, \dots, r_{47}$ .

For each of the entries in the table, we now do the following. We view the entry as the bits 9, 11,  $\dots$ , 47 of the LFSR. We now shift the LFSR two positions to the left. The feedback bit, which we call  $s_{20}$ , that is shifted in second could be either 0 or 1; not knowing the even numbered bits of the LFSR nor the low numbered odd ones, we have no information about the feedback. We can check, however, which of the two possibilities for  $s_{20}$  matches with the keystream, i.e., which satisfy  $f(s_1, s_2, \dots, s_{20}) = b_2$ . If only a single value of  $s_{20}$  matches, we extend the entry in our table by  $s_{20}$ . If



both match, we duplicate the entry, extending it once with 0 and once with 1. If neither matches, we delete the entry. On average, 1/4 of the time we duplicate an entry, 1/4 of the time we delete an entry, and 1/2 of the time we only extend the entry. Therefore, the table stays, approximately, of size  $2^{19}$ .

We repeat this procedure for the bits  $b_4, b_6, \dots, b_{n-1}$  of the keystream. This way we obtain a table of approximately  $2^{19}$  entries  $s_0 s_1 \dots s_{19+n/2}$  with the property that  $f(s_i, s_{i+1}, \dots, s_{i+19}) = b_{2i}$  for all  $i \in \{0, 1, \dots, n/2\}$ . Consequently, the sequences  $\bar{r}$  of the LFSR that we are looking for must have one of the entries of this table as its bits  $r_9, r_{11}, \dots, r_{47+n}$ .

Similarly, we obtain a table of approximately  $2^{19}$  entries  $t_0 t_1 \dots t_{19+n/2}$  with the property that  $f(t_i, t_{i+1}, \dots, t_{i+19}) = b_{2i+1}$  for all  $i \in \{0, 1, \dots, n/2\}$ .

Note that after only 4 extensions of each table, when all entries have length 24, one could try every entry  $s_0 s_1 \dots s_{23}$  in the first table with every entry  $t_0 t_1 \dots t_{23}$  in the second table to see if  $s_0 t_0 s_1 \dots t_{23}$  generates the correct keystream. Note that this already reduces the search complexity from  $2^{48}$  in the brute force case to  $(2^{19})^2 = 2^{38}$ .

To further reduce the search complexity, we now look at the feedback of the LFSR. Consider an entry  $\bar{s} = s_0 s_1 \dots s_{19+n/2}$  of the first table and an entry  $\bar{t} = t_0 t_1 \dots t_{19+n/2}$  of the second table. In order that  $\bar{r} = s_0 t_0 s_1 \dots t_{19+n/2}$  is indeed generated by the LFSR, it is necessary (and sufficient) that every 49 consecutive bits satisfy the LFSR relation (4.5), i.e., the 49th must be the feedback generated by the previous 48 bits.

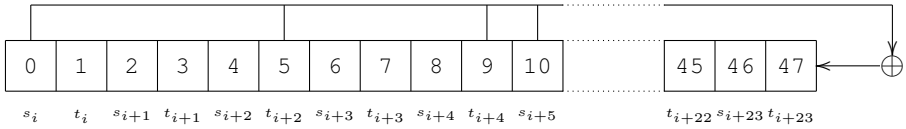


Figure 4.10: Subsequences  $\bar{s}$  and  $\bar{t}$ .

So, for every subsequence  $s_i s_{i+1} \dots s_{i+24}$  of 25 consecutive bits of  $\bar{s}$  we compute its contribution  $b_i^{1, \bar{s}} = s_i \oplus s_{i+5} \oplus s_{i+6} \oplus s_{i+7} \oplus s_{i+12} \oplus s_{i+21} \oplus s_{i+24}$  of the LFSR relation and for every subsequence  $t_i t_{i+1} \dots t_{i+23}$  of 24 consecutive bits of  $\bar{t}$  we compute  $b_i^{2, \bar{t}} = t_{i+2} \oplus t_{i+4} \oplus t_{i+7} \oplus t_{i+8} \oplus t_{i+9} \oplus t_{i+12} \oplus t_{i+13} \oplus t_{i+14} \oplus t_{i+17} \oplus t_{i+19} \oplus t_{i+20} \oplus t_{i+21}$ . See Figure 4.10. If  $s_0 t_0 s_1 \dots t_{n/2}$  is indeed generated by the LFSR, then

$$b_i^{1, \bar{s}} = b_i^{2, \bar{t}} \text{ for all } i \in \{0, \dots, n/2 - 5\}. \quad (4.7)$$

Symmetrically, for every subsequence of 24 consecutive bits of  $\bar{s}$  and corresponding 25 consecutive bits of  $\bar{t}$ , we compute  $\tilde{b}_i^{1, \bar{s}} = s_{i+2} \oplus s_{i+4} \oplus s_{i+7} \oplus s_{i+8} \oplus s_{i+9} \oplus s_{i+12} \oplus s_{i+13} \oplus s_{i+14} \oplus s_{i+17} \oplus s_{i+19} \oplus s_{i+20} \oplus s_{i+21}$  and  $\tilde{b}_i^{2, \bar{t}} = t_i \oplus t_{i+5} \oplus t_{i+6} \oplus t_{i+7} \oplus t_{i+12} \oplus t_{i+21} \oplus t_{i+24}$ . Also here, if  $s_0 t_0 s_1 \dots t_{n/2}$  is indeed generated by the LFSR, then

$$\tilde{b}_i^{1, \bar{s}} = \tilde{b}_i^{2, \bar{t}} \text{ for all } i \in \{0, \dots, n/2 - 5\}. \quad (4.8)$$

One readily sees that together, conditions (4.7) and (4.8) are equivalent to equation (4.5).

To efficiently determine the LFSR state sequences that we are looking for, we sort the first table by the newly computed bits  $b_0^{1,\bar{s}} \dots b_{n/2-5}^{1,\bar{s}} \tilde{b}_0^{1,\bar{s}} \dots \tilde{b}_{n/2-5}^{1,\bar{s}}$ , and the second table by  $b_0^{2,\bar{t}} \dots b_{n/2-5}^{2,\bar{t}} \tilde{b}_0^{2,\bar{t}} \dots \tilde{b}_{n/2-5}^{2,\bar{t}}$ .

Since  $s_0 t_0 s_1 \dots t_{n/2}$  is generated by the LFSR if and only if  $b^{1,\bar{s}} \tilde{b}^{1,\bar{s}} = b^{2,\bar{t}} \tilde{b}^{2,\bar{t}}$  and since by construction it generates the required keystream, we do not even have to search anymore. The complexity now reduces to  $n$  loops over two tables of size approximately  $2^{19}$  and two sortings of these two tables. For completeness sake, note that from our tables we retrieve  $r_9 r_{10} \dots r_{46+n}$ . So to obtain the state of the LFSR at the start of the keystream, we have to roll back the state  $r_9 r_{10} \dots r_{58}$  9 steps.

In a variant of this method, applicable if we have sufficiently many bits of keystream available (64 will do), we only generate one of the two tables. For each of the approximately  $2^{19}$  entries of the table, the LFSR relation (4.4) can then be used to express the ‘missing’ bits as linear combinations (over  $\mathbb{F}_2$ ) of the bits of the entry. We can then check if it produces the required keystream.

This construction has been implemented in two ways. First of all as C code that recovers states from keystreams. Secondly also as a logical theory that has been verified in the theorem prover PVS [ORSvH95]. The latter involves a logical formalization of many aspects of the Mifare Classic [JWS11].

## 4.7 Attacking Mifare Classic

This section describes different attacks on the Mifare Classic card. Section 4.7.1 and 4.7.2 describe the first two practical attacks that exploit weaknesses in the CRYPTO1 cipher of the Mifare Classic. Then, Section 4.7.4 gives a brief overview of subsequent attacks that are card-only, have improved attack times and/or need less traces.

### 4.7.1 Attack one

An attacker can recover the secret key from a Mifare reader as follows.

First, the attacker generates the table of (lfsr, ks) tuples as described in Section 4.6.1. This one terabyte table can be computed in one afternoon on standard hardware and can be reused.

Next, the attacker initiates  $4096 = 2^{12}$  authentication sessions and computes  $ks_2, ks_3$  for each of these sessions as described in Section 4.5.1. Note that this only requires access to a reader and not to a tag. As explained in Section 4.6.1, it is possible to recover the state of the LFSR prior to feeding in  $n_R$ . Then, as explained in Section 4.6.2, it is also possible to recover the state prior to feeding in  $n_T \oplus uid$ , i.e., the secret key is recovered!

Experiments show that it is typically possible to gather between 5 and 35 partial authentication sessions per second from a Mifare reader, depending on whether or

not the reader is online. This means that gathering 4096 sessions takes between 2 and 14 minutes.

### 4.7.2 Attack two

Instead of using the table, we can also use the invertibility of  $f$  described in Section 4.6.3 to recover the state of the LFSR at the end of the authentication. This way, we only need a single (partial) authentication session.

Note that this attack cannot be stopped by fixing the readers to not continue communication after authentication fails. With the knowledge of just  $ks_2$ , we can invert  $f$  to find approximately 65536 candidate keys; these can be checked against another authentication session.

In practice, a relatively straightforward implementation of this attack takes less than one second of computation and only about 8 MB of memory on ordinary hardware to recover the secret key. Moreover, it does not require any kind of pre-computation, rainbow tables, etc. A highly optimized implementation of the single table variant consumes virtually no memory and recovers the secret key within 0.1 second on the same hardware.

### 4.7.3 Multiple-sector authentication

Many systems authenticate for more than one sector. Starting with the second authentication the protocol is slightly different. Since there is already a session key established, the new authentication command is sent encrypted with this key. At this stage the secret key  $K'$  for the new sector is loaded into the LFSR. The difference is that now the tag nonce  $n_T$  is sent encrypted with  $K'$  while it is fed into the LFSR (resembling the way the reader nonce is fed in). From this point on the protocol continues exactly as before, i.e., the reader nonce is fed in, etc.

To clone a card, one typically needs to recover all the information read by the reader and this usually involves a few sectors. To do so, we first eavesdrop a single, complete session which contains authentications for multiple sectors. Once we have recovered the key for the first sector as described in Section 4.7, we proceed to the next sector read by the reader. The authentication request is now encrypted with the previous session key, but this is not a problem: we just recovered that key, so we can decrypt the authentication request. The issue now is that we need the tag nonce  $n_T$  to mount our attacks and it is encrypted with the key  $K'$  which we do not yet know. We can, of course, simply try all  $2^{16}$  possible tag nonces to execute our attack.

Using the parity bits, however, the number of possible tag nonces can be drastically reduced. The first three parity bits, say  $p_0, p_1, p_2$ , of the tag nonce  $n_T$  are encrypted with the keystream bits that are also used to encrypt bits  $n_8, n_{16}$ , and  $n_{24}$  of  $n_T$ . That is, from the communication we can observe  $p_0 \oplus b_8, n_8 \oplus b_8$ , where  $b_8$  is the keystream bit that is used to encrypt  $n_8$ , and similarly for the other two parity bits. From this we can see whether or not  $p_0$ , the parity of the first byte of  $n_T$ , is equal

to  $n_8$ , the first bit of the second byte of  $n_T$ . This information decreases the number of potential nonces by a factor of 2. The same holds for the other 2 parity bits in  $n_T$  and for the 7 parity bits in  $\text{suc}^2(n_T)$  and  $\text{suc}^3(n_T)$ . In total, the search space is reduced from  $2^{16}$  nonces to only  $2^{16}/2^{10} = 64$  nonces.

By running many tests we found a way to select almost immediately the correct nonce out of those 64 candidates. The pseudo-random number generator of the tag keeps shifting during the communication in a predictable way. This enables us to predict the distance  $d(n_T, n'_T)$  between the tag nonce  $n_T$  used in one authentication session and the tag nonce  $n'_T$  used in the next. Distance here means the number of times the pseudo-random number generator has to shift after outputting  $n_T$  before it outputs  $n'_T$ . The relation we found experimentally is  $d(n_T, n'_T) = 8t - 55c - 400$ , where  $t$  is the time between the sending of the encrypted reader nonce in the first authentication session and the authenticate command that starts the next session (expressed in bit-periods, the time it takes to send a single bit, approximately  $9.44\mu\text{s}$ ) and  $c$  is the number of commands the reader sends in the first session. However, we do not know precisely why this relation holds and if it holds under all circumstances. In practice, the correct nonce is nearly always the one (from the 64 candidates) whose distance to  $n_T$  is closest to  $d(n_T, n'_T)$ . Consequently, keys for subsequent sectors are obtained at the same speed as the key for the first sector.

#### 4.7.4 Improved attacks

We will now briefly visit other attacks on Mifare Classic that soon followed the disclosure of the algorithm and authentication protocol described in this chapter. The previously described attacks always involve a genuine reader in order to obtain valid traces between a reader and a card. These traces are then used to recover the secret keys that were used during authentication. A very straightforward and logical next step would be to attack a Mifare card without the involvement of a genuine reader. The first card-only attack was presented by Garcia et al. [GvRVWS09] at the IEEE Symposium on Security and Privacy in 2009. This paper makes use of several weaknesses.

- In the Mifare Classic protocol, parity bits are computed over the plaintext and encrypted with the same keystream bit that is used to encrypt the next data bit. This leaks information about the keystream bits.
- When the card receives a wrong authentication attempt from the reader, but with correct parity bits, it will answer with an encrypted NACK of 4 bits ( $0 \times 05$ ) which again leaks keystream bits.
- Once authenticated for one sector, it is possible to start authenticating for another sector. The tag nonce  $n_T$  is encrypted using the key of this other sector. This means that 32-bits of keystream are leaked since it is possible to predict the nonce using a precise timing of our authentication request. This attack is called the nested authentication attack.

Furthermore, an important precondition to mount attacks in practice is to have control over the parity bits that are sent. The development of the *libnfc* library for ACR122 readers makes it possible to do this using much cheaper equipment compared to the Proxmark.

Soon after this first card-only attack Nicolas Courtois published another card-only attack [Cou09] which needed fewer authentication attempts (around 300) and no pre-computation at all.

## 4.8 Conclusion

We have reverse engineered the security mechanisms of the Mifare Classic chip. We found several vulnerabilities and successfully managed to exploit them, retrieving the secret key from a genuine reader. We have presented two very practical attacks that, to retrieve the secret key, do not require access to a genuine tag at any point.

In particular, the second attack recovers a secret key from just one or two authentication attempts with a genuine reader (without access to a genuine tag) in less than a second on ordinary hardware and without any pre-computation. Furthermore, an attacker that is capable of eavesdropping the communication between a tag and a reader can recover all keys used in this communication. This enables an attacker to decrypt the whole trace and clone the tag.

What the actual implications are for real life systems deploying the Mifare Classic depends, of course, on the system as a whole: contactless smart cards are generally not the only security mechanism in place. For instance, public transport payment systems such as the OV-chipkaart have a back-end system recording transactions and attempting to detect fraudulent activities (such as traveling on a cloned card). Systems like these have to deal with the fact that it is fairly easy to read and clone Mifare Classic cards.

In general, we believe that it is far better to use well-established and peer-reviewed cryptographic primitives and protocols than proprietary ones. As was already formulated by Auguste Kerckhoffs in 1883, and what is now known as Kerckhoffs' Principle, the security of a cryptographic system should not depend on the secrecy of the system itself, but only on the secrecy of the key [Ker83]. Once again it is proven that details of a proprietary system will eventually become public; the previous obscurity then only leads to a less well-reviewed system that is prone to have design and implementation flaws.



## Chapter 5

---

# Dismantling iClass and iClass Elite

*“Should you find yourself in a chronically leaking boat, energy devoted to changing vessels is likely to be more productive than energy devoted to patching leaks”*

Warren Buffett

With more than 300 million cards sold [HID10], HID iClass is – after the Mifare Classic – the most popular high frequency contactless smart card for access control on the market. It is widely used for access control, secure login and payment systems. The card uses 64-bit keys to provide authenticity and integrity. The cipher and key diversification algorithms are proprietary and little information about them is publicly available. In this chapter we reverse engineer all iClass security mechanisms including cipher, authentication protocol and key diversification algorithms, which we publish in full detail. Additionally, we analyze their security and show that these cards do not provide the level of security that is claimed by their manufacturer.

iClass is an ISO/IEC 15693 [ISO00, ISO06, ISO09] compatible contactless smart card manufactured by HID Global. It was introduced in the market back in 2002 as a secure replacement of the HID Prox card which did not have any cryptographic capabilities. The iClass cards are widely used in access control of secured buildings such as The Bank of America Merrill Lynch, the International Airport of Mexico City and the United States Navy base of Pearl Harbor [Cum06] among many others<sup>1</sup>. Other applications include secure user authentication such as in the naviGO system included in Dell’s Latitude and Precision laptops; e-payment like in the FreedomPay and SmartCentric systems; and billing of electric vehicle charging such as in the Liberty PlugIns system. iClass has also been incorporated into the new BlackBerry phones which support Near Field Communication (NFC).

iClass uses a proprietary cipher to provide data integrity and mutual authentication between card and reader. The cipher uses a 64-bit diversified key which is derived from a 56-bit master key and the serial number of the card. This key diversification algorithm is built into all iClass readers. The technology used in the card is covered by US Patent 6058481 and EP 0890157. The precise description of both the cipher and the key diversification algorithms are kept secret by the manufacturer following the principles of security by obscurity. HID distinguishes two system configurations for iClass, namely iClass Standard and iClass Elite. The main differences

---

<sup>1</sup><http://hidglobal.com/mediacenter.php?cat2=2>

between iClass Standard and iClass Elite lies in their key management and key diversification algorithms. Remarkably, all iClass Standard cards worldwide share the same master key for the iClass application. This master key is stored in the EEPROM memory of every iClass reader. Our analysis uncovers this key. In iClass Elite, however, it is possible to let HID generate and manage a custom key for your system if you are willing to pay a higher price. The iClass Elite Program (a.k.a., High Security) uses an additional key diversification algorithm (on top of the iClass Standard key diversification) and a custom master key per system which according to HID provides “the highest level of security” [HID09].

In this chapter we describe the reverse engineering of the cipher, authentication protocol and key diversification algorithms. Moreover, we find critical weaknesses in both iClass Standard and iClass Elite. We exploit these weaknesses in three attacks, one on the iClass Standard key diversification, one that recovers a master key on iClass Standard and one that recovers the master key on iClass Elite. The first attack against iClass Standard key diversification only uses weaknesses related to the key diversification algorithm. Here, the master key can only be recovered using a strong adversarial model where the adversary controls a genuine reader and is able to send key update commands. Later in the second attack, we are able to recover the master key from an iClass Standard system with an adversary that is more limited in its capabilities. Here, the adversary no longer needs to control the reader. This second attack uses weaknesses that were found in the cipher and card implementation which allows recovery of the secret card key. This attack requires one authentication attempt with a legitimate reader and  $2^{22}$  queries to a card and has a computational complexity of  $2^{40}$  MAC computations. The whole attack can be executed within a day on ordinary hardware. Remarkably our last attack, against the supposedly more secure system iClass Elite, is significantly faster. It directly recovers the master key from only 15 authentication attempts with a legitimate reader. The computational complexity of this attack is lower than  $2^{25}$  MAC computations, which means that it can be fully executed within 5 seconds on an ordinary laptop.

## 5.1 Research context and related work

Over the last few years, much attention has been paid to the (in)security of the cryptographic mechanisms used in contactless smart cards [GdKGM<sup>+</sup>08, GvRVWS10, PN12, VGB12]. Experience has shown that the secrecy of proprietary ciphers does not contribute to its cryptographic strength. Most notably the Mifare Classic, which we investigated in Chapter 4, has been thoroughly broken in the last few years [NESP08, dKGGH08, GdKGM<sup>+</sup>08, GvRVWS09, Cou09]. Other prominent examples include KeeLoq [Bog07, KKMP09] and Hitag2 [COQ09, SNC09, vN11, SHXZ11, VGB12] used in car keys, CryptoRF [GvRVWS10, BKZ11, BGV<sup>+</sup>12] used in access control and payment systems and the A5/1 [Gol97], DECT [LST<sup>+</sup>09] and GMR [DHW<sup>+</sup>12] ciphers used in cordless phones. HID proposes iClass as a migration option for systems us-



ing Mifare Classic, boosting that iClass provides “improved security, performance and data integrity”<sup>2</sup>. The details of the security mechanisms of iClass remained secret for almost one decade.

During the course of our research Kim, Jung, Lee, Jung and Han have made a technical report [KJL<sup>+</sup>11] available online describing independent reverse engineering of the cipher used in iClass. Their research takes a very different, hardware oriented approach. They recovered most of the cipher by slicing the chip and analyzing the circuits with a microscope. Our approach, however, is radically different as our reverse engineering is based on the disassembly of the reader’s firmware and the study of the communication behavior of tags and readers. Furthermore, the description of the cipher by Kim et al. contains a major flaw. Concretely, their key byte selection function in the cipher is different from the one used in iClass which results in incompatible keys. Kim et al. have proposed two key recovery attacks. The first one is theoretical, in the sense that it assumes that an adversary has access to a MAC oracle over messages of arbitrary length. This assumption is unrealistic since neither the card nor the reader provide access to such a powerful oracle. Their second attack requires full control over a legitimate reader in order to issue arbitrary commands. Besides this assumption, it requires  $2^{42}$  online authentication queries which, in practice, would take more than 710 years to gather. Our attacks, however, are practical in the sense that they can be executed within a day and require only wireless communication with a genuine iClass card/reader.

### Research contribution

The contribution of this chapter consists of several parts. First it describes the reverse engineering of the built-in key diversification algorithm of iClass Standard security. The basic diversification algorithm, which also forms the basis for iClass Elite key diversification, consists of two parts: a cipher that is used to encrypt the identity of the card; and a key fortification function, called *hash0* in HID documentation, which is intended to add extra protection to the master key.

We show that the key fortification function *hash0* is actually not one-way nor collision resistant and therefore it adds little protection to the master key. To demonstrate this, we give the inverse function  $hash0^{-1}$  that on input of a 64 bit bitstring outputs a modest amount (on average 4) of candidate pre-images. This results in our first attack on the iClass Standard key diversification that recovers a master key from an iClass reader which is of comparable complexity to that of breaking single DES. It only uses weaknesses in the key diversification algorithm. Since in the end it comes down to breaking DES, it can be accomplished within a few days on a RIVYERA<sup>3</sup>. This is extremely sensitive since there is only one master key for all iClass Standard readers and from this master key all diversified card keys can be computed. As a

---

<sup>2</sup><http://www.hidglobal.com/pr.php?id=393>

<sup>3</sup>A generic massively parallel FPGA-computer which is designed to run parallel tasks at speeds that approach the processing speed of custom built hardware. See <http://www.sciengines.com>

faster alternative, it is possible to emulate a predefined card identity and use a DES rainbow table [Hel80,Oec03] based on this identity to perform the attack. This allows an adversary to recover the master key even within minutes.

Furthermore, we have fully reverse engineered iClass's proprietary cipher and authentication protocol. This task of reverse engineering is not trivial since it was first necessary to bypass the read protection mechanisms of the microcontroller used in the readers in order to retrieve its firmware. We also found serious vulnerabilities in the cipher that enable an adversary to recover the secret card key by just wirelessly communicating with the card. The potential impact of this second and improved attack against iClass Standard is vast since when it is combined with the vulnerabilities in the key diversification algorithm, which we exploited earlier, it allows an adversary to use this secret key to recover the master key. Additionally, we have reverse engineered the iClass Elite key diversification algorithm which we also describe in full detail. We show that this algorithm has even more serious vulnerabilities than the iClass Standard key diversification. In our third and last attack, an adversary is able to directly recover an "Elite" master key by simply communicating with a legitimate iClass reader.

Concretely, we propose three key recovery attacks: one on the iClass Standard key diversification, one against iClass Standard and one against iClass Elite. All attacks allow an adversary to recover the master key.

- The first attack, against iClass Standard key diversification, exploits the fact that the key diversification algorithm can be inverted. An adversary needs to let the genuine reader issue a key update command. The card key will be updated and from the eavesdropped communication the adversary learns the card key. The adversary proceeds by inverting the key diversification which results in a modest amount of pre-images. Now, only a bruteforce attack on single DES will reveal which master key was used.
- The second attack, against iClass Standard, exploits a total of *four* weaknesses in the cipher, key diversification algorithm and card implementation. In order to execute this attack the adversary first needs to eavesdrop one legitimate authentication session between the card and reader. Then it runs  $2^{19}$  key updates and  $2^{22}$  authentication attempts with the card. This takes less than six hours to accomplish (when using a Proxmark III as a reader) and recovers 24 bits of the card key. Finally, off-line, the adversary needs to search for the remaining 40 bits of the key. Having recovered the card key, the adversary gains full control over the card. Furthermore, computing the master key from the card key is as hard as breaking single DES and is done like in the first attack.
- The third attack, concerning iClass Elite, exploits *two* weaknesses in the key diversification algorithm and recovers the master key directly. In order to run this attack the adversary only needs to run 15 authentication attempts with a legitimate reader. Afterwards, off-line, the adversary needs to compute only

$2^{25}$  DES encryptions in order to recover the master key. This attack, from beginning to end runs within 5 seconds on ordinary hardware.

We have executed all attacks in practice and verified these claims and attack times. For eavesdropping and card emulation we used a Proxmark III, see Section 2.2.

## Chapter outline

This chapter is organized as follows. Section 5.2 starts with a description of the iClass architecture, the functionality of the card, the cryptographic algorithms. Then, Section 5.3 describes the reverse engineering of the key diversification scheme that is used in iClass Standard. Here, we also give an attack against this iClass Standard key diversification that recovers the master key from a diversified key. This attack method forms the basis for the second attack against iClass Standard where it is used to recover the master key in its last step. The second attack itself is described in Section 5.5 after the reverse engineering and description of the cipher in Section 5.4. Finally, Section 5.6 describes the key diversification in iClass Elite and presents an attack against this scheme. We conclude with Section 5.7.

## 5.2 iClass

An HID iClass card is in fact a pre-configured and re-branded PicoPass card manufactured by Inside Secure<sup>4</sup>. HID configures and locks the cards so that the configuration settings can no longer be modified. This section describes in detail the functionality and security mechanisms of iClass and it also describes the reverse engineering process. Let us first introduce notation.

**Notation 5.1.** *Throughout this chapter  $\epsilon$  denotes the empty bitstring.  $\oplus$  denotes the bitwise exclusive or.  $\boxplus$  denotes addition modulo 256. Given two bitstrings  $x$  and  $y$ ,  $xy$  denotes their concatenation. Sometimes we write this concatenation explicitly with  $x \cdot y$  to improve readability.  $\bar{x}$  denotes the bitwise complement of  $x$ .  $0^n$  denotes a bitstring of  $n$  zero-bits. Similarly,  $1^n$  denotes a bitstring of  $n$  one-bits. Furthermore, given a bitstring  $x \in (\mathbb{F}_2^k)^l$ , we denote with  $x_{[i]}$  the  $i$ -th element  $y \in \mathbb{F}_2^k$  of  $x$ . We write  $y_i$  to denote the  $i$ -th bit of  $y$ . For example, given the bitstring  $x = 0x010203 \in (\mathbb{F}_2^8)^3$  and  $y := x_{[2]}$  then  $y = 0x03$  and  $y_6 = 1$ .*

**Remark 5.1** (Byte representation). *Throughout this chapter, bytes are represented with their most significant bit on the left. However, the least significant bit is transmitted first over the air (compliant with ISO/IEC 15693). This is the same order in which the bits are input to the cryptographic functions. In other words,  $0x0a0b0c$  is transmitted and processed as input  $0x50d030$ .*

<sup>4</sup><http://www.insidesecond.com/eng/Products/Secure-Solutions/PicoPass>

Block	Content	Description	
0	Card serial number	Identifier $id$	publicly readable
1	Configuration		publicly readable
2	e-Purse	Card challenge $c_C$	publicly readable
3	Key for application 1	Diversified debit key $k_1$	publicly readable
4	Key for application 2	Diversified credit key $k_2$	publicly readable
5	Application issuer area		publicly readable
6...18	Application 1	HID application	write-only after authentication
19...n	Application 2	User defined memory	read-write after authentication

Figure 5.1: Memory layout of an iClass card

### 5.2.1 Functionality

iClass cards come in two versions called 2KS and 16KS with respectively 256 and 4096 bytes of memory. The memory of the card is divided into blocks of eight bytes as shown in Figure 5.1. Memory blocks 0, 1, 2 and 5 are publicly readable. They contain the card identifier  $id$ , configuration bits, the card challenge  $c_C$  and issuer information. Block 3 and 4 contain two diversified cryptographic keys  $k_1$  and  $k_2$  which are derived from two different master keys  $\mathcal{K}_1$  and  $\mathcal{K}_2$ . These master keys are referred to in the documentation as debit key and credit key. The card only stores the diversified keys  $k_1$  and  $k_2$ . The remaining memory blocks are divided into two areas, which are represented by the host software as applications. The size of these applications is defined by the configuration block.

The first application of an iClass card is the *HID application* which stores the card identifier, PIN code, password and other information used in access control systems. Read and write access to the HID application requires a valid mutual authentication using the cipher to prove knowledge of  $k_1$ . The master key of the HID application is a global key known to all iClass Standard compatible readers. The globally used key  $\mathcal{K}_1$  is kept secret by HID Global and is not shared with any customer or industrial partner. Recovery of this key undermines the security of all systems using iClass Standard. Two methods have been proposed [Mer10, GdKGV11] to recover this key. To circumvent the obvious limitations of having only a global master key, iClass Elite uses a different key diversification algorithm that allows having custom master keys. The details regarding iClass Elite can be found in Section 5.6.1. The second global master key  $\mathcal{K}_2$  is used in both iClass Standard and Elite systems and it is available to any developer who signs a non-disclosure agreement with HID global. It is possible to extract this key from publicly available software binaries [GdKGV11]. In addition, the document [HID06] contains this master key and is available online. This key  $\mathcal{K}_2$  can be used by developers to protect the second application, although in practice,  $\mathcal{K}_2$  is hardly ever used or modified.

