

The (in)security of proprietary cryptography

Roel Verdult

KU LEUVEN

Radboud University Nijmegen



Copyright © Roel Verdult, 2015

ISBN: 978-94-6259-622-1

IPA Dissertation Series: 2015-10

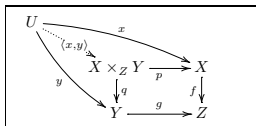
URL: http://roel.verdult.xyz/publications/phd_thesis-roel_verdult.pdf

Typeset using L^AT_EX

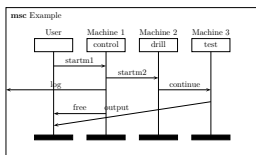


The work in this dissertation has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). For more information, visit

<http://www.win.tue.nl/ipa/>



X_Y-pic is used for typesetting graphs and diagrams in schematic representations of *logical composition of visual components*. X_Y-pic allows the style of pictures to match well with the exquisite quality of the surrounding T_EX typeset material [RM99]. For more information, visit <http://xy-pic.sourceforge.net/>



The message sequence diagrams, charts and protocols in this dissertation are facilitated by the MSC macro package [MB01, BvDKM13]. It allows L^AT_EX users to easily include Message Sequence Charts in their texts. For more information, visit

<http://satoss.uni.lu/software/mscpackage/>



The graphical art of this work 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/>

The (in)security of proprietary cryptography

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen op gezag van
de rector magnificus prof. dr. Th.L.M. Engelen
volgens besluit van het college van decanen

en

ter verkrijging van de graad van doctor in de ingenieurswetenschappen
aan de KU Leuven op gezag van
de rector prof. dr. R. Torfs,

in het openbaar te verdedigen op dinsdag 21 april 2015
om 14:30 uur precies

door

Roel Verdult

geboren op 20 oktober 1982
te Zevenaar, Nederland.

Promotoren:

Prof. dr. Bart Jacobs

Prof. dr. ir. Ingrid Verbauwhede
KU Leuven, België

Copromotoren:

Dr. Lejla Batina

Dr. Claudia Diaz Martinez
KU Leuven, België

Manuscriptcommissie:

Prof. dr. Eric Verheul

Dr. Jaap-Henk Hoepman

Prof. dr. Herbert Bos
Vrije Universiteit Amsterdam, Nederland

Prof. dr. Srdjan Čapkun
ETH Zurich, Zwitserland

Prof. dr. Thorsten Holz
Ruhr-Universität Bochum, Duitsland

KU Leuven Examencommissie:

Prof. dr. Wim Dehaene
KU Leuven, België

Prof. dr. Bart Preneel
KU Leuven, België

Prof. dr. Herman Neuckermans
KU Leuven, België

Prof. dr. Gildas Avoine
Université Catholique de Louvain, België

The (in)security of proprietary cryptography

DOCTORAL THESIS

to obtain the degree of doctor
from Radboud University Nijmegen on the authority of
the rector magnificus prof. dr. Th.L.M. Engelen
according to the decision of the Council of Deans

and

to obtain the degree of doctor of Engineering Science
from KU Leuven on the authority of
the rector prof. dr. R. Torfs,

to be defended in public on Tuesday, 21 April 2015
at exactly 14:30 hours

by

Roel Verdult

born in Zevenaar, Nederland
on 20 October 1982.

Supervisors:

Prof. dr. Bart Jacobs

Prof. dr. ir. Ingrid Verbauwhede
KU Leuven, Belgium

Co-supervisors:

Dr. Lejla Batina

Dr. Claudia Diaz Martinez
KU Leuven, Belgium

Doctoral Thesis Committee:

Prof. dr. Eric Verheul

Dr. Jaap-Henk Hoepman

Prof. dr. Herbert Bos
Vrije Universiteit Amsterdam, The Netherlands

Prof. dr. Srdjan Čapkun
ETH Zurich, Switzerland

Prof. dr. Thorsten Holz
Ruhr-Universität Bochum, Germany

KU Leuven Examination Board:

Prof. dr. Wim Dehaene
KU Leuven, Belgium

Prof. dr. Bart Preneel
KU Leuven, Belgium

Prof. dr. Herman Neuckermans
KU Leuven, Belgium

Prof. dr. Gildas Avoine
Université Catholique de Louvain, Belgium

A tribute to my dearest family

Irma, Twan and Sten

Acknowledgements

It has been a great honour to work with my direct colleagues at the Digital Security group of the Radboud University, who worked with me as co-authors, but most of all, as best friends. They inspired me to explore the path of science, which enabled me to write this doctoral dissertation. I am extremely grateful for their guidance, didactics, insights and support.

I would like to specially thank Flavio Garcia and Gerhard de Koning Gans, who have been working besides me from day one. Without their feedback and continuous support, I would not have realized my admiration for science and discovered the contribution that I have to offer. Furthermore, I'm grateful to my first promotor Bart Jacobs, who is an excellent example of a dedicated and outstanding scientist that is concurrently involved in society.

My gratitude goes to all the members of the Radboud University reading committee and KU Leuven examination board, Eric Verheul, Jaap-Henk Hoepman, Wim Dehaene, Bart Preneel, Herman Neuckermans, Gildas Avoine, Herbert Bos, Srdjan Čapkun and Thorsten Holz, who helped improving this doctoral dissertation a lot by their valuable and professional comments. Additionally, I would like to thank my friends and family, and especially my father Ad Verdult, for helping me improving the language of this thesis.

It was pleasant to work with my second promotor Ingrid Verbauwhede and her research group at the KU Leuven. Specifically, I would like to thank my direct colleague from Belgium, Josep Balasch. Besides our scientific collaboration, he also assisted me numerous times to comply with rules and administrative protocols of the KU Leuven.

During the security research that I performed with my colleagues, several friendly and prominent contacts were established. We collaborated extensively with governments, secret services, nation wide police forces and large corporations. However, despite our efforts to carefully disclose sensitive information in a responsible way, there were a few unfortunate events where we faced legal pressure that tried to re-

strain us from publishing the details of our work. During such an event we have always been firmly supported by the legal department and executive board of the Radboud University. My gratitude goes especially to Bas Kortmann, former rector magnificus of the Radboud University, and Dorine Gebbink, head of legal affairs.

Although it is impossible to name every person, I would like to show my gratitude to all the people who supported me in many ways over the last years. During this period I have acquired a lot of knowledge, skills and experience. I'm very thankful for that.

Finally, I would like to express my admiration, gratitude and love to my wife and children. They have always supported me during my research and the long days that I've spent writing of this thesis. I will be forever grateful for their love and care.

Roel Verdult
Oosterbeek, March 2015

Abstract

Proprietary cryptography is a term used to describe custom encryption techniques that are kept secret by its designers to add additional security. It is questionable if such an approach increases the cryptographic strength of the underlying mathematical algorithms. The security of proprietary encryption techniques relies entirely on the competence of the semi-conductor companies, which keep the technical description strictly confidential after designing. It is difficult to give a public and independent security assessment of the cryptography, without having access to the detailed information of the design.

Proprietary cryptography is currently deployed in many products which are used on a daily basis by the majority of people world-wide. It is embedded in the computational core of many wireless and contactless devices used in access control systems and vehicle immobilizers.

Contactless access control cards are used in various security systems. Examples include the use in public transport, payment terminals, office buildings and even in highly secure facilities such as ministries, banks, nuclear power plants and prisons. Many of these access control cards are based on proprietary encryption techniques. Prominent examples are the widely deployed contactless access control systems that use the *MIFARE Classic*, *iClass* and *Cryptomemory* technology.

A vehicle immobilizer is an electronic device that prevents the engine of the vehicle from starting when the corresponding transponder is not present. This transponder is a wireless radio frequency chip which is typically embedded in the plastic casing of the car key. When the driver tries to start the vehicle, the car authenticates the transponder before starting the engine, thus preventing hot-wiring. According to European Commission directive (95/56/EC) it is mandatory that all cars, sold in the EU from 1995 onwards, are fitted with an electronic immobilizer. In practice, almost all recently sold cars in Europe are protected by transponders that embed one of the two proprietary encryption techniques *Hitag2* or *Megamos Crypto*.

In this doctoral thesis well-known techniques are combined with novel methods

to analyze the workings of the previously mentioned proprietary cryptosystems. The cryptographic strength and security features of each system are comprehensively evaluated. The technical chapters describe various weaknesses and practical cryptanalytic attacks which can be mounted by an adversary that uses only ordinary and consumer grade hardware. This emphasizes the seriousness and relevance to the level of protection that is offered. The identified vulnerabilities are often plain design mistakes, which makes the cryptosystems exploitable since their introduction.

The first part of this dissertation is dedicated to an introduction of the general field of computer security and cryptography. It includes an extensive description of the theoretical background that refers to related literature and gives a summary of well-known cryptographic attack techniques. Additionally, a broad summary of related scientific research on proprietary cryptography is given. Finally, the technical part of this doctoral dissertation presents serious weaknesses in widely deployed proprietary cryptosystems, which are still actively used by billions of consumers in their daily lives.

Samenvatting (Dutch Summary)

De term *propriétaire cryptografie* beschrijft een eigengemaakte en specifiek ontworpen versleutelingstechniek die vaak geheim gehouden wordt met als doel hierdoor extra veiligheid te bemachtigen. Het is maar de vraag of een dergelijke aanpak de cryptografische sterkte van de onderliggende wiskundige algoritmes ten goede komt. De veiligheid van *propriétaire* versleutelingstechnieken rust volledig op de competentie van de semi-conductor bedrijven, die na het ontwerpen de technische omschrijvingen strikt geheim houden. Het is moeilijk om een publieke en onafhankelijke veiligheidsbeoordeling van de cryptografie te geven, zonder dat er gedetailleerde informatie beschikbaar is over het ontwerp.

Propriétaire cryptografie wordt momenteel gebruikt in een groot aantal producten die dagelijks gebruikt worden door de meerderheid van de wereldbevolking. Het is geïntegreerd in de rekenkern van verschillende draadloze en contactloze apparaten die toegepast worden in bijvoorbeeld toegangscontrolesystemen en startonderbrekers.

Contactloze toegangscontrolekaarten worden gebruikt in diverse beveiligingssystemen. Ze worden bijvoorbeeld gebruikt in het openbaar vervoer, betaalautomaten, bedrijfsgebouwen en zelfs in de beter beveiligde omgevingen zoals ministeries, banken, kernreactoren en gevangenissen. Veel van deze toegangspassen zijn gebaseerd op *propriétaire* versleutelingstechnieken. Prominente voorbeelden zijn de op grote schaal uitgerolde toegangscontrole systemen die gebruik maken van de *MIFARE Classic*, *iClass* and *Cryptomemory* technologie.

Een startonderbreker is een elektronisch apparaat dat het starten van een auto voorkomt wanneer de bijbehorende transponder niet in de buurt is. De transponder is een draadloze radiografische chip die doorgaans verwerkt zit in het plastic omhulsel van de autosleutel. Om autodiefstal tegen te gaan zal, wanneer de bestuurder de auto probeert te starten, de startonderbreker eerst de autosleutel proberen te authenticeren voordat de auto daadwerkelijk wordt gestart. Volgens de Europese richtlijn (95/56/EC) is het verplicht dat alle auto's, die zijn verkocht na 1995, worden uitgerust met een dergelijke startonderbreker. In praktijk zijn bijna alle recent verkochte auto's

in Europa beschermd met transponders die gebruik maken van één van de twee propriëtaire cryptografische algoritmes *Hitag2* of *Megamos Crypto*.

In dit proefschrift worden bestaande technieken gecombineerd met nieuwe methodes om de werking van de eerder genoemde propriëtaire cryptosystemen te analyseren. De cryptografische sterkte en veiligheid van elk systeem is uitgebreid geëvalueerd. De technische hoofdstukken beschrijven verschillende zwakheden en praktische cryptanalytische aanvallen die kwaadwillenden kunnen uitvoeren met behulp van doorsnee consumentenapparatuur. Dit benadrukt de ernst en de relevantie met betrekking tot de mate van bescherming die wordt geboden. De geïdentificeerde kwetsbaarheden zijn vaak simpele ontwerpfouten, waar de cryptosystemen al sinds hun introductie vatbaar voor zijn.

Het eerste deel van dit proefschrift is gewijd aan een algemene introductie van computerbeveiliging en cryptografie. Het bevat een uitgebreide beschrijving van de theoretische achtergrond die verwijst naar verwante literatuur en geeft een overzicht van bekende cryptografische aanvalstechnieken. Daarnaast wordt er een breed overzicht gegeven van gerelateerde wetenschappelijke onderzoeken met betrekking tot propriëtaire cryptografie. Ten slotte presenteert het technische gedeelte van dit proefschrift ernstige zwakheden in breed ingezette propriëtaire cryptosystemen, die nog steeds actief gebruikt worden door miljarden consumenten in hun dagelijks leven.

Contents

Acknowledgements	ix
Abstract	xi
Samenvatting (Dutch Summary)	xiii
1 Introduction	1
1.1 Introduction to digital security	2
1.2 Inspiration and motivation	4
1.3 Proprietary and secret cryptography	4
1.4 Scientific and social value	7
1.5 Responsible disclosure	8
1.6 Scope and focus	9
1.7 Outline, contributions and results	10
1.8 Notation	14
1.8.1 Mathematical symbols	14
1.8.2 Byte representation	14
I Theoretical background	15
2 Introduction to cryptography	17
2.1 History	17
2.1.1 Caesar substitution	18
2.1.2 Scytale transposition	19

2.2	Cryptosystems	20
2.2.1	Cipher specification	20
2.2.2	Cryptographic strength	21
2.2.3	Design methodology	22
2.2.4	Application and deployment	23
2.3	Cryptographic algorithms	24
2.3.1	Symmetric and asymmetric cryptography	24
2.3.2	Stream ciphers	24
2.3.3	Block ciphers	27
2.3.4	Hashes, MAC's and Digital signatures	28
2.4	Protocols and access conditions	29
2.4.1	Authentication	29
2.4.2	Authorization	31
2.5	Weaknesses in symmetric cryptosystems	32
2.5.1	Cipher design	33
2.5.2	Authentication protocol	33
2.5.3	Cipher initialization	34
2.5.4	Encryption oracle	35
2.5.5	Authorization model	35
2.5.6	Communication protocol	36
2.5.7	Implementation	37
2.5.8	Deployment	38
3	Attack scenarios	41
3.1	Authentication and communication attacks	41
3.1.1	Passive eavesdropping	42
3.1.2	Relay attack	42
3.1.3	Replay attack	45
3.1.4	Reflection attack	46
3.1.5	Reorder and inject messages	47
3.1.6	Block the communication	48
3.2	Cryptographic attacks	49
3.2.1	Malleability attack	50
3.2.2	Divide-and-conquer attack	51
3.2.3	Correlation attack	53
3.2.4	Guess-and-determine attack	55

3.2.5	Differential cryptanalysis	58
3.2.6	Algebraic attacks	60
3.2.7	Meet-in-the-middle attack	62
3.3	Physical attacks	65
3.3.1	Non-invasive attacks	65
3.3.2	Invasive attacks	66
3.3.3	Semi-invasive attacks	66
4	Scientific security assessments of proprietary cryptosystems	69
4.1	Access control systems and electronic locks	69
4.1.1	Legic	70
4.1.2	SimonsVoss G1	70
4.1.3	SimonsVoss G2	71
4.2	Electronic vehicle immobilizers	71
4.2.1	DST	72
4.2.2	KeeLoq	72
4.3	Cellular-, cordless- and sat-phones	73
4.3.1	ORYX	73
4.3.2	A3, A8 and COMP128	74
4.3.3	A5/1	74
4.3.4	A5/2	75
4.3.5	A5/3 (KASUMI)	76
4.3.6	DSAA	76
4.3.7	DSC	77
4.3.8	GMR-1	77
4.3.9	GMR-2	78
4.4	Various encryption techniques	78
4.4.1	CSS	79
4.4.2	E0	79
4.4.3	Skipjack	80
4.4.4	RC4	81
4.4.5	WEP	82
5	Introduction to Radio Frequency Identification (RFID)	83
5.1	Technology and application	83
5.1.1	Signal modulation and encoding	84

5.1.2	Communication standards	84
5.1.3	Near Field Communication (NFC)	85
5.1.4	Contactless smart cards	86
5.1.5	Proprietary cryptography in RFID devices	86
5.2	Research tools	87
5.2.1	GNURadio framework	88
5.2.2	Proxmark hardware device	89

II Technical section 91

6 MIFARE Classic 93

6.1	Introduction	93
6.2	Background	97
6.2.1	Hardware setup	97
6.2.2	Communication	97
6.2.3	Memory structure of the Mifare Classic	98
6.3	Reverse-engineering MIFARE Classic	98
6.3.1	Tag and reader authentication protocol	100
6.3.2	Initialization	101
6.3.3	Filter function	102
6.3.4	CRYPTO1	103
6.3.5	Rollback	105
6.4	Weaknesses	106
6.4.1	Odd Inputs to the Filter Function	106
6.4.2	Parity weaknesses	109
6.4.3	Nested authentications	110
6.4.4	Known Plaintext	111
6.5	Attacks	112
6.5.1	Brute-force attack	112
6.5.2	Varying the reader nonce	112
6.5.3	Varying the tag nonce	116
6.5.4	Nested authentication attack	118
6.6	Conclusions	120
6.7	Acknowledgements	121

7	Hitag2	123
7.1	Introduction	123
7.2	Hardware setup	128
7.3	Hitag2	130
7.3.1	Functionality	130
7.3.2	Memory	130
7.3.3	Communication	130
7.3.4	Cipher	132
7.3.5	Authentication protocol	133
7.3.6	Cipher Initialization	134
7.3.7	Rollback	134
7.4	Hitag2 weaknesses	135
7.4.1	Arbitrary length keystream oracle	135
7.4.2	Dependencies between sessions	136
7.4.3	Approximation of the filter function	137
7.5	Attacks	137
7.5.1	Malleability attack	138
7.5.2	Time/memory tradeoff attack	138
7.5.3	Cryptanalytic attack	139
7.6	Starting a car	141
7.7	Implementation weaknesses	142
7.7.1	Weak random number generators	142
7.7.2	Low entropy keys	143
7.7.3	Readable keys	144
7.7.4	Predictable transponder passwords	144
7.7.5	Identifier pickpocketing	144
7.8	Mitigation	145
7.9	Conclusions	146
7.10	Acknowledgments	147
8	SecureMemory, CryptoMemory and CryptoRF	149
8.1	Introduction	149
8.2	Background	152
8.3	The ciphers	152
8.3.1	Initialization and authentication	155
8.4	Attacking SecureMemory	158

8.4.1	Recovering the internal state	159
8.4.2	Unrolling the cipher	161
8.4.3	Recovering the key	161
8.4.4	Complexity and time	162
8.5	Attacking CryptoMemory	163
8.5.1	Recovering the internal state	163
8.5.2	Unrolling the cipher	165
8.5.3	Recovering the key	165
8.5.4	Complexity and time	166
8.6	Conclusion	167
9	iClass and iClass Elite	169
9.1	Introduction	169
9.2	Research context and related work	170
9.2.1	Research contribution	171
9.2.2	Outline	172
9.3	iClass	173
9.3.1	Functionality	173
9.3.2	Authentication protocol	175
9.4	iClass standard	176
9.4.1	Black box reverse engineering	176
9.4.2	The function <i>hash0</i>	182
9.4.3	Weaknesses in iClass Standard key diversification	184
9.4.4	Attacking iClass Standard key diversification	186
9.5	The iClass cipher	187
9.5.1	Firmware reverse engineering	187
9.5.2	The cipher	189
9.6	Weakness in iClass	191
9.6.1	Weak keys	191
9.6.2	XOR key update weakness	191
9.6.3	Privilege escalation	192
9.6.4	Lower card key entropy	192
9.6.5	Key recovery attack on iClass Standard	192
9.7	iClass Elite	194
9.7.1	Key diversification on iClass Elite	194
9.7.2	Weaknesses in iClass Elite key diversification	196

9.7.3	Key recovery attack on iClass Elite	197
9.8	Conclusion	198
9.9	Acknowledgments	199
10	Megamos Crypto	201
10.1	Disclaimer	201
10.2	Historical claim	201
III	Back matter	203
	Conclusion	205
	Index	209
	Acronyms	211
	List of figures	215
	Curriculum vitae	219
	Bibliography	221

Chapter 1

Introduction

It is hard to imagine a world without computers. Especially without small computers, called micro-controllers, which are embedded into a variety of modern devices. Well-known examples of such devices are electronic passports, contactless banking cards, access control tokens, car keys, phones, televisions and even small appliances like toasters. The general public uses such devices many times a day without considering the potential security and privacy risks they get exposed to.