The card provides basic memory operations like read and write. These operations have some non-standard behavior and therefore we describe them in detail.

- The `read` command takes as input an application number  $a$  and a memory block number  $n$  and returns the memory content of this block. This command has the side effect of selecting the corresponding key ( $k_1$  for application 1 or  $k_2$  for application 2) in the cipher and then it feeds the content of block  $n$  into the internal state of the cipher. Cryptographic keys are not readable. When the block number  $n$  corresponds to the address where a cryptographic key is stored, then `read` returns a bitstring of 64 ones.
- The `write` command takes as input a block number  $n$ , an eight-byte payload  $p$  and a MAC of the payload  $\text{MAC}(k, n \cdot p)$ , where  $k$  is a diversified card key. When successful, it writes  $p$  in memory and it returns a copy of  $p$  for verification purposes. This command has the side effect of resetting the internal state of the cipher. In addition, when the block number  $n$  corresponds to the address where a cryptographic key  $k$  is stored, the payload is XOR-ed to the previous value instead of overwriting it, i.e., it assigns  $k := k \oplus p$ .

Therefore, in order to update a key  $k$  to  $k'$ , the reader must issue a `write` command with  $k \oplus k'$  as payload. In this way the card will store  $k \oplus k \oplus k' = k'$  as the new key. On the one hand, this particular key update procedure has the special feature that in case an adversary eavesdrops a key update he is unable to learn the newly assigned key, provided that he does not know  $k$ . On the other hand this introduces a new weakness which we describe in Section 5.5.2.

**Listing 5.1: Authenticate and decrement card challenge  $c_C$  using  $k_1 = 0 \times \text{E033CA419AEE43F9}$**

Sender	Hex	Abstract
Reader	0C 00 73 33	Read identifier
Tag	47 47 6C 00 F7 FF 12 E0	Card serial number $id$
Reader	0C 01 FA 22	Read configuration
Tag	12 FF FF FF E9 1F FF 3C	iClass 16KS configuration
Reader	88 02	Read $c_C$ and select $k_1$
Tag	FE FF FF FF FF FF FF FF	Card challenge $c_C$
Reader	05 00 00 00 00 1D 49 C9 DA	Reader nonce $n_R = 0$ , $\text{MAC}(k_1, c_C \cdot n_R)$
Tag	5A A2 AF 92	Response $\text{MAC}(k_1, c_C \cdot n_R \cdot 0^{32})$
Reader	87 02 FD FF FF FF FF FF FF	Write on block 02 followed by
	CF 3B D4 6A	$\text{MAC}(k_1, 02 \cdot c_C - 1)$
Tag	FF FF FF FF FD FF FF FF	Update successful

Before being able to execute `read` or `write` commands on the protected memory of a card, the reader needs to get access to the corresponding application by running a successful authentication protocol described in Section 5.2.2. Cryptographic keys  $k_1$  and  $k_2$  can be seen as part of application 1 and 2, respectively. This means that in order to modify a key e.g.,  $k_1$ , the reader first needs to run a successful authentication with  $k_1$ .

### 5.2.2 Authentication protocol

This section describes the authentication protocol between an iClass card and reader. This protocol is depicted in Figure 5.2 and an example trace is shown in Listing 5.1. First, during the anti-collision protocol, the reader learns the identity of the card  $id$ . Then, the reader chooses an application and issues a `read` command on the card challenge  $c_C$ . This  $c_C$  is called ‘e-purse’ in the iClass documentation [IC04] and it is a special memory block in the sense that it is intended to provide freshness. In the next step, the reader issues an `authenticate` command. This command sends to the card a reader nonce  $n_R$  and a MAC of the card challenge  $c_C$  concatenated with  $n_R$ . This MAC is computed using a diversified card key  $k$ . Finally, the card answers with a MAC of  $c_C$ ,  $n_R$  followed by 32 zero bits. For more details over the MAC function see Section 5.4.2.

After a successful authentication on  $c_C$  the reader is granted read and write access within the selected application.

**Remark 5.2.** *Since the card lacks a pseudo-random number generator, the reader should decrement  $c_C$  after a successful authentication in order to provide freshness for the next authentication, see Listing 5.1. Note that this is not enforced by the card.*

## 5.3 iClass Standard

In this chapter we first reverse engineer the iClass Standard key diversification. Then, we describe its weaknesses in Section 5.3.3. Finally, we describe the first attack against iClass Standard in Section 5.3.4.

Our first approach for reverse engineering is in line with that of [GdKGM<sup>+</sup>08, LST<sup>+</sup>09, GvRVWS10] and consists of analyzing the update card key messages sent by an iClass compatible reader while we produce small modifications on the key, just after the DES operation and just before it is passed to the fortification function  $hash0$ . We used an Omnikey reader that supports iClass. Still, we first had to bypass the encryption layer of the Omnikey Secure Mode that is used in its USB communication in order to control the reader messages. We reverse engineered the Omnikey Se-

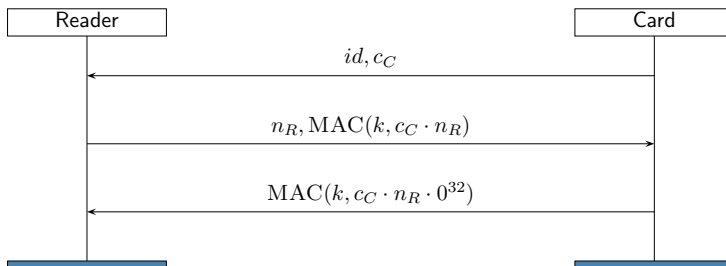


Figure 5.2: Authentication protocol

cure Mode and wrote a library that is capable of communicating in Omnikey Secure Mode to any Omnikey reader. To eavesdrop the contactless interface we have built a custom firmware for the Proxmark III in order to intercept ISO/IEC 15693 [ISO09] frames. We have released the library, firmware and an implementation of *hash0* under the GNU General Public License and they are available at the Proxmark website<sup>5</sup>.

Later in this chapter, in Section 5.4, we use a different approach for reverse engineering the cipher and the key diversification for iClass Elite. In this approach we first recover the firmware from an iClass reader. Then, by disassembling the firmware we are able to recover the cipher and key diversification for iClass Elite. The knowledge about the structure of *hash0* which we describe in this section did help a lot in identifying the interesting parts of the firmware for reverse engineering.

### 5.3.1 Black box reverse engineering

This section describes how *hash0* [Cum06] (a.k.a. *h0* [Cum03]) was reverse engineered. The final description of *hash0* is given in Section 5.3.2. The method used to reverse engineer *hash0* studies the input-output relations of *hash0* in order to recover its internal structure. The primary goal is to learn how a card key  $k$  is derived from a master key  $\mathcal{K}$  and the card identity  $id$ . The general structure of the key derivation is known. First, the iClass reader encrypts a card identity  $id$  with the master key  $\mathcal{K}$ , using single DES. The resulting ciphertext is then input to *hash0* which outputs the diversified key  $k$ .

$$k = \text{hash0}(\text{DES}_{\text{enc}}(\mathcal{K}, id))$$

We define the function *flip* that takes an input  $c$  and flips a specific bit in  $c$ . By flipping a bit we mean taking the complement of this bit. The definition *flip* is as follows.

**Definition 5.1.** Let the function *flip*:  $\mathbb{F}_2^{64} \times \mathbb{N} \rightarrow \mathbb{F}_2^{64}$  be defined as

$$\text{flip}(c, m) = c_{63} \dots c_{m+1} \cdot \overline{c_m} \cdot c_{m-1} \dots c_0$$

Since we only learn the XOR difference between two *hash0* outputs we define the function *diff* that we use to express these XOR differences. The function *diff* computes the output difference of two *hash0* calls and is defined as follows.

**Definition 5.2.** Let the function *diff*:  $\mathbb{F}_2^{64} \times \mathbb{N} \rightarrow \mathbb{F}_2^{64}$  be defined as

$$\text{diff}(c, m) = \text{hash0}(c) \oplus \text{hash0}(\text{flip}(c, m))$$

Now we use this definition of output difference to describe accumulative output differences of an input set  $\mathcal{C}$ .

$$k^{\wedge m} = \bigwedge_{c \in \mathcal{C}} \text{diff}(c, m), \quad k^{\vee m} = \bigvee_{c \in \mathcal{C}} \text{diff}(c, m)$$

<sup>5</sup><http://www.proxmark.org>

The results are grouped by the position of the flipped bit  $m$ . Then, the AND and OR is computed of all the results in a group. These cumulative AND and OR values for 64 bits that were flipped in 3000 random bitstrings  $c \in \mathcal{C}$  are presented in Figure 5.4, 5.5 and 5.6. The output difference for flipping all possible bits is abbreviated as follows.

$$k^\wedge = \bigwedge_{m=0}^{63} k^{\wedge m}, \quad k^\vee = \bigwedge_{m=0}^{63} k^{\vee m}$$

### Gathering input-output pairs

In this section we explain how we gather the input-output pairs for *hash0* and calculate the output differences. In our setup we have complete control over an iClass reader for which we can set and update the keys that are used. Furthermore, we are able to emulate iClass cards and learn all communication between the controlled reader and (emulated) iClass card. First, we analyze the input-output relations of *hash0* on bit level. This requires complete control over the input  $c$  of *hash0* which can be achieved in our test setup. In this test setup we emulate a card identity  $id$  and also know, or even can change, which master key  $\mathcal{K}$  is used. The following steps deliver XOR differences between two *hash0* evaluations that differ only one bit in the input:

- generate a large set of random bitstrings  $c \in \mathbb{F}_2^{64}$ .
- for each  $c$ 
  - calculate  $id = \text{DES}_{\text{dec}}(c, \mathcal{K})$  and  $id^m = \text{DES}_{\text{dec}}(\text{flip}(c, m), \mathcal{K})$  for  $m = 0 \dots 63$
  - for each  $m$  authenticate with  $id$ , perform a key update, the reader requests the card identity again, now use  $id^m$  instead of  $id$

Keep the master key  $\mathcal{K}$  constant during the key updates described above. This delivers the XOR of two function evaluations of the form  $\text{diff}(c, m) = \text{hash0}(c) \oplus \text{hash0}(\text{flip}(c, m))$ . We performed this procedure for 3000 random values  $c \in \mathcal{C}$  since the procedure to retrieve these values was slow and this was the amount of data that could be obtained in a couple of days.

### Function input partitioning

Figure 5.4 and 5.5 show the accumulated differences for the 48 rightmost output bits at input  $c$ . The results for the remaining 16 leftmost output bits are shown in Figure 5.6. These differences reveal that the input  $c$  of *hash0* is of the form  $c = x \cdot y \cdot z_{[0]} \dots z_{[7]}$  with  $x, y \in \mathbb{F}_2^8$  and  $z_{[i]} \in \mathbb{F}_2^6$ . The eight output bytes are defined as  $k_{[0]} \dots k_{[7]}$  and constitute the diversified key  $k$ . The structure of the masks in Figure 5.4, 5.5 and 5.6 are computed for 3000 values with  $x = y = 0^8$  and  $z$  a random bitstring. This leads to the following observations:

- $z_{[0]} \dots z_{[3]}$  affects  $k_{[4]} \dots k_{[7]}$  and  $z_{[4]} \dots z_{[7]}$  affects  $k_{[0]} \dots k_{[3]}$ .



- $z_{[0]} \dots z_{[3]}$  and  $z_{[4]} \dots z_{[7]}$  generate a similar structure in the output but are mutually independent. This suggests the use of a subfunction that is called twice, once with input  $z_{[0]} \dots z_{[3]}$  and once with input  $z_{[4]}, \dots, z_{[7]}$ . We call this function *check*.
- $y_i$  affects  $k_{[i]}$  for  $i = 0 \dots 7$ . The OR-mask for  $y$  indicates a complement operation on the output while the AND-mask indicates an injective function that maps  $y_i$  to  $k_{[i]_0}$ .
- $x$  defines a permutation. The output is scrambled after flipping a single bit within  $x$ . The AND-mask shows that  $k_{[i]_7}$  is exclusively affected by  $x$  for  $i = 0 \dots 7$ .
- flipping bits in  $z$  never affects  $k_{[i]_0}$  or  $k_{[i]_7}$ . This is inferred from the occurrences of  $0 \times 7e$  (01111110 in binary representation) in Figure 5.4 and 5.5.

The above observations suggest that we can recover different parts of the function independently. Figure 5.3 summarizes how different parts of the input affect specific parts of the output. Note that from the last observation we know that the subfunction *check* operates on  $z_{[i]_0} \dots z_{[i]_5}$  and affects  $k_{[i]_1} \dots k_{[i]_6}$ . Furthermore, it is observed that the leftmost bit of all output bytes  $k_{[i]_0}$  and the permutation of  $z_{[i]}$  to  $k_{[i]_1} \dots k_{[i]_6}$  is determined by  $x$ . Finally, every input bit  $y_i$  is copied to output bit  $k_{[i]_7}$ .

Summarizing, *hash0* can be split into three different parts. The first part is the subfunction *check* which applies a similar operation on  $z_{[0]} \dots z_{[3]}$  and  $z_{[4]} \dots z_{[7]}$ . In the second part a bitwise complement operation is computed based on bits from the input byte  $y$ . The last part applies a permutation that is defined by the input byte  $x$ . The following sections discuss the reverse engineering of these identified parts of *hash0*. Finally, the complete *hash0* definition is given in Section 5.6.

### Subfunction *check*

This section describes the reverse engineering of the subfunction *check* which operates on two times four 6-bit input values  $z_{[0]} \dots z_{[3]}$  and  $z_{[4]} \dots z_{[7]}$ . In order to recover

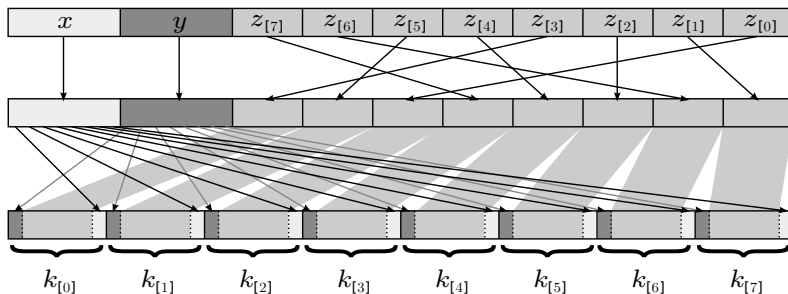


Figure 5.3: Schematic representation of the function *hash0*

bit ↓	OR-mask of differences in output $k$	AND-mask of differences in output $k$
$z[7]$ { 63	0x7e7e7e7e00000000	0x0400000000000000
62	0x7e7e7e7e00000000	0x0400000000000000
61	0x7a7e7e7e00000000	0x0800000000000000
60	0x727e7e7e00000000	0x1000000000000000
59	0x627e7e7e00000000	0x2000000000000000
58	0x427e7e7e00000000	0x4000000000000000
57	0x007e7e7e00000000	0x0000000000000000
$z[6]$ { ...	...	...
52	0x007e7e7e00000000	0x0000000000000000
51	0x00007e7e00000000	0x0000000000000000
$z[5]$ { ...	...	...
46	0x00007e7e00000000	0x0000000000000000
45	0x0000007e00000000	0x0000000000000000
$z[4]$ { ...	...	...
40	0x0000007e00000000	0x0000000000000000

Figure 5.4: OR and AND-mask for flipping bits 40 . . . 63 in input  $c$ 

bit ↓	OR-mask of differences in output $k$	AND-mask of differences in output $k$
$z[3]$ { 39	0x0000000027e7e7e	0x0000000020000000
38	0x0000000047e7e7e	0x0000000040000000
37	0x0000000087e7e7e	0x0000000080000000
36	0x0000000107e7e7e	0x0000000100000000
35	0x0000000207e7e7e	0x0000000200000000
34	0x0000000407e7e7e	0x0000000400000000
33	0x0000000007e7e7e	0x0000000000000000
$z[2]$ { ...	...	...
28	0x0000000007e7e7e	0x0000000000000000
27	0x000000000007e7e	0x0000000000000000
$z[1]$ { ...	...	...
22	0x000000000007e7e	0x0000000000000000
21	0x00000000000007e	0x0000000000000000
$z[0]$ { ...	...	...
16	0x000000000000007e	0x0000000000000000

Figure 5.5: OR and AND-mask for flipping bits 16 . . . 39 in input  $c$ 

this part of the function we keep  $x = y = 0^8$  and let  $z$  vary over random bitstrings. According to Figure 5.4 and 5.5 only flipping bits in  $z$  (positions 16 to 63 of input  $c$ ) does matter for *check*. We write  $modified(x)$  to indicate changes in  $x$  between two different test cases. We make modifications to the input and see where it affects the output. We start by looking at the following rules that are deduced from Figure 5.4 an 5.5.

$$\begin{aligned}
 modified(k_{[0]}) &\rightarrow modified(z_{[7]}) \wedge \neg modified(z_{[0]} \dots z_{[6]}) \\
 modified(k_{[4]}) &\rightarrow modified(z_{[3]}) \wedge \neg modified(z_{[0]} \dots z_{[2]}) \wedge \neg modified(z_{[4]} \dots z_{[7]})
 \end{aligned}$$

bit ↓	OR-mask of differences in output $k$	AND-mask of differences in output $k$
15	0xfc00000000000000	0x8000000000000000
14	0x00fc000000000000	0x0080000000000000
13	0x0000fc0000000000	0x0000800000000000
12	0x000000fc00000000	0x0000008000000000
11	0x00000000fe000000	0x00000000fe000000
10	0x0000000000fe0000	0x0000000000fe0000
9	0x000000000000fe00	0x000000000000fe00
8	0x00000000000000fe	0x00000000000000fe
7	0x7f7f7f7e7e7f7f7f	0x0101010000010101
6	0x000007f7e7f000000	0x0000010001000000
5	0x7f7e7e7e7f000000	0x0100000001000000
4	0x7f7e7e7e7e7f0000	0x0100000000010000
3	0x00007f7e7e7e7f00	0x0000010000000100
2	0x7f7e7f7f7f7f00	0x0100010101010100
1	0x7f7e7f7e7e7f7f00	0x0100010000010100
0	0x7f7e7f7e7f7e7f00	0x0100010001000100

Figure 5.6: OR and AND-mask for flipping bits 0 . . . 15 in input  $c$

Note that

$$k_{[4]_1} \dots k_{[4]_6} = z_{[3]}. \tag{5.1}$$

For  $k_{[0]}$  it is harder to find a function since flipping a single bit in  $z_{[7]}$  may affect multiple bits in  $k_{[0]}$ . The relation between  $z_{[7]}$  and  $k_{[0]}$  becomes more clear when we look at specific input patterns and their corresponding output difference in Figure 5.7. The stars in the input pattern for  $z_{[7]}$  denote a bit that can be either 0 or 1 without affecting the output difference of  $k_{[0]}$ . Note that, of course, the input bit that is being flipped can also be either 0 or 1 and is therefore also denoted by a star. We try to capture the output differences for flipping all possible bits between two different inputs  $c$ .

$z_{[7]}$ of $c$	$diff(c, 63)_{[0]}$	$z_{[7]}$ of $c$	$diff(c, 62)_{[0]}$
***0*	06	****0	04
**01*	0e	**0*1	0c
*011*	1e	*01*1	1c
*0111*	3e	*011*1	3c
11111*	7c	0111*1	7c
01111*	7e	1111*1	7e

Figure 5.7: Input-output relations for  $z_{[7]} \leftrightarrow k_{[0]}$

The output difference  $k_{[0]}^\vee$  based on flipping bits in  $z_{[7]}$  is:

$$k_{[0]_1}^\vee \dots k_{[0]_6}^\vee = (z_7 \bmod 63) + 1 \oplus (z'_7 \bmod 63) + 1$$

from which we deduce that

$$k_{[0]_1} \dots k_{[0]_6} = (z_7 \bmod 63) + 1 \tag{5.2}$$

%	$z_{[6]}$ of $c$	$z_{[7]}$ of $c$	$diff(c, 57)_{[1]}$	
0.97	*****	*****	0c	} bit-flip $z_{6[5]}$
0.03	00010*	000101	0c	
	10011*	101000	52	
	11001*	110100	6c	
	...	...	...	
%	$z_{[6]}$ of $c$	$z_{[7]}$ of $c$	$diff(c, 56)_{[1]}$	
0.97	*****	*****	0c	} bit-flip $z_{6[4]}$
	***1**	*****	1c	
	*011**	*****	3c	
	1111**	*****	78	
	0111**	*****	7c	
0.03	0010*0	001001	1a	
	0110*0	011001	3a	
	1001*0	100111	4e	
	1100*1	110100	64	
	...	...	...	

Figure 5.8: Input-output relations for  $z_{[6]} \cdot z_{[7]} \leftrightarrow k_{[1]}$

Our next step is to find  $k_{[1]_1} \dots k_{[1]_6}$  which depends on two input values  $z_{[6]}$  and  $z_{[7]}$ .

$$modified(k_{[1]}) \rightarrow modified(z_{[7]}) \vee modified(z_{[6]}) \wedge \neg modified(z_{[0]} \dots z_{[5]})$$

Again, we construct an overview of all input-output relations (Fig. 5.8). The observations for flipping  $z_{6[5]}$  and  $z_{6[4]}$  show that sometimes  $z_{[6]}$  and  $z_{[7]}$  are independent and sometimes they are not, e.g., when  $z_{[7]} = *****$ , this means that they are independent. From the cases where both  $z_{[6]}$  and  $z_{[7]}$  have a specific value (see Fig. 5.8), it is clear that flipping a bit in  $z_{[6]}$  affects the equality  $z_{[6]} + 1 = z_{[7]}$ .

We look at an example where  $diff(c, 56)_{[1]} = 0x3c$ :

$$\begin{array}{rcl} z_{[6]} = 0011\mathbf{0}1, & z_{[6]} + 2 & = 001111 \\ z'_{[6]} = 0011\mathbf{1}1, & z'_{[6]} + 2 & = \frac{010001 \oplus}{011110} \end{array}$$

As a result  $k_{[1]_1} \dots k_{[1]_6} = 011110$  which reads as  $0x3c$  in the output. Another example is the case where  $diff(c, 56)_{[1]} = 0x78$ . Careful inspection of many samples shows that there is a modulo operation involved. Input  $z_{[6]}$  is taken modulo 62, which is 111110 in binary. So, an example for  $diff(c, 56)_{[1]} = 0x78$  is:

$$\begin{array}{rcl} z_{[6]} = 1111\mathbf{0}0, & (z_{[6]} \bmod 62) + 2 & = 111110 \\ z'_{[6]} = 1111\mathbf{1}0, & (z'_{[6]} \bmod 62) + 2 & = \frac{000010 \oplus}{111100} \end{array}$$

This results in  $k_{[1]_1} \dots k_{[1]_6} = 111100$  which reads as  $0x78$  in the output. The last thing that we try to reveal is how the equality  $z_{[6]} + 1 = z_{[7]}$  affects the output. Close

inspection of our samples shows that  $k_{[1]_1} \dots k_{[1]_6} = 1$  when the relation holds and  $k_{[1]_1} \dots k_{[1]_6} = (z_{[6]} \bmod 62) + 2$  when it does not hold. An illustrating example is  $\text{diff}(c, 56)_{[1]} = 0x4e$ :

$$\begin{array}{rcl} z_{[6]} = 1001\mathbf{00}, (z_{[6]} \bmod 62) + 2 & = & 100110 \\ z'_{[6]} = 1001\mathbf{10}, ((z'_{[6]} \bmod 62) + 1 = z_{[7]}) & = & \frac{000001 \oplus}{100111} \end{array}$$

This results in  $k_{[1]_1} \dots k_{[1]_6} = 100111$  which reads as  $0x4e$  in the output. Eventually, the function for  $k_{[1]_1} \dots k_{[1]_6}$  is:

$$k_{[1]_1} \dots k_{[1]_6} = \begin{cases} 1, & \text{if } (z_{[6]} \bmod 62) + 1 = (z_{[7]} \bmod 63); \\ (z_{[6]} \bmod 62) + 2, & \text{otherwise.} \end{cases} \quad (5.3)$$

From here we proceed by recovering  $k_{[2]}$  and  $k_{[3]}$ . It is helpful to look for similarities and repeating patterns, like for example the continuation of the decreasing modulus for  $z_{[5]}$  and  $z_{[4]}$ .

$$\begin{array}{rcl} z_{[7]} \bmod 63 & \xrightarrow{?} & z_{[5]} \bmod 61 \\ z_{[6]} \bmod 62 & & z_{[4]} \bmod 60 \end{array}$$

Once  $k_{[0]} \dots k_{[3]}$  is recovered, we use the uncovered relations and check whether they can be applied to describe  $k_{[4]} \dots k_{[7]}$  as well. We give  $k_{[2]_1} \dots k_{[2]_6}$  to show how the structure of the function evolves:

$$k_{[2]_1} \dots k_{[2]_6} = \begin{cases} 2, & \text{if } (z_{[5]} \bmod 61) + 1 = (z_{[6]} \bmod 62) \wedge (z_{[7]} \bmod 63) \neq 0; \\ 1, & \text{if } (z_{[5]} \bmod 61) + 1 = (z_{[6]} \bmod 62) \wedge (z_{[7]} \bmod 63) = 0; \\ 1, & \text{if } (z_{[5]} \bmod 61) + 2 = (z_{[7]} \bmod 63); \\ (z_{[5]} \bmod 61) + 3, & \text{otherwise.} \end{cases}$$

A concise and more refined definition of the function is given in Section 5.3.2. Eventually, the modulo operations are separated from the subfunction *check* and defined in the main function *hash0*. Also, the definition in Section 5.3.2 clarifies why the subfunction is called *check*. It checks equalities between the different components of  $z$  and affects the output accordingly.

### Complement byte

The second byte of the input  $c$  is the complement byte  $y$ . It performs a complement operation on the output of the function as Figure 5.6 clearly shows. Flipping bit  $y_i$  results in the complement of  $k_{[i]_7}$  in the output, for  $i = 0 \dots 7$ . Note that no other input bit influences any least significant output bit of the output bytes  $k_{[i]_7}$ . Furthermore,  $k_{[2]_1} \dots k_{[2]_6}$  are flipped, however, keep in mind that we do not involve the action of byte  $x$  at this point, which prevents any permutation of the output.

Finally, every  $k_{[i]_0}$  is not affected. It is important to observe that for  $k_{[4]} \dots k_{[7]}$  the OR and AND-mask agree that the left 7 bits are always flipped while for  $k_{[0]} \dots k_{[3]}$  this is not true. To be precise, the bits  $k_{[i]_6}$  for  $i = 0 \dots 3$  are *never* flipped. We found

that the output value  $z_{[j]}$  that constitutes output byte  $k_{[i]}$  is decremented by one if  $j \leq 3$  except when  $y_i = 0$ . Example for  $\text{diff}(c, 15)_{[0]} = 0\text{xfc}$  with unknown bit  $t$ :

$$\begin{aligned} z_{[j]} &= 101101, & \text{where } j \leq 3 \\ y_7 = 0, & \quad k_{[0]} = y_7 \cdot \frac{z_{[j]} \cdot t}{1} = 0101101t \\ y'_7 = 1, & \quad k'_{[0]} = y'_7 \cdot \frac{z_{[j]} - 1 \cdot t}{1} = \frac{1010011t \oplus}{11111100} \end{aligned}$$

This results in  $k_{[0]} = 11111100$  which reads as  $0\text{xfc}$  in the output.

### Permute byte

Finally, the byte  $x$  defines a permutation. Iterating over  $x$  while  $y$  and  $z_{[0]} \dots z_{[7]}$  are constants shows that  $x$  is taken modulo 70. This follows from the fact that the output values repeat every 70 inputs. The cumulative bitmasks of the output differences, shown in Figure 5.6, do not provide information about the permutation but do show that  $k_{[i]_7}$  is affected. Experiments show that  $x$  is an injective mapping on  $k_{[i]_7}$  for  $i = 0 \dots 7$ . This means that it is possible to learn  $x$  by looking at the least significant output bits  $k_{[i]_7}$ .

Furthermore, we conclude that the permutation is independent of  $y$  and  $z$ . This means that a permutation function *permute* can be constructed which takes  $x \bmod 70$  as input and returns a particular mapping. We could recover this permutation because the values for  $k_{[i]_7}$ , for  $i = 0 \dots 7$ , directly relate to a unique mapping of the  $z$  input. The *hash0* function can be split up into *check* and *permute* subfunctions and is defined in Section 5.3.2.

### 5.3.2 The function *hash0*

The following sequence of definitions describe the recovered function *hash0* in detail.

**Definition 5.3.** Let the function *check*:  $(\mathbb{F}_2^6)^8 \rightarrow (\mathbb{F}_2^6)^8$  be defined as

$$\text{check}(z_{[0]} \dots z_{[7]}) = \text{ck}(3, 2, z_{[0]} \dots z_{[3]}) \cdot \text{ck}(3, 2, z_{[4]} \dots z_{[7]})$$

where *ck*:  $\mathbb{N} \times \mathbb{N} \times (\mathbb{F}_2^6)^4 \rightarrow (\mathbb{F}_2^6)^4$  is defined as

$$\text{ck}(1, -1, z_{[0]} \dots z_{[3]}) = z_{[0]} \dots z_{[3]}$$

$$\text{ck}(i, -1, z_{[0]} \dots z_{[3]}) = \text{ck}(i - 1, i - 2, z_{[0]} \dots z_{[3]})$$

$$\text{ck}(i, j, z_{[0]} \dots z_{[3]}) = \begin{cases} \text{ck}(i, j - 1, z_{[0]} \dots z_{[i]} \leftarrow j \dots z_{[3]}), & \text{if } z_{[i]} = z_{[j]}; \\ \text{ck}(i, j - 1, z_{[0]} \dots z_{[3]}), & \text{otherwise.} \end{cases}$$

**Definition 5.4.** Define the function *permute*:  $\mathbb{F}_2^n \times (\mathbb{F}_2^6)^8 \times \mathbb{N} \times \mathbb{N} \rightarrow (\mathbb{F}_2^6)^8$  as

$$\text{permute}(\epsilon, z, l, r) = \epsilon$$

$$\text{permute}(p_0 \dots p_n, z, l, r) = \begin{cases} (z_{[l]} + 1) \cdot \text{permute}(p_0 \dots p_{n-1}, z, l + 1, r), & \text{if } p_n = 1; \\ z_{[r]} \cdot \text{permute}(p_0 \dots p_{n-1}, z, l, r + 1), & \text{otherwise.} \end{cases}$$

**Definition 5.5.** Define the bitstring  $\pi \in (\mathbb{F}_2^8)^{35}$  in hexadecimal notation as

$$\begin{aligned} \pi = & 0x0F171B1D1E272B2D2E333539363A3C474B \\ & 4D4E535556595A5C636566696A6C71727478 \end{aligned}$$

Each byte in this sequence is a permutation of the bitstring  $00001111$ . Note that this list contains only the half of all possible permutations. The other half can be computed by taking the bit complement of each element in the list.