During the last decade, many of these devices started using customly designed wireless communication interfaces. The ability to operate autonomously with remote access to all sorts of information enables several advanced features. However, the security and privacy risks of such systems increase with every additional feature. It is important to mitigate the risks and protect these wireless interfaces against malicious adversaries in the best possible way.

It is not trivial to design an algorithm (effective method expressed as a finite list of mathematical calculations) that establishes a secure communicate channel. Many experts, affiliated with top universities, leading industries and influential governments proposed several cryptographic designs. Surprisingly, only very few cryptographic algorithms proved to be secure over time. When used properly, such algorithms offer appropriate protection against malicious adversaries. However, despite their existence, these secure algorithms are still rarely utilized in commercial devices.

Mathematicians and experienced practitioners advocate that a cryptosystem should undergo a long and careful peer-review process that consists of academic and public scrutiny. Contrarily, businesses have a tendency to operate in isolation and protect their intellectual property. Accordingly, they design confidential proprietary cryptosystems which embed secret (authentication) protocols. Unfortunately, history shows that their proprietary designs tend to be insecure and susceptible to severe practical attacks.

In this thesis the security strengths of various proprietary cryptosystems are independently assessed. Advanced techniques, mathematical proofs and practical examples demonstrate how minor design mistakes can drastically reduce the security of a complete cryptosystem. Furthermore, it didactically guides and educates the reader to be creative and to think more as an adversary when assessing novel security designs.

This chapter introduces the motivation, objectives and outline of this dissertation.

A general introduction to security in the digital world is given in Section 1.1. The inspiration and motivation for this research is outlined in Section 1.2, followed by careful explanation of its scientific and social value in Section 1.4. Section 1.6 gives an overview of the research scope. Section 1.7 gives an outline of the chapters and results that were achieved during this study. Finally, Section 1.8 introduces the notation used throughout this dissertation.

1.1 Introduction to digital security

Privacy protection, secure data communication, remote authentication and proof of ownership are well-known examples of digital security mechanisms. Digital security, sometimes also referred to as information security, plays an important role in the daily tasks of an average person. It is hardly noticeable that such security mechanisms are used when email is checked, mobile phones are activated or a digital television is switched on. However, each of these tasks performs various mathematical computations to validate the user's credentials and protect the confidentiality of the transmitted data. Such mathematical computations are also called cryptographic operations. With the use of cryptography it is possible to achieve several goals of digital security; this section introduces five of these goals: confidentiality, data integrity, entity authentication, message authentication, availability and non-repudiation.

Confidentiality

There can be many reasons why a transmission of data should be protected against eavesdroppers. This is extremely important for content of high value, like privacy sensitive communication, confidential company information and data that involves governmental secrecy. In these cases it is important to establish a confidential communication channel where only the intended genuine sender and receiver can interpret the communication. Such a communication channel, which is kept confidential between the legitimate parties, is called a secure channel. The mathematical tool that is used for this is called encryption. More details about its background and specific workings are available in Chapter 2.

Data integrity

Data integrity enables a receiving party to determine if the transmitted data was unaltered and finished within the expected time-frame. When such an integrity check fails, it could be just a transmission error, or the data was intentionally altered by an adversary. Such alternation of data is referred to as message tampering. A communication is only considered secure, when all transmitted messages are delivered, none of the messages are tampered with, and the order and delivery time occurred as specified. They are commonly referred to as authentication protocols, see Section 2.4.1 for more details.

Entity authentication

Entity authentication verifies the authenticity of the transmitting party (entity). There are mathematical ways to prove that the data is originating from the genuine transmitting party. They are commonly referred to as authentication protocols, see Section 2.4.1 for more details. Such authentication protocols enable an authenticated secret key establishment between both entities [BM03], which is useful to set up an encrypted communication channel. Besides entity authentication, there are several more types of authentication, please refer to [MVOV10] for more details on this topic.

Message authentication

To trust a communication channel the involved parties should provide both: entity authentication and data integrity. It proves the authenticity of the message data and data origin. Such authenticity is established when the recipient can verify that the sender has access to secret credentials of the genuine originator. After verification the recipient has a certain level of confidence that the message was not illegitimately forged or tampered with. This is usually provided by the utilization of digital signatures. More information about such signatures is given in Section 2.3.4.

Authenticity is used for many purposes. For instance to prove the validity of an e-passport, ensure the genuineness of a website, protect against product counterfeiting or prevent fraud in digitalized public transport systems.

Availability

Availability is not always directly considered as a security requirement. However, recent events show that even websites of international banks and large government institutions can become inaccessible when enough network load is allocated to overload their servers. Therefore, the security is considered compromised when the system is made unavailable by an adversary, even when confidentiality and integrity are still guaranteed.

Non-repudiation

Non-repudiation allows the communicating parties to commit to their actions in such a way that it assures the origin, transport and delivery of the message. It is primarily of interest in the context of legal or financial transactions. The involved parties commit to their actions using cryptographic operations in such a way that they cannot deny their involvement afterwards.

Availability and non-repudiation are indisputably important security requirements. However, they are not directly applicable to the systems addressed in this study. Therefore, the focus is put toward the security concepts and cryptographic properties of confidentiality, integrity and authenticity. A more general overview of the security requirements is available in [Lan01, Sti05, MVOV10]. Additionally, a business oriented study about the process and financial impact of computer security is available in [LCPW01, GL02, BGL05]

1.2 Inspiration and motivation

In everyday life an average person is surrounded by cryptographically enabled electronic devices which operate remotely through a contactless interface. They are utilized to improve usability, transaction speed and reliability. Well-known examples include: using your access control card at the entrance of your office building; traveling with a contactless public transport card; authenticating to vehicle immobilizers using your car key; or paying for lunch with your cell phone.

The general public often uses devices without giving their security much attention, let alone questioning the actual strength of the applied security measures. They just work great, it often feels a bit magical to open a door from a distance using (battery-less) wireless devices. Such a feeling tends to give a false sense of confidence about the provided security. It looks very complicated, so it is probably pretty hard to understand, let alone to attack, break and abuse.

Surprisingly, the opposite seems to be true when looking deeper into the cryptography, authentication protocols and actual implementations of widely deployed cryptosystems. There are numerous examples where the security mechanisms of these systems fail. Subsequently, the general public is using many devices that embed weak security features, ranging from easy to break proprietary cryptography to unreliable and insecure communication protocols. It is no wonder that people start to feel like “*Everything is Hackable*”.

The problem is not the lack of decent and secure cryptographic algorithms and protocols. Several secure cryptosystems were introduced over the last decades [Smi71, DH76, RSA78, DR98]. The origin of the problem is related to the design methodology of a cryptosystem. A secret and proprietary design inherently hinders an open discussion about the security of a system. Moreover, it counteracts independent scientific assessments and public proposals to mitigate and improve the utilized security mechanisms.

1.3 Proprietary and secret cryptography

In 1883, more than a century ago, the Dutch linguist and cryptographer Auguste Kerckhoffs published a set of principles in the *Journal des sciences militaires* that should apply to a cryptosystem [Ker83]. The second principle, often referred to as *the Kerckhoffs’s principle*, represents one of the most cited and critical ground rules that apply to the design of a cryptosystem:

*Il faut qu’il n’exige pas le secret, et qu’il puisse sans inconvénient tomber entre les mains de l’ennemi*¹

An approximate English translation of the French text:

¹http://www.petitcolas.net/kerckhoffs/crypto_militaire_1.pdf

It must not require secrecy and it can without disadvantage fall into the hands of the enemy

Many cryptographers refer to Kerckhoffs's principle in the literature and likewise advocate that the security of a cryptosystem should not depend on the secrecy of the algorithm. Instead, its security should solely rely on the secrecy of the key.

According to Kerckhoff's principle the design of a cryptosystem should be considered known to an adversary. This does not necessarily imply that the method should be made public. Moreover, history teaches us that the security of a cryptosystem benefits from public scrutiny.

Over time, many proprietary cryptosystems proved to be weak and easily broken. The proposed attack methodologies are surprisingly simple and efficient. Algorithms and protocols are often compromised with use of only elementary cryptanalysis techniques, see Chapter 3 for more details. Note that such techniques are sometimes known in the literature for several decades. For instance, many proprietary cryptosystems are vulnerable to algebraic attacks, which were proposed in the literature almost a century ago in 1929 [Hil29].

Designing secure cryptographic algorithms has proven to be a difficult task without feedback from the scientific community [Ker83, JS97, fSN97]. The workings of proprietary cryptosystems are often kept secret to provide security-through-obscurity. The industry often claims that their products provide 'state-of-the-art', 'field-proven', 'high-level' and 'unbreakable' security, but it is hard to know what this means and how much security you actually get. There are numerous examples in the literature [Gol97a, WSK97, KSW97, Gol97b, WSD⁺99, Ste99, BBS99, HN00, FMS01, BGS⁺05, Bog07c, GdKGM⁺08, dKGGH08, LST⁺09, GvRVWS09, Cou09, COQ09, GvRVWS10, NTW10, KCR⁺10, BKZ11, VK11, CMK⁺11, GdKGV11, PN12, DHW⁺12, GdKGV12, BGV⁺12, VGB12, VGE13, WMT⁺13, SDK⁺13, OSS⁺13, MOPS13, BK14, GdKGV14] showing that once the secrecy of an algorithm is lost, so is its security.

Proprietary cryptographic algorithms do not always comply with open and community-reviewed encryption techniques. Therefore, the metrics of publicly scrutinized cryptography do not always hold for a proprietary cryptosystem with similar specifications, like the same secret key length and operation speed. Furthermore, the security features of a proprietary cryptosystem are undefined until they are independently and thoroughly assessed. Only if a comparable assessment and selection process is performed to construct a proprietary cipher, it is possible to match the strength of public schemes.

In 1997, the United States (US) Department of Defence (DoD) commissioned an objective study to measure and specify cryptographic strength [JS97]. The report explicitly states that if an algorithm is publicly proposed and published in the open literature, it should benefit from a broader base of criticism. Such criticism initiates a more severe verification of the algorithm and stimulates society to propose additional improvements.

Weaknesses make a cryptosystem vulnerable to malicious adversaries. Such weaknesses, also known as vulnerabilities, can be identified, yet they are never created. It is sensible to filter out as many weaknesses as possible before a cryptosystem is deployed. Vulnerabilities that are identified after deployment are often hard to mitigate. This is one of the main reasons why new cryptosystems are first publicly proposed in the scientific literature.

An important note needs to be made on proprietary versus secret cryptography. Cryptographic algorithms that are kept secret are not necessarily a fully customized design like most proprietary algorithms. Such designs are often similar to widely adopted and publicly scrutinized algorithms. For instance, by compromising on resources and encryption speed the strength of the algorithm can be increased.

Large institutions with access to substantial resources, such as government agencies and military establishments, have the capability to organize internal events to design new proprietary and secret cryptographic algorithms. These proposals are then subject to an internal but independent rigorous selection process, where weak schemes are eliminated and the best and strongest cryptographic design is selected. However, such a procedure is costly, requires many employees with a strong cryptographic background and would take many years of research. Finally, the secrecy of an algorithm should never be utilized to cover up negligent designs.

Despite the efforts of designing a secret algorithm, it should be noted that secrecy of an algorithm offers only marginal improvements to the security of a cryptosystem. As Kerckhoffs pointed out, it is just a matter of time before the inner workings of the algorithm are exposed to an adversary. Furthermore, it is not a measurable security property, since it is unknown whether the adversary has already gained knowledge about the algorithm or not. Especially in the longer term, it is wise to assume that such knowledge becomes public knowledge.

There are several ways to recover the mathematical operations of a cryptographic algorithm. For instance, a practical method to discover the workings of an algorithm is the use of reverse-engineering techniques [CC90, SS02], such as decompilation [Hou73, Cif94, CG95, SSB11] and side-channel analysis [Cla04, DLMV05]. With the assumption that an algorithm eventually gets exposed to the general public, and is likely to be peer-reviewed by the scientific community, it might be a wise strategy to invoke such procedures from the start.

The publicly proposed cryptosystems are carefully audited and scrutinized by fellow academics. After identification of vulnerabilities in a design it is immediately discouraged for further use, in theory and practice. Conversely, cryptosystems that resist comprehensive assessments and prove secure over time [Smi71, DH76, RSA78, DR98] are praised by the scientific, as well as the industrial security community. Such cryptosystems are utilized worldwide in many major computer systems that require a high level of security. Unfortunately, most small consumer devices do not embed strong cryptography. Instead, they still use ad hoc created and poorly tested secret and proprietary cryptosystems.

1.4 Scientific and social value

A very interesting point of discussion is the scientific and social value of studies that determine the strength of widely used proprietary cryptosystems. A separation can be made between the short-term and the long-term contributions.

Short-term value

The direct impact is the immediate response that is given by the users and other stake-holders. Once security issues of a system become public knowledge, the general public demands that the involved parties take responsibility and carry out appropriate actions to strengthen the system in such a way that it compensates for the loss of security.

Scientists play a crucial role in objectively informing society about important developments and serious security issues which concern the general public. Such activity includes the verification of open academic designs as well as the assessments of proprietary industry designs.

There is a risk that adversaries take advantage of the moment that a vulnerability becomes public knowledge, while not all security weaknesses are taken care of. However, it is important to note that unpublished vulnerabilities are not less dangerous than published ones. Weaknesses are identified, not created. Nevertheless, the risk of exploitation can be reduced if weaknesses are taken care of before the adversary learns that they exist. However, it is hard to estimate the kind of knowledge an adversary has access to.

Long-term value

Furthermore, it generally motivates the industry to migrate to well-studied and publicly reviewed cryptographic algorithms and formally verified authentication protocols. It has been known for many years that weak ciphers that solely depend on security by obscurity do not prevail in a hostile environment. However, industrial parties often only acknowledge the problem when their weak cryptographic algorithm is openly proven to be insecure and vulnerable in practice. Subsequently, this policy opens doors for third parties like criminals and government agencies to abuse its insecurity. The temptation is high for such parties to keep the knowledge of cryptographic weaknesses hidden and keep exploiting them in secrecy.

This study concretely addresses cryptographic security vulnerabilities which exist in widely used weak proprietary cryptosystems. It enables architects of such systems, academics as well as engineers, to learn from serious design and implementation mistakes. The presented security assessments of various cryptosystems educate architects to identify security vulnerabilities and prevents similar weaknesses in future designs.

1.5 Responsible disclosure

Even though the long term goal is to improve our security systems, there is an ethical consideration in the fact that publishing information regarding weaknesses in a cryptosystem could, in the short term, make it easier for criminals to misuse the system. To deal with this problem there is the widely accepted principles of *responsible disclosure*.

The National Cyber Security Centre (NCSC), part of the Dutch Ministry of Security and Justice, released formal guidelines which define responsible disclosure as:

Revealing Information and Communication Technology (ICT) vulnerabilities in a responsible manner in joint consultation between discloser and organisation based on a responsible disclosure policy set by organisations.

The guidelines from the NCSC encourage security researchers to disclose security vulnerabilities following this principle². A detailed overview of the responsible disclosure objectives are defined in the Request for Comments (RFC) draft [CW02], which was published in 2002 by the Internet Engineering Task Force (IETF).

Once a vulnerability has been found, stakeholders agree to allow a period of time for the vulnerability to be fixed before publishing the details. Developers of hardware and software often require time and resources to repair their mistakes. Computer security scientists have the opinion that it is their social responsibility to make the public aware of vulnerabilities with a high impact. Hiding these problems could cause a feeling of false security. To avoid this, the involved parties join forces and agree on a period of time for repairing the vulnerability and preventing any future damage. Depending on the potential impact of the vulnerability, this period may vary between a few days and several months.

The Dutch NCSC defines that if the vulnerability is to be made public, the organisation will set a date for when the vulnerability will be made public in consultation with the discloser. A reasonably standard term that can be used for software vulnerabilities is 60 days. Remedying hardware vulnerabilities is much more difficult; for these, a term of six months can be considered reasonable under normal circumstances.

Another authoritative organization is the Computer Emergency Response Team (CERT), which is part of the Software Engineering Institute (SEI), based at the Carnegie Mellon University. It is the US counterpart of the Dutch NCSC and dedicates itself to improve the security and resilience of computer systems and networks. Vulnerabilities reported to the CERT are by default disclosed to the public 45 days after the initial report, regardless of the existence or availability of patches or workarounds from affected vendors. Extenuating circumstances, such as active exploitation, threats of an especially serious (or trivial) nature, or situations that require changes to an established standard may result in earlier or later disclosure.

The vulnerabilities that are identified in the Technical Section of this dissertation

²<https://www.ncsc.nl/english/current-topics/news/responsible-disclosure-guideline.html>

are published in line with the principles of responsible disclosure, where all affected vendors were apprised at least a half year in advance of the publication. Together with the affected vendors an alternate publication schedule was negotiated when required. It is the goal of this policy to balance the need of the public to be informed of security vulnerabilities with vendors' need for time to respond effectively. The final determination of a publication schedule will be based on the best interests of the community overall.

1.6 Scope and focus

Designing a cryptosystem is a complex task. A slight variation of a secure design could undermine its protection against various attack and exploitation techniques. The variety of secure applications and their design specifics make it hard to compile one general solution. Instead, there are many different kinds of cryptosystems using various types of cryptography.

Weaknesses in a cryptosystem often depend on the specific use of certain algorithms and protocols. For instance, weaknesses of asymmetric cryptography based systems differ radically from those used in symmetric systems. Furthermore, the specifics and usage of symmetric algorithms like hash functions, stream and block ciphers define another additional subset of particular vulnerabilities. The subject is comprehensive and difficult to address entirely. Therefore, the scope of this study is limited to a subset of cryptosystems that are widely used in proprietary cryptographic designs.

A general categorization of cryptography is illustrated in Figure 1.1. The specific details of the mentioned categories are later discussed in Chapter 2, which embodies the introduction to cryptography. Within the scope of cryptography, the focus of this study is put on proprietary synchronous stream ciphers. Such stream ciphers represent a limited subcategory within the broader context of symmetric cryptography. For more details about stream ciphers, please refer to Section 2.3.2.

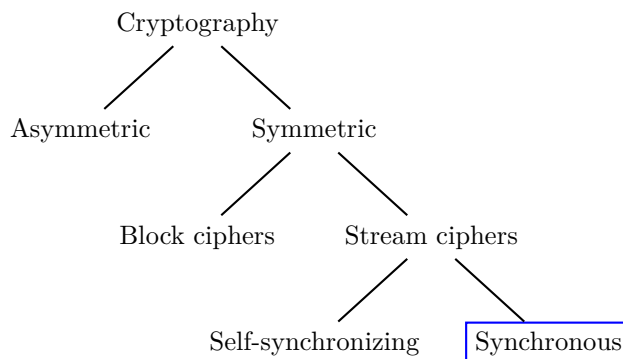


Figure 1.1: General categorization of cryptography and study focus

Radio Frequency Identification (RFID) is a contactless technology which offers a wireless communication interface to micro-controllers that operate with very limited energy. The technology is deployed in many security applications such as in secure storage, access control systems and anti-theft devices.

The majority of RFID devices use proprietary cryptosystems as a security mechanism to achieve confidentiality and integrity. The heavy dependence on proprietary cryptography makes them ideal subjects for the scope of this study.

1.7 Outline, contributions and results

The contribution of this study is a dissertation that introduces, summarizes and applies various cryptographic attack methodologies on proprietary ciphers. It is divided into three main parts, each containing several self-contained chapters. A quick overview of the parts and chapters follows.

Part I – Theoretical background

The first part covers the theoretical background which introduces the techniques and methodologies used during the study. It consists of four chapters.

Chapter 2 – Introduction to cryptography

The core aspects of cryptography are introduced in this chapter. It starts with a few historical examples and explains the different types of cryptographic algorithms. Furthermore, it presents the different properties of the components embedded in a cryptosystem. The chapter shows the reader the meaning of cryptographic strength and how to identify weaknesses in a cryptosystem.

Chapter 3 – Attack scenarios

Elementary attacking techniques are introduced in the second chapter. It first shows attacks on a protocol level, where the communication channel is compromised. Then, cryptographic weaknesses are exploited and attacked with various cryptanalytic techniques.