Finally, the definition of *hash0* is as follows.

**Definition 5.6.** Let the function  $\text{hash0}: \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times (\mathbb{F}_2^6)^8 \rightarrow (\mathbb{F}_2^8)^8$  be defined as

$$\text{hash0}(x, y, z_{[0]} \dots z_{[7]}) = k_{[0]} \dots k_{[7]}$$

where

$$\begin{aligned} z'_{[i]} &= (z_{[i]} \bmod (63 - i)) + i & i = 0 \dots 3 \\ z'_{[i+4]} &= (z_{[i+4]} \bmod (64 - i)) + i & i = 0 \dots 3 \\ \hat{z} &= \text{check}(z') \\ p &= \begin{cases} \overline{\pi_{[x \bmod 35]}}, & \text{if } x_0 = 1; \\ \pi_{[x \bmod 35]}, & \text{otherwise.} \end{cases} \\ \tilde{z} &= \text{permute}(p, \hat{z}, 0, 4) \\ k_{[i]} &= \begin{cases} y_i \cdot \overline{\tilde{z}_{[i]}} \cdot p_i + 1, & \text{if } y_i = 1; \\ y_i \cdot \tilde{z}_{[i]} \cdot \overline{p_i}, & \text{otherwise.} \end{cases} & i = 0 \dots 7 \end{aligned}$$

This concludes the reverse engineering of the key diversification algorithm that is used in iClass Standard and defined as follows.

$$k = \text{hash0}(\text{DES}_{\text{enc}}(\mathcal{K}, id))$$

We reverse engineered all parts of this function by analysing the input-output relations of the function that was running on an Omnikey reader.

### 5.3.3 Weaknesses in iClass Standard key diversification

This section describes weaknesses in the design of the Omnikey Secure Mode and on the iClass built-in key diversification and fortification algorithms. These weaknesses will be later exploited in Section 5.3.4.

## Omniquey Secure Mode

Even though encrypting the communication over USB is in principle a good practice, the way it is implemented in the Omnikey Secure Mode adds little security. The shared key  $k_{CUW}$  that is used for this practice is the same for all Omnikey readers. This key is included in software that is publicly available online, which only gives a false feeling of security.

## Weak key fortification

This section clarifies why  $hash0$  is not strengthening the diversified key  $k_{id}$  at all. First, note that only the modulo operations in  $hash0$  on  $x \pmod{\frac{256}{70}}$  and  $z_{[0]}, \dots, z_{[7]}$  are responsible for collisions in the output. The expected number of pre-images for an output of  $hash0$  is given by

$$\frac{256}{70} \times \frac{64}{60} \times \prod_{n=61}^{63} \left( \frac{64}{n} \right)^2 \approx 4.72$$

When we want to invert the function  $hash0$  we need to find the possible inputs that generate one specific output. Once we find a pre-image, we need to determine if there exists other values within the input domain that leads to the same output when the modulus is taken. Note that each input value  $z_{[i]}$  may have a second pre-image that maps to the same output. Furthermore, every permute byte  $x$  has at least two other values that map to the same output and in some cases there is even a third one. This means that the minimal number of pre-images is three. The probability  $p$  that for a given random input  $c$  there are only two other pre-images is

$$\frac{24}{70} \times \frac{60}{64} \times \prod_{n=61}^{63} \left( \frac{n}{64} \right)^2 \approx 0.27$$

This means that  $hash0$  does not add much additional protection. For example, imagine an adversary who can learn the output  $k_{id}$  of  $hash0(\text{DES}_{\text{enc}}(\mathcal{K}, id))$  for arbitrary values  $id$ . Then, the probability  $p'$  for an adversary to obtain an output  $k_{id}$  which has only three pre-images is  $1 - (1 - p)^n$ , where  $n$  is the number of function calls using random identities  $id$ . For  $n = 15$  this probability becomes  $p' > 0.99$ .

## Inverting $hash0$

It is relatively easy to compute the inverse of the function  $hash0$ . Let us first compute the inverse of the function  $check$ . Observe that the function  $check^{-1}$  is defined just as  $check$  except for one case where the condition and assignment are swapped, see Definition 5.7.

**Definition 5.7.** Let the function  $check^{-1}: (\mathbb{F}_2^6)^4 \rightarrow (\mathbb{F}_2^6)^4$  be defined as  $check(z_{[0]} \dots z_{[7]})$



in Definition 5.3 except for the following case where

$$ck^{-1}(i, j, z_{[0]} \dots z_{[3]}) = \begin{cases} ck^{-1}(i, j-1, z_{[0]} \dots z_{[i]} \leftarrow z_{[j]} \dots z_{[3]}), & \text{if } z_{[i]} = j; \\ ck^{-1}(i, j-1, z_{[0]} \dots z_{[3]}), & \text{otherwise.} \end{cases}$$

**Definition 5.8.** Define the function  $permute^{-1}: \mathbb{F}_2^n \times (\mathbb{F}_2^6)^8 \times \mathbb{N} \times \mathbb{N} \rightarrow (\mathbb{F}_2^6)^8$  as

$$permute^{-1}(p, z, l = 12, r) = \epsilon$$

$$permute^{-1}(p, z, l < 4, r) = \begin{cases} (z_{[r]} - 1) \cdot permute^{-1}(p, z, l+1, r+1), & \text{if } p_r = 1; \\ permute^{-1}(p, z, l, r+1), & \text{otherwise.} \end{cases}$$

$$permute^{-1}(p, z, l \geq 4, r) = \begin{cases} z_{[l-4]} \cdot permute^{-1}(p, z, l+1, r), & \text{if } p_{l-4} = 0; \\ permute^{-1}(p, z, l+1, r), & \text{otherwise.} \end{cases}$$

Next, we define the function  $hash0^{-1}$ , the inverse of  $hash0$ . This function outputs a set  $\mathcal{C}$  of candidate pre-images. These pre-images output the same key  $k$  when applying  $hash0$ . The definition of  $hash0^{-1}$  is as follows.

**Definition 5.9.** Let the function  $hash0^{-1}: (\mathbb{F}_2^8)^8 \rightarrow \{\mathbb{F}_2^8 \times \mathbb{F}_2^8 \times (\mathbb{F}_2^6)^8\}$  be defined as

$$hash0^{-1}(k_0 \dots k_7) = \mathcal{C}$$

where

$$\begin{aligned} \mathcal{C} = & \{x|x \equiv x' \pmod{70}\} \times \{y\} \times \\ & \{z_7|z_7 \equiv \dot{z}_7 \pmod{61}\} \times \{z_6|z_6 \equiv \dot{z}_6 \pmod{62}\} \times \\ & \{z_5|z_5 \equiv \dot{z}_5 \pmod{63}\} \times \{z_4|z_4 \equiv \dot{z}_4 \pmod{64}\} \times \\ & \{z_3|z_3 \equiv \dot{z}_3 \pmod{60}\} \times \{z_2|z_2 \equiv \dot{z}_2 \pmod{61}\} \times \\ & \{z_1|z_1 \equiv \dot{z}_1 \pmod{62}\} \times \{z_0|z_0 \equiv \dot{z}_0 \pmod{63}\} \end{aligned}$$

$$x' \text{ is the unique element in } \mathbb{F}_2^8 \text{ s.t. } \begin{cases} p = \overline{\pi_{[x' \pmod{35}]} \leftrightarrow x'_0 = 1} \\ p = \pi_{[x' \pmod{35}] \leftrightarrow x'_0 = 0} \end{cases}$$

$$\dot{z}_{[i]} = z'_{[i]} - (i \pmod{4}) \quad i = 0 \dots 7$$

$$z' = check^{-1}(\hat{z})$$

$$\hat{z} = permute^{-1}(p, \tilde{z}, 0, 0)$$

$$\tilde{z}_{[i]} = k'_{[i]_6} \dots k'_{[i]_1} \quad i = 0 \dots 7$$

$$p_i = \overline{k'_{[i]_0}} \quad i = 0 \dots 7$$

$$k'_{[i]} = \begin{cases} \overline{k_{[i]} - 1}, & \text{if } y_i = 1; \\ k_{[i]}, & \text{otherwise.} \end{cases} \quad i = 0 \dots 7$$

$$y_i = k_{[i]_7} \quad i = 0 \dots 7$$

### Weak key diversification algorithm

The iClass Standard key diversification algorithm uses a combination of single DES and the proprietary function called *hash0*, which we reverse engineered. Based on our findings in the preceding sections, we conclude that the function *hash0* is not one-way nor collision resistant. In fact, it is possible to compute the inverse function  $\text{hash0}^{-1}$  having a modest amount (on average 4) of candidate pre-images. After recovering a secret card key, recovering an iClass master key is not harder than a chosen plaintext attack on single DES. The use of single DES encryption for key diversification results in weak protection of the master key. This is a critical weakness, especially considering that there is only one master key for the HID application of all iClass cards. Furthermore, the composition of single DES with the function *hash0* does not strengthen the secret card key in any way. Even worse, when we look at the modulo operations that are applied on the  $z$  component of the *hash0* function input, we see that this reduces the entropy of the diversified card key with 2.23 bits.

#### 5.3.4 Attacking iClass Standard key diversification

From the weaknesses that were explained in the previous section it can be concluded that *hash0* does not significantly increase the complexity of an attack on the master key  $\mathcal{K}$ . In fact, the attack explained in this section requires one brute force run on DES. For this key recovery attack we need a strong adversary model where the adversary controls a genuine reader and is able to issue key update commands. Section 5.5.5 introduces an attack that allows a more restricted adversary. In this case, we use a strong adversary that controls a genuine reader, like an Omnikey reader in Secure Mode. The adversary controls this reader and is able to issue key update commands. An attack consists of two phases and an adversary  $A$  needs to:

##### Phase 1

- emulate a random identity  $id$  to the reader;
- issue an update key command that updates from a known user defined master key  $\mathcal{K}'$  to the unknown master key  $\mathcal{K}$  that  $A$  wants to recover. Now,  $A$  can obtain  $k_{id} = \text{hash0}(\text{DES}_{\text{enc}}(\mathcal{K}, id))$  from the XOR difference;
- compute the set of pre-images  $\mathcal{C}$  by  $\text{hash0}^{-1}(k_{id})$ ;
- repeats Phase 1 until  $A$  obtains an output  $k_{id}$  with  $|\mathcal{C}| = 3$ .

##### Phase 2

- $A$  checks for every candidate DES key  $\mathcal{K}^* \in \{0, 1\}^{56}$  if  $\text{DES}_{\text{enc}}(\mathcal{K}^*, id) = c$ , for every  $c \in \mathcal{C}$ ;
- when the check above succeeds,  $A$  verifies the corresponding key  $\mathcal{K}^*$  against another set of  $id$  and  $k_{id}$ .

We have verified this attack on the two master keys  $k_1$  and  $k_2$  that are stored in the Omnikey reader for the iClass application. The key  $k_2$  was also stored in the naviGO software and we could check the key against pre-images that were selected as described above. Although we did not find  $k_1$  stored in software we were still able to verify it since we could dump the EEPROM of a reader where  $k_1$  was stored, see Section 5.4.1. It would have been possible to recover  $hash_0$  from the EEPROM as well, although the prior knowledge about  $hash_0$  allowed us to identify more quickly where the remaining cryptographic functions were located in the EEPROM.

The attack above comes down to a brute force attack on single DES. A slightly different variant is to keep the card identity  $id$  fixed and use a DES rainbow table [Hel80] that is constructed for a specific plaintext and runs through all possible encryptions of this plaintext. Note that the rainbow table needs to be pre-computed and thus a fixed plaintext must be chosen on forehand. This means that one fixed predefined  $id$  is to be used in the attack. The number of pre-images can no longer be controlled. In the worst case, the total number of pre-images is 512.

Finally, note that we need a strong adversary model in this attack. The adversary needs to control a genuine reader, by which we mean that the adversary is able to let the reader issue card key update commands. In a real-life setup this is not really feasible. The reverse engineering of the cipher and authentication protocol of iClass in Section 5.4 did not only reveal the iClass security mechanisms, but also more weaknesses that are described in Section 5.5. We use some of these weaknesses to lower the requirements on the adversary and deploy a second attack on iClass Standard where an adversary does not control the reader in Section 5.3.4

## 5.4 The iClass cipher

This section first describes the reverse engineering process employed to recover the iClass cipher and to recover the iClass Elite key diversification algorithm. Then, we only describe the reverse engineered iClass cipher. We use this in Section 5.5 to mount a second (improved) attack on iClass Standard. The recovered key diversification for iClass Elite and its corresponding weaknesses lead to the third attack which is described in Section 5.6.

### 5.4.1 Firmware reverse engineering

In order to reverse engineer the cipher and the key diversification algorithm, we have first recovered the firmware from an iClass reader. For this we used a technique introduced in [Mer10] and later used in [GdKGV11]. Next, we will briefly describe this technique. iClass readers (Fig. 5.9), as many other embedded devices, rely on the popular PIC microcontroller (Fig. 5.9b) to perform their computations. These microcontrollers are very versatile and can be flashed with a custom firmware. The (program) memory of the microcontroller is divided into a number of blocks, each of them having access control bits determining whether this block is readable/writable.

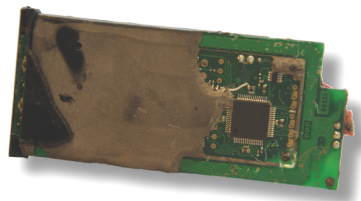
Even when the PIC is configured to be non-writable, it is always possible to reset the access control bits by erasing the memory of the chip. At first glance this feature does not seem very helpful to our reverse engineering goals since it erases the data in the memory. Conveniently enough, even when the most common programming environments do not allow it, the microcontroller supports erasure of a single block. After patching the PIC programmer software to support this feature, it is possible to perform the following attack to recover the firmware:

- Buy two iClass RW400 (6121AKN0000) readers.
- Erase block 0 on one of the readers. This resets the access control bits on block 0 to readable, writable.
- Write a small dumper program on block 0 that reads blocks  $1, \dots, n$  and outputs the data via one of the microcontroller's output pins.
- Use the serial port of a computer to record the data. This procedure recovers blocks  $1, \dots, n$ .
- Proceed similarly with the other reader, but erasing blocks  $1, \dots, n$ . This in fact fills each block with NOP operations.
- At the end of block  $n$  write a dumper program for block 0.
- At some point the program will jump to an empty block and then reach the dumper program that outputs the missing block 0.

Once we have recovered the firmware, it is possible to use IDA Pro and MPLAB<sup>6</sup> to reverse engineer the algorithms.



(a) iClass reader.



(b) iClass reader where the epoxy resin has been partially removed to expose the PIC microcontroller.

**Figure 5.9:** iClass readers

---

<sup>6</sup>Tools that can be used for disassembling, debugging and reverse engineering of software.

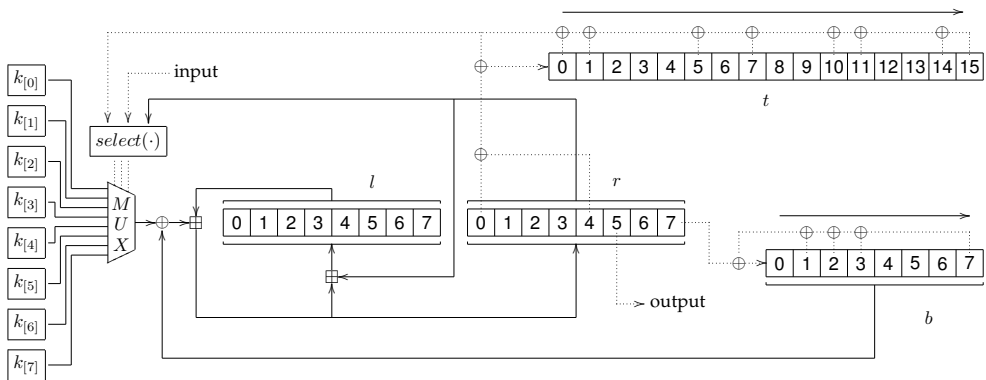


Figure 5.10: The iClass cipher

### 5.4.2 The cipher

This section describes the iClass cipher that we recovered from the firmware. This cipher is interesting from an academic and didactic perspective as it combines two important techniques in the design of stream ciphers from the '80s and beginning of the '90s, i.e., Fibonacci generators and Linear Feedback Shift Registers (LFSRs). The internal state of the iClass cipher consists of four registers as can be seen in Figure 5.10. Two of these registers, which we call left ( $l$ ) and right ( $r$ ) are part of the Fibonacci generator. The other two registers constitute linear feedback shift registers top ( $t$ ) and bottom ( $b$ ). In order to understand the description of the cipher correctly, take into account that the solid lines in Figure 5.10 represent byte operations while dotted lines represent bit operations.

**Definition 5.10** (Cipher state). *A cipher state of iClass  $s$  is an element of  $\mathbb{F}_2^{40}$  consisting of the following four components:*

- the left register  $l = (l_0 \dots l_7) \in \mathbb{F}_2^8$ ;
- the right register  $r = (r_0 \dots r_7) \in \mathbb{F}_2^8$ ;
- the top register  $t = (t_0 \dots t_{15}) \in \mathbb{F}_2^{16}$ ;
- the bottom register  $b = (b_0 \dots b_7) \in \mathbb{F}_2^8$ .

The cipher has an input bit which is used (among others) during authentication to shift in the card challenge  $c_C$  and the reader nonce  $n_R$ . With every clock tick a cipher state  $s$  evolves to a successor state  $s'$ . Both LFSRs shift to the right and the Fibonacci generator iterates using one byte of the key (chosen by the  $\text{select}(\cdot)$  function) and the bottom LFSR as input. During this iteration each of these components is updated, receiving additional input from the other components of the cipher. With each iteration, the cipher produces one output bit. The following sequence of definitions describe the cipher in detail; see also Figure 5.10.

**Definition 5.11.** The feedback function for the top register  $T: \mathbb{F}_2^{16} \rightarrow \mathbb{F}_2$  is defined by

$$T(x_0x_1 \dots x_{15}) = x_0 \oplus x_1 \oplus x_5 \oplus x_7 \oplus x_{10} \oplus x_{11} \oplus x_{14} \oplus x_{15}.$$

**Definition 5.12.** The feedback function for the bottom register  $B: \mathbb{F}_2^8 \rightarrow \mathbb{F}_2$  is defined by

$$B(x_0x_1 \dots x_7) = x_1 \oplus x_2 \oplus x_3 \oplus x_7.$$

**Definition 5.13 (Selection function).** The selection function  $select: \mathbb{F}_2 \times \mathbb{F}_2 \times \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^3$  is defined by

$$select(x, y, r) = z_0z_1z_2$$

where

$$z_0 = (r_0 \wedge r_2) \oplus (r_1 \wedge \bar{r}_3) \oplus (r_2 \vee r_4)$$

$$z_1 = (r_0 \vee r_2) \oplus (r_5 \vee r_7) \oplus r_1 \oplus r_6 \oplus x \oplus y$$

$$z_2 = (r_3 \wedge \bar{r}_5) \oplus (r_4 \wedge r_6) \oplus r_7 \oplus x$$

**Definition 5.14 (Successor state).** Let  $s = \langle l, r, t, b \rangle$  be a cipher state,  $k \in (\mathbb{F}_2^8)^8$  be a key and  $y \in \mathbb{F}_2$  be an input bit. Then, the successor cipher state  $s' = \langle l', r', t', b' \rangle$  is defined as

$$\begin{aligned} t' &:= (T(t) \oplus r_0 \oplus r_4)t_0 \dots t_{14} & l' &:= (k_{[select(T(t), y, r)]} \oplus b') \boxplus l \boxplus r \\ b' &:= (B(b) \oplus r_7)b_0 \dots b_6 & r' &:= (k_{[select(T(t), y, r)]} \oplus b') \boxplus l \end{aligned}$$

We define the successor function  $suc$  which takes a key  $k \in (\mathbb{F}_2^8)^8$ , a state  $s$  and an input  $y \in \mathbb{F}_2$  and outputs the successor state  $s'$ . We overload the function  $suc$  to multiple bit input  $x \in \mathbb{F}_2^n$  which we define as

$$suc(k, s, \epsilon) = s$$

$$suc(k, s, x_0 \dots x_n) = suc(k, suc(k, s, x_0 \dots x_{n-1}), x_n)$$

**Definition 5.15 (Output).** Define the function  $output$  which takes an internal state  $s = \langle l, r, t, b \rangle$  and returns the bit  $r_5$ . We also define the function  $output$  on multiple input bits which takes a key  $k$ , a state  $s$  and an input  $x \in \mathbb{F}_2^n$  as

$$output(k, s, \epsilon) = \epsilon$$

$$output(k, s, x_0 \dots x_n) = output(s) \cdot output(k, s', x_1 \dots x_n)$$

$$\text{where } s' = suc(k, s, x_0).$$

**Definition 5.16 (Initial state).** Define the function  $init$  which takes as input a key  $k \in (\mathbb{F}_2^8)^8$  and outputs the initial cipher state  $s = \langle l, r, t, b \rangle$  where

$$t := 0xE012$$

$$l := (k_{[0]} \oplus 0x4C) \boxplus 0xEC$$

$$b := 0x4C$$

$$r := (k_{[0]} \oplus 0x4C) \boxplus 0x21$$

**Definition 5.17 (MAC function).** Define the function  $MAC: (\mathbb{F}_2^8)^8 \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^{32}$  as

$$MAC(k, m) = output(k, suc(k, init(k), m), 0^{32})$$

## 5.5 Weakness in iClass

This section describes weaknesses in the design and implementation of iClass. We present four weaknesses that are later exploited in Section 5.5.5 to mount an attack that recovers the systems master key.

### 5.5.1 Weak keys

The cipher has a clear weakness when the three rightmost bits of each key byte are the same. Let us elaborate on that.

**Proposition 5.2.** *Let  $\beta$  be a bitstring of length three. Then, for all keys  $k \in \mathbb{F}_2^{64}$  of the form  $k = \alpha_{[0]}\beta \dots \alpha_{[7]}\beta$  with  $\alpha_{[i]} \in \mathbb{F}_2^5$  the cipher outputs a constant  $C_\beta$ .*

This is due to the fact that only the three rightmost bits of register  $r$  define the output of the cipher and only the rightmost bit of  $r$  influences register  $b$ . But these, in turn, are only influenced by the three rightmost bits of the key bytes. This means that the 5 leftmost bits of  $r$  and the 5 leftmost bits of each key byte affect only the key byte selection, but for the key under consideration this does not affect the output. The same holds for  $c_C$  and  $n_R$  as they are just input to the  $select(\cdot)$  function. Figure 5.11 shows the corresponding MAC value for each possible value of  $\beta$ .

$\beta$	$C_\beta = \text{MAC}(k, c_C \cdot n_R)$
000	BF 5D 67 7F
001	10 ED 6F 11
010	53 35 42 0F
011	AB 47 4D A0
100	F6 CF 43 36
101	59 7F 4B 58
110	1A A7 66 46
111	E2 D5 69 E9

**Figure 5.11:** Corresponding MAC for each value of  $\beta$

The manufacturer seems to be aware of this feature of the cipher since the function  $hash0$ , used in key diversification, prevents such a key from being used. This weakness combined with the weakness described in Section 5.5.2 and 5.5.3 results in a vulnerability exploited in Section 5.5.5.

### 5.5.2 XOR key update weakness

In order to update a card key, the iClass reader does not send the new key to the card in the clear but instead it sends the XOR of the old and the new key (see Section 5.2.1). This simple mechanism prevents an adversary from eavesdropping the

new key during key update. Although, this key update mechanism introduces a new weakness, namely, it makes it possible for an adversary to make partial modifications to the existing key. A key update should be an atomic operation. Otherwise it allows an adversary to split the search space in a time-memory trade-off. Moreover, in case the cipher has some weak keys like the ones described in Section 5.5.1, it allows an adversary to force the usage of one of these keys.

### 5.5.3 Privilege escalation

Several privilege escalation attacks have been described in the literature [KSRW04, DDSW11]. The privilege escalation weakness in iClass concerns the management of access rights over an application within the card. After a successful authentication for application 1 has been executed, the reader is granted read and write access to this application. Then, it is possible to execute a `read` command for a block within application 2 without losing the previously acquired access rights. More precisely, a `read` command on block  $n$  within application 2, with  $n \neq c_C$ , returns a sequence of 64 ones which indicates that permission is denied to read this block. Surprisingly, this read attempt on application 2 does not affect the previously acquired access rights on application 1. This `read` command though, has the side effect of loading the key  $k_2$  into the internal state of the cipher. In particular, from this moment on the card accepts `write` commands on application 1 that have a valid MAC computed using key  $k_2$ .

### 5.5.4 Lower card key entropy

After careful inspection of the function *hash0* (Section 5.3.3) it becomes clear that this function attempts to fix the weak key weakness presented in this section. The function *hash0* makes sure that, when looking at the last bit of each key byte, exactly four of them are zeros (and the other four of them are ones). Due to this restriction there are only  $\frac{8!}{(4!)^2} = 70$  possibilities for the last bits of each key byte, instead of  $2^8 = 256$ , reducing the entropy of the key by 1.87 bits. This constitutes the biggest part of the 2.23 bits entropy loss (Section 5.3.3) that is caused by *hash0*.

### 5.5.5 Key recovery attack on iClass Standard

This section shows how the weaknesses described in Section 5.5 can be exploited. Concretely, we propose an attack that allows an adversary to recover a card key by wirelessly communicating with a card and a reader. Once the card key has been recovered, the weak key diversification weakness described in Section 5.3.3 can be exploited in order to recover the master key. Next, we describe the attack in detail.

In order to recover a target card key  $k_1$  from application 1, an adversary  $A$  proceeds as follows. First,  $A$  eavesdrops a legitimate authentication trace on the e-purse with key  $k_1$ , while making sure that the e-purse is not updated. If the reader attempts



to update the e-purse, this can be prevented by playing as man-in-the-middle or by simply jamming the e-purse update message. Next, the adversary replays this authentication trace to the card. At this point the adversary gains read and write access to application 1. Although, in order to actually be able to write, the adversary still needs to send a valid MAC with  $k1$  of the payload. To circumvent this problem, the adversary proceeds as described in Section 5.5.3, exploiting the privilege escalation weakness. At this point the adversary still has read and write access to application 1 but he is now able to issue `write` commands using MACs generated with the known key  $k2$  to write on application 1. In particular,  $A$  is now able to modify  $k1$  at will. Exploiting the XOR key update weakness described in Section 5.5.2, the adversary modifies the card key  $k1$  into a weak key by setting the three rightmost bits of each key byte the same. Concretely, the adversary runs  $2^{3 \times 7} = 2^{21}$  key updates on the card with  $\Delta = 0^5 \delta_{[0]} \dots 0^5 \delta_{[6]} 0^8 \in \mathbb{F}_2^{64}$  and  $\delta_{[i]} = abc \in \mathbb{F}_2^3$  for all possible bits  $a, b$  and  $c$ . One of these key updates will produce a weak key, i.e., a key of the form  $k = \alpha_{[0]}\beta \dots \alpha_{[7]}\beta$  with  $\alpha_{[i]} \in \mathbb{F}_2^5$ . Exploiting the weak key weakness described in Section 5.5.1, after each key update  $A$  runs 8 authentication attempts, one for each possible value of  $\beta$ , using the MAC values shown in Figure 5.11. Note that a failed authentication will not affect the previously acquired access rights. As soon as an authentication attempt succeeds, the card responds with a MAC value that univocally determines  $\beta$  as stated in Proposition 5.2. Knowing  $\beta$ , the adversary is able to recover the three rightmost bits of  $k1_{[i]}$  by computing  $\beta \oplus \delta_{[i]}$  for  $i = 0 \dots 6$ . Furthermore, the three rightmost bits of  $k_{[7]}$  are equal to  $\beta \oplus 000 = \beta$ . In this way, the adversary recovers  $3 \times 8 = 24$  bits of  $k1$  and only has to search the remaining 40 bits of the key, using the legitimate trace eavesdropped in the beginning for verification.

This attack can be further optimized. The restriction on the last bit of each byte imposed by `hash0`, described at the end of Section 5.5.4, reduces the number of required key updates from  $2^{21}$  to almost  $2^{19}$ . Therefore, it reduces the total number of authentication attempts to  $2^{19} \times 8 = 2^{22}$ . Once the adversary has recovered the card key  $k1$ , as we already mention in Section 5.5.4, recovering the master key is just as hard as breaking single DES.

## 5.6 iClass Elite

This section describes in detail the built-in key diversification algorithm of iClass Elite. Besides the obvious purpose of deriving a card key from a master key, this algorithm intends to circumvent weaknesses in the cipher by preventing the usage of certain ‘weak’ keys. In this way, it is patching a weakness in the iClass cipher. After the description of the iClass Elite key diversification in Section 5.6.1 we describe the weaknesses of this scheme in Section 5.6.2. Finally, the third and fastest attack of this chapter, concerning iClass Elite, is given in Section 5.6.3.