There are many complex mathematical notations to represent such techniques. However, this chapter puts forward the most commonly used cryptanalytic techniques by using comprehensible examples. Finally, it shows the possibility to compromise a cryptosystem on a different level by directly attacking the hardware. Most attacks addressed in this chapter are mounted in practice on a real cryptosystem in the technical section of this dissertation.

Chapter 4 – Related attacks on proprietary cryptosystems

The academic literature includes a large quantity of cryptanalytic contributions that assess the security of various proprietary cryptosystems. This chapter describes several studies which are most relevant to the research presented in this

dissertation. The results and conclusion of the related work strongly support the reasons to choose publicly reviewed cryptosystems over proprietary designs.

Chapter 5 – Introduction to Radio Frequency Identification (RFID)

The fourth chapter links the theoretical study on proprietary cryptography to practical examples of deployment in the RFID industry. To fully understand the features and limitations of the technology, a short introduction is given to the communication techniques and operational characteristics. Additionally, this chapter presents a set of hardware research tools which can be used to verify the results presented throughout this dissertation.

Part II – Technical section

The technical section is embedded in the second part of this dissertation. It contains the core contributions of the scientific study of proprietary cryptosystems. It includes comprehensive methods to identify weaknesses, novel attack techniques, precise cryptographic strength assessments and constructive procedures to mitigate attacks. Part II consists of eight original contributions published in the literature. The related articles are combined in five chapters which are summarized below.

Chapter 6 – MIFARE Classic

This chapter uncovers serious security weaknesses in the widely sold MIFARE Classic RFID chip. The chip is embedded in more than a billion contactless smart cards. It was originally designed for use in public transport systems. However, many highly secured facilities adopted RFID tokens for their access control system.

The chapter illustrates how the workings of the cryptosystem were revealed and presents a mathematical representation of the used algorithms. Furthermore, it identifies several weaknesses, proposes several attacks and demonstrates their feasibility. The practical impact of this study is vast. After exposure, many ministries, military bases, banks, nuclear power plants and prisons quickly migrated to more secure alternatives that use publicly reviewed cryptography.

Contribution

The work is based on two articles, *Dismantling MIFARE Classic* [GdKGM⁺08] and *Wirelessly Pickpocketing a MIFARE Classic Card* [GvRVWS09].

The first article [GdKGM⁺08] is written together with *Flavio Garcia, Gerhard de Koning Gans, Ruben Muijrs, Peter van Rossum, Ronny Wichers Schreur and Bart Jacobs*. The contribution of the author to this article is manifold. Firstly, the reverse-engineering process was performed by the author in close cooperation with Gerhard de Koning Gans. Secondly, one of the two proposed attack ideas originates from the author. Finally, implementation of the cipher, authentication protocol and attacks in software was largely done by the author.

The second article [GvRVWS09] is written together with *Flavio Garcia, Peter van Rossum, Ronny Wichers Schreur*. The author identified additional weaknesses, contributed a fully working novel attack himself and designed another attack together with the co-authors.

The practicality of both attacks was demonstrated by a fully working implementation written by the author. Furthermore, he designed a experimental set-up which could mount all proposed attacks and depends only on low-cost off-the-shelf hardware.

Chapter 7 – Hitag2

Hitag2 is one of the most used RFID chips in the car immobilizer industry. Such an immobilizer is an anti-theft device which prevents the engine of the vehicle from starting when the corresponding chip is not present. The chip is a passive RFID tag which is embedded in the key of the vehicle. Hitag2 uses a proprietary cryptosystem for authentication and confidentiality. This chapter reveals several cryptographic weaknesses and presents three practical attacks which depend only on wireless communication.

Contribution

The chapter is based on the article *Gone in 360 Seconds: Hijacking with Hitag2* [VGB12] which is written together with Flavio D. Garcia and Josep Balasch. The major part of this study is performed by the author. This includes the design and implementation of the weaknesses, attacks, experimental set-up and mitigating measures. However, the scientific article was composed in close collaboration with both co-authors.

Chapter 8 – SecureMemory, CryptoMemory and CryptoRF

This chapter assesses the security of an algorithm used in three different chip families, namely SecureMemory, CryptoMemory and CryptoRF. They are widely deployed and applied in many industries which include anti-counterfeiting, secure storage and contactless smart cards.

The chips use a proprietary stream cipher to guarantee origin authenticity, confidentiality, and data integrity. This chapter describes the cipher in detail and points out several weaknesses.

Contribution

This chapter represents the article *Dismantling SecureMemory, CryptoMemory and CryptoRF* [GvRVWS10]. The research load was evenly shared between the co-authors. The author contributed to several activities such as engineering, cryptanalysis, computing mathematical complexity estimations and writing of the article.

Chapter 9 – iClass and iClass Elite

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. According

to the manufacturer, more than 300 million iClass cards have been sold. The algorithms used in iClass are proprietary and little information about them is publicly available. This chapter presents six critical weaknesses and two novel cryptanalytic attacks, one against iClass Standard and one against the more expensive iClass Elite variant.

Contribution

The chapter embeds three scientific articles. The first two are conference papers, *Exposing iClass Key Diversification* [GdKGV11] and *Dismantling iClass and iClass Elite* [GdKGV12], the third is a more comprehensive story published by a journal *Wirelessly Lockpicking a Smart Card Reader* [GdKGV14].

The contribution to each paper is shared amongst co-authors. The author contributed a large amount of work to the cryptanalysis embedded the second paper. Furthermore, the author was closely involved in the design, development and verification of both attacks presented in this chapter.

Chapter 10 – Megamos Crypto

The Megamos Crypto chip is another widely deployed electronic vehicle immobilizer. The original scientific article was carefully peer-reviewed and accepted for publication. It demonstrates the lack of existing security in various car makes. The article reveals inherent weaknesses on the basis of mathematical calculations.

The intention was to embed the article in this chapter. However, due to a interim injunction, ordered by the High Court of London on Tuesday the 25th of June, 2013, the author is restrained from publishing the technical contents of the scientific article *Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer* [VGE13] until further notice.

Therefore, this chapter only defines a historical claim, in the form of a cryptographic message digest, of the original scientific article. This way, with restraints lifted in the future, the genuineness of the article in a post publication procedure can still be validated.

Contribution

The author supplied a comprehensive contribution to this article. A major share of the ground-work, design, implementation, experiments and writing is performed by the author.

Part III – Back matter

The back matter material is covered in the third part. It includes an extensive bibliography which reflects all consulted sources, index of commonly used phrases, overview of acronyms, list of figures and the author's curriculum vitae.

1.8 Notation

It is difficult to define certain cryptographic properties without a proper notation. Part I of this dissertation contains only limited use of elementary mathematical symbols, while Part II explicitly uses a more complex mathematical notation to support the statements that are specified in the text. Throughout this dissertation the notation that is used for mathematical symbols is specified in Section 1.8.1 and the representation of a byte is defined in Section 1.8.2

1.8.1 Mathematical symbols

The mathematical symbols are defined as follows:

Let $\mathbb{F}_2 = \{0, 1\}$ be the field of two elements (or the set of Booleans). The vector space \mathbb{F}_2^n represents a bitstring of length n . Given two bitstrings x and y , xy denotes their concatenation. Sometimes, this concatenation is written explicitly with $x \cdot y$ to improve readability.

Given a bitstring $x \in (\mathbb{F}_2^n)^l$, then $x_{[i]}$ denotes the i -th element $y \in \mathbb{F}_2^n$ of x . Likewise, y_i denotes the i -th bit of y . For example, given the bitstring $x = 0x010203 \in (\mathbb{F}_2^8)^3$ and $y \leftarrow x_{[2]}$ then the byte $y = 0x03$ and the bits $y_6 = y_7 = 1$.

The symbol ϵ represents the empty bitstring, 0^n a bitstring of n zero-bits and similarly, 1^n denotes a bitstring of n one-bits. \oplus denotes the bitwise exclusive-or (XOR), \boxplus denotes addition modulo 256, \bar{x} denotes the bitwise complement of x and $x \leftarrow y$ denotes the assignment of a value y to variable x . Sometimes, encryptions are explicitly denoted by $\{-\}$ to improve readability.

1.8.2 Byte representation

During a Radio Frequency Identification (RFID) transmission, the *most* significant bit is transmitted first over the air. However, the two leading RFID standards ISO/IEC 15693 [ISO00] and ISO/IEC 14443 [ISO01] specify that bytes should be represented with the *least* significant bit on the left. This means that an ISO/IEC compliant tool represents an over the air transmitted value of `0x0a0b0c` as `0x50d030`.

Unfortunately, the cryptosystems, specified in Part II, use the air transmission order to feed input bits into the cryptographic functions. Therefore, we adopt the air transmission convention throughout this dissertation (with the *most* significant bit left, since that has nicer mathematical properties). However, we make an exception when we show a communication trace, such that we maintain representational consistency with tools and command codes that are compliant with the ISO/IEC specification.

Part I

Theoretical background

Chapter 2

Introduction to cryptography

There are many situations where a private communication channel is required in order to exchange confidential messages. It is not hard to imagine such a requirement for governmental matters that concern national security. Moreover, it also applies for business information that conceals company trade-secrets or privacy sensitive information in someone's love-letters. The use of Cryptography can ensure that only the intended recipient can read the message.

Cryptography can be used to transform an original piece of text (plaintext) into a concealed piece of text (ciphertext) and vice-versa. The basic idea behind such cryptography is that a sender conceals (encrypts) a message using a mathematical algorithm (cipher) and a secret. A recipient that knows the cipher and secret can reveal (decrypt) the original message. The legitimate parties, sometimes referred to as (genuine) entities, can transform plaintext to ciphertext and ciphertext to plaintext. However, a third party (e.g. the adversary), has no knowledge of the secret and therefore does not have access to the plaintext.

Although this chapter tries to introduce the use of cryptography, it mostly focusses on encryption and authentication protocols. However, the applicability of cryptographic algorithms extends far beyond this scope. A much more extensive overview of cryptography in general is described in [Sti05] and [MVOV10].

Section 2.1 demonstrates the basic principle and usage of a cryptographic algorithm by introducing two historical ciphers. Section 2.2 gives a short introduction to cryptosystems. It defines the specification, cryptographic strength, design methodology and utilization of a cipher. The different types of modern cryptographic algorithms are introduced in Section 2.3. Section 2.4 introduces authentication protocols and authorization models and carefully illustrates the difference between them. Finally, the various weaknesses and vulnerabilities of a cryptosystem are presented in Section 2.5.

2.1 History

Preliminary forms of cryptography were already used by legendary civilizations like Ancient Greece [Rei62] and the Roman Empire [Sin66]. In fact, the word cryptography is derived from Greek κρυπτός ('kryptos' means 'hidden') and γράφειν ('graphein' means 'to write').

There are many ways to implement a symmetric cipher [Sha49]. However, most of the time it is a combination of the substitution and transposition technique. It is easy to explain those two techniques by applying two famous ancient ciphers. These ciphers were used to exchange secret military strategies between allied forces. The *Caesar substitution* was used by the Roman Empire and the *Scytale transposition* was used in Ancient Greece. In the next paragraphs, both ciphers are introduced.

2.1.1 Caesar substitution

The Caesar substitution cipher is a simple cryptographic algorithm. Many people played with it when they were kids to send a secret message to one of their friends. The basic concept of the algorithm is easy: every character is replaced (substituted) by a different character that is a number of positions later in the alphabet. The alphabet is rotated, so if the end of the alphabet is reached, the next substitution is with the first letter of the alphabet. The secret key in this algorithm is the number of positions by which the characters are rotated.



The procedure of this cipher follows. Take all the characters from the alphabet and put them from *A* to *Z* next to each other in one sequence. Rotate the characters by a few positions and write the same characters set below the original sequence. Figure 2.1 shows a graphical representation of the Caesar substitution cipher with a secret key a rotation by three positions.

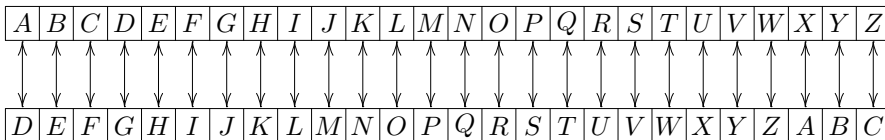


Figure 2.1: Caesar substitution using a rotation of three positions

Figure 2.2 shows an example of encryption and decryption of the word “ATTACK”. The top (plaintext) characters are substituted with the bottom (ciphertext) characters as specified in Figure 2.1. This example makes it rather easy to spot one of the obvious weaknesses in this cryptosystem: it changes the characters yet not the structure of the plaintext. An adversary can immediately deduce the length of the plaintext and that several characters are similar.

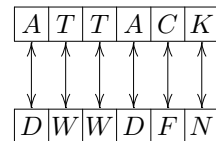


Figure 2.2:
Substitution
example

In Figure 2.2 the 1st character is the same as the 4th and the 2nd is the same as the 3rd. There are not so many (English) words of six characters that have such a structure. Additionally, inside information about the context allows for more optimized plaintext recovery. For instance, when the message illustrated in Figure 2.2 can be correlated to battlefield terminology, the set of generic English words can be

reduced directly to the original plaintext. More information about the cryptographic strength of the Caesar substitution cipher is explained in Section 2.2.2.

2.1.2 Scytale transposition

In the era of Ancient Greece, Spartan generals and Greek leaders $\epsilon\pi\eta\rho\rho\varsigma$ ('ephors' means 'one who oversees') used a cylindrical staff as a tool to apply an elementary transposition cipher. Such a staff was called a $\sigma\kappa\upsilon\tau\alpha\lambda\eta$ ('scytale' meaning 'baton'). It is assumed that generals were able to secretly communicate with one another by using a scytale of similar size [Rei62].



To encrypt a message, the general would wind a ribbon of papyrus vertically around the scytale and write the text on it horizontally. After completing the message, the ribbon was unwound and delivered. The characters on the ribbon are transposed and would appear to be written in a jumbled form. However, the original content of the message is revealed when the recipient rewound the ribbon on a scytale with similar shape and dimensions. Encryption with a scytale only depends on thickness of the scytale, which represents the characters split position. Such split position can be seen as the secret key of a transposition cipher. Figure 2.3 shows a simple transposition encryption performed on a famous quote from [Sun94] using a split position of seven characters.

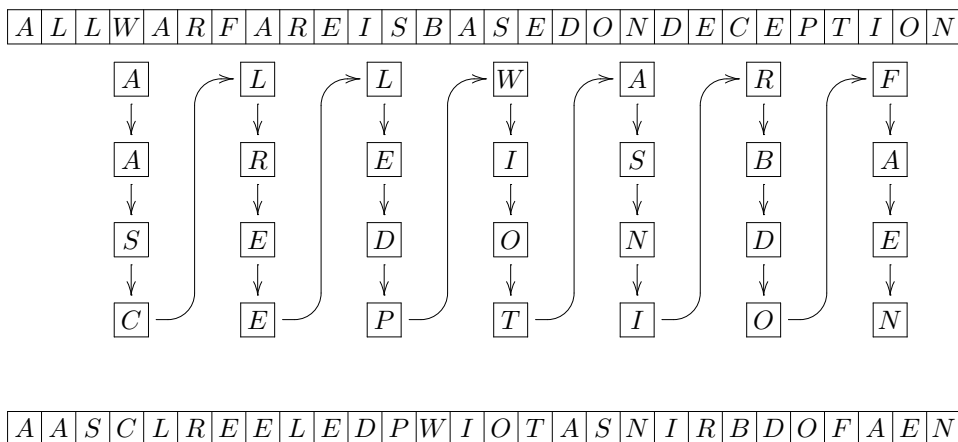


Figure 2.3: Scytale transposition using a split position of seven characters

With respect to the Caesar substitution cipher, the structure of the original plaintext is much harder to reveal from the the transposed characters. However, re-arranging positions with a scytale is very limited and can be rather easily recovered using an exhaustive search of all the possibilities.

There are several ways to attack more complex transposition ciphers. For example,

the original character representation and distribution are not hidden when a transposition cipher is used solely. This enables an adversary to make a statistical analysis of the frequent of occurring syllables in the language that were used in plaintext. Such analysis would easily reveal the secret transposition split position. An example of a cryptographic attack that uses frequency analysis is given in Section 2.2.2.

2.2 Cryptosystems

A system that provides a usable platform with cryptographic features is often referred to as a cryptosystem. The building blocks of such a cryptosystem are the cipher specification, cryptographic algorithm, implementations, usage and applications of the system. This chapter explains them in more detail.

Section 2.2.1 shows the details from a cipher specification and configuration properties it may embed. Section 2.2.2 gives a short introduction into measuring the cryptographic strength of a cipher. The design process of a cipher is handled in Section 2.2.3. Finally, Section 2.2.4 addresses the application and deployment of a cryptosystem.

2.2.1 Cipher specification

The specification of a cipher defines properties about the secret key, in- and output message lengths, operational details and computational complexity.

All cryptographic algorithms require a secret key to operate. The length of this key, often specified by its number of bits, depends heavily on the working of a cryptographic algorithm. For example, as mentioned in Section 2.3.1, asymmetric cryptography requires much larger keys than symmetric cryptography.

The length of an in- and output message differs per algorithm. It depends on the operation of the cipher. Block ciphers use a fixed message length of multiple digits, while hardware optimized stream ciphers often operate bit-wise. This property does not directly influence the strength of an algorithm. However, some configurations may require specific use of the algorithm to avoid cryptographic vulnerabilities. The environmental configuration and application of ciphers is further discussed in Section 2.2.4.

Some cryptosystems have special requirements that vary per algorithm. They specify operational directives for initial computations so that the output becomes less proportional to the input, which introduces an increase of non-linearity. For instance, the output produced during the first computational rounds of a cipher often contains direct linear relations to the input. Therefore, the specification can define a minimum number of initial encryption rounds to significantly decrease the linear dependencies between the cipher internals and the computed encryption bits. The specifics of (non-)linearity are further defined in Section 3.2.6.

The use of a cryptographic algorithm requires a certain amount of computation, commonly referred to as computational complexity. This is often deterministic for a certain length of input. Such complexity defines the minimal number of bit-wise operations in terms of hardware gates (transistors) that are required to assemble a chip which can perform the complete cryptographic computation. More information about the computational complexity is described in Section 2.2.2.

2.2.2 Cryptographic strength

The strength of a cryptographic algorithm is expressed in the total amount of computations an adversary needs to perform to recover the secret key. It is often referred to as the computational complexity of the cipher.

For a perfectly secure cipher, the computational complexity is the same as the key space, sometimes referred to as the *entropy* of the key. The key space refers to the set of all possible keys and represents the total number of combinations using all secret key bits. The size of the key (key-size) is the amount of bits n which define the size of the complete key space 2^n .

The naive method to recover the secret key is to try simply all combinations. Such methodology is often referred to as an exhaustive search or a brute-force attack. It would most likely require almost 2^n computations to determine the secret key. To be precise, on average an adversary finds the secret key halfway. When lucky, the key can be determined at an early stage, but it might just as well be one of the last tries. Interestingly, this property already shows that the number of computations required to determine the key is by definition lower than the actual key space.

A cipher that is perfect should be the base for a secure cryptosystem. In practice however, most ciphers are (slightly) weaker than the full entropy of their secret key. With clever optimizations it is often possible to find the secret key by doing far fewer computations than the actual entropy would require. This is called the actual attack complexity of a cipher.

To explain the attack complexity we use a substitution cipher which is similar to the Caesar cipher introduced in Section 2.1.1. However, the algorithm does not depend on a rotation in the second sequence of characters. This time it re-arranges the characters completely. Figure 2.4 shows an example of a random character permutation sequence.

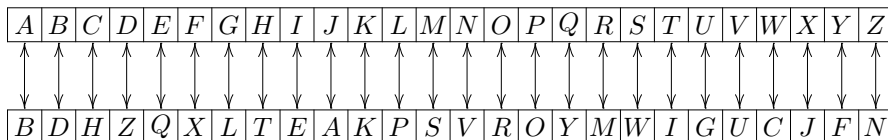


Figure 2.4: Modified Caesar substitution using a random permutation

This allows at most $26!$ permutations, where each permutation is uniquely defined by a secret (key) value. This results in the fact that the total entropy is close to

$26! \approx 2^{88.4}$, which represents a key of 88 bits. Such a relatively big key space is still considered secure. In 2014, an average consumer laptop is able to perform $2^{40} = 1,099,511,627,776$ computations within an hour¹. Although it looks impressive to perform more than a trillion computations within one hour, with an entropy of $2^{88.4}$, the laptop needs to compute for $2^{88.4-40} \times 2^{40} = 2^{48.4} \times 2^{40} \approx 366,791,447,164,915$ hours.