First, recall the key diversification of the iClass Standard system that we described in Section 5.3.2. In this scheme, the iClass reader first encrypts the card

identity  $id$  with the master key  $\mathcal{K}$ , using single DES. The resulting ciphertext is then input to a function called  $hash0$  which outputs the diversified key  $k$ , i.e.,

$$k = hash0(DES_{enc}(\mathcal{K}, id)).$$

Here the DES encryption of  $id$  with master key  $\mathcal{K}$  outputs a cryptogram  $c$  of 64 bits. These 64 bits are divided as  $c = \langle x, y, z_{[0]}, \dots, z_{[7]} \rangle \in \mathbb{F}_2^8 \times \mathbb{F}_2^8 \times (\mathbb{F}_2^6)^8$  which is used as input to the  $hash0$  function. This function introduces some obfuscation by performing a number of permutations, complement and modulo operations. Besides that, it checks for and removes patterns like similar key bytes, which could produce a strong bias on the cipher. Finally, the output of  $hash0$  is the diversified card key  $k = k_{[0]}, \dots, k_{[7]} \in (\mathbb{F}_2^8)^8$ .

**Remark 5.3.** *The DES implementation used in iClass is non-compliant with the NIST standard [FIP99]. Concretely, iClass deviates from the standard in the way of representing keys. According to the standard a DES key is of the form  $\langle k_0 \dots k_{55} p_0, \dots, k_{47} \dots k_{55} p_7 \rangle$  where  $k_0 \dots k_{55}$  are the actual key bits and  $p_0 \dots p_7$  are parity bits. Instead, in iClass, a DES key is of the form  $\langle k_0 \dots k_{55} p_0 \dots p_7 \rangle$ .*

### 5.6.1 Key diversification on iClass Elite

The iClass Elite system is sold as a more secure and advanced solution than the iClass Standard variant. HID introduces iClass Elite (a.k.a. High Security) as the solution for “those who want a boost in security” [Cum03]. iClass Elite aims to solve the obvious limitations of having just one single world-wide master key for all iClass systems. Instead, iClass Elite allows customers to have a personalized master key for their own system. To this purpose, HID has modified the key diversification algorithm, described in Section 5.3.2 by adding an additional layer to it. This modification only affects the way in which readers compute the corresponding card key but does not change anything on the cards themselves. This section describes this key diversification algorithm in detail. Then, Section 5.6.2 describes two weaknesses that are later exploited in Section 5.6.3.

We first need to introduce a number of auxiliary functions and then we explain this algorithm in detail.

**Definition 5.18** (Auxiliary functions). *Let us define the following auxiliary functions. The bit-rotate left function*

$$rl: \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8 \text{ as } rl(x_0 \dots x_7) = x_1 \dots x_7 x_0.$$

*The bit-rotate right function*

$$rr: \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8 \text{ as } rr(x_0 \dots x_7) = x_7 x_0 \dots x_6.$$

*The nibble-swap function swap*

$$swap: \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8 \text{ as } swap(x_0 \dots x_7) = x_4 \dots x_7 x_0 \dots x_3.$$

**Definition 5.19.** Let the function  $hash1: (\mathbb{F}_2^8)^8 \rightarrow (\mathbb{F}_2^8)^8$  be defined as

$$hash1(id_{[0]} \dots id_{[7]}) = k_{[0]} \dots k_{[7]}$$

where

$$\begin{aligned} k_{[i]} &= k'_{[i]} \bmod 128, & i &= 0 \dots 7 \\ k'_{[0]} &= id_{[0]} \oplus \dots \oplus id_{[7]} & k'_{[4]} &= \overline{rr(id_{[4]} \boxplus k'_{[2]})} + 1 \\ k'_{[1]} &= id_{[0]} \boxplus \dots \boxplus id_{[7]} & k'_{[5]} &= \overline{rl(id_{[5]} \boxplus k'_{[3]})} + 1 \\ k'_{[2]} &= rr(\text{swap}(id_{[2]} \boxplus k'_{[1]})) & k'_{[6]} &= rr(id_{[6]} \boxplus (k'_{[4]} \oplus 0x3C)) \\ k'_{[3]} &= rl(\text{swap}(id_{[3]} \boxplus k'_{[0]})) & k'_{[7]} &= rl(id_{[7]} \boxplus (k'_{[5]} \oplus 0xC3)) \end{aligned}$$

**Definition 5.20.** Define the rotate key function  $rk: (\mathbb{F}_2^8)^8 \times \mathbb{N} \rightarrow (\mathbb{F}_2^8)^8$  as

$$\begin{aligned} rk(x_{[0]} \dots x_{[7]}, 0) &= x_{[0]} \dots x_{[7]} \\ rk(x_{[0]} \dots x_{[7]}, n + 1) &= rk(rl(x_{[0]}) \dots rl(x_{[7]}), n) \end{aligned}$$

**Definition 5.21.** Let the function  $hash2: (\mathbb{F}_2^8)^8 \rightarrow (\mathbb{F}_2^{64})^{16}$  be defined as

$$hash2(\mathcal{K}^{cus}) = y_{[0]}z_{[0]} \dots y_{[7]}z_{[7]}$$

where

$$\begin{aligned} z_{[0]} &= \text{DES}_{enc}(\mathcal{K}^{cus}, \overline{\mathcal{K}^{cus}}); & z_{[i]} &= \text{DES}_{dec}(rk(\mathcal{K}^{cus}, i), z_{[i-1]}) & i &= 1 \dots 7 \\ y_{[0]} &= \text{DES}_{dec}(z_{[0]}, \overline{\mathcal{K}^{cus}}); & y_{[i]} &= \text{DES}_{enc}(rk(\mathcal{K}^{cus}, i), y_{[i-1]}) & i &= 1 \dots 7 \end{aligned}$$

Next we introduce the Selected key. This key is used as input to the standard iClass key diversification algorithm. It is computed by taking a selection of bytes from  $hash2(\mathcal{K}^{cus})$ . This selection is determined by each byte of  $hash1(id)$  seen as a byte offset within the bitstring  $hash2(\mathcal{K}^{cus})$ .

**Definition 5.22.** Let  $h \in (\mathbb{F}_2^8)^{128}$ . Let  $k^{sel} \in (\mathbb{F}_2^8)^8$  be the Selected key defined as

$$h := hash2(\mathcal{K}^{cus}); \quad k_{[i]}^{sel} := h_{[hash1(id)_{[i]}]} \quad i = 0 \dots 7$$

The last step to compute the diversified card key is just like in iClass (see Section 5.3.2).

$$k := hash0(\text{DES}_{enc}(k^{sel}, id))$$

## 5.6.2 Weaknesses in iClass Elite key diversification

This section describes two weaknesses in the key diversification algorithm of iClass Elite. These weaknesses are exploited in Section 5.6.3 to mount an attack against iClass Elite that recovers the custom master key.

### Redundant key diversification on iClass Elite

Assume that an adversary somehow learns the first 16 bytes of  $hash2(\mathcal{K}^{cus})$ , i.e.,  $y_{[0]}$  and  $z_{[0]}$ . Then he can simply recover the master custom key  $\mathcal{K}^{cus}$  by computing

$$\mathcal{K}^{cus} = \overline{DES_{enc}(z_{[0]}, y_{[0]})}.$$

Furthermore, the adversary is able to verify that he has the correct  $\mathcal{K}^{cus}$  by checking the following equality

$$z_{[0]} = DES_{enc}(\mathcal{K}^{cus}, \overline{\mathcal{K}^{cus}}).$$

### Weak key-byte selection on iClass Elite

Yet another weakness within the key diversification algorithm of iClass Elite has to do with the way in which bytes from  $hash2(\mathcal{K}^{cus})$  are selected in order to construct the key  $k^{sel}$ . As described in Section 5.6.1, the selection of key bytes from  $hash2(\mathcal{K}^{cus})$  is determined by  $hash1(id)$ . This means that only the card's identity decides which bytes of  $hash2(\mathcal{K}^{cus})$  are used for  $k^{sel}$ . This constitutes a serious weakness since no secret is used in the selection of key bytes at all. Especially considering that, for some card identities, the same bytes of  $hash2(\mathcal{K}^{cus})$  are chosen multiple times by  $hash1(id)$ . In particular, this implies that some card keys have significantly lower entropy than others. What is even more worrying, an adversary can compute by himself which card identities have this feature.

## 5.6.3 Key recovery attack on iClass Elite

In order to recover a master key  $\mathcal{K}^{cus}$ , an adversary proceeds as follows. First, exploiting the weakness described in Section 5.6.2, the adversary builds a list of chosen card identities like the ones shown in Listing 5.2.

Listing 5.2: Chosen card identities

Card identity $id$	$hash1(id)$	Recovery
00 0B 0F FF F7 FF 12 e0	<b>01 01 00 00 45 01 45 45</b>	Byte 00 and 01 in $2^{24}$
00 04 0E 08 F7 FF 12 e0	<b>78 02</b> 00 00 45 01 45 45	Byte 02 in $2^{16}$
00 09 0D 05 F7 FF 12 e0	<b>7B 03</b> 00 00 45 01 45 45	Byte 03 in $2^{16}$
00 0A 0C 06 F7 FF 12 e0	<b>7A 04</b> 00 00 45 01 45 45	Byte 04 in $2^{16}$
00 0F 0B 03 F7 FF 12 e0	<b>7D 05</b> 00 00 45 01 45 45	Byte 05 in $2^{16}$
00 08 0A 0C F7 FF 12 e0	<b>74 06</b> 00 00 45 01 45 45	Byte 06 in $2^{16}$
00 0D 09 09 F7 FF 12 e0	<b>77 07</b> 00 00 45 01 45 45	Byte 07 in $2^{16}$
00 0E 08 0A F7 FF 12 e0	<b>76 08</b> 00 00 45 01 45 45	Byte 08 in $2^{16}$
00 03 07 17 F7 FF 12 e0	<b>69 09</b> 00 00 45 01 45 45	Byte 09 in $2^{16}$
00 3C 06 E0 F7 FF 12 e0	<b>20 0A</b> 00 00 45 01 45 45	Byte 0A in $2^{16}$
00 01 05 1D F7 FF 12 e0	<b>63 0B</b> 00 00 45 01 45 45	Byte 0B in $2^{16}$
00 02 04 1E F7 FF 12 e0	<b>62 0C</b> 00 00 45 01 45 45	Byte 0C in $2^{16}$
00 07 03 1B F7 FF 12 e0	<b>65 0D</b> 00 00 45 01 45 45	Byte 0D in $2^{16}$
00 00 02 24 F7 FF 12 e0	<b>5C 0E</b> 00 00 45 01 45 45	Byte 0E in $2^{16}$
00 05 01 21 F7 FF 12 e0	<b>5F 0F</b> 00 00 45 01 45 45	Byte 0F in $2^{16}$

This listing contains a list of 15 card identities and their corresponding key-byte selection indices  $hash1(id)$ . The selection of card identities in this list is malicious. They are chosen such that the resulting key  $k^{sel}$  has very low entropy (in fact, it is possible to find several lists with similar characteristics).

For the first card identity in the list, the resulting key  $k^{sel}$  is built out of only three different bytes from  $hash2(\mathcal{K}^{cus})$ , namely  $0x00$ ,  $0x01$  and  $0x45$ . Therefore, this key has as little as 24 bits of entropy (instead of 56). Next, the adversary will initiate an authentication protocol run with a legitimate reader, pretending to be a card with identity  $id = 0x000B0FFFF7FF12E0$  as shown in the list. Following the authentication protocol, the reader will return a message containing a nonce  $n_R$  and a MAC using  $k$ . The adversary will repeat this procedure for each card identity in the list, storing a tuple  $\langle id, n_C, n_R, MAC \rangle$  for each entry. Afterwards, off-line, the adversary tries all  $2^{24}$  possibilities for bytes  $0x00$ ,  $0x01$  and  $0x45$  for the first key identity. For each try, he computes the resulting  $k$  and recomputes the authentication run until he finds a MAC equal to the one he got from the reader. Then he has recovered bytes  $0x00$ ,  $0x01$  and  $0x45$  from  $hash2(\mathcal{K}^{cus})$ .

The adversary proceeds similarly for the remaining card identities from the list. Although, this time he already knows bytes  $0x00$ ,  $0x01$  and  $0x45$  and therefore only two bytes per identity need to be explored. This lowers the complexity to  $2^{16}$  for each of the remaining entries in the list. The bytes that need to be explored at each step are highlighted with boldface in the list. At this point the adversary has recovered the first 16 bytes of  $hash2(\mathcal{K}^{cus})$ . Finally, exploiting the weakness described in Section 5.6.2, the adversary is able to recover the custom master key  $\mathcal{K}^{cus}$  with a total computational complexity of  $2^{25}$  DES encryptions.

## 5.7 Conclusion

In this chapter we have shown that the security of several building blocks of iClass is unsatisfactory. Again, obscurity does not provide extra security and there is always a risk that it can be circumvented. In fact, experience shows that instead of adding extra security it often covers up negligent designs.

It is hard to imagine why HID decided, back in 2002, to use single DES for key diversification considering that DES was already broken in practice in 1997 [LG98]. Especially when most (if not all) HID readers are capable of computing 3DES. Another unfortunate choice was to design their proprietary *hash0* function instead of using an openly designed and community reviewed hash function like SHA-1. From a cryptographic perspective, their proprietary function *hash0* fails to achieve any desirable security goal.

Furthermore, we have found many vulnerabilities in the cryptography and implementation of iClass that result in two key recovery attacks. Our first attack requires one eavesdropped authentication trace with a genuine reader (which takes about 10ms). Next, the adversary needs  $2^{22}$  authentication attempts with a card, which in practice takes approximately six hours. To conclude the attack, the adversary needs only  $2^{40}$  off-line MAC computations to recover the card key. The whole attack can be executed within a day. For the attack against iClass Elite, an adversary only needs 15 authentication attempts with a genuine reader to recover the custom master key. The computational complexity of this attack is negligible, i.e.,  $2^{25}$  DES encryptions. This attack can be executed from beginning to end in less than five seconds. We have successfully executed both attacks in practice and verified the claimed attack times.

This chapter reinforces the point that has been made many times: security by obscurity often covers up negligent designs. The built-in key diversification and especially the function *hash0* is advertised as a security feature but in fact it is a patch to circumvent weaknesses in the cipher. The cipher is a basic building block for any secure protocol. Experience shows that once a weakness in a cipher has been found, it is extremely difficult to patch it in a satisfactory manner. Using a well known and community reviewed cipher is a better alternative. The technique described in [RSH<sup>+</sup>12] could be considered as a palliating countermeasure for our first attack. More is not always better: the key diversification algorithm of iClass Elite requires fifteen DES operations more than iClass Standard while it achieves inferior security. Instead, it would have been more secure and efficient to use 3DES than computing 16 single DES operations in an ad hoc manner. Furthermore, NIST have proposed a statistical test suite [RSN<sup>+</sup>01] that can be used to measure the cryptographic strength of a cipher. Although this might identify weaknesses in a cipher, still many weaknesses arise from mistakes in the implementation. In order to find these problems, it is good practice to incorporate some form of formal verification in the development and implementation of security products, see for instance [FL12]. Also, systematic and automated model checking techniques proposed in [Tre08] can help to detect

and avoid implementation weaknesses like the privilege escalation in iClass. Alternatively, formalizing the whole design in a theorem prover [Bla01, JWS11] may reveal additional weaknesses. It remains an open question whether the unusual data structures and functions that we recovered in this chapter can be recovered using automated techniques, like for example with Howard [SSB11]. Automated techniques might speed up and assist in the reverse engineering of algorithms and data structures from software binaries. In line with the principles of responsible disclosure, we have notified the manufacturer HID Global and informed them of our findings back in November 2011. HID has established a product security reporting center<sup>7</sup> to encourage and improve this type of communication.

---

<sup>7</sup><http://www.hidglobal.com/main/product-security-reporting-center/>





## Chapter 6

---

# A desynchronization resistant privacy preserving RFID protocol

*“If you think technology can solve your security problems,  
then you don’t understand the problems and you don’t  
understand the technology.”*

Bruce Schneier

Over the last few years, the use of Radio Frequency Identification (RFID) technology has expanded enormously. It is currently deployed in electronic passports, labels for consumer goods, public transport ticketing systems, race timing, and countless other applications. In the preceding chapters we have seen that the security mechanisms of several large RFID systems are not satisfactory. Chapter 4 discussed the most used contactless smart card, the Mifare Classic, which suffered from several security vulnerabilities. In Chapter 5 we have seen that bad use of cryptographic primitives in iClass leads to systems that can easily be compromised. The primary security goals in these chapters were confidentiality, integrity and authenticity. In this last chapter we focus on privacy as a security goal. In the context of RFID systems, privacy is achieved when an adversary is not able to identify, trace, or link tags within the system [Vau07]. The RFID tags that we focus on in this chapter have little computing power. The reason for this is that they are used in product labels. The additional value of the tag should be as low as possible since, for example, the customer does not want to pay US\$ 0.50 more on a product that normally costs US\$ 1.00. Accordingly, this cost restriction makes it hard to design a privacy-friendly protocol for these tags since no public-key cryptography can be used. The difficulty of this problem can be seen in the numerous articles that cover this specific subject [OSK03, JW05, Tsu06, PLHCETR06a, Vau07, BdMM08, BBEG09, ACCP10, FHV10, HCPLPT10, AD11, HSH11, ACM12a, ACM12b].

RFID technology has recently become popular as a replacement for traditional barcodes in the consumer supply chain and is increasingly incorporated into product labels. It is used to automatically identify products and to track products along the supply chain in industry [Att07], the medical sector [WCO<sup>+</sup>07], libraries [MW04b] and many other situations where barcodes are already employed. Even though RFID tags have indeed advantages over barcodes, they also have some drawbacks. RFID tags can be read faster than barcodes and have less restrictions on their physical positioning. These advantages do not necessarily imply that barcodes will get completely replaced by RFID tags. It is still useful to have some backup identification

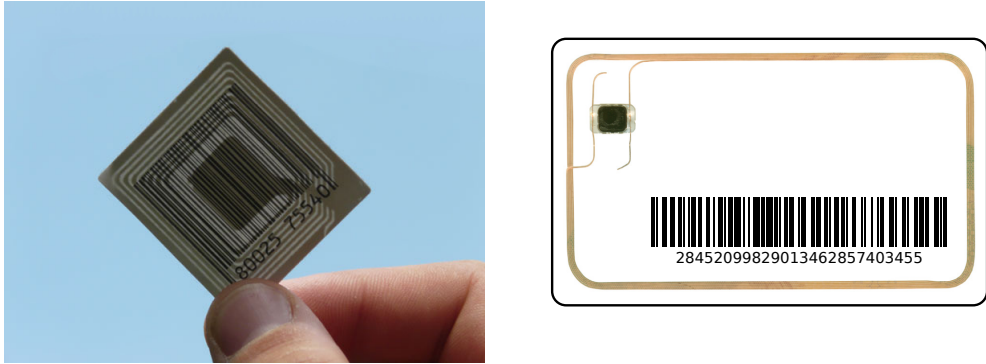


Figure 6.1: Two combined barcode-and-RFID labels

option. For instance when an RFID tag breaks down, it is still possible to switch to barcode identification. RFID and barcode labels, so-called dual labels, do exist as shown in Figure 6.1. Barcodes are often printed directly on the product (wrapping) and therefore are currently cheaper (below US\$ 0.01) than RFID tags. Furthermore, barcodes are deeply entrenched in many systems and complete replacement is not going to happen in the near future [WNLY06]. Actually, barcodes and RFID systems will be used in parallel for many more years to come.

One of the problems with inexpensive RFID tags is that their widespread use introduces several security threats including privacy concerns [JRS03, PLHCETR06b]. Most RFID tags send a unique identifier to every reader that attempts to communicate with them. This behavior can be used by an adversary to construct an “RFID profile” of an individual, i.e., the collection of unique tag identifiers that a specific individual usually carries. This profile could then be used to track this person, or to infer the behavior such as spending or traveling patterns, jeopardizing this person’s privacy.

At the same time, the supply chain requires RFID tags for product identification that are as inexpensive as possible. This is important since the cost of an RFID tag directly adds up to the product price which should remain competitive. The so-called Electronic Product Code (EPC) tags are of this most simple form. If we focus on these inexpensive EPC-like tags, think of the tags that are attached to clothes in a shop, we observe that RFID tags are often used in parallel with barcodes, instead of replacing them. The combination of barcode and RFID tag can be found on several products nowadays (see Fig. 6.1). In this chapter we exploit this duality by combining the barcode and RFID tag into one label in order to get the best of each technology. On the one hand, we get flexible reading and unique identification, on the other hand we retain the infeasibility for an adversary to track goods at will. We present a practical solution where both the RFID tag and the barcode are combined in order to provide privacy while the problem of tag desynchronization, which we will introduce in Section 6.3, is being addressed.

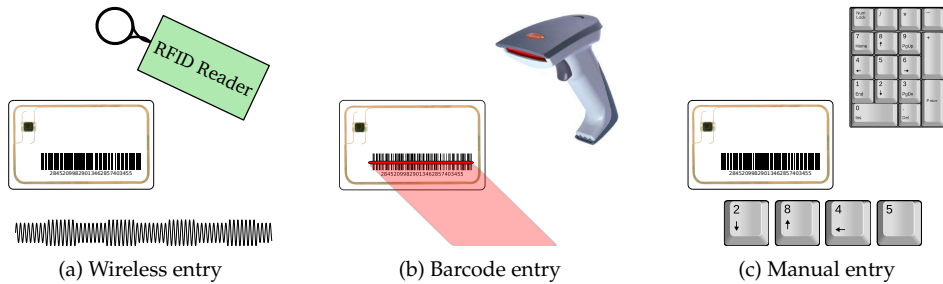


Figure 6.2: Different entry methods

## 6.1 RFID next to barcodes

Before we elaborate on the privacy issues that arise when using RFID tags in supply chains, let us first briefly visit three different identification techniques and discuss how they relate to each other in terms of throughput. The first and most simple method is to print an identification number (or product number) on the product. In order to identify this product, its number has to be manually entered in the system. Of course, in most systems this identification number is accompanied by a barcode. This barcode can be read by a barcode scanner that has to be positioned in a certain way to correctly read the identification number. This is already much of an improvement and gives a higher efficiency. Still, the identification number is also printed in numbers next to its barcode representation. This is a backup mechanism in case the barcode reader fails to recognize the barcode pattern. In this scenario it is possible to fall back on the slower method of entering the identification number manually. In this chapter we extend this idea of interoperability with a third identification method, namely the application of RFID. The use of RFID technology can be seen as the next level in fast identification in supply chain management. It is not limited to the line-of-sight restriction that barcodes have and allows to read the product identification number through all kinds of materials and from many different positions. In this chapter, we treat the RFID technology as another identification method on top of the existing barcode and not as a replacement of the barcode. First, the identification process tries to read the identification number from the RFID tag using wireless technology as depicted in Figure 6.2. This can be done in a fully automatic way. In case the identification using RFID fails, one switches back to reading the barcode using a reader as depicted in Figure 6.2b. It might require some manual intervention in order to get the barcode positioned correctly in front of the reader. In the end, when this second identification method also fails, we fall back to manual entry of the identification number using a keypad as depicted in Figure 6.2c. In Figure 6.2, method (a) has a higher throughput than (b) and (b) has a higher throughput than (c).

It has often been questioned whether and when RFID will replace the barcode [WSRE04, She04, Jue06, Tsu06]. Since this is questionable, it is surprising that many solutions do not even consider the simultaneous use of RFID and barcodes [Yu07], and just assume that RFID will replace all barcodes. We consider in this chapter the coexistence of both technologies as a valuable and realistic option. It is hard to give a precise estimate of the price range that RFID tags should reach in order to be cost-effective [Wan06], but in general the estimate is around US\$ 0.05 [WSRE04, Wan06]. New developments in organic circuits [PBR<sup>+</sup>05] make it possible to print circuits directly on material like paper, plastic or even cloth. Opposed to inorganic silicon, organic circuits are carbon-based and can be printed (referred to as IC-printing) right on product wrappings. Although organic circuits have a lower performance compared to inorganic circuits, research has shown that it is possible to assemble RFID tags using this technique [SDVM<sup>+</sup>06]. In other words, IC-printing of RFID tags has become practically feasible. Using this method, RFID tags can be printed on flexible materials. Some studies point out that it might be possible to produce RFID tags for less than US\$ 0.02 [SFC<sup>+</sup>05, SSS<sup>+</sup>05] using IC-printing technology. This includes the circuit and the complete antenna. In 2010, a block cipher was proposed by Knudsen et al. in [KLPR10] that especially focuses on cryptography in low-cost printed circuits that could also be used in RFID applications. Knudsen et al. also point out that several companies already deliver printed RFID circuits [Pol12, Kov12]. The application of RFID in the supply chain is not only a matter of costs but also raises privacy concerns. Therefore, we study the option where the RFID tag and barcode are combined into one label. We propose a solution that benefits the efficiency of RFID technology and at the same time does prevent the tracing of these labels by an attacker. This way we preserve the privacy of the individual that holds the product label.

## 6.2 Forward privacy

Many privacy notions have been discussed in the literature. We use the notion of forward privacy that is defined in the privacy model of Vaudenay [Vau07]. A formal definition follows in Section 6.5. Forward privacy requires that an adversary, who has control over the communication channel, should not be able to tell whether two protocol instances involve the same tag or not. Moreover, even when all secret information in the tag is revealed to the adversary, this should not enable him to link this tag with previously recorded protocol runs. In order to achieve such a strong security notion, it is necessary that the tag updates its state (using a one-way function) with every authentication attempt. The back office should also be aware of this tag updating in order to ‘follow’ the tag state. This continuous updating might lead to desynchronization between the back office and the tag. This desynchronization can be both induced by an adversary or simply due to poor physical circumstances like a too big distance between tag and reader.

A large number of protocols have been proposed in the literature that aim to achieve privacy [JW05, Tsu06, BdMM08]. Furthermore, there are proposals that aim more concretely at forward-privacy [OSK03, Vau07, BBEG09]. Unfortunately, many of these proposals turn out to be either impractical due to the resource-constrained nature of RFID or suffer from desynchronization. Achieving forward privacy without the use of public-key cryptography has shown to be a very challenging task. In fact, Vaudenay [Vau07] showed that having a forward-private stateless RFID scheme implies key agreement, which is believed to require the use of public-key cryptography. Achieving forward-privacy with symmetric cryptography requires heavy workload on the reader side and these protocols often suffer from desynchronization. A distinguished example is due to Avoine [AO05], who proposed a scheme based on the Ohkubo-Suzuki-Kinoshita (OSK) protocol [OSK03] that achieves forward-privacy. Unfortunately this protocol suffers from desynchronization which has impact on availability. The scheme of Dimitriou [Dim05] is reminiscent of the Hash-Locking scheme of Weis [WSRE04] but it also suffers from desynchronization. For a complete survey of related work we refer the reader to [Jue06] and [ACM12a].

In this chapter, we propose a forward-private RFID authentication protocol that incorporates a mechanism for re-synchronization. We exploit the coexistence of RFID and barcodes in the protocol design in order to achieve an efficient search procedure on the reader side. The main idea of the protocol resembles that of OSK, except that we allow a limited and small number of failed authentication attempts. This dramatically reduces the search space on the reader side. Should this limit be exceeded, then the barcode allows us to re-synchronize the reader and the tag. This re-synchronization procedure is part of the authentication protocol so that it does not differentiate from normal protocol runs. This way, the adversary is not able to distinguish the different protocol runs. We propose a model for RFID privacy using provable security techniques, following the lines of [Avo05, Vau07, JW09, GvR10]. Within this model we define correctness, forward-privacy and synchronization. Finally, we show that our protocol satisfies all these security notions using the random oracle methodology.

The remainder of this chapter is organized as follows. First, Section 6.3 explains the desynchronization problem. Then, Section 6.4 describes the system and adversarial models. Section 6.5 provides definitions for security, (forward-)privacy, (strong-)correctness and desynchronization. Section 6.6 describes our protocol and Section 6.7 substantiates the security claims. Finally, Section 6.8 concludes this chapter and discusses the results.

## 6.3 The desynchronization problem

Our goal is a practical RFID protocol that provides forward privacy. The meaning of the adjective “practical” heavily depends on the resources and restrictions that are given. A good first attempt is the protocol shown in Figure 6.3 where a tag  $T$  sends the hash of its identity  $id$  concatenated with some random value  $r$  and  $r$  itself

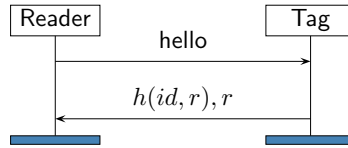


Figure 6.3: A first attempt

to a reader  $R$ . Assuming a perfect hash function and random number generator it is impossible for an eavesdropper to retrieve the identity  $id$ . This small protocol is more or less what was proposed as the Randomized Hash-Locking scheme by Weis et al. in [WSRE04]. Note that this protocol does not provide forward privacy. Once the identity  $id$  is known to an eavesdropper it is possible to link communications in the past by simply recalculating the hash of this  $id$  combined with the plaintext random value  $r$  and check the outcome. Furthermore, the reader is connected to a back-end where a database is maintained with all tag identities. Apart from its vulnerability to a replay attack, a big drawback of this solution is its search procedure. To look up a tag, every identity in the database needs to be hashed in combination with the random  $r$  and then needs to be compared with the value  $h(id, r)$  that was sent by the tag. This drastically reduces the applicability of this solution to only small systems with only a limited number of tags.