However, when an adversary attacks the algorithm using the character frequency of a language, the attack complexity drops dramatically. The letter ‘E’ occurs on average the most often in any piece of written English text. An adversary just counts the total number of symbols on one page and constructs a list of all characters and their occurring frequency. It is most likely that in a sorted frequency list in descending order, the top character represents the letter ‘E’. This means that the adversary found an attack to drop the complexity to $25! \approx 2^{83.7}$. Repetition of the attack with vowels and other frequently occurring consonants, the attack complexity vastly drops to an entropy below 2^{40} , which is easily solved with an average laptop.

2.2.3 Design methodology

To perform encryption and decryption, the algorithm needs to be known by the sender as well as by the recipient. When all parties are trusted entities it may seem useful to reveal only the algorithm to those involved. However, this means the initiator needs to design its own algorithm and supply that in secret to the other parties. This way, the algorithm actually becomes an extension of the secret key. In the late 90’s, Bruce Schneier already stated [Sch98a, Sch98b] that it is far from difficult to reverse-engineer an algorithm. When the algorithm becomes public, it may get much worse, since it was designed internally with little feedback from the large scientific and industrial cryptographic community.

As mentioned in Section 1.2, most scientists advocate that the strength of a cryptosystem should depend on secrecy of the key and not of the algorithm [Ker83]. However, for industrial parties it seems attractive to keep the workings of their product secret. Instead of security, the main objective shifts to protect themselves against counterfeiting by competitors. Such designs often prove to be weak and offer a much lower attack complexity compared to their claimed computational complexity, see Section 2.2.2.

Many insecure proprietary algorithms were proposed in the last decades, see Section 1.2 for more details. Almost all of them have an attack complexity that is lower than an average laptop can compute in a small amount of time. Therefore, they are considered practically broken and insecure to use. An interesting fact is that after a proprietary algorithm is reverse-engineered, it often takes only a few weeks before a successful attack is mounted.

¹Results gathered from various computational experiments

Since the 70's the approach to design a strong cryptographic algorithm became somewhat different. Experts in cryptography of the US government decided that it would be more satisfactory if there was a world-wide call for participation to design a new cryptographic algorithm that complies with a certain cipher specification, like announced in [FSN97]. The challenges are organized by the National Institute of Standards and Technology (NIST), formerly called National Bureau of Standards (NBS). They specify that proposed cryptosystems should undergo a longer period (several years) of peer-reviews and public scrutiny. The cipher with the highest attack complexity and best performance is selected as a new public encryption standard. Prominent ciphers competitions were held for the Data Encryption Standard (DES) [FIP77] in 1977, the Secure Hash Standard (SHS) [FIP93] in 1993, the Advanced Encryption Standard (AES) [FIP01] in 2001, and the Secure Hash Algorithm-3 (SHA-3) [FIP14] in 2014.

The success of this selection method can be measured by examining the number of years that attack complexity is above the computational capabilities of high-end computers. For example, the cryptographic attacks on DES presented in the literature [BS93, Mat94] still require a complexity of at least 2^{39} computations, see Section 3.2.5 for more details about the attack technique. An important remark on those attacks should be made: they additionally need more than 70 terra bytes of chosen plaintext-ciphertext pairs. Hence, the feasibility to mount an attack against DES in practice is very slim.

As a result, although the DES algorithm is clearly out-dated (1977), has a only very limited key-size (56 bits) and can easily be broken with a big cluster of computers [KPP⁺06], it is still considered (moderately) secure.

2.2.4 Application and deployment

A cryptographic algorithm has to be used in the correct way. In many cases it is just a building block of a larger cryptosystem to provide an authentication method (see Section 2.4.1) or data integrity check (see Section 2.3.4). The purpose of the application and its working environment define how the encryption functionality of the cipher is used. An authentication method could in theory be specified independently of the cipher. However, in practice it is usual to design the authentication protocol in such a way that it is optimized for the characteristics of the applied cipher. Examples could be to adjust the size of the input and output such that it matches the cipher specification as explained in Section 2.2.1.

The purpose of some cryptosystems is to authenticate entities rather than to encrypt confidential data. In such a system the security depends on the combination of cipher strength (as described in Section 2.2.2), the used authentication protocol and the way they are implemented. A more detailed overview of various weaknesses that could be present in a cryptosystem is described in Section 2.5.

2.3 Cryptographic algorithms

As illustrated in Figure 1.1, there are several categories of encryption algorithms within cryptography. This section explains the classification and covers the most prominent variants and utilization of modern ciphers.

The difference between symmetric and asymmetric ciphers is explained in Section 2.3.1. Symmetric cryptographic algorithms can be divided into two categories, stream ciphers and block ciphers. Section 2.3.2 introduces stream ciphers and Section 2.3.3 block ciphers. Finally, the way a cipher can be used for other purposes than confidentiality is covered by Section 2.3.4.

2.3.1 Symmetric and asymmetric cryptography

Cryptographic algorithms can be divided into two categories, symmetric and asymmetric. A cryptographic cipher is considered to be symmetric when it uses the same secret key for both encryption and decryption. An asymmetric cipher, on the other hand uses two different keys. One key is kept secret, this is called the private key. Contrarily, the second key is revealed to allow encryption of messages which are specifically directed to the owner of the corresponding private key. This second key is often revealed to the general public, since in most cases the recipient wants to make the possibility of sending encrypted messages to him available to everyone. Therefore, this key is referred to as public key. A message that is encrypted with the public key can only be decrypted with the corresponding private key and vice-versa. This can be very useful in situations where one party wants to establish a secure communication with another party without being in contact on beforehand to exchanged a shared secret key (like applied in symmetric cryptography).

The basis for asymmetric cryptography is a mathematical problem which admits no efficient solution, like integer factorization, discrete logarithm, or elliptic curve relationships. These problems are time-consuming to solve, but usually faster than trying all possible keys by brute-force and therefore require keys of larger sizes. Keys of an asymmetric cipher are at least twice as big as symmetric cipher keys with comparable security. Due to hardware limitations of small embedded systems, the lighter requirements of symmetric cryptography are often preferred to avoid computational expensive big number multiplication and storage of large keys in an asymmetric cryptography system.

2.3.2 Stream ciphers

A stream cipher performs an encryption which is similar to the One-time Pad (OTP) encryption technique. It produces a large chunk of secret, random looking data and combines it with the plaintext to produce ciphertext. Without the exact same data chunk, the plaintext cannot be uncovered from the ciphertext. The random data

represents a stream of bits which is derived from the secret key and is commonly referred to as *keystream*. A stream cipher contains some persistent memory, called the internal cipher state, which is initialized by the secret key and propagates to a successor state after each encryption step. The output of a strong stream cipher is comparable to (and should be indistinguishable from) a contiguous bit stream produced by a Pseudo Random Number Generator (PRNG).

To be more precise, we embed the remarks made in [Sha04] and define a stream cipher as follows: an encryption function which operates on individual plaintext digits (usually bits) where its internal state is initialized with the secret key prior to encryption. The keystream varies, depending on the initialized secret key and the moment of encryption with respect to the propagation of the internal state. Encryption of plaintext and decryption of ciphertext are both performed by the exclusive-or (XOR) operation, which is denoted by a \oplus symbol and represents a bit-wise addition modulo two. A useful mathematical property of this operator is that it can be inverted. Therefore, it can be applied for encryption as well as decryption.

There are two types of stream ciphers, synchronous and self-synchronizing. In a synchronous stream cipher, the encryption bits are computed independently from the plaintext. Such ciphers are useful in situations when a communication channel is more prone to error. It might happen that just one badly transmitted bit is wrongly interpreted, which however does not directly affect the other bits that were transferred in a correct manner. Therefore, stream ciphers are very useful to encrypt streaming media where the speed of data-traffic is more important than the completeness and integrity of the data. Contrarily, a self-synchronizing stream cipher computes the successor of its internal state with a function over the previous state and the ciphertext. The internal state diverts from its original propagation path when a transmission error occurs. This dissertation focusses itself on the most widely used and best studied of the two, the synchronous stream ciphers. Therefore, a general reference to a stream cipher refers to a synchronous stream cipher.

An important objective of a stream cipher is to avoid a direct relation between the input (secret key) and output (keystream) of the cipher. Because the entropy of a stream cipher is limited to the size of the internal state, the produced keystream will eventually repeat itself. Note, that this is not a property of a regular One-time Pad (OTP).

Pure One-time Pad encryption can provide *perfect secrecy* when the keystream is truly random and uniquely generated for each message that is transmitted [Sha49]. In such a setting, the keystream should consist of a unique bit string that contains uniformly distributed random bits. However, in practice it is difficult to generate truly random data. Alternative methods, like using the complete contents of a random book, drastically limit the number of possible keystreams. Moreover, reuse of the same keystream is very insecure. With access to previous plaintext and ciphertext, an adversary would be able to extract the keystream. If the same keystream is used in a second transmission, the adversary can use the recovered keystream and reveal

the second plaintext. Exactly for this particular reason, the encryption technique is called One-time Pad. The keystream that represents the secret key should only be used once.

Keystream can be seen as an unique set of bits which must be as long as the plaintext. However, continuing distribution of fresh keystream for long data sequences is undesirable. With an increase in electronic transmissions of large transcripts in the 20th century, the need for alternative solutions grew. In response several stream cipher encryption techniques were introduced. For instance, in the 1930s stream ciphers were mainly used in the form of physical rotor machines which operated mostly mechanically. A well-known example of such a rotor machine is the Enigma, which is illustrated in Figure 2.5. A few decades later, the introduction of large scale computer networks increased the demand for more hard- and software oriented stream ciphers which supported automated communication.



Figure 2.5: Enigma machine²

One of the most popular techniques in the sixties and seventies was the non-linear binary sequence stream cipher [Gol67, Gro71, Key76, Ple77, DH79]. It produces a binary keystream that allows a regular One-time Pad encryption without the requirement of a very large secret key. A typical cryptosystem based on a non-linear stream cipher is illustrated in Figure 2.6. This type of ciphers was very popular because of their small hardware footprint. Most of the proprietary ciphers that are assessed in Part II are a derivative of the non-linear stream cipher.