Another well-known RFID protocol from the literature is the OSK protocol proposed by Ohkubo et al. [OSK03]. In the OSK protocol, the tag secret is updated in every protocol run regardless whether it was a successful run or not. This is done by means of a hash chain where  $h^i(s)$  means that the tag secret  $s$  is successively hashed  $i$  times. The eventual value that is sent to the reader is hashed another time using a different hash function  $g$ . This ensures that when an adversary eavesdrops

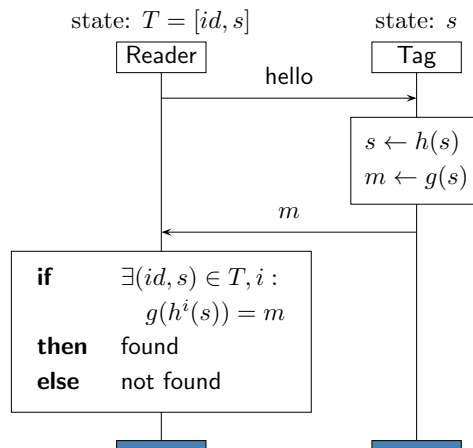


Figure 6.4: The OSK protocol

two protocol runs with the same tag it is still not possible to tell from the messages  $m$  whether they were produced by the same tag. The one-wayness of  $g$  prevents disclosure of the underlying chain of tag secrets. The OSK protocol makes forward privacy possible [Vau07] since once an eavesdropper learns the current tag state  $s$  it is impossible for this eavesdropper to get back to earlier tag states that were used in earlier communications. If an eavesdropper could derive earlier tag states this would mean that one of the primitives, the hash function  $h$ , is broken. The protocol is depicted in Figure 6.4. In [CC08] it is shown that the OSK scheme is also synchronizable. The example of the OSK protocol illustrates the fact that a synchronizable protocol is not automatically efficient in its search procedure. The reader state  $T = [id, s]$  is a list of identities  $id$  and their associated tag secret  $s$ . In theory a reader always finds the tag identity when it is generated from a known tag secret  $s$ . In practice, however, it takes too much time to search for tag secrets that are updated many times. Ohkubo et al. suggest to store the latest value  $h^i(s)$  that was observed in order to optimize the search procedure on the reader side. Another issue is that a Denial-of-Service (DOS) attack might be induced on the reader side by simply sending a random value to the reader. In this case the search procedure of the reader will never end. Assume a normal protocol run with a genuine tag. The reader will start searching for a solution that satisfies  $\exists(id, s) \in T : g(h^i(s)) = m$ . The problem here is that somehow the number of iterations  $i$  should be bounded in order to prevent an endless search. When we adapt the OSK protocol such that the number of hash iterations is bounded by some value  $N$ , an adversary can mount a desynchronization attack by issuing  $N + 1$  authentication attempts to a tag. This will result in the tag updating its internal secret to  $h^{N+1}(s)$  which is out of scope of the reader. The reader will not be able to find the identity of the genuine tag. In this case we say that the reader and the tag are desynchronized.

### 6.3.1 Barcode analogy

The protocol that we propose in this chapter can be best explained in analogy to the traditional and very successful barcode. A well known daily example of barcodes can be found in a shop. The cashier scans the barcodes (Fig. 6.2b) of products that the customer wants to buy. From time to time the scanner might not be able to read a barcode. In such cases the cashier enters the serial number by hand using a keypad (Fig. 6.2c). This backup procedure costs more time and effort, but in the end, the checkout procedure is far more efficient than it would be when every product was entered manually at default.

The number of times that the cashier has to fall back to the manual input procedure is very low, otherwise the use of barcodes would become questionable. Actually, we face the same problem in privacy friendly RFID. Here, the tag and reader need to stay synchronized in some way. To the best of our knowledge, all attempts to design a protocol that keeps up with these discrepancies try to achieve this without any human intervention. Many proposals try to prevent desynchronization purely

by means of the wireless link. This becomes a very hard task when, at the same time, an adversary is allowed to exhaustively query a tag. In practice desynchronization is a problem that should be handled, merely because it may also occur due to physical problems in the reading process. Now, recall the same shop as mentioned before but let the products be equipped with RFID tags. When a tag is no longer synchronized with a genuine reader and the system fails to identify a tag (Fig. 6.2a), we fall back to the use of a *second channel* which provides the reader with the needed identity. This identity can then be read from a barcode or serial number which is physically printed on the RFID tag (Fig. 6.2b).

**Remark 6.1.** *When multiple tags are scanned at the same time, which is a useful feature in a supply chain, it might be troublesome to single out a tag that causes problems in the identification procedure. We would like to point out that it makes no difference whether this problem occurs with a tag that has a fixed identifier or with a tag that follows the protocol defined in this chapter. Both cases result in the same amount of overhead.*

A protocol run in which a *second channel* is used to synchronize the tag and reader state is called a *synchronization run*. Since a *synchronization run* involves additional actions apart from running the protocol it can be treated as a special instance of the protocol. In general, these special instances occur scarcely in practical settings. We will now further elaborate on a system like sketched above.

## 6.4 System model

Consider a scheme where readers have a secure communication channel with the back office. We assume that readers are single threaded, i.e., can only have one active protocol instance with a tag at a time. After running a protocol with a tag, the reader has an output that is typically the identity of the tag. New readers and tags can be added to the system at will. The formal definition follows.

**Definition 6.1** (RFID scheme). *An RFID scheme  $\Pi$  consists of:*

- *a probabilistic polynomial-time algorithm SetupSystem that takes as input the security parameter  $1^n$  and outputs the public key pair  $(sk, pk)$  of the system.*
- *a probabilistic polynomial-time algorithm SetupReader that takes as input the secret key of the system  $sk$  and outputs the initial state of the reader  $s$  and the reader's secret  $k$ .*
- *a probabilistic polynomial-time algorithm SetupTag that takes as input the secret key of the system  $sk$  and outputs the initial state of the tag  $s$  and the tag's secret  $k$ .*
- *a polynomial-time interactive protocol between a reader and a tag, where the reader returns Output. Output is typically the identity of the tag.*



An adversary is a probabilistic polynomial-time algorithm that interacts with the system by means of different oracles. The environment keeps track of the state of each element in the system and answers the oracle queries according to the protocol. Besides adding new tags and readers to the system and being able to communicate with them, an adversary can also corrupt tags. This models techniques like differential power analysis and chip slicing. By corrupting a tag, an adversary retrieves its internal state.

**Definition 6.2** (Adversary). *An adversary is a probabilistic polynomial-time algorithm that takes as input the system public key  $pk$  and has access to the following oracles:*

- $\text{CreateReader}(\mathcal{R})$  creates a new reader by calling  $\text{SetupReader}(sk)$  and updates the state of the back-office. This new reader is referenced as  $\mathcal{R}$ .
- $\text{CreateTag}(\mathcal{T})$  creates a new tag  $\mathcal{T}$  by calling  $\text{SetupTag}(sk)$  and updates the state of the back-office. This new tag is referenced as  $\mathcal{T}$ .
- $\text{CorruptTag}(\mathcal{T})$  returns the internal state  $s$  of the tag  $\mathcal{T}$ .
- $\text{Launch}(\mathcal{R})$  attempts to initiate a new protocol instance at reader  $\mathcal{R}$ . If  $\mathcal{R}$  has already an active protocol instance then  $\text{Launch}$  fails and returns zero. Otherwise it starts a new protocol instance and returns one.
- $\text{Send}(m, A)$  sends a message  $m$  to the entity  $A$  and returns its response  $m'$ . The entity  $A$  can either be a reader  $\mathcal{R}$  or a tag  $\mathcal{T}$ .
- $\text{Result}(\mathcal{R})$  outputs whether or not the output of the last finished protocol instance at reader  $\mathcal{R}$  is not  $\perp$ , i.e.,  $\text{Output} \neq \perp$ .

**Definition 6.3.** We denote by  $\mathcal{O}$  the set of oracles  $\{\text{CreateReader}, \text{CreateTag}, \text{CorruptTag}, \text{Launch}, \text{Send}, \text{Result}\}$ .

## 6.5 Security definitions

This section elaborates on the security and privacy definitions from the literature; much of it is standard. The main goal of an RFID system is security, which means that readers are able to authenticate legitimate tags. Throughout this chapter we focus on privacy. For the sake of self containment, we include here the following security definition which is an adapted version of the security definition proposed in [Vau07].

**Definition 6.4** (Security). *An RFID scheme is secure if for all adversaries  $\mathcal{A}$  and for all readers  $\mathcal{R}$ , the probability that  $\mathcal{R}$  outputs the identity of a legitimate tag  $\mathcal{T}$  is a negligible function of  $\eta$  when the last finished protocol instance at reader  $\mathcal{R}$  and tag  $\mathcal{T}$  did not have any matching conversation. Matching conversation here means that  $\mathcal{R}$  and the tag (successfully) executed the authentication protocol.*

Next we define privacy composing the definitions of Juels and Weis [JW09] and Vaudenay [Vau07] since each of them has its advantages: the former is indistinguishability based, which makes it more practical; the latter has the drawback of being simulation based but is stronger and allows for a variety of adversaries with custom capabilities. Privacy is defined in an IND-CCA like fashion where the adversary tries to win the privacy game. In this game, the environment creates system parameters by calling  $\text{SetupSystem}$ . Then it gives the public key of the system  $pk$  to the adversary  $\mathcal{A}_0$ . This adversary has access to the set of oracles  $\mathcal{O}$ . Eventually,  $\mathcal{A}_0$  must output two uncorrupted challenge tags  $\mathcal{T}_0^*$  and  $\mathcal{T}_1^*$ . Then, the environment chooses a random bit  $b$  and gives the adversary  $\mathcal{A}_1$  access to  $\mathcal{T}_b^*$ . At this point, the original references to  $\mathcal{T}_0^*$  and  $\mathcal{T}_1^*$  are no longer valid. Again, the adversary has access to all oracles  $\mathcal{O}$ . Finally, the adversary outputs a guess bit  $b'$ . The adversary wins the game if  $b = b'$ . The formal definition follows.

**Definition 6.5** (Privacy game).

*Priv-Game* $_{\Pi, \mathcal{A}}(\eta)$  :

$(sk, pk) \leftarrow \text{SetupSystem}(1^\eta)$   
 $\mathcal{T}_0^*, \mathcal{T}_1^* \leftarrow \mathcal{A}_0^{\mathcal{O}}(pk)$   
 $b \leftarrow \{0, 1\}$   
 $b' \leftarrow \mathcal{A}_1^{\mathcal{O}}(\mathcal{T}_b^*)$   
*winif*  $b = b'$ .

The challenge tags  $\mathcal{T}_0^*$  and  $\mathcal{T}_1^*$  must be uncorrupted. This means that no  $\text{CorruptTag}(\mathcal{T}_{\{0,1\}}^*)$  query has been made. Adversaries implicitly pass state.

In general, it is hard to define a realistic adversarial model as different applications have different requirements. Following the lines of Vaudenay [Vau07], we consider different classes of adversaries depending on their capabilities. The model of Vaudenay defines the notions of forward, weak and narrow adversaries. In this chapter we would like to add the notion of a thin adversary in order to handle protocols that use a *second channel*. Intuitively, a *forward* adversary is an adversary that observes communication between tags and readers and later on acquires one of these tags and tries to link it with some of the past sessions, compromising its privacy. If the adversary succeeds to do so, with non-negligible probability, we say that this is a *winning* adversary. A *weak* adversary is an adversary that is unable to corrupt tags. In real life scenarios it is often realistic to assume that an adversary can see the outcome of an authentication attempt. For instance, this is the case of transport ticketing systems where an adversary could observe whether the gate of the metro opens or not, for a specific tag. An adversary that is unable to do so is called *narrow*. In line with the *narrow* adversary we introduce the *thin* adversary. A *thin* adversary cannot see additional information that is provided to the reader. Think for example of additional identifying information to make the search procedure more efficient.

**Definition 6.6** (Types of adversaries). A *forward adversary* is an adversary that has access to all oracles  $\mathcal{O}$ . A *weak adversary* cannot perform any  $\text{CorruptTag}$  query at all. A

narrow adversary does never query the Result oracle. Finally, we introduce the notion of thin adversary which, like the narrow adversary, does never query the Result oracle. Furthermore, a thin adversary cannot see synchronization runs and thus cannot see protocol runs where information is used that is obtained by the second channel.

**Remark 6.2.** Note that this notion of forward adversary is stronger than the one proposed by Vaudenay and closer to the notion of Juels and Weis.

**Definition 6.7 (Privacy).** Let  $C$  be a class of adversaries in  $\{\text{forward, weak, narrow, thin}\}$ . An RFID scheme is said to be  $C$ -private if for all probabilistic polynomial-time adversaries  $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1) \in C$

$$\mathbf{P}[\text{Priv-Game}_{\Pi, \mathcal{A}}(\eta)] - \frac{1}{2}$$

is a negligible function of  $\eta$ .

In our definition of *desynchronization* we follow [CC08]. Consider a valid tag which is referenced by  $id$ . Let its corresponding key  $k$  be denoted  $k_{id}$ . Every tag is initialized by SetupTag using the initial key  $k_{id}^0$ . Then,  $k_{id}^i$  denotes the tag key after  $i$  updates. Since both reader and tag keep track of their own instance of  $k_{id}$ , we write  $rk_{id}$  for the reader instance and  $tk_{id}$  for the tag instance of  $k_{id}$ . Usually,  $rk_{id} = tk_{id} = k_{id}^*$ , but when the tag and reader are no longer synchronized we have  $tk_{id} = k_{id}^i$  and  $rk_{id} = k_{id}^j$  with  $i \neq j$ . In order to allow reasoning about desynchronization, first *correctness* is defined, then the definition of a *strong correctness game* follows. In its turn this game is used to define *strong correctness*. Finally, we define when an RFID scheme can be subject to *desynchronization*.

**Definition 6.8 (Correctness).** An RFID system is said to be correct when the reader outputs  $\perp$  after an authentication protocol  $\pi$  with a non-legitimate tag and outputs the tag id after an authentication protocol  $\pi$  with a legitimate tag.

The **Strong Correctness Game** is comparable to the **Privacy-Game** and its setup is also indistinguishability based. Again, the challenger generates system parameters by calling SetupSystem. Then, the public key  $pk$  is given to an adversary  $\mathcal{A}$  which has access to the set of oracles  $\mathcal{O}$ . At some point  $\mathcal{A}$  outputs an uncorrupted challenge tag  $\mathcal{T}^*$ . Then, the environment runs the authentication protocol with  $\mathcal{T}^*$ . This yields an output  $\perp$  when the tag was not recognized as legitimate or an identifier  $id$  when a legitimate tag was found. Finally, the adversary wins if the reader outputs  $\perp$  and cannot identify  $\mathcal{T}^*$ .

**Definition 6.9 (Strong Correctness Game).**

**Strong-Corr-Game** $_{\Pi, \mathcal{A}}(\eta)$  :

$(sk, pk) \leftarrow \text{SetupSystem}(1^\eta)$   
 $\mathcal{T}^* \leftarrow \mathcal{A}^{\mathcal{O}}(pk)$   
Execute( $\mathcal{R}^*, \mathcal{T}^*$ )  
 $b \leftarrow \text{Result}(\mathcal{R}^*)$   
**winif**  $b = 0$ .

where  $\text{Execute}(\mathcal{R}, \mathcal{T})$  runs the authentication protocol between the reader  $\mathcal{R}$  and the tag  $\mathcal{T}$ . The challenge tag  $\mathcal{T}^*$  must be uncorrupted, which means that no  $\text{CorruptTag}(\mathcal{T}^*)$  query has been made.

**Definition 6.10** (Strong Correctness). Let  $C$  be a class of adversaries in  $\{\text{forward}, \text{weak}, \text{narrow}, \text{thin}\}$ . An RFID system is said to be  $C$ -strong correct if for all probabilistic polynomial-time adversaries  $\mathcal{A} \in C$

$$\mathbf{P}[\text{Strong-Corr-Game}_{\Pi, \mathcal{A}}(\eta)] - \frac{1}{2}$$

is a negligible function of  $\eta$ .

**Definition 6.11** (Key shifts). A key shift in an RFID scheme is the increment of  $|i - j|$  by 1 for an arbitrary tag  $\mathcal{T}$  with  $tk_{id}^i$  and reader  $\mathcal{R}$  with  $rk_{id}^j$ . The value  $|i - j| \in \mathbb{N}$  is called number of key shifts.

**Remark 6.3.** Note that our definition of key shift corresponds with the definition of desynchronization in [CC08]. We prefer to define desynchronization as the state where synchronization between a tag and reader is no longer possible.

The desynchronization value is a pair  $(D_{\mathcal{R}}, D_{\mathcal{T}})$  where  $D_{\mathcal{R}}$  is the maximum number of key shifts  $j - i$  with  $rk_{id}^i \neq tk_{id}^j$  and  $i < j$ , while  $D_{\mathcal{T}}$  is the maximum number of key shifts  $i - j$  with  $rk_{id}^i \neq tk_{id}^j$  and  $i > j$ . Correspondingly, the resynchronization value is a pair  $(R_{\mathcal{R}}, R_{\mathcal{T}})$  where  $R_{\mathcal{R}}$  and  $R_{\mathcal{T}}$  are the maximum number of possible key shifts after which the RFID system still is *strong correct*. An RFID scheme is said to be synchronizable when both  $D_{\mathcal{R}} \leq R_{\mathcal{R}}$  and  $D_{\mathcal{T}} \leq R_{\mathcal{T}}$ .

**Definition 6.12** (Desynchronization). An RFID scheme is subject to desynchronization when  $D_{\mathcal{R}} > R_{\mathcal{R}}$  or  $D_{\mathcal{T}} > R_{\mathcal{T}}$ .

## 6.6 Protocol description

This section introduces a protocol that exploits the use of a *second channel* to achieve *thin-forward* privacy. The protocol should not be subject to desynchronization. Even when a tag is queried an unbounded (yet polynomial) number of times, this should not result in a Denial-of-Service (DOS) or in identification failure. First, we briefly elaborate on the notion of *second channel* that we use, then we define the tag and reader state in this protocol, and finally we discuss the protocol itself.

### 6.6.1 Second channel

The protocol uses a *second channel* which is a channel between the tag and reader that allows a tag to send its tag identity to the reader. This channel uses other physical means than the wireless link and is therefore out of the scope of a *narrow* adversary. Like *narrow* adversaries cannot perform the Result query [Vau07], i.e. cannot learn

outgoing messages on channels other than the wireless link, they also cannot learn incoming messages that are sent on channels other than the wireless link. An example of an outgoing message on a *second channel* is for instance a door that opens when a tag is successfully authenticated. An example of an incoming message is for instance a barcode scanner or keypad connected to an RFID reader that communicates the tag identity to the reader. Of course, this identity still needs to be verified by the reader using the wireless link. The *second channel* speeds up the search process at the reader side when the tag and reader keys are relatively shifted. It does not replace the wireless link.

## 6.6.2 Tag and reader state

In order to keep track of all the state changes and to achieve an RFID system that cannot be desynchronized, the state is managed as follows. First we introduce some notation in Listing 6.1.

Listing 6.1: Notation

Notation	Meaning
$id$	The tag identifier
$k$	The session key; this key is updated in every protocol run
$\tilde{k}$	The synchronization key; for tag-reader synchronization
$h^i(x)$	$i$ times successively hashing of $x$

Every tag has an identifier  $id$ , but this identifier is not part of the tag state. However, a reader needs to relate this tag state somehow to the identifier of the tag. The tag state consists of a session key  $k$  and a synchronization key  $\tilde{k}$ . This pair of keys  $(k, \tilde{k})$  uniquely identifies a tag and thus can be related to  $id$ . The session key is updated in every protocol run, while the synchronization key is only updated after an authenticated message from the reader. A tag always starts to execute an internal key update before it sends any message. The purpose of  $\tilde{k}$  is to allow synchronization between the tag and reader. Finally, it should be possible to extract the identity  $id$  from the tag using a *second channel*. For example, the identity  $id$  can be printed on the tag as a barcode, which allows a barcode scanner to send  $id$  over the *second channel*. The reader state contains, apart from  $k$  and  $\tilde{k}$ , also the tag identifier  $id$ . To distinguish the keys in the reader state from the keys in the tag state we write  $rk_{id}$ ,  $r\tilde{k}_{id}$  and  $tk_{id}$ ,  $t\tilde{k}_{id}$ , respectively. There are two ways in which the reader identifies a tag.

- The reader pre-computes  $h(h^i(rk_{id}), n_r)$  for all  $i < N$ , all tag ids, and some nonce  $n_r$ . Now, identification is a look-up in its pre-computed table (See Listing 6.2 and 6.3).
- The reader obtains the identity  $id$  via the *second channel*. Now,  $id$  allows the reader to look up the synchronization key  $r\tilde{k}_{id}$ , which in its turn is used to induce synchronization of the tag and reader state.

The first method solely uses the wireless link whereas the second method also uses the *second channel*. The synchronization is needed when the tag's session key is beyond the scope  $N$  of the reader. It allows a reader to quickly identify which tag it is targeting.

The protocol design is such that after a synchronization attempt a tag could either update its synchronization key or not. Depending on the situation there are two tag states possible. Therefore, the reader keeps track of two states for each tag simultaneously. The next protocol run in which this particular tag participates resolves then which of the two states is valid. In the tables in Listing 6.2 and 6.3 the two states are captured by the record status  $st$ , which can either be 'old' (O) or 'new' (N). This makes the reader state consist of at most two tuples  $(id, st, k, \tilde{k})$  per tag.

Listing 6.2: Reader Database

$id$	Status $st$	Key $k$	Sync key $\tilde{k}$	Identifier 1	...	Identifier $i$
$id_1$	O	$k_1$	$\tilde{k}_1$	$h(h^1(k_1), n_r)$	...	$h(h^i(k_1), n_r)$
$id_1$	N	$k'_1$	$\tilde{k}'_1$	$h(h^1(k'_1), n_r)$	...	$h(h^i(k'_1), n_r)$
$id_2$	O	$k_2$	$\tilde{k}_2$	$h(h^1(k_2), n_r)$	...	$h(h^i(k_2), n_r)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$id_n$	N	$k_n$	$\tilde{k}_n$	$h(h^1(k_n), n_r)$	...	$h(h^i(k_n), n_r)$

### 6.6.3 Success, failure and synchronization run

This section discusses the *success*, *failure* and *synchronization run*. The authentication protocol is depicted in Figure 6.5 and 6.6. The *success run* is a protocol run in which a reader is able to successfully identify a tag and updates the identifiers in the database accordingly, see Figure 6.5. This update might just concern the next identifier  $k$  and deleting any other record with the same identifier  $id$  or identification might fail. Whenever a reader fails to identify a tag, the corresponding protocol run is called a *failure run*. After a failure run, the reader needs to be provided with the tag identifier  $id$  using the *second channel*, see Figure 6.6. Now,  $id$  can be used to select the tag in the database and find the corresponding synchronization key  $\tilde{k}$  which can be used to execute a *synchronization run* and update both  $k$  and  $\tilde{k}$ . An adversary should not be able to distinguish the different run types.

A *success run* starts with the reader sending a challenge nonce  $n_r$ , see Figure 6.5. In all cases, the tag computes the successive tag key  $k \leftarrow h(k)$  before it sends any response message. This key update on the tag is executed regardless of the number of challenges that have been made before. After the key update, the tag sends identifier  $m_1$ , which directly depends on  $k$  as  $m_1 \leftarrow h(k, n_r)$ . Due to this dependence

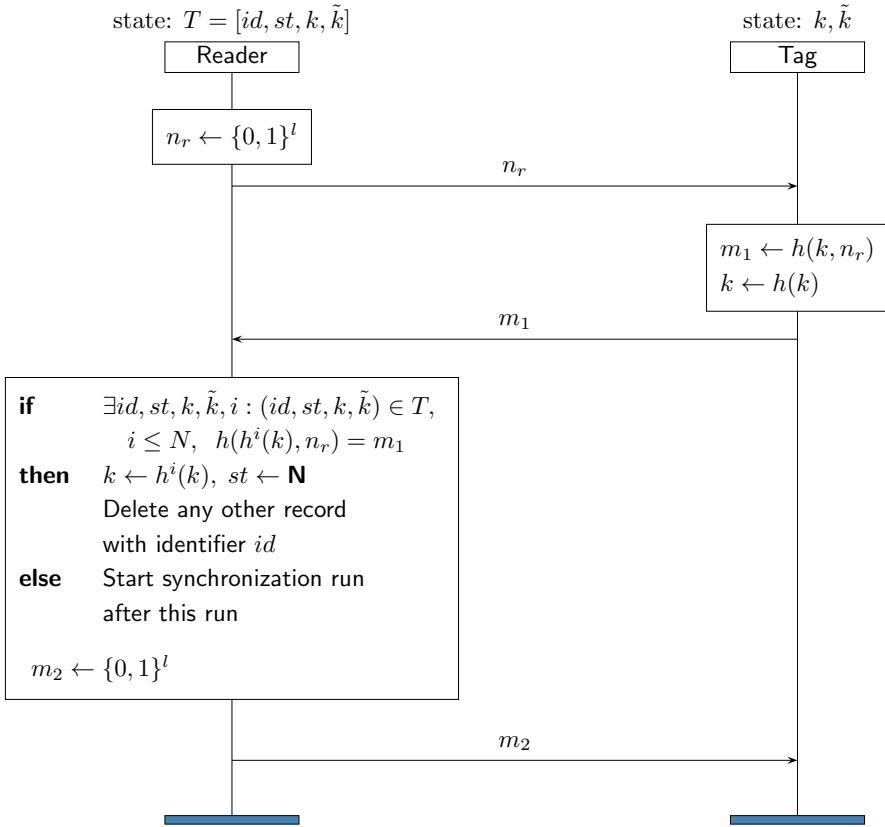


Figure 6.5: The success/failure run

on  $k$ , the successive tag identifiers might run beyond the identifiable scope  $N$  of the reader. The variable  $N$  determines the maximum number of key updates considered in a look-up attempt on the reader side. In the success run we consider a lookup successful when it is of the form  $\exists(id, k) \in T, i \leq N : h(h^i(k), n_r) = m_1$ . The corresponding identity  $id$ , key  $k$  and resynchronization key  $\tilde{k}$  of the tag are resolved, which completes the identification of the tag. In order to make all protocol runs look similar, the reader finishes by sending a random message  $m_2$ .

The *failure run* starts like every run with the reader sending a challenge nonce  $n_r$ , see Figure 6.5. In its turn, the tag first computes the next tag key  $k \leftarrow h(k)$  before it sends any response  $m_1$ . In contrast to a *success run*, the reader is unable to resolve the tag's identity from message  $m_1$ . Since  $m_2$  can be a random message, the reader is still able to finish the protocol. The protocol run still looks similar to any other run. However, the reader failed to identify the tag and has to obtain the tag identifier  $id$  by using a second channel, e.g. the  $id$  could also be available as a barcode. Of course, an adversary could obtain the  $id$  as well, but the tracking effort per tag is big compared to the tracking of RFID tags with a fixed identifier. For this reason, we reduce the problem of tracking RFID tags to the problem of tracking barcodes.

Finally, the *synchronization run* is used once the identifier  $id$  is obtained by the reader, see Figure 6.6. The identifier  $id$  can be provided over the second channel and allows the lookup of  $k$  and  $\tilde{k}$ , which are used later in this run. Again, the reader starts the protocol by sending a nonce  $n_r$ . The tag computes  $m_1 \leftarrow h(k, n_r)$  and updates the tag key  $k \leftarrow h(k)$ , then it sends  $m_1$ . The message  $m_1$  is used as unpredictable input for the last message  $m_2$ . The reader proves to the tag that it knows the synchronization key  $\tilde{k}$  in message  $m_2$ . In order to do so,  $m_2$  is constructed from  $m_1$  and  $\tilde{k}$  as  $m_2 \leftarrow h(m_1, \tilde{k})$ . The tag knows  $\tilde{k}$  and is therefore able to check the validity of  $m_2$ . If it is indeed a valid message, the tag updates the tag key  $k \leftarrow h(\tilde{k}, m_1)$  and the synchronization key  $\tilde{k} \leftarrow h(\tilde{k})$ . The tag does not send any confirmation to the reader after it updated its tag key. This means that the reader does not know whether the key update was successful. For this reason the reader keeps track of an old (O) and new (N) state. Another protocol run should reveal whether the synchronization run was successful or not. Note that adding a final tag-to-reader message, to confirm the key update, does not eliminate the need to keep the old record. This is due to the fact that an adversary might simply block the last message. Clearly, this reduces again to the 3-message protocol that we already have.

Listing 6.3: Look-up tables for different reader randoms

Identifier	$id$	$st$	Identifier	$id$	$st$
$h(h^1(k_1), n_r)$	$id_1$	O	$h(h^1(k_1), n'_r)$	$id_1$	O
$h(h^1(k'_1), n_r)$	$id_1$	N	$h(h^1(k'_1), n'_r)$	$id_1$	N
$h(h^2(k_1), n_r)$	$id_1$	O	$h(h^2(k_1), n'_r)$	$id_1$	O
$h(h^2(k'_1), n_r)$	$id_1$	N	$h(h^2(k'_1), n'_r)$	$id_1$	N
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$h(h^i(k_1), n_r)$	$id_1$	N	$h(h^i(k_1), n'_r)$	$id_1$	N
$h(h^1(k_2), n_r)$	$id_2$	O	$h(h^1(k_2), n'_r)$	$id_2$	O
$h(h^2(k_2), n_r)$	$id_2$	O	$h(h^2(k_2), n'_r)$	$id_2$	O
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$h(h^i(k_2), n_r)$	$id_2$	O	$h(h^i(k_2), n'_r)$	$id_2$	O

### 6.6.4 Precomputation and state resolution

Two important questions need to be answered. First, how can the reader construct a precomputed table for look-up while a random nonce  $n_r$  is used in the protocol of Figure 6.5. Second, how can the number of possible tag states be limited in such a way that state resolution is always possible.

The reader state is stored as shown in the table in Listing 6.2. For every tag the reader precomputes the identifiers  $h(h^i(rk_{id}), n_r)$  for all  $i < N$ . In practice,  $N = 3$



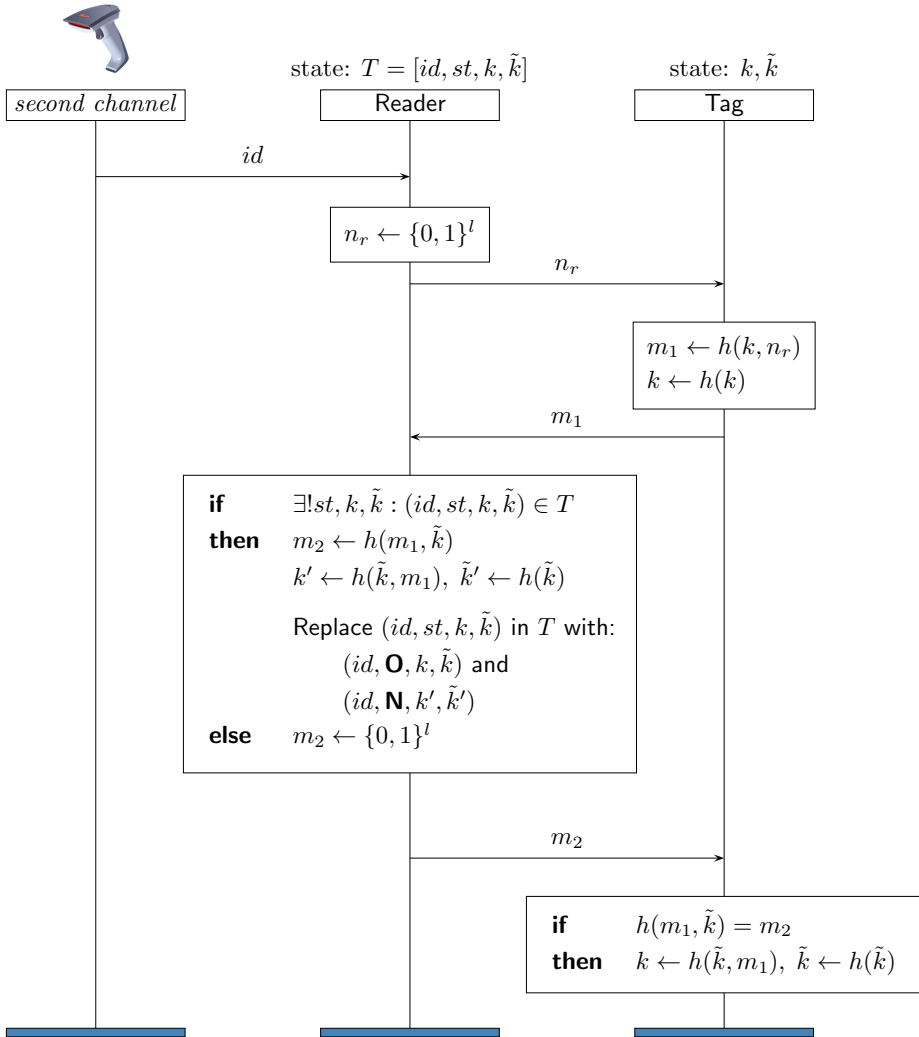


Figure 6.6: The synchronization run

might already be a good choice to withstand desynchronizations that occur due to bad physical circumstances. Since the reader cannot know  $id$  in advance, all nonces  $n_r$  in the precomputed table need to be the same. During idle time the reader can precalculate several tables as shown in Listing 6.2 for different values  $n_r$ . A different representation for two of these tables with different values  $n_r$  is given in Listing 6.3.

When a *synchronization run* is needed, first the identifier  $id$  is obtained by using the *second channel*. Then, the reader executes a *synchronization run*, immediately followed by a normal run. This second run makes clear whether the key update on the tag side was successful or not. If it was successful the reader is able to lookup the tag identifier in the database. However, in case of a *failure run* it is unclear whether the

update was successful but the second run failed, or if the update already failed in the first place. For both scenarios the reader keeps a record corresponding to  $id$ , namely  $\mathbf{O}$  and  $\mathbf{N}$ . In order to prevent desynchronization on this level, this specific tag can be labeled as ‘suspicious’ to indicate that something went wrong in the synchronization run. The tag needs then to be synchronized in a safe environment. Every other attempt of a reader to synchronize would potentially leak information to an adversary and should therefore not be executed.

## 6.7 Security analysis

This section analyzes the security of the proposed protocol in the random oracle model. In the *resynchronization run* the last message  $m_2$  of the protocol leaks location information. For this reason, and in general because forward privacy cannot be achieved for any type of synchronized symmetric protocol construction [NSMSN09], we use the slightly more restricted *thin* adversary. First, we show that our protocol is *thin-forward* private. Then we show that the protocol is not subject to desynchronization.

**Theorem 6.1.** *The protocol depicted in Figure 6.5 and 6.6 is thin-forward private in the random oracle model.*

The proof closely follows the *narrow-forward privacy* proof of modified OSK which is given in [GvR10]. In short, it introduces a simulator  $\mathcal{S}$  which keeps track of all oracle calls  $\mathcal{H}$  and stores them as an entry of the form  $\langle \text{IN}, \text{OUT} \rangle$  in a table  $\mathcal{T}_{\mathcal{H}}$ . Then,  $\mathcal{T}_{\mathcal{H}}$  is adapted such that the protocol messages and thus the resulting view of a particular adversary  $\mathcal{A}_1$  remain the same while the keys, and thus the tag identities, are swapped. This leads to a contradiction.

*Proof (Sketch).* Suppose that there exists an adversary  $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$  which is able to win the **Priv-Game** <sub>$\Pi$</sub>  given in Definition 6.5 with non-negligible probability. Then, imagine a simulator  $\mathcal{S}$  that first initializes the system and then runs the adversary  $\mathcal{A}_0$ . Every oracle call of  $\mathcal{A}_0$  to the oracle  $\mathcal{H}$  is simulated as usual by a table  $T_{\mathcal{H}}$  which contains all previous queries with their corresponding answers. At some point  $\mathcal{A}_0$  finishes and chooses two tags  $\mathcal{T}_0^*$  and  $\mathcal{T}_1^*$ . Let  $(k_0, \tilde{k}_0)$  be the key pair of  $\mathcal{T}_0^*$  and  $(k_1, \tilde{k}_1)$  be the key pair of  $\mathcal{T}_1^*$  after they are returned by  $\mathcal{A}_0$ . As in the game,  $\mathcal{S}$  will draw a random bit  $b$ . Next,  $\mathcal{S}$  runs  $\mathcal{A}_1^{\mathcal{O}}(T_b^*)$  which at some point outputs a guess bit  $b'$ . By hypothesis we get that  $b' = b$  with probability significantly higher than  $\frac{1}{2}$ . By  $\dagger$  we identify the predecessor value of a key, so the predecessor of  $k_0$  is  $k_0^\dagger$ . Now  $\mathcal{S}$  swaps all occurrences of  $k_0$  with  $k_1$  in all entries of  $T_{\mathcal{H}}$ . Note that either the entry  $\langle h(\tilde{k}_0^\dagger, \_), k_0 \rangle$  or the entry  $\langle h(k_0^\dagger), k_0 \rangle$  is present in  $T_{\mathcal{H}}$ . The first one occurs when the last update of  $k$  was in a *synchronization run*. The latter one occurs when the last update of  $k$  was in a non-synchronization protocol run. The replacement of  $k_0$  by  $k_1$  and vice versa does not affect the protocol messages since  $k_0$  and  $k_1$  are not involved in any protocol messages after the oracle call entries defined above.

Furthermore,  $m_2 \leftarrow h(m_1, \tilde{k})$  is the only message that involves  $\tilde{k}$  and only occurs in a *synchronization run*. Since  $\mathcal{A}_1$  is thin, it is clear that  $\tilde{k}$  does not have any influence on the view of the adversary.

Now,  $\mathcal{S}$  runs adversary  $\mathcal{A}_1^\mathcal{O}(T_{1-b}^*)$  with the adjusted  $T_{\mathcal{H}}$ . Again by hypothesis, we get that  $\mathcal{A}_1$  outputs  $b' = 1 - b$  with probability significantly higher than  $\frac{1}{2}$ . Since  $\mathcal{A}_1$  is thin, its view is exactly the same as in the previous run, which leads to a contradiction.  $\square$

**Theorem 6.2.** *The protocol depicted in Figure 6.5 and 6.6 is not subject to desynchronization in the random oracle model.*

*Proof (Sketch).* In order to show that desynchronization is impossible we have to show that both  $D_{\mathcal{R}} \leq R_{\mathcal{R}}$  and  $D_{\mathcal{T}} \leq R_{\mathcal{T}}$  hold. The tag state is the tuple  $(k, \tilde{k})$ . First,  $k$  is always updated,  $\tilde{k}$  is only updated after a *synchronization run*. Therefore, we focus on  $\tilde{k}$  to induce key shifts since only then a desynchronization is possible. From the protocol definition we deduce that  $D_{\mathcal{R}} = R_{\mathcal{R}} = 1$  since a reader only starts a synchronization run when it was able to look up  $\tilde{k}$  in one of the two possible tag states. Furthermore, we know that  $D_{\mathcal{T}} = 0$  from which follows that  $D_{\mathcal{T}} \leq R_{\mathcal{T}}$  since  $R_{\mathcal{T}}$  has to be positive. Suppose that either  $D_{\mathcal{R}} > R_{\mathcal{R}}$  or  $D_{\mathcal{T}} \leq R_{\mathcal{T}}$  is true, then there exists an adversary  $\mathcal{A}$  that wins the **Strong-Corr-Game** $_{\Pi}$  given in Definition 6.9 with non-negligible probability. This means that  $\mathcal{A}$  outputs a tag  $\mathcal{T}^*$  with key  $t\tilde{k}^i$  while the reader has no matching key  $r\tilde{k}^{j-1}$  or  $r\tilde{k}^j$ , since  $i \neq j - 1$  and  $i \neq j$  has to be true. There are two ways for the adversary to achieve this:

$i > j$  : The tag key is updated  $(i - j)$ -times more than the reader key. The only way to induce a key update on the tag side is to construct the message  $m_2 = h(m_1, \tilde{k})$ . Because of the one-wayness of  $h$  and since the adversary cannot call `CorruptTag`, the key  $\tilde{k}$  is not known and it is impossible to construct  $m_2$  for the adversary. Only the reader  $\mathcal{R}$  is able to construct  $m_2 = h(m_1, \tilde{k})$ , but inherent to this generation of  $m_2$  is the storage of the new reader keys  $(rk^{j+1}, r\tilde{k}^{j+1})$  while at the same time the old keys  $(rk^j, r\tilde{k}^j)$  are maintained. The last option would be a replay of  $m_2$ , but this is rendered impossible by the use of  $n_r$  in  $m_1$ , and thus in  $m_2$ , which introduces freshness in every protocol run. To conclude, it is not possible to obtain  $i > j$ .

$i < j - 1$  : The reader key is updated  $(j - i)$ -times more than the tag key. By hypothesis we know that  $i < j - 1$  since  $i \neq j$ ,  $i \neq j - 1$  and  $i \not\geq j$  as concluded in the previous case. Let  $i = j$ , the only way to update  $r\tilde{k}^j$  to  $r\tilde{k}^{j+1}$  comes with the generation of  $m_2 = h(m_1, \tilde{k})$ . If  $m_2$  is received by the tag it will update its key from  $t\tilde{k}^i$  to  $t\tilde{k}^{i+1}$  and consequently  $i = j$  again. Obviously, to prevent incrementation of  $i$  is to block or replace  $m_2$  since then the tag does not update its key and as a result  $i = j - 1$ . Next, the adversary needs to go one step further since the reader is still able to identify the tag ( $t\tilde{k}^i = r\tilde{k}^{j-1}$ ). To induce another reader key update, the reader has to be provided with the tag identifier  $id$  by using the *second channel*. When the last synchronization attempt turned out to

be unsuccessful, which is stored in the reader state belonging to  $id$ , the reader just sends random data for  $m_2$ . In this situation resynchronization has to be done in a safe environment. The tag state either contains  $t\tilde{k}^i$  when in the last synchronization attempt  $m_2$  was blocked or the tag state contains  $t\tilde{k}^{i+1}$  when the last synchronization run was successful. In the latter case the reader is able to identify the tag since it knows  $r\tilde{k}^j$  which equals  $t\tilde{k}^{i+1}$ , respectively. To conclude, the adversary needs to induce a synchronization run, which can be done by first querying the tag more than  $N$  times. Then, before the reader starts a *synchronization run* it retrieves  $id$ . By looking up the correct entry using  $id$  the reader has enough information to decide on the execution of another *synchronization run*. If the last attempt was unsuccessful this indicates that something suspicious is going on and resynchronization should be done in a safe environment. If the last attempt was successful the reader is sure that  $i = j$ . So,  $\max(|i - j|) = 1$  where  $i < j$ , which is not enough to satisfy  $i < j - 1$ .

Finally, from the two possible strategies to win the **Strong-Corr-Game<sub>II</sub>** we conclude that both  $i > j$  and  $i < j - 1$  cannot be satisfied, therefore contradicting the assumption that such an adversary  $\mathcal{A}$  exists.  $\square$

## 6.8 Conclusion

In this chapter we have presented a new approach to tackle the desynchronization problem. This desynchronization problem is actually an unwanted side effect of a solution to another problem: forward privacy for RFID tags. Many solutions tend to solve this problem by introducing a stateful protocol. A main challenge of these protocols is to keep the tag and reader state synchronized while at the same time no information can be leaked that enables an adversary to track a specific tag. To the best of our knowledge there have been no attempts to seek the solution beyond the bounds of the wireless link. In line with the abilities of a *narrow* adversary, introduced by Vaudenay in [Vau07], in which an adversary is unable to see the result of a protocol run like a gate that opens, we propose to use this information flow also in the opposite direction. This means that additional information is made available to the reader which it can use to identify and resynchronize with the tag. A *narrow* adversary does not have access to this information since it is not send on the wireless link but some other communication channel which is introduced in this chapter as the *second channel*. We add some mild restrictions to the *narrow* adversary and introduce this as the *thin* adversary which is needed to prove forward-privacy under mild additional assumptions. Suppose that barcode scanners are used as *second channel* and RFID tags are also equipped with barcodes. Furthermore, assume a protocol  $P$  that uses the *second channel* such that it provides *thin-forward* privacy and is not subject to desynchronization. Then, tracking tags in this system has become as hard as tracking barcodes.

The *second channel* can be used in new protocol designs and relaxes the workload

---

of the reader and/or database. It allows to solve the desynchronization problem in an elegant way and eliminates the need for restrictions on the number of key updates that can be induced by an adversary between two synchronizations. In order to show that such a protocol can be constructed we proposed a protocol that only uses hash functions. We have shown that it provides *thin-forward* privacy in the random oracle model. Furthermore, we followed the definition of [CC08] to show that the protocol is not subject to desynchronization.



---

## Bibliography

- [ABV12] G. Alpár, L. Batina, and R. Verdult. Using NFC phones for proving credentials. In *16th Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB&DFT 2012)*, volume 7201 of *Lecture Notes in Computer Science*, pages 317–330. Springer-Verlag, 2012.
- [ACCP10] B. Alomair, A. Clark, J. Cuellar, and R. Poovendran. Scalable RFID systems: a privacy-preserving protocol with constant-time identification. In *40th International Conference on Dependable Systems and Networks (DSN 2010)*, pages 1–10. IEEE Computer Society, 2010.
- [ACM12a] G. Avoine, X. Carpent, and B. Martin. Privacy-friendly synchronized ultralightweight authentication protocols in the storm. *Journal of Network and Computer Applications*, 35(2):826–843, February 2012.
- [ACM12b] G. Avoine, I. Coisel, and T. Martin. A privacy-restoring mechanism for offline RFID systems. In *5th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec 2012)*, pages 63–74, Tucson, Arizona, USA, April 2012. ACM.
- [AD11] M. Asadpour and M. Dashti. A privacy-friendly RFID protocol using reusable anonymous tickets. In *10th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2011)*, pages 206–213. IEEE Computer Society, 2011.
- [AG97] M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. In *4th ACM Conference on Computer and Communications Security (CCS 1997)*, pages 36–47. ACM, 1997.
- [AO05] G. Avoine and P. Oechslin. A scalable and provably secure hash based RFID protocol. In *3rd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops)*, pages 110–114. IEEE Computer Society, 2005.
- [AR02] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
- [Att07] M. Attaran. RFID: an enabler of supply chain operations. *Supply Chain Management: An International Journal*, 12(4):249–257, 2007.

- [Avo05] G. Avoine. Adversary model for radio frequency identification. Technical report, Swiss Federal Institute of Technology (EPFL), Security and Cryptography Laboratory (LASEC), 2005.
- [BBEG09] C. Berbain, O. Billet, J. Etrog, and H. Gilbert. An efficient forward private RFID protocol. In *16th ACM Conference on Computer and Communications Security (CCS 2009)*, pages 43–53, New York, NY, USA, 2009. ACM.
- [BBLF11] A. Barisani, D. Bianco, A. Laurie, and Z. Franken. Chip & PIN is definitely broken. Presentation at CanSecWest Applied Security Conference, Vancouver, 2011. Slides available at [http://dev.inversepath.com/download/emv/emv\\_2011.pdf](http://dev.inversepath.com/download/emv/emv_2011.pdf).
- [BC94] S. Brands and D. Chaum. Distance-bounding protocols. In *Advances in Cryptology (EUROCRYPT 1993)*, volume 765 of *Lecture Notes in Computer Science*, pages 344–359. Springer-Verlag, 1994.
- [BD91] T. Beth and Y. Desmedt. Identification tokens—or: Solving the chess grandmaster problem. pages 169–176. Springer-Verlag, 1991.
- [BdKGP<sup>+</sup>12] A. Blom, G. de Koning Gans, E. Poll, J. de Ruiter, and R. Verdult. Designed to fail: A USB-connected reader for online banking. In *17th Nordic Conference on Secure IT Systems (NordSec 2012)*, volume 7617 of *Lecture Notes in Computer Science*. Springer-Verlag, 2012.
- [BdMM08] M. Burmester, B. de Medeiros, and R. Motta. Anonymous RFID authentication supporting constant-cost key-lookup against active adversaries. *Journal of Applied Cryptography*, 1(2):79–90, 2008.
- [BGV<sup>+</sup>12] J. Balasch, B. Gierlichs, R. Verdult, L. Batina, and I. Verbauwhede. Power analysis of Atmel CryptoMemory - recovering keys from secure EEPROMs. In *12th Cryptographers' Track at the RSA Conference (CT-RSA 2012)*, volume 7178 of *Lecture Notes in Computer Science*, pages 19–34. Springer-Verlag, 2012.
- [BKZ11] A. Biryukov, I. Kizhvatov, and B. Zhang. Cryptanalysis of the Atmel cipher in SecureMemory, CryptoMemory and CryptoRF. In *9th Applied Cryptography and Network Security (ACNS 2011)*, volume 6715 of *Lecture Notes in Computer Science*, pages 91–109. Springer-Verlag, 2011.
- [Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE workshop on Computer Security Foundations (CSFW 2001)*, pages 82–96. IEEE Computer Society, 2001.
- [Bog07] A. Bogdanov. Linear slide attacks on the KeeLoq block cipher. In *3rd International Conference on Information Security and Cryptology (INSCRYPT 2007)*, volume 4990 of *Lecture Notes in Computer Science*, pages 66–80. Springer-Verlag, 2007.
- [BS80] G. Bochmann and C. Sunshine. Formal methods in communication protocol design. *IEEE Transactions on Communications*, 28(4):624–631, 1980.
- [BSI00] Identification card systems. inter-sector electronic purse. dataelements and interchanges. BS EN 1546-3:2000, 2000.
- [CC08] S. Canard and I. Coisel. Data synchronization in privacy-preserving RFID authentication schemes. In *3rd Workshop on RFID Security and Privacy (RFID-Sec 2007)*, volume 17 of *Lecture Notes in Electrical Engineering*. Springer-Verlag, 2008.



- [Cha09] R. Charette. This car runs on code. *IEEE Spectrum*, 46(3):3, 2009.
- [Cho10a] K. Chopra. Physics behind RFID smart card security in context of privacy. Master's thesis, The University of Texas at Arlington, 2010.
- [Cho10b] O. Choudary. The smart card detective: A hand-held EMV interceptor. Master's thesis, University of Cambridge, 2010.
- [CMK<sup>+</sup>11] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX Security Symposium (USENIX Security 2012)*, pages 77–91, 2011.
- [COQ09] N. T. Courtois, S. O'Neil, and J.-J. Quisquater. Practical algebraic attacks on the Hitag2 stream cipher. In *12th Information Security Conference (ISC 2009)*, volume 5735 of *Lecture Notes in Computer Science*, pages 167–176. Springer-Verlag, 2009.
- [Cou09] N. T. Courtois. The dark side of security by obscurity - and cloning MIFARE Classic rail and building passes, anywhere, anytime. In *4th International Conference on Security and Cryptography (SECRYPT 2009)*, pages 331–338. INSTICC Press, 2009.
- [Cum03] N. Cummings. iClass levels of security, April 2003.
- [Cum06] N. Cummings. Sales training. Slides from HID Technologies, March 2006.
- [DDS<sup>+</sup>11] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege escalation attacks on Android. In *13th Information Security Conference (ISC 2010)*, volume 6531 of *Lecture Notes in Computer Science*, pages 346–360. Springer-Verlag, 2011.
- [DGB88] Y. Desmedt, C. Goutier, and S. Bengio. Special uses and abuses of the fiat-shamir passport protocol. In *Advances in Cryptology (CRYPTO'87)*, pages 21–39. Springer-Verlag, 1988.
- [DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [DHW<sup>+</sup>12] B. Driessen, R. Hund, C. Willems, C. Paar, and T. Holz. Don't trust satellite phones: A security analysis of two satphone standards. In *33rd IEEE Symposium on Security and Privacy (S&P 2012)*, pages 128–142. IEEE Computer Society, 2012.
- [Dim05] T. Dimitriou. A lightweight RFID protocol to protect against traceability and cloning attacks. In *1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005)*, pages 59–66. IEEE Computer Society, 2005.
- [DK02] H. Delfs and H. Knebl. *Introduction to Cryptography: Principles and Applications*. Springer-Verlag, 2002.
- [dKG08] G. de Koning Gans. Analysis of the MIFARE classic used in the OV-chipkaart project. Master's thesis, Radboud University Nijmegen, 2008.
- [dKGdR12] G. de Koning Gans and J. de Ruiter. The smartlogic tool: Analysing and testing smart card protocols. In *5th International Conference on Software Testing, Verification, and Validation (ICST 2012)*, pages 864–871. IEEE Computer Society, 2012.

- [dKGG10] G. de Koning Gans and F. D. Garcia. Towards a practical solution to the RFID desynchronization problem. In S. O. Yalcin, editor, *6th Workshop on RFID Security (RFIDSec 2010)*, volume 6370 of *Lecture Notes in Computer Science*, pages 203–219. Springer-Verlag, 2010.
- [dKGHG08] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia. A practical attack on the MIFARE Classic. In *8th Smart Card Research and Advanced Applications Conference (CARDIS 2008)*, volume 5189 of *Lecture Notes in Computer Science*, pages 267–282. Springer-Verlag, 2008.
- [dKGV11] G. de Koning Gans and E. Verheul. Best effort and practice activation codes. In *8th International on Trust, Privacy and Security in Digital Business (Trust-Bus 2011)*, volume 6863 of *Lecture Notes in Computer Science*, pages 98–112. Springer -Verlag, 2011.
- [DM07] S. Drimer and S. Murdoch. Keep your enemies close: Distance bounding against smartcard relayattacks. In *16th USENIX Security Symposium (USENIX Security 2007)*, pages 1–16. USENIX Association, 2007.
- [DR02] J. Daemen and V. Rijmen. *The design of Rijndael: AES – the advanced encryption standard*. Springer, 2002.
- [EMV08a] EMVCo. EMV– integrated circuit card specifications for payment systems, book 1: Application independent icc to terminal interface requirements, 2008.
- [EMV08b] EMVCo. EMV– integrated circuit card specifications for payment systems, book 2: Security and key management, 2008.
- [EMV08c] EMVCo. EMV– integrated circuit card specifications for payment systems, book 3: Application specification, 2008.
- [EMV08d] EMVCo. EMV– integrated circuit card specifications for payment systems, book 4: Cardholder, attendant, and acquirer interface requirements, 2008.
- [Ess11] L. Essers. Banken dichten skimgat in het nieuwe pinnen (6-4-2011). <http://webwereld.nl/nieuws/106271/banken-dichten-skimgat-in-het-nieuwe-pinnen.html>, 2011.
- [Fei73] H. Feistel. Cryptography and computer privacy. *Scientific American*, 228:15–23, 1973.
- [FHV10] J. Fan, J. Hermans, and F. Vercauteren. On the claimed privacy of ec-rac iii. In S. O. Yalcin, editor, *6th Workshop on RFID Security (RFIDSec 2010)*, volume 6370 of *Lecture Notes in Computer Science*, pages 66–74. Springer-Verlag, 2010.
- [FIP99] FIPS 46-3, Data Encryption Standard (DES). National Institute for Standards and Technology (NIST), Gaithersburg, MD, USA, 1999.
- [FL12] R. Focardi and F. L. Luccio. Secure recharge of disposable RFID tickets. In *8th International Workshop on Formal Aspects of Security and Trust (FAST 2011)*, volume 7140 of *Lecture Notes in Computer Science*, pages 85–99. Springer-Verlag, 2012.
- [Gar05] S. Garfinkel. History’s worst software bugs, 2005.
- [Gar08] F. D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. PhD thesis, Radboud University Nijmegen, 2008.

- [GdKGM<sup>+</sup>08] F. D. Garcia, G. de Koning Gans, R. Muijers, P. van Rossum, R. Verdult, R. Wichers Schreur, and B. Jacobs. Dismantling MIFARE Classic. In *13th European Symposium on Research in Computer Security (ESORICS 2008)*, volume 5283 of *Lecture Notes in Computer Science*, pages 97–114. Springer-Verlag, 2008.
- [GdKGV11] F. D. Garcia, G. de Koning Gans, and R. Verdult. Exposing iClass key diversification. In *5th USENIX Workshop on Offensive Technologies (USENIX WOOT 2011)*, pages 128–136. USENIX Association, 2011.
- [GdKGV12] F. D. Garcia, G. de Koning Gans, and R. Verdult. Tutorial: Proxmark, the swiss army knife for RFID security research. Technical report, Radboud University Nijmegen, 2012.
- [GdKGV12] F. D. Garcia, G. de Koning Gans, R. Verdult, and M. Meriac. Dismantling iClass and iClass Elite. In *17th European Symposium on Research in Computer Security (ESORICS 2012)*, *Lecture Notes in Computer Science*. Springer-Verlag, 2012.
- [GGvR08] D. Galindo, F. D. Garcia, and P. van Rossum. Computational soundness of non-malleable commitments. In L. Chen, Y. Mu, and W. Susilo, editors, *4th Information Security Practice and Experience Conference (ISPEC 2008)*, volume 4266 of *Lecture Notes in Computer Science*, pages 361–376. Springer Verlag, 2008.
- [GHPvR05] F. D. Garcia, I. Hasuo, W. Pieters, and P. van Rossum. Provable anonymity. In R. Küsters and J. Mitchell, editors, *3rd ACM Workshop on Formal Methods in Security Engineering (FMSE 2005)*, pages 63–72. ACM Press, November 2005.
- [GJ12] F. D. Garcia and B. Jacobs. The fall of a tiny star. <http://www.cs.ru.nl/~flaviog/publications/Tiny.Star.pdf>, 2012.
- [Gol97] J. D. Golic. Cryptanalysis of alleged A5 stream cipher. In *16th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1997)*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1997.
- [GSM95] Digital cellular telecommunications system (phase 2+); specification of the subscriber identity module — mobile equipment (SIM-ME) interface, 1995.
- [GvR06a] F. D. Garcia and P. van Rossum. Sound computational interpretation of formal hashes. Technical Report ICIS-R06001, Nijmegen Institute for Computing and Information Sciences, <http://www.cs.ru.nl/research/reports/info/ICIS-R06001.html>, 2006.
- [GvR06b] F. D. Garcia and P. van Rossum. Sound computational interpretation of symbolic hashes in the standard model. In H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, and S. Kawamura, editors, *Advances in Information and Computer Security. International Workshop on Security (IWSEC 2006)*, volume 4266 of *Lecture Notes in Computer Science*, pages 33–47. Springer Verlag, Oct 23–24 2006.
- [GvR08] F. D. Garcia and P. van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Theoretical Computer Science*, 394(1–2):112–133, 2008.

- [GvR10] F. D. Garcia and P. van Rossum. Modeling privacy for off-line RFID systems. In D. Gollmann and J.-L. Lanet, editors, *9th Smart Card Research and Advanced Applications (CARDIS 2010)*, volume 6035 of *Lecture Notes in Computer Science*, pages 194–208. Springer Verlag, 2010.
- [GvRVWS09] F. D. Garcia, P. van Rossum, R. Verdult, and R. Wichers Schreur. Wirelessly pickpocketing a MIFARE Classic card. In *30th IEEE Symposium on Security and Privacy (S&P 2009)*, pages 3–15. IEEE Computer Society, 2009.
- [GvRVWS10] F. D. Garcia, P. van Rossum, R. Verdult, and R. Wichers Schreur. Dismantling SecureMemory, CryptoMemory and CryptoRF. In *17th ACM Conference on Computer and Communications Security (CCS 2010)*, pages 250–259. ACM, 2010.
- [Han11] G. Hancke. Design of a secure distance-bounding channel for RFID. *Journal of Network and Computer Applications*, 34(3):877–887, 2011.
- [HCPLPT10] J. C. Hernandez-Castro, P. Peris-Lopez, R. Phan, and J. Tapiador. Cryptanalysis of the David-Prasad RFID ultralightweight authentication protocol. In S. O. Yalcin, editor, *6th Workshop on RFID Security (RFIDSec 2010)*, volume 6370 of *Lecture Notes in Computer Science*, pages 22–34. Springer-Verlag, 2010.
- [Hel80] M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
- [HHJ<sup>+</sup>06] J.-H. Hoepman, E. Hubbers, B. Jacobs, M. Oostdijk, and R. Schreur. Crossing borders: Security and privacy issues of the european e-passport. In H. Yoshiura, K. Sakurai, K. Rannenber, Y. Murayama, and S. Kawamura, editors, *1st International Workshop on Security, Advances in Information and Computer Security (IWSEC 2006)*, volume 4266 of *Lecture Notes in Computer Science*, pages 152–167. Springer-Verlag, 2006.
- [HID06] HID Global. HID management key letter, November 2006.
- [HID09] HID Global. iClass RW100, RW150, RW300, RW400 readers, 2009.
- [HID10] HID Global. Controlvault identity protection (brochure), 2010.
- [HJSW06] J. Halamka, A. Juels, A. Stubblefield, and J. Westhues. The security implications of verichip cloning. *Journal of the American Medical Informatics Association*, 13(6):601–607, 2006.
- [HK05] G. Hancke and M. Kuhn. An RFID distance bounding protocol. In *1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005)*, pages 67–73. IEEE Computer Society, 2005.
- [HSH11] T. Halevi, N. Saxena, and S. Halevi. Tree-based HB protocols for privacy-preserving authentication of rfid tags. *Journal of Computer Security*, 19(2):343–363, 2011.
- [IC04] PicoPass 2KS. Product Datasheet, Nov 2004. Inside Contactless.
- [ISO00] ISO/IEC 15693-1. Identification cards — Contactless integrated circuit cards — Vicinity cards — Part 1: Physical characteristics. International Organization for Standardization (ISO), Geneva, Switzerland, 2000.
- [ISO01] ISO/IEC 14443. Identification cards — Contactless integrated circuit cards — Proximity cards. International Organization for Standardization (ISO), Geneva, Switzerland, 2001.

- [ISO06] ISO/IEC 15693-2. Identification cards — Contactless integrated circuit cards — Vicinity cards — Part 2: Air interface and initialization. International Organization for Standardization (ISO), Geneva, Switzerland, 2006.
- [ISO07a] ISO/IEC 7816-3. Identification cards — Integrated circuit card programming interfaces — Part 3: Cards with contacts — Electrical interface and transmission protocols. International Organization for Standardization (ISO), Geneva, Switzerland, 2007.
- [ISO07b] ISO/IEC 7816-4. Identification cards — Integrated circuit card programming interfaces — Part 4: Organization, security and commands for interchange. International Organization for Standardization (ISO), Geneva, Switzerland, 2007.
- [ISO08] ISO/IEC 27005. Information technology — Security techniques — Information security risk management. International Organization for Standardization (ISO), Geneva, Switzerland, 2008.
- [ISO09] ISO/IEC 15693-3. Identification cards — Contactless integrated circuit cards — Vicinity cards — Part 3: Anticollision and transmission protocol. International Organization for Standardization (ISO), Geneva, Switzerland, 2009.
- [JRS03] A. Juels, R. L. Rivest, and M. Szydło. The blocker tag: Selective blocking of RFID tags for consumer privacy. In *10th ACM Conference on Computer and Communications Security (CCS 2003)*, pages 103–111. ACM, 2003.
- [Jue06] A. Juels. RFID security and privacy: A research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, 2006.
- [JW05] A. Juels and S. Weis. Authenticating pervasive devices with human protocols. In V. Shoup, editor, *Advances in Cryptology (CRYPTO 2005)*, volume 3126 of *Lecture Notes in Computer Science*, pages 293–308, Santa Barbara, California, USA, 2005. Springer-Verlag.
- [JW09] A. Juels and S. Weis. Defining strong privacy for RFID. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):1–23, 2009.
- [JWS11] B. Jacobs and R. Wichers Schreur. Logical formalisation and analysis of the MIFARE Classic card in PVS. In *2nd International Conference on Interactive Theorem Proving*, volume 6898 of *Lecture Notes in Computer Science*, pages 3–17. Springer-Verlag, 2011.
- [KAK<sup>+</sup>09] C. Kim, G. Avoine, F. Koeune, F. Standaert, and O. Pereira. The swiss-knife RFID distance bounding protocol. In *11th International Conference on Information Security and Cryptology (ICISC 2008)*, volume 5461 of *Lecture Notes in Computer Science*, pages 98–115. Springer-Verlag, 2009.
- [Ker83] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, 9(1):5–38, 1883.
- [KJL<sup>+</sup>11] C. Kim, E.-G. Jung, D. H. Lee, C.-H. Jung, and D. Han. Cryptanalysis of INCrypt32 in HID’s iClass systems. *Cryptology ePrint Archive*, Report 2011/469, 2011.
- [KKMP09] M. Kasper, T. Kasper, A. Moradi, and C. Paar. Breaking KeeLoq in a flash: on extracting keys at lightning speed. In *2nd International Conference on Cryptology in Africa, Progress in Cryptology (AFRICACRYPT 2009)*, volume 5580 of *Lecture Notes in Computer Science*, pages 403–420. Springer-Verlag, 2009.

- [KLPR10] L. Knudsen, G. Leander, A. Poschmann, and M. Robshaw. PRINTcipher: a block cipher for ic-printing. In *12th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2010)*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer-Verlag, 2010.
- [Knu08] L. Knudsen. The security of Feistel ciphers with six rounds or less. volume 15, pages 207–222. Springer Verlag, 2008.
- [Kov12] Kovio 2K. <http://www.kovio.com>, 2012.
- [KSRW04] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an electronic voting system. In *25th IEEE Symposium on Security and Privacy (S&P 2004)*, pages 27–40. IEEE Computer Society, 2004.
- [Leo12] Smart cards: Leon devices. <http://www.citi.umich.edu/projects/smartcard/leon.html>, 2012.
- [LG98] M. Loukides and J. Gilmore, editors. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998.
- [Low96] G. Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. *Software - Concepts and Tools*, 17(3):93–102, 1996.
- [LST<sup>+</sup>09] S. Lucks, A. Schuler, E. Tews, R.-P. Weinmann, and M. Wenzel. Attacks on the DECT authentication mechanisms. In *9th Cryptographers' Track at the RSA Conference (CT-RSA 2009)*, volume 5473 of *Lecture Notes in Computer Science*, pages 48–65. Springer-Verlag, 2009.
- [Lub96] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.
- [MDAB10] S. Murdoch, S. Drimer, R. Anderson, and M. Bond. Chip and PIN is broken. In *27th IEEE Symposium on Security and Privacy (S&P 2006)*, pages 433–446. IEEE Computer Society, 2010.
- [Mer10] M. Meriac. Heart of darkness - exploring the uncharted backwaters of HID iClass security. In *27th Chaos Computer Congress (27C3)*, December 2010.
- [MW04a] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In M. Naor, editor, *1st Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer-Verlag, 2004.
- [MW04b] D. Molnar and D. Wagner. Privacy and security in library RFID: Issues, practices, and architectures. In *11th ACM Conference on Computer and Communications Security (CCS 2004)*, pages 210–219. ACM, 2004.
- [NESP08] K. Nohl, D. Evans, Starbug, and H. Plötz. Reverse engineering a cryptographic RFID tag. In *17th USENIX Security Symposium (USENIX Security 2008)*, pages 185–193. USENIX Association, 2008.
- [NGEF99] X. Nie, L. Gazsi, F. Engel, and G. Fettweis. A new network processor architecture for high-speed communications. In *4th IEEE Workshop on Signal Processing Systems (SiPS 1999)*, pages 548–557. IEEE Computer Society, 1999.
- [NP07] K. Nohl and H. Plötz. Mifare, little security, despite obscurity. *Presentation on the 24th Congress of the Chaos Computer Club in Berlin (24C3)*, December 2007.

- [NR99] M. Naor and O. Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. volume 12, pages 29–66. Springer Verlag, 1999.
- [NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
- [NSMSN09] C. Ng, W. Susilo, Y. Mu, and R. Safavi-Naini. New privacy results on synchronized RFID authentication protocols against tag tracing. In *14th European Symposium on Research in Computer Security (ESORICS 2009)*, volume 5789 of *Lecture Notes in Computer Science*, page 321. Springer-Verlag, 2009.
- [NXP02] MIFARE Standard 4KByte Card IC Functional Specification. NXP Semiconductors, <http://www.nxp.com>, 2002. Revision 3.1.
- [NXP07] MIFARE Standard 4KByte Card IC Functional Specification. NXP Semiconductors, <http://www.nxp.com>, 2007.
- [NXP10] P5Cx012/02x/40/73/80/144 family: Secure dual interface and contact PKI smart card controller. NXP Semiconductors, <http://www.nxp.com>, 2010.
- [Nyq24] H. Nyquist. Certain factors affecting telegraph speed. *Bell System Technical Journal*, 3:324–346, 1924.
- [Nyq28] H. Nyquist. Certain topics in telegraph transmission theory. *American Institute of Electrical Engineers, Transactions of the*, 47(2):617–644, april 1928.
- [Oec03] P. Oechslin. Making a faster cryptanalytic time-memory trade-off. pages 617–630. Springer-Verlag, 2003.
- [ORSvH95] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomenato the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, 1995.
- [OSK03] M. Ohkubo, K. Suzuki, and S. Kinoshita. Cryptographic approach to “privacy-friendly” tags. In *RFID Privacy Workshop*, volume 82. MIT, Cambridge, MA, 2003.
- [OSM12] <http://bb.osmocom.org/trac/wiki/SIMtrace>, 2012. Osmocom SIMtrace.
- [Pat92] J. Patarin. New results on pseudorandom permutation generators based on the DES scheme. In *Advances in Cryptology (CRYPTO '91)*, volume 537, pages 301–312. Springer Verlag, 1992.
- [Pat98] J. Patarin. About Feistel schemes with six (or more) rounds. In *Fast Software Encryption, 5th International Workshop, (FSE '98)*, volume 1372, pages 103–121. Springer Verlag, 1998.
- [Pat04] J. Patarin. Security of random Feistel schemes with 5 or more rounds. In *Advances in Cryptology (CRYPTO 2004)*, pages 106–122. Springer Verlag, 2004.
- [PBR<sup>+</sup>05] R. Parashkov, E. Becker, T. Riedl, H. Johannes, and W. Kowalsky. Large area electronics using printing methods. *Proceedings of the IEEE*, 93(7):1321–1329, 2005.
- [PLHCETR06a] P. Peris-Lopez, J. C. Hernandez-Castro, J. Estevez-Tapiador, and A. Ribagorda. EMAP: An efficient mutual-authentication protocol for low-cost

- RFID tags. In *On the Move to Meaningful Internet Systems (OTM 2006)*, volume 4278 of *Lecture Notes in Computer Science*, pages 352–361. Springer-Verlag, 2006.
- [PLHCETR06b] P. Peris-Lopez, J. C. Hernandez-Castro, J. Estevez-Tapiador, and A. Ribagorda. RFID systems: A survey on security threats and proposed solutions. In *11th International Conference on Personal Wireless Communications (PWC 2006)*, volume 4217 of *Lecture Notes in Computer Science*, pages 159–170. Springer-Verlag, 2006.
- [PN12] H. Plötz and K. Nohl. Peeling away layers of an RFID security system. In *16th International Conference on Financial Cryptography and Data Security (FC 2012)*, volume 7035 of *Lecture Notes in Computer Science*, pages 205–219. Springer-Verlag, 2012.
- [Pol12] PolyIC – The chip printers.  
<http://www.polyic.com>, 2012. (Retrieved on May 16).
- [PRI11] Private communication with ABN-AMRO, 2011.
- [RCT05] M. Rieback, B. Crispo, and A. Tanenbaum. RFID guardian: A battery-powered mobile device for RFID privacy management. In *10th Australasian Conference on Information Security and Privacy (ACISP 2005)*, volume 3574 of *Lecture Notes in Computer Science*. Springer -Verlag, 2005.
- [RE10] W. Rankl and W. Effing. *Smart Card Handbook*. Wiley, 2010.
- [Reb12] RebelSim APDU Scanner, 2012.  
<http://rebelsimcard.com/network-sim-apdu-scanner.html>.
- [Repa] The Proxmark repository.  
<https://code.google.com/p/proxmark3>.
- [Repb] The SmartLogic repository.  
<https://code.google.com/p/smartlogictool>.
- [RHU08] Counter Expertise Review of the TNO Security Analysis of the Dutch OV-chipkaart, 2008.
- [Rie08] M. Rieback. *Security and privacy of radio frequency identification*. Vrije Universiteit, 2008. Dissertation.
- [Rob03] S. Robinson. Still guarding secrets after years of attacks, RSA earns accolades for its founders. *SIAM News*, 36(5):1–4, 2003.
- [RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [RSH<sup>+</sup>12] A. Rahmati, M. Salajegheh, D. Holcomb, J. Sorber, W. P. Burleson, and K. Fu. TARDIS: Time and remanence decay in SRAM to implement secure protocols on embedded devices without clocks. In *21st USENIX Security Symposium (USENIX Security 2012)*, pages 221–236. USENIX Association, 2012.
- [RSN<sup>+</sup>01] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications. *NIST Special Publication (800-22)*, 22:1–152, 2001.



- [SDVM<sup>+</sup>06] S. Steudel, S. De Vusser, K. Myny, M. Lenes, J. Genoe, and P. Heremans. Comparison of organic diode structures regarding high-frequency rectification behavior in radio-frequency identification tags. *Journal of applied physics*, 99:114519, 2006.
- [SEA12] Season3 smart card logger. <http://www.cardman.com/loggers.html>, 2012.
- [SFC<sup>+</sup>05] V. Subramanian, J. Frechet, P. Chang, D. Huang, J. Lee, S. Molesa, A. Murphy, D. Redinger, and S. Volkman. Progress toward development of all-printed RFID tags: Materials, processes, and devices. *Proceedings of the IEEE*, 93(7):1330–1338, 2005.
- [Sha49] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.
- [She04] Y. Sheffi. RFID and the innovation cycle. *International Journal of Logistics Management, The*, 15(1):1–10, 2004.
- [SHXZ11] S. Sun, L. Hu, Y. Xie, and X. Zeng. Cube cryptanalysis of Hitag2 stream cipher. In *10th International Conference on Cryptology and Network Security (CANS 2011)*, volume 7092 of *Lecture Notes in Computer Science*, pages 15–25. Springer-Verlag, 2011.
- [Smi93] J. Smith. AFC project in oslo. *Smart Card News Ltd.*, pages 106–108, June 1993.
- [Smi94] J. Smith. Mikron austria bid for world ticket card. *Smart Card News Ltd.*, pages 161–164, September 1994.
- [Smi95] J. Smith. Philips takes over mikron. *Smart Card News Ltd.*, pages 124–125, July 1995.
- [Smi96] J. Smith. Korean bus fare system expands. *Smart Card News Ltd.*, 5:166, 1996.
- [Smi97] J. Smith. French La Poste to use MIFARE. *Smart Card News Ltd.*, 6:105, 1997.
- [Smi04] J. Smith. Netherlands receives worlds first full contactless transport system. *Smart Card News Ltd.*, 13:2, 2004.
- [SNC09] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In *12th International Conference on Theory and Applications of Satisfiability Testing (SAT 2009)*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer-Verlag, 2009.
- [SSB11] A. Slowinska, T. Stancescu, and H. Bos. Howard: a dynamic excavator for reverse engineering data structures. In *18th Network and Distributed System Security Symposium (NDSS 2011)*, San Diego, CA, 2011. The Internet Society.
- [SSS<sup>+</sup>05] R. Sangoi, C. Smith, M. Seymour, J. Venkataraman, D. Clark, M. Kleper, and B. Kahn. Printing radio frequency identification (rfid) tag antennas using inks containing silver dispersions. *Journal of dispersion science and technology*, 25(4):513–521, 2005.
- [Tan09] W. Tan. Practical attacks on the MIFARE classic. Master’s thesis, Imperial College London, 2009.
- [TNO08] Security Analysis of the Dutch OV-Chipkaart, February 2008. Public excerpt of TNO report 34642.

- [Tre08] J. Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing (FORTEST 2008)*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer-Verlag, 2008.
- [Tsu06] G. Tsudik. YA-TRAP: Yet another trivial RFID authentication protocol. In *4th IEEE International Conference on Pervasive Computing and Communications (PerCom 2006)*, Pisa, Italy, March 2006. IEEE Computer Society.
- [Vau07] S. Vaudenay. On privacy models for RFID. In *13th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2007)*, volume 4833 of *Lecture Notes in Computer Science*, pages 68–87. Springer-Verlag, 2007.
- [VdKG09] R. Verdult and G. de Koning Gans. Proxmark.org - A Radio Frequency Identification tool. <http://www.proxmark.org>, 2009.
- [VdKGG12] R. Verdult, G. de Koning Gans, and F. D. Garcia. A toolbox for RFID protocol analysis. In *4th International EURASIP Workshop on RFID Technology (EURASIP RFID 2012)*. IEEE Computer Society, 2012.
- [Ver08a] R. Verdult. Proof of concept, cloning the OV-chip card. Technical report, 2008.
- [Ver08b] R. Verdult. Security analysis of RFID tags. Master’s thesis, Radboud University Nijmegen, 2008.
- [VGB12] R. Verdult, F. D. Garcia, and J. Balasch. Gone in 360 seconds: Hijacking with Hitag2. In *21st USENIX Security Symposium (USENIX Security 2012)*, pages 237–252. USENIX Association, 2012.
- [VK11] R. Verdult and F. Kooman. Practical attacks on NFC enabled cell phones. In *3rd International Workshop on Near Field Communication (NFC 2011)*, pages 77–82. IEEE Computer Society, 2011.
- [vN11] P. Štembera and M. Novotný. Breaking Hitag2 with reconfigurable hardware. In *14th Euromicro Conference on Digital System Design (DSD 2011)*, pages 558–563. IEEE Computer Society, 2011.
- [Wan06] R. Want. An introduction to RFID technology. *Pervasive Computing, IEEE*, 5(1):25–33, 2006.
- [WCO<sup>+</sup>07] S. Wanga, W. Chenb, C. Onga, L. Liuc, and Y. Chuangb. RFID applications in hospitals: a case study on a demonstration RFID project in a Taiwan hospital. *hospitals*, 8:33, 2007.
- [Wes12] J. Westhues. A test instrument for HF/LF RFID. <http://cq.cx/proxmark3.pl>, 2012. (Retrieved on May 16).
- [WNLY06] N. Wu, M. Nystrom, T. Lin, and H. Yu. Challenges to global RFID adoption. *Technovation*, 26(12):1317–1323, 2006.
- [WSRE04] S. A. Weis, S. E. Sarma, R. L. Rivest, and D. W. Engels. Security and privacy aspects of low-cost radio frequency identification systems. In *1st International Conference on Security in Pervasive Computing (SPC 2003)*, volume 2802 of *Lecture Notes in Computer Science*, pages 50–59. Springer-Verlag, 2004.

- [WSvRG<sup>+</sup>08] R. Wichers Schreur, P. van Rossum, F. D. Garcia, W. Teepe, J.-H. Hoepman, B. Jacobs, G. de Koning Gans, R. Verdult, R. Muijrrers, R. Kali, and V. Kali. Security flaw in MIFARE Classic. *Press release, Digital Security group, Radboud University Nijmegen, The Netherlands*, March 2008.
- [Yu07] S. Yu. RFID implementation and benefits in libraries. *The Electronic Library*, 25(1):54–64, 2007.
- [Zie] S. Ziegenbalg. ZTEX Hardware. <http://ztex.de>.
- [Zim80] H. Zimmermann. OSI Reference model—the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.



- access control, 20, 39, 65, 67, 70, 77, 99, 100, 104, 119, 120
- active RFID, 4
- adaptive progressive thresholding, 37
- ADC, 34
- algorithm
  - fortification, 115
  - key diversification, 10, 99–102, 104, 106, 118, 119, 125–128, 130
  - probabilistic polynomial-time, 140
- antenna circuit, 26, 31, 32
- anticollision, 33, 71, 73, 76, 106
- APDU, 41
- ARM, 33
- ASK, 30
- asymmetric-key crypto, *see* cryptography
- ATR, 41, 45, 60
- attack
  - brute force, 77, 93, 118, 119
  - card-only, 68, 94, 96, 97
  - Chipknip, 59, 60
  - chosen ciphertext, 142
  - chosen plaintext, 128
  - cloning, 39, 63, 66, 69, 95, 97
  - desynchronization, 139
  - DOS, 11, 139, 144
  - eavesdropping, 13, 38, 45, 47
  - EMV, 50
  - fault injection, 17
  - iClass Elite, 128
  - iClass Standard, 118, 124
  - internet banking, 56
  - key recovery, 102, 118, 124, 128
  - known plaintext, 77, 85
  - Man-in-the-Middle, 13, 50
    - active, 45
    - passive, 13
  - Mifare Classic, 94
  - privilege escalation, 124
  - reflection, 15
  - relay, 13, 59
  - replay, 15, 60, 79
  - side channel, 16
  - SIM, 62
  - skimming, 49
  - timing, 47
  - tracking, 10, 12, 17, 18, 147, 152
- authentication, 5, 9, 11, 51, 62, 73, 106, 137
- authenticity, 5, 9, 84, 99, 133
- barcode, 133
- baudrate, 45, 46
  - detection, 45
- block cipher, *see* cipher
- Caesar cipher, *see* cipher
- challenge-response, 14, 15, 51, 52, 74, 104
- Chipknip, 60
- chosen ciphertext attack, *see* attack
- cipher, 100
  - block, 6, 7, 101
  - Caesar, 5
  - stream, 6
- cipher feedback, 78, 86, 121
- cloning, *see* attack
- collision, 9, 101, 116
- collision resistance, 9
- communication
  - full-duplex, 30

- half-duplex, 30
  - simplex, 30
- confidentiality, 5, 11, 133
- cryptanalysis, 6, 68
- cryptography, 3, 5, 6, 130, 133, 136, 137
  - asymmetric-key, 6, 8
  - symmetric-key, 6, 65
- CRYPTO1 cipher, *see* Mifare Classic
  
- data authentication, *see* EMV
- decoding, 35, 36
- decryption, 6
- delay time, 60
- demodulation, 35, 36, 38
- DES, 1, 101, 102, 106–108, 118, 119, 125, 126, 129, 130
- desynchronization, 137
- distance bounding, 14
- DOS, 11, 139, 144
  
- e.dentifier2, 56
- EAS, 4
- eavesdropping, 45, 47
- electronic passport, 65, 133
- emulation, 47
- EMV, 49, 50
  - action codes, 51
  - attack, 52, 54
  - cardholder verification, 51
  - data authentication, 51
  - EMV-CAP, 56
- encoding
  - Manchester, 29
  - Miller, 28
  - Modified Miller, 28
  - non-return-to-zero, 27
- encryption, 6, 103, 106, 118, 119, 126, 129, 130
  - asymmetric, 8
- entropy, 74, 118, 124, 128, 129
- EPC, 4, 134
- ETU, 28
  
- Fibonacci generator, 121
- formal verification, 11, 130
- forward adversary, 142
- forward privacy, 136, 137, 139, 144, 150, 152
  - narrow-, 150
  - thin-, 152
- FPGA, 28, 32, 34, 43
- FSK, 30
- function
  - hash, 9, 137–139, 153
  - key fortification, 101, 116
  - MAC, 9, 51, 53, 125
  - XOR, 6, 7
  
- hash chain, 138
- hash function, *see* function
- HID application, 104, 118
- HID Global, 99, 100, 103, 104, 126, 130, 131
- high frequency, 4, 26, 35, 99
- Hitag2 cipher, 86–88, 90, 100
  
- iClass, 103
  - authentication protocol, 106
  - cipher, 121
  - commands, 104
  - Elite, 125
  - hash0* function, 114, 115
  - hash1* function, 126
  - hash2* function, 127
  - key diversification, 106, 118, 125, 126, 128
  - key recovery attack, 102, 118, 124, 128
  - key update, 105
  - MAC function, 122
  - Standard, 106
- identity, 3, 15, 16, 62, 101, 102, 106–108, 118, 119, 126, 128, 129, 137–141, 144, 145, 147
- IND-CCA, 142
- indistinguishability, 142, 143
- integrity, 5, 9, 11, 71, 99, 101, 133
- internet banking, 56
- ISO/IEC 14443, 25
- ISO/IEC 15693, 25
- ISO/IEC 7816, 41, 60
  
- key derivation, 107
- key diversification, *see* algorithm
- key fortification, *see* function
- key management, 100
- keystream mapping, 78, 79
- keystream recovery, 77
- known plaintext, 85

- Leon Device, *see* tool
- LFSR, 7, 74, 75, 84–89, 91–95, 121
  - rollback, 91
  - state recovery, 89
- libnfc*, 97
- low frequency, 4, 26, 32, 34, 86
- MAC, 9, 122, 123
- microcontroller, 33
- Mifare Classic, 70
  - attacks, 94
  - authentication protocol, 73, 83
  - commands, 73, 81
  - communication layer, 71
  - CRYPTO1 cipher, 38, 65–67, 69, 77, 85–88, 90
  - filter function, 87, 92
  - memory, 71
  - weaknesses, 89
- modulation, 29
  - analog, 29
  - digital, 29
- multiple sector authentication, 95
- narrow adversary, 142
- NFC, 99
- NXP Semiconductors, 28, 38, 65, 67, 68, 70, 71, 80, 81, 86
- Omnikey, 106, 107, 115, 116
- OOK, 29
- OpenPCD, *see* tool
- OSI model, 23, 24, 26
- OSK protocol, 136, 138
- OV-chipkaart, 65, 67–70, 81, 97
- Oyster card, 65, 67, 69
- passive RFID, 4, 16
- payload, 105, 125
- physical layer, 24, 26, 41
- pre-image, 9, 116–119
- pre-image resistance, 9
- privacy, 141
- privilege escalation, 124
- proprietary, 2, 12, 20, 21, 56, 60, 65, 66, 69, 71, 97, 99, 100, 102, 118, 130
- protocol, 144
  - anticollision, 73, 76, 106
  - authentication, 73, 100, 106, 137
  - communication, 25
  - EMV, 51
  - smart card, 18, 19, 25
- Proxmark, 31
- pseudo-random number generator, 7, 66, 69, 74, 77, 78, 96, 106
- PSK, 31
- public transport ticketing, 2, 10, 65, 67, 70, 77, 81, 97, 133
- public-key cryptography, *see* asymmetric-key crypto
- randomness, 6, 7
- reverse engineering, 20, 21, 68, 69, 77, 83, 97, 100–103, 106, 107, 109, 115, 118–120
- RFID, 3
- RFID Guardian, *see* tool
- sampling, 35, 36
- Season3, *see* tool
- second channel*, 139, 144
- sector zero, 80
- security, 10
  - goals
    - authenticity, *see* authenticity, 11
    - availability, 11
    - confidentiality, 11
    - integrity, 11, 99
    - mutual authentication, 99
    - non-repudiation, 11
    - privacy, 11
- security protocol, 10–12
  - verification, 12
- session key, 63
- signature, 9, 51, 53
- SIM, 62
  - sharing, 62
- smart card, 2–5, 7, 8, 12, 18, 24, 25, 40–46, 50, 51, 56, 58–60, 62, 63
  - contact-based, 3, 18, 19, 23, 25–27, 40, 46
  - contactless, 2–4, 12, 17–20, 25–27, 46, 99, 100, 133
  - emulation, 60
  - sharing, 44, 62
- Smart Card Detective, 46
- SmartLogic, *see* tool

- software, 44
- stream cipher, *see* cipher
- supply chain, 133–136, 140
  
- thin adversary, 142, 152
- tool
  - Ghost, 39
  - IDA Pro, 120
  - Leon Device, 46
  - MPLAB, 120
  - OpenPCD, 38
  - Osmocom SIMtrace, 45
  - Proxmark, 31
  - RebelSim, 45
  - RFID Guardian, 39
  - Season3, 46
  - Smart Card Detective, 46
  - SmartLogic, 40, 52, 60
- tracking, 10, 12, 17, 18, 147, 152
- transaction, 49, 51–54, 57, 59, 62, 69, 75, 77, 78, 80, 97
  - counter, 49, 61
  - EMV, 56
  - off-line, 49
  - online, 52
  
- ultra-high frequency, 4
  
- XOR, *see* function
  
- ZTEX board, 25, 41–45



---

# Abbreviations

<b>ABS</b>	Anti-lock Braking System
<b>AAC</b>	Application Authentication Cryptogram
<b>AC</b>	Access Conditions
<b>AC</b>	Application Cryptogram
<b>ADC</b>	Analog-to-Digital Converter
<b>AES</b>	Advanced Encryption Standard
<b>AFC</b>	Automatic Fare Collection
<b>AM</b>	Amplitude Modulation
<b>APDU</b>	Application Protocol Data Unit
<b>ARM</b>	Advanced RISC Machine
<b>ARQC</b>	Authorization Request Cryptogram
<b>ASK</b>	Amplitude Shift Keying
<b>ATM</b>	Automated Teller Machine
<b>ATR</b>	Answer-to-Reset
<b>BCC</b>	Bit Count Check
<b>BPSK</b>	Binary Phase Shift Keying
<b>CAP</b>	Chip Authentication Program
<b>CIA</b>	Confidentiality-Integrity-Availability
<b>CV</b>	Cardholder Verification
<b>CVM</b>	Cardholder Verification Method
<b>DES</b>	Data Encryption Standard
<b>DDA</b>	Dynamic Data Authentication
<b>DDOS</b>	Distributed Denial-of-Service
<b>DOS</b>	Denial-of-Service
<b>DSP</b>	Digital Signal Processing

---

<b>EAS</b>	Electronic Article Surveillance
<b>ECC</b>	Elliptic Curve Cryptography
<b>ECU</b>	Engine Control Unit
<b>EEPROM</b>	Electrically Erasable Programmable Read-Only Memory
<b>EMV</b>	Europay, Mastercard and VISA
<b>EOF</b>	End-of-Frame
<b>EPC</b>	Electronic Product Code
<b>ETU</b>	Elementary Time Unit
<b>FM</b>	Frequency Modulation
<b>FPGA</b>	Field Programmable Gate Array
<b>FSK</b>	Frequency Shift Keying
<b>GSM</b>	Global System for Mobile communications (originally Groupe Spécial Mobile)
<b>HID</b>	Human Interface Device
<b>HF</b>	High Frequency
<b>IC</b>	Integrated Circuit
<b>ICC</b>	Integrated Circuit Card
<b>IP</b>	Internet Protocol
<b>JCB</b>	Japan Credit Bureau
<b>LF</b>	Low Frequency
<b>LFSR</b>	Linear Feedback Shift Register
<b>LSB</b>	Least Significant Bit
<b>MAC</b>	Message Authentication Code
<b>MitM</b>	Man-in-the-Middle
<b>MSB</b>	Most Significant Bit
<b>NFC</b>	Near Field Communication
<b>OOK</b>	On-Off Keying
<b>OSI</b>	Open Systems Interconnection
<b>OSK</b>	Ohkubo-Suzuki-Kinoshita
<b>PCD</b>	Proximity Coupling Device
<b>PICC</b>	Proximity Integrated Circuit Card
<b>PIN</b>	Personal Identification Number
<b>POS</b>	Point-of-Sale
<b>PSK</b>	Phase Shift Keying
<b>RFID</b>	Radio Frequency Identification
<b>RISC</b>	Reduced Instruction Set Computing
<b>RSA</b>	Rivest-Shamir-Adleman

---

<b>RTT</b>	Round-Trip Time
<b>SAM</b>	Secure Application Module
<b>SIM</b>	Subscriber Identity Module
<b>SOF</b>	Start-of-Frame
<b>SSL</b>	Secure Socket Layer
<b>SSP</b>	Simple Serial Protocol
<b>TCP</b>	Transmission Control Protocol
<b>UART</b>	Universal Asynchronous Receiver Transmitter
<b>UHF</b>	Ultra High Frequency
<b>UID</b>	Unique Identifier
<b>USB</b>	Universal Serial Bus
<b>VHDL</b>	VHSIC Hardware Description Language
<b>VHSIC</b>	Very-High-Speed Integrated Circuit
<b>WT</b>	Waiting Time
<b>WYSIWYS</b>	What-you-see-is-what-you-sign
<b>XOR</b>	Exclusive or



---

# Samenvatting

Dit proefschrift behandelt de wijze waarop smartcards worden geïmplementeerd in hedendaagse systemen als het openbaar vervoer, toegangsbeveiliging en internetbankieren. Het onderzoek richt zich voornamelijk op de digitale beveiligingsmechanismen die hier worden toegepast. Een smartcard, letterlijk vertaald 'slimme kaart', bevat een chip die kan worden gezien als een volwaardige kleine computer met een processor, werkgeheugen en opslagruimte om gegevens langer te bewaren. Enkel de randapparatuur zoals een beeldscherm en toetsenbord ontbreekt.

We kennen in de eerste plaats smartcards die contact-gebaseerd zijn, dat wil zeggen dat de kaart is voorzien van contactpunten die fysiek in verbinding kunnen worden gebracht met een kaartlezer. Daarnaast kennen we ook smartcards die contactloos een verbinding met een kaartlezer kunnen aangaan. Deze zogenaamde contactloze kaarten zijn uitgerust met RFID technologie. RFID staat voor Radio Frequency Identification, wat eigenlijk wil zeggen dat identificatie mogelijk wordt gemaakt met behulp van radiogolven. Deze communicatie tussen de kaartlezer en de kaart van zowel de contactloze als de contact-gebaseerde kaart moet, afhankelijk van de eisen die worden gesteld, afdoende beveiliging bieden.

In het eerste hoofdstuk worden enkele van deze belangrijke veiligheidseisen, ook wel security goals genoemd, beschreven. Security goals zijn doelen die worden gesteld waaraan de beveiliging van een systeem moet voldoen. Data integriteit, het voorkomen dat gegevens al dan niet intentioneel worden veranderd, kan een dergelijk doel zijn. Hiertoe moeten onlogische veranderingen in deze gegevens kunnen worden gedetecteerd, en eventueel gecorrigeerd. Security goals als deze kunnen worden bereikt met behulp van beveiligingstechnieken waarvan er een aantal in hoofdstuk 1 worden geïntroduceerd. Er zijn vervolgens vele manieren om te toetsen of de doelstellingen worden bereikt. Dit kan bijvoorbeeld door de specificatie en de implementatie van een systeem te formaliseren in een wiskundig model. Vervolgens kan met behulp van formele methoden worden geverifieerd of de implementatie voldoet aan de specificatie. De schaduwzijde van deze aanpak is dat het in veel gevallen nog steeds gaat om een geïdealiseerde beschrijving van de implementatie. Wanneer de implementatie vervolgens wordt vertaald naar een productieomgeving is de kans aanwezig dat er toch nog fouten worden geïntroduceerd. Het is van belang te weten welke fouten er kunnen optreden. Het is in de beveiligingswereld dan ook niet ongebruikelijk om te redeneren vanuit het perspectief van een aanvaller. Hoofdstuk 1 sluit daarom af met een beschrijving van een aantal veel voorkomende aanvalsscenario's.

Hoofdstuk 2 beschrijft twee tools die onderzoekers in staat stellen om de communicatie tussen kaartlezers en kaarten grondig te bestuderen en mogelijke aanvalsscenario's na te bootsen. Het gaat hier om de Proxmark voor contactloze kaarten en de SmartLogic voor contact-

gebaseerde kaarten. In beide gevallen is het mogelijk om de communicatie volledig te controleren. Dat betekent onder andere dat de communicatie tot op bit-niveau kan worden geregeld. De implementatie van de signaalverwerking en de aansturing van de hogere lagen in de communicatie worden hier toegelicht. De bijdrage in dit hoofdstuk ligt voornamelijk in het bieden van een goede infrastructuur waarmee de verdere onderzoeken die later in dit proefschrift aan bod komen sterk konden worden gefaciliteerd. De beschreven tools en bijbehorende software zijn open source en beschikbaar in het publieke domein.

In hoofdstuk 3 laten we een aantal praktijkvoorbeelden zien waarin de SmartLogic is ingezet om de communicatie tussen contact-gebaseerde kaarten te bestuderen. Een prominent voorbeeld is het gebruik van de chip in betaalpassen in plaats van het gebruik van de magneetstrip. De zogenaamde EMV chip<sup>1</sup> is eind 2011 ook in heel Nederland doorgevoerd en wordt gepromoot als “Het nieuwe pinnen”. Andere landen die al gebruik maakten van EMV betalingen zijn onder andere Groot-Brittannië, Ierland en België. Het gebruik van de chip maakt het mogelijk om gegevens versleuteld uit te wisselen en gegevens beter te beschermen tegen ongeoorloofd kopiëren. In maart 2011 toonden onderzoekers<sup>2</sup> in Groot-Brittannië echter aan [BBLF11] dat het nog steeds mogelijk was om de pincode te achterhalen door een betaalautomaat te overtuigen dat de betreffende kaart geen versleuteling ondersteunde. Dit experiment hebben wij ook in Nederland herhaald [dKGdR12] met behulp van de SmartLogic. Ook hier kon de pincode worden achterhaald, zij het dat de transactie vervolgens wel werd geblokkeerd door de bank. Toch was dit een fout in de beveiliging die niet had mogen optreden, de pincode moet namelijk geheim blijven en had versleuteld moeten worden verzonden. Naar aanleiding van deze bevindingen is de software van de betreffende betaalautomaten in Nederland bijgewerkt. Dit hoofdstuk behandelt verder ook het delen van één simkaart met meerdere telefoons en het kopen van snoep met een Chipknip die zich 20 kilometer van de snoepautomaat bevindt. Ten slotte kijken we naar het internetbankieren met behulp van een USB-kabel. Ook hier treffen we een fout aan in de implementatie.

De laatste drie hoofdstukken richten zich volledig op contactloze kaarten, oftewel RFID kaarten. Hoofdstuk 4 gaat over de Mifare Classic. Tot 2007 was dit een naam die vooral toeleveranciers en system integrators bekend was. CRYPTO1 is het beveiligingsmechanisme, of cryptografisch algoritme, waarvan de Mifare Classic gebruik maakt en was destijds nog zo onbekend dat de enige relevante verwijzing van de zoekmachine Google verwees naar een forumdiscussie over de geslotenheid van het algoritme. Die geslotenheid bestond uit het feit dat de fabrikant<sup>3</sup> geheim hield hoe het beveiligingsmechanisme CRYPTO1 daadwerkelijk in elkaar stak. Hoofdstuk 4 begint met een korte historische inleiding op de Mifare Classic. Een kaart die al in 1994 op de markt kwam en waarvan het gebruik in al die jaren een indrukwekkende groei doormaakte. Uiteindelijk heeft de Mifare Classic zelfs het grootste marktaandeel voor contactloze kaarten veroverd. De kaart is, in verhouding tot andere kaarten die ook beveiligingsmechanismen kennen, vele malen goedkoper. De kostprijs is vaak doorslaggevend in grote projecten waar contactloze kaarten worden gebruikt, zoals in de toegangsbeveiliging. Zo ook in het project dat in 2004 werd opgezet in Nederland om het eerste nationale openbaar vervoersnetwerk te creëren waar met één kaart kon worden gereisd. Een heel ambitieus project met een kaart die we nu kennen als de OV-chipkaart. Toen de kaart nog moest worden uitgerold en alleen nog in testfase werd gebruikt in Rotterdam werd al duidelijk dat het beveiligingsmechanisme CRYPTO1 grote fouten bevatte. Die fouten waren

---

<sup>1</sup>EMV staat voor Eurocard, Mastercard en Visa.

<sup>2</sup>Andrea Barisani en Daniele Bianco.

<sup>3</sup>NXP Semiconductors, voorheen Philips.

dermate ernstig dat men niet meer kon spreken van een afdoende beveiligd systeem. Hoofdstuk 4 beschrijft alle technische details van de Mifare Classic en CRYPTO1. In het hoofdstuk worden de eerste praktisch uitvoerbare aanvallen uitgewerkt die op de Mifare Classic mogelijk zijn. Wat begint bij het ongeautoriseerd uitlezen van een beperkt stukje geheugen van de kaart (zonder enige kennis van geheime sleutels of CRYPTO1), eindigt bij een aanval waarbij alle sleutels kunnen worden achterhaald en het complete geheugen binnen een minuut kan worden uitgelezen. De Proxmark, geïntroduceerd in hoofdstuk 2, heeft het mogelijk gemaakt om alle aanvallen ook daadwerkelijk praktisch uit te voeren en te verifiëren, hetgeen een onmisbare overtuigingskracht heeft gehad in de communicatie met de fabrikant en de overheid.

In hoofdstuk 5 wordt de iClass kaart behandeld. De fabrikant<sup>4</sup> van de iClass prijst dit product aan als een goede vervanger van de Mifare Classic. De veel duurdere iClass lijkt in eerste opzicht inderdaad betere bescherming te bieden door het gebruik van een beveiligingsmechanisme (DES) wat al sinds 1977 door vele specialisten en onderzoekers kritisch wordt bekeken. Dit mechanisme wordt overigens alleen gebruikt voor het afleiden van geheime sleutels. Het iClass systeem kent een geheime procedure waarin individuele kaartsleutels worden afgeleid van één hoofdsleutel. Deze procedure wordt ook wel sleutel diversificatie genoemd. iClass is beschikbaar in twee soorten: iClass Standard en iClass Elite. Het verschil tussen deze systemen ligt hoofdzakelijk in de wijze waarop kaartsleutels worden afgeleid van een hoofdsleutel. Naast deze sleutel diversificatie procedures is de iClass ook uitgerust met een beveiligingsmechanisme dat zorg draagt voor de wederzijdse authenticatie tussen de kaart en de kaartlezer. De werking van dit beveiligingsmechanisme werd door de fabrikant geheim gehouden. In dit hoofdstuk worden verschillende technieken gebruikt om de geheime procedures te ontrafelen. De eerste sleutel diversificatie procedure van de iClass Standard is ontrafeld door steeds wisselende invoer te geven en vervolgens de bijbehorende uitvoer te bestuderen. Uiteindelijk konden, deels geautomatiseerd, patronen in kaart worden gebracht die de relaties tussen invoer en uitvoer bloot legden. Hierbij werd gebruik gemaakt van speciaal voor de Proxmark ontwikkelde signaalverwerkingssoftware die het mogelijk maakt om alle communicatie van zowel iClass kaarten als lezers te genereren of op te vangen. Een tweede techniek waarbij de overige geheime procedures werden teruggehaald is gefundeerd op het analyseren van de machinecode uit iClass leesapparatuur. Dit werk resulteerde uiteindelijk in een reconstructie van het gehele iClass systeem. Eenmaal gereconstrueerd is het mogelijk om het systeem grondig te analyseren op eventuele beveiligingslekken. Hoofdstuk 5 beschrijft al de gevonden zwakheden en bevat ook drie aanvallen die gebruik maken van deze zwakheden. Ook in dit geval zijn al deze aanvallen geverifieerd in de praktijk met behulp van de Proxmark.

Naast beveiliging is privacy ook een belangrijk doel dat in sommige toepassingen moet worden nagestreefd, denk bijvoorbeeld aan het elektronisch stemmen. Het stemgeheim dat we kennen in Nederland vereist dat het in een elektronisch systeem niet terug te halen is wat iemand heeft gestemd. Toch moeten alle stemmen kunnen worden geteld. Er bestaan methoden om dit te realiseren. Ook het gebruik van RFID roept automatisch vragen op rond privacy. Het laatste hoofdstuk gaat in op privacy aspecten van RFID en richt zich, anders dan de voorgaande hoofdstukken, niet op RFID kaarten maar op RFID labels. Dit zijn labels in bijvoorbeeld de vorm van stickers waarin een kleine RFID chip met een antenne is verwerkt. RFID labels worden steeds meer gebruikt in productieketens om producten op een efficiënte manier te kunnen volgen. Omdat het om grote hoeveelheden labels gaat en omdat ze ook

---

<sup>4</sup>HID Global.

worden gebruikt op goedkope producten is het belangrijk dat deze labels zelf een zo laag mogelijke kostprijs hebben. Over het algemeen mogen ze niet duurder worden dan enkele eurocenten. Een gevolg hiervan is dat er nauwelijks tot geen beveiligingsmechanismen worden gebruikt, omdat die de chip in het RFID label te complex en dus te duur maken. Al die RFID labels die in en op producten worden gebruikt hebben wel een nadelig bijeffect. De unieke identificeerbaarheid, een eigenschap die ze per definitie hebben, maakt dat een persoon die deze kleding draagt, of producten met zich voert die zijn voorzien van deze labels, buiten zijn weten automatisch volgbaar is. Oplossingen waarbij deze labels steeds wisselende identificatie nummers uitzenden lijken de privacy te waarborgen. Een nadelig effect van deze oplossingen is het desynchronisatieprobleem. Dit is een probleem waarbij de eigenaar van het label, bijvoorbeeld een fabrikant, het label zelf ook niet meer kan identificeren. In dit hoofdstuk maken we gebruik van de omstandigheid dat de traditionele barcode vaak alsnog op het product wordt gedrukt. De informatie van de barcode gebruiken we om identificatie in het geval van desynchronisatie weer mogelijk te maken en de backend weer te synchroniseren met het RFID label. We stellen een mechanisme voor dat het met behulp van barcodes mogelijk moet maken om labels altijd weer te kunnen synchroniseren met de backend. Op deze manier kan dus de privacy van eindgebruikers worden gewaarborgd terwijl de productidentificatie met RFID labels mogelijk blijft.

Dit proefschrift beoogt zo een bijdrage te leveren aan de noodzakelijke beveiliging van toepassingen die gebruik maken van smartcards en RFID. Tevens toont het ook aan dat de wetenschap een onmisbare partner is van de overheid om op verantwoorde wijze de samenleving in het digitale tijdperk te dienen en te beschermen.



---

## Curriculum vitae

Gerhard de Koning Gans was born on the 1st of June, 1983, in Zwolle, the Netherlands. In 2000, he started his studies in computer science at Windesheim University of Applied Sciences in Zwolle. After he obtained his bachelor's degree in 2004, he continued his master's degree at the Radboud University in Nijmegen. Gerhard followed the security track of the computer science program and obtained his master's degree, cum laude, in 2008. His thesis entitled "Analysis of the MIFARE Classic used in the OV-chipkaart project" has been awarded with the *Aia Software Master Award*.

In September 2008, Gerhard started as a Ph.D. student in the Digital Security group of the Radboud University in Nijmegen where he was supervised by prof.dr. Bart Jacobs and dr. Flavio Garcia. His research subject was mainly security and privacy in RFID. Since October 2012, Gerhard works as a digital expert at Team High Tech Crime (THTC), a Dutch police department for digital crime investigation.



## Titles in the IPA Dissertation Series since 2007

- H.A. de Jong.** *Flexible Heterogeneous Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01
- N.K. Kavaldjiev.** *A run-time reconfigurable Network-on-Chip for streaming DSP applications.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02
- M. van Veelen.** *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems.* Faculty of Mathematics and Computing Sciences, RUG. 2007-03
- T.D. Vu.** *Semantics and Applications of Process and Program Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04
- L. Brandán Briones.** *Theories for Model-based Testing: Real-time and Coverage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05
- I. Loeb.** *Natural Deduction: Sharing by Presentation.* Faculty of Science, Mathematics and Computer Science, RU. 2007-06
- M.W.A. Streppel.** *Multifunctional Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2007-07
- N. Trčka.** *Silent Steps in Transition Systems and Markov Chains.* Faculty of Mathematics and Computer Science, TU/e. 2007-08
- R. Brinkman.** *Searching in encrypted data.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09
- A. van Weelden.** *Putting types to good use.* Faculty of Science, Mathematics and Computer Science, RU. 2007-10
- J.A.R. Noppen.** *Imperfect Information in Software Development Processes.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11
- R. Boumen.** *Integration and Test plans for Complex Manufacturing Systems.* Faculty of Mechanical Engineering, TU/e. 2007-12
- A.J. Wijs.** *What to do Next?: Analysing and Optimising System Behaviour in Time.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13
- C.F.J. Lange.** *Assessing and Improving the Quality of Modeling: A Series of Empirical Studies about the UML.* Faculty of Mathematics and Computer Science, TU/e. 2007-14
- T. van der Storm.** *Component-based Configuration, Integration and Delivery.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-15
- B.S. Graaf.** *Model-Driven Evolution of Software Architectures.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16
- A.H.J. Mathijssen.** *Logical Calculi for Reasoning with Binding.* Faculty of Mathematics and Computer Science, TU/e. 2007-17
- D. Jarnikov.** *QoS framework for Video Streaming in Home Networks.* Faculty of Mathematics and Computer Science, TU/e. 2007-18
- M. A. Abam.** *New Data Structures and Algorithms for Mobile Data.* Faculty of Mathematics and Computer Science, TU/e. 2007-19
- W. Pieters.** *La Volonté Machinale: Understanding the Electronic Voting Controversy.* Faculty of Science, Mathematics and Computer Science, RU. 2008-01
- A.L. de Groot.** *Practical Automaton Proofs in PVS.* Faculty of Science, Mathematics and Computer Science, RU. 2008-02
- M. Bruntink.** *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03
- A.M. Marin.** *An Integrated System to Manage Crosscutting Concerns in Source Code.* Faculty

of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

**N.C.W.M. Braspenning.** *Model-based Integration and Testing of High-tech Multi-disciplinary Systems.* Faculty of Mechanical Engineering, TU/e. 2008-05

**M. Bravenboer.** *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates.* Faculty of Science, UU. 2008-06

**M. Torabi Dashti.** *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

**I.S.M. de Jong.** *Integration and Test Strategies for Complex Manufacturing Machines.* Faculty of Mechanical Engineering, TU/e. 2008-08

**I. Hasuo.** *Tracing Anonymity with Coalgebras.* Faculty of Science, Mathematics and Computer Science, RU. 2008-09

**L.G.W.A. Cleophas.** *Tree Algorithms: Two Taxonomies and a Toolkit.* Faculty of Mathematics and Computer Science, TU/e. 2008-10

**I.S. Zapreev.** *Model Checking Markov Chains: Techniques and Tools.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

**M. Farshi.** *A Theoretical and Experimental Study of Geometric Networks.* Faculty of Mathematics and Computer Science, TU/e. 2008-12

**G. Gulesir.** *Evolvable Behavior Specifications Using Context-Sensitive Wildcards.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

**F.D. Garcia.** *Formal and Computational Cryptography: Protocols, Hashes and Commitments.* Faculty of Science, Mathematics and Computer Science, RU. 2008-14

**P. E. A. Dürr.** *Resource-based Verification for Robust Composition of Aspects.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

**E.M. Bortnik.** *Formal Methods in Support of SMC Design.* Faculty of Mechanical Engineering, TU/e. 2008-16

**R.H. Mak.** *Design and Performance Analysis of Data-Independent Stream Processing Systems.* Faculty of Mathematics and Computer Science, TU/e. 2008-17

**M. van der Horst.** *Scalable Block Processing Algorithms.* Faculty of Mathematics and Computer Science, TU/e. 2008-18

**C.M. Gray.** *Algorithms for Fat Objects: Decompositions and Applications.* Faculty of Mathematics and Computer Science, TU/e. 2008-19

**J.R. Calamé.** *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

**E. Mumford.** *Drawing Graphs for Cartographic Applications.* Faculty of Mathematics and Computer Science, TU/e. 2008-21

**E.H. de Graaf.** *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation.* Faculty of Mathematics and Natural Sciences, UL. 2008-22

**R. Brijder.** *Models of Natural Computation: Gene Assembly and Membrane Systems.* Faculty of Mathematics and Natural Sciences, UL. 2008-23

**A. Koprowski.** *Termination of Rewriting and Its Certification.* Faculty of Mathematics and Computer Science, TU/e. 2008-24

**U. Khadim.** *Process Algebras for Hybrid Systems: Comparison and Development.* Faculty of Mathematics and Computer Science, TU/e. 2008-25

- J. Markovski.** *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26
- H. Kastenbergh.** *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27
- I.R. Buhan.** *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28
- R.S. Marin-Perianu.** *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29
- M.H.G. Verhoef.** *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01
- M. de Mol.** *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02
- M. Lormans.** *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03
- M.P.W.J. van Osch.** *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04
- H. Sozer.** *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05
- M.J. van Weerdenburg.** *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06
- H.H. Hansen.** *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07
- A. Mesbah.** *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08
- A.L. Rodriguez Yakushev.** *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9
- K.R. Olmos Joffré.** *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10
- J.A.G.M. van den Berg.** *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11
- M.G. Khatib.** *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12
- S.G.M. Cornelissen.** *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13
- D. Bolzoni.** *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14
- H.L. Jonker.** *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15
- M.R. Czenko.** *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16
- T. Chen.** *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

- C. Kaliszyk.** *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web.* Faculty of Science, Mathematics and Computer Science, RU. 2009-18
- R.S.S. O'Connor.** *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory.* Faculty of Science, Mathematics and Computer Science, RU. 2009-19
- B. Ploeger.** *Improved Verification Methods for Concurrent Systems.* Faculty of Mathematics and Computer Science, TU/e. 2009-20
- T. Han.** *Diagnosis, Synthesis and Analysis of Probabilistic Models.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21
- R. Li.** *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis.* Faculty of Mathematics and Natural Sciences, UL. 2009-22
- J.H.P. Kwisthout.** *The Computational Complexity of Probabilistic Networks.* Faculty of Science, UU. 2009-23
- T.K. Cocx.** *Algorithmic Tools for Data-Oriented Law Enforcement.* Faculty of Mathematics and Natural Sciences, UL. 2009-24
- A.I. Baars.** *Embedded Compilers.* Faculty of Science, UU. 2009-25
- M.A.C. Dekker.** *Flexible Access Control for Dynamic Collaborative Environments.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26
- J.F.J. Laros.** *Metrics and Visualisation for Crime Analysis and Genomics.* Faculty of Mathematics and Natural Sciences, UL. 2009-27
- C.J. Boogerd.** *Focusing Automatic Code Inspections.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01
- M.R. Neuhäuser.** *Model Checking Nondeterministic and Randomly Timed Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02
- J. Endrullis.** *Termination and Productivity.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03
- T. Staijen.** *Graph-Based Specification and Verification for Aspect-Oriented Languages.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04
- Y. Wang.** *Epistemic Modelling and Protocol Dynamics.* Faculty of Science, UvA. 2010-05
- J.K. Berendsen.** *Abstraction, Prices and Probability in Model Checking Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2010-06
- A. Nugroho.** *The Effects of UML Modeling on the Quality of Software.* Faculty of Mathematics and Natural Sciences, UL. 2010-07
- A. Silva.** *Kleene Coalgebra.* Faculty of Science, Mathematics and Computer Science, RU. 2010-08
- J.S. de Bruin.** *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications.* Faculty of Mathematics and Natural Sciences, UL. 2010-09
- D. Costa.** *Formal Models for Component Connectors.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10
- M.M. Jaghoori.** *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services.* Faculty of Mathematics and Natural Sciences, UL. 2010-11
- R. Bakhshi.** *Gossiping Models: Formal Analysis of Epidemic Protocols.* Faculty of Sciences, Department of Computer Science, VUA. 2011-01
- B.J. Arnoldus.** *An Illumination of the Template Enigma: Software Code Generation with Templates.* Faculty of Mathematics and Computer Science, TU/e. 2011-02
- E. Zambon.** *Towards Optimal IT Availability Planning: Methods and Tools.* Faculty of

Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

**L. Astefanoaei.** *An Executable Theory of Multi-Agent Systems Refinement.* Faculty of Mathematics and Natural Sciences, UL. 2011-04

**J. Proença.** *Synchronous coordination of distributed components.* Faculty of Mathematics and Natural Sciences, UL. 2011-05

**A. Morali.** *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

**M. van der Bijl.** *On changing models in Model-Based Testing.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

**C. Krause.** *Reconfigurable Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-08

**M.E. Andrés.** *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems.* Faculty of Science, Mathematics and Computer Science, RU. 2011-09

**M. Atif.** *Formal Modeling and Verification of Distributed Failure Detectors.* Faculty of Mathematics and Computer Science, TU/e. 2011-10

**P.J.A. van Tilburg.** *From Computability to Executability – A process-theoretic view on automata theory.* Faculty of Mathematics and Computer Science, TU/e. 2011-11

**Z. Protic.** *Configuration management for models: Generic methods for model comparison and model co-evolution.* Faculty of Mathematics and Computer Science, TU/e. 2011-12

**S. Georgievska.** *Probability and Hiding in Concurrent Processes.* Faculty of Mathematics and Computer Science, TU/e. 2011-13

**S. Malakuti.** *Event Composition Model: Achieving Naturalness in Runtime Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

**M. Raffelsieper.** *Cell Libraries and Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-15

**C.P. Tsirogiannis.** *Analysis of Flow and Visibility on Triangulated Terrains.* Faculty of Mathematics and Computer Science, TU/e. 2011-16

**Y.-J. Moon.** *Stochastic Models for Quality of Service of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-17

**R. Middelkoop.** *Capturing and Exploiting Abstract Views of States in OO Verification.* Faculty of Mathematics and Computer Science, TU/e. 2011-18

**M.F. van Amstel.** *Assessing and Improving the Quality of Model Transformations.* Faculty of Mathematics and Computer Science, TU/e. 2011-19

**A.N. Tamalet.** *Towards Correct Programs in Practice.* Faculty of Science, Mathematics and Computer Science, RU. 2011-20

**H.J.S. Basten.** *Ambiguity Detection for Programming Language Grammars.* Faculty of Science, UvA. 2011-21

**M. Izadi.** *Model Checking of Component Connectors.* Faculty of Mathematics and Natural Sciences, UL. 2011-22

**L.C.L. Kats.** *Building Blocks for Language Workbenches.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23

**S. Kemper.** *Modelling and Analysis of Real-Time Coordination Patterns.* Faculty of Mathematics and Natural Sciences, UL. 2011-24

**J. Wang.** *Spiking Neural P Systems.* Faculty of Mathematics and Natural Sciences, UL. 2011-25

- A. Khosravi.** *Optimal Geometric Data Structures.* Faculty of Mathematics and Computer Science, TU/e. 2012-01
- A. Middelkoop.** *Inference of Program Properties with Attribute Grammars, Revisited.* Faculty of Science, UU. 2012-02
- Z. Hemel.** *Methods and Techniques for the Design and Implementation of Domain-Specific Languages.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03
- T. Dimkov.** *Alignment of Organizational Security Policies: Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04
- S. Sedghi.** *Towards Provably Secure Efficiently Searchable Encryption.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05
- F. Heidarian Dehkordi.** *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference.* Faculty of Science, Mathematics and Computer Science, RU. 2012-06
- K. Verbeek.** *Algorithms for Cartographic Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2012-07
- D.E. Nadas Agut.** *A Compositional Interchange Format for Hybrid Systems: Design and Implementation.* Faculty of Mechanical Engineering, TU/e. 2012-08
- H. Rahmani.** *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2012-09
- S.D. Vermolen.** *Software Language Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10
- L.J.P. Engelen.** *From Napkin Sketches to Reliable Software.* Faculty of Mathematics and Computer Science, TU/e. 2012-11
- F.P.M. Stappers.** *Bridging Formal Models – An Engineering Perspective.* Faculty of Mathematics and Computer Science, TU/e. 2012-12
- W. Heijstek.** *Software Architecture Design in Global and Model-Centric Software Development.* Faculty of Mathematics and Natural Sciences, UL. 2012-13
- C. Kop.** *Higher Order Termination.* Faculty of Sciences, Department of Computer Science, VUA. 2012-14
- A. Osaiweran.** *Formal Development of Control Software in the Medical Systems Domain.* Faculty of Mathematics and Computer Science, TU/e. 2012-15
- W. Kuijper.** *Compositional Synthesis of Safety Controllers.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16
- H. Beohar.** *Refinement of Communication and States in Models of Embedded Systems.* Faculty of Mathematics and Computer Science, TU/e. 2013-01
- G. Igna.** *Performance Analysis of Real-Time Task Systems using Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2013-02
- E. Zambon.** *Abstract Graph Transformation – Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03
- B. Lijnse.** *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications.* Faculty of Science, Mathematics and Computer Science, RU. 2013-04
- G.T. de Koning Gans.** *Outsmarting Smart Cards.* Faculty of Science, Mathematics and Computer Science, RU. 2013-05