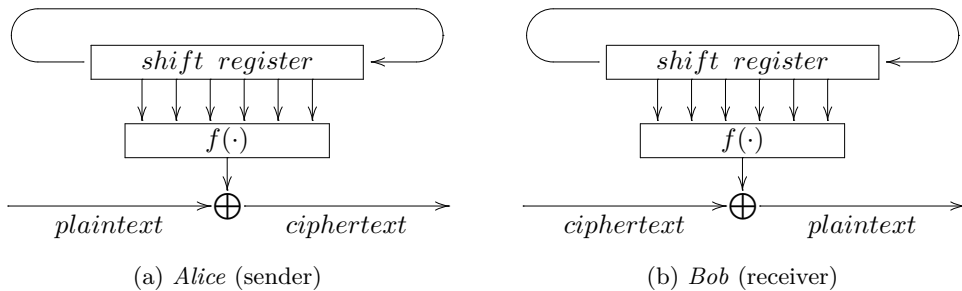


Figure 2.6: Typical non-linear stream cipher system

The cryptographic algorithm illustrated in Figure 2.6 embeds a rotating shift register, which represents the internal state of the cipher. After the computation of a new cipher bit, the successor function updates the internal state by a linear function to preserve as much entropy to the cipher. Then, the output component applies a

²<http://www.securityninja.co.uk/wp-content/uploads/2010/09/4-rotor-enigma-open-copy-BW.jpg>

non-linear filter function $f(\cdot)$ to compute the next keystream bit. The keystream bits are used by the sender (Figure 2.6a) to encrypt the plaintext bits by combining both bit strings with the exclusive-or (XOR) operation. The resulting ciphertext is transmitted over an insecure channel. The receiver (Figure 2.6b) performs the exact same computations and applies another XOR operation, this time on the ciphertext bits in combination with the keystream bits. The keystream bits, already embedded in the ciphertext, are cancelled out and the original plaintext is revealed to the receiver.

The sender and the receiver use the non-linear stream cipher to compute exactly the same keystream. Then, the sender combines the keystream with the plaintext to produce the ciphertext by using the XOR operation. The receiver performs the same technique on the ciphertext together with the keystream to reconstruct and reveal the plaintext.

In most cryptosystems it is important to link multiple encrypted messages in one cryptographic session, this is called chaining of encryption. Stream ciphers inherently provide this feature since their ciphertext is produced incrementally. It uses the previous internal state and a successor function to step forward.

Besides these historical stream cipher designs there are several new proposals in the literature [AM97, EJ03b, DC06, HJM07]. Despite their advantages in flexibility and speed, stream ciphers are currently scarcely used in secure systems that provide strong cryptographic security. Typical stream cipher attacks aim to separate the plaintext from the encryption bits. For instance, a malleability attack exploits a general and unavoidable weakness in traditional stream ciphers where the keystream is generated independently from the plaintext. Small alterations (bitflips) to the ciphertext might be sufficient to perform the attack without actually recovering the secret key. More details are given in Section 3.2.1.

The security that is provided by the underlying building blocks of stream ciphers are well-studied. However, the security implications of these separate components may not hold when they are combined and used together in one cryptographic algorithm. Instead, the comprehensive security implications of block ciphers are better understood [BKL⁺07, Bir04]. The next Section 2.3.3 explains block cipher designs and their (dis)advantages in more detail.

2.3.3 Block ciphers

A block cipher operates on data blocks of a fixed size, instead of on individual bits. The size of the blocks are predetermined and defined in the cipher specification. The length of the input data needs to meet the exact specified length. This often requires padding of the input to align the data with the cipher input requirements.

A useful property of block ciphers is that the encryption output of block ciphers heavily depends on the content of the data itself. The output changes almost completely when only one bit of the input is changed. This property makes it much harder

to correlate between encrypted messages that contain only slight differences in the plaintext.

A disadvantage of block ciphers is that, by default, multiple encryptions are performed independently from each other. To maintain a linked cryptographic session an additional method should be employed. Over the years, several techniques were proposed in the literature to chain sequential block cipher encryptions. A well-known example is the Cipher Block Chaining (CBC) method which can be used to cryptographically link encrypted data blocks. The chaining method was proposed in 1976 and patented by IBM [EMST78]. It is used in many block cipher cryptosystems. Although CBC encryption is useful for session confidentiality, it lacks session integrity and is proven to be vulnerable to practical attacks [DR11].

In addition, alternative techniques were proposed that additionally provide session authenticity, confidentiality and integrity. Well-known and widely deployed alternatives are Counter with CBC-MAC (CCM), proposed and evaluated in [WHF02, Jon03], and Galois Counter Mode (GCM), introduced in [MV04]. Both chaining methods are currently recommended by the National Institute of Standards and Technology (NIST) in [Dwo04, Dwo07] and are used for many secure applications. For instance, most wireless consumer networks, secured by Wi-Fi Protected Access II (WPA2), depend on the Counter with CBC-MAC (CCM) technique which combines the Counter (CTR) chaining mode encryption together with a CBC-MAC to provide both confidentiality, data integrity and authenticity. The second technique, Galois Counter Mode encryption chaining, is used in various network and infrastructural security solutions such as Internet Protocol Security (IPsec), Secure Shell (SSH) and Transport Layer Security (TLS).

Block ciphers are computationally more expensive than stream ciphers, yet their security properties are better understood [BKL+07]. Furthermore, traditional block cipher encryptions (not keystream generation) suffer less from typical stream cipher weaknesses, such as the vulnerability to malleability attacks. In fact, an interesting development in algorithm designs is the shift to more hybrid cipher schemes [HCJ02, WFY+02]. Underneath those recently proposed stream cipher designs, there is actually a block cipher component that operate in chaining mode [McG02, Bir04].

2.3.4 Hashes, MAC's and Digital signatures

There exist cryptographic techniques that are not solely used for confidentiality. Well-known examples are cryptographic hash functions, message authentication codes and digital signatures.

A cryptographic hash function generates from an arbitrary length input bit string (message) a collision resistant, very hard to forge, yet verifiable output bit string (digest) of a fixed length. It implements a One-Way Function (OWF), which suggests an algorithm that is easy to compute and very hard to invert. There is a broad com-

munity of academics that study the properties of these mathematical functions. More detailed information about the workings of hash functions and their cryptographic strength is available in the literature [PGV94, PS96, CDMP05, WY05, MVOV10, Ste13].

A Message Authentication Code (MAC) is a bit string that can only be computed with knowledge of a secret key and is transmitted alongside the plaintext or encrypted message. Both parties, sender and receiver, have knowledge of the secret key. Therefore, it can be used to verify the data integrity and authenticity of the corresponding message. MAC algorithms are often constructed from other cryptographic primitives, such as a Hashed Message Authentication Code (HMAC) which utilizes a cryptographic hash function [FIP08], or a Cipher-based Message Authentication Code (CMAC) that is based on a symmetric key block cipher [Dwo05].

Digital signature also offer verification of data integrity and authenticity of a message. However, they differ from a MAC in the sense that they depend on asymmetric cryptography which uses public and private keys, see 2.3.1 for more detail. The public key of the sender is often embedded into a certificate (chain) which is ultimately signed by an authority that is known and trusted by the recipient. The receiver can establish trust in the public key of the sender by verifying the certificate and signature from the trusted party. After the verification of trust, the recipient can use the public key to validate the signature over the corresponding message and establish confidence about the data integrity and authenticity of the message.

2.4 Protocols and access conditions

Cryptographic algorithms are often used for confidentiality reasons. However, the encryption and decryption functionality can be applied in a particular manner such that it can be used to prove authenticity of one entity to another. This process is defined in the authentication protocol of the cryptosystem and is further explained in Section 2.4.1. A cryptosystem can embed various security levels with different access conditions for each entity. The access conditions specify the acquired rights and authorization of a certain entity. A further introduction to the details of an authorization model is given in Section 2.4.2.

2.4.1 Authentication

A symmetric cryptographic algorithm is used to identify and authenticate two parties to each other. This operation is specified in a so-called authentication protocol. An authentication protocol consists of a set of predefined cryptographic operations which describe the tasks of each party in proving their legitimacy to the other party without revealing the secret key. It typically relies on a technique that is called cryptographic challenge-response verification. First a random number is transmitted as challenge, followed by a response which proves knowledge of the secret key. The response is

derived from the output of the cryptographic computation that uses the secret key and challenge as Initialization Vector (IV). Legitimate parties can validate the response, yet an adversary should not be able to derive any secret from a challenge-response pair.

It is possible to simultaneously authenticate many parties to each other. However, the scope of this section is limited to authentication protocols which are designed to authenticate two parties to each other using symmetric cryptography. For explanatory reasons it is useful to make a distinction between these parties. Therefore, we define the first (initiating) party as *Alice* (\mathcal{A}) and the second (responding) party as *Bob* (\mathcal{B}). To increase readability, this section presents rather conceptual versions of the mentioned authentication protocols.

The first scheme is a limited authentication protocol, illustrated in Figure 2.7, which performs a single authentication that only proves the legitimacy of one party. Then, a basic mutual authentication is introduced where both parties prove their legitimacy concurrently to each other, see Figure 2.8. Finally, a slightly more complex, yet much more secure version of a mutual authentication protocol is presented in Figure 2.9.

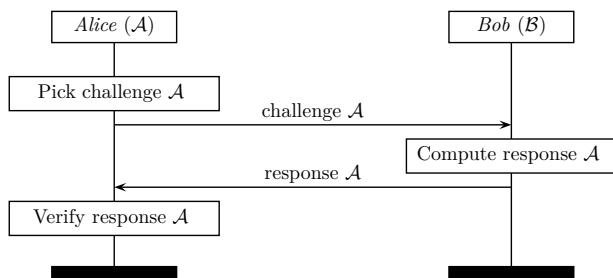


Figure 2.7: Two-pass single authentication protocol

In a single authentication, sometimes referred to as unilateral authentication, only one side proves legitimacy to the other. Figure 2.7 shows an example where *Bob* authenticates himself (proves who he is) to *Alice* using a simplified two-pass single authentication protocol. In this scheme *Bob* proves his legitimacy by answering with a cryptographic response to the challenge from *Alice*. Since only the legitimacy of *Bob* is verified, an adversary could easily impersonate *Alice* and gather valid challenge-response pairs. Such a set-up defines a typical form of an encryption oracle and is further explored in Section 2.5.4.

To achieve bilateral trust it is more common to use a mutual authentication protocol. In this case both parties, *Alice* and *Bob*, prove concurrently that they are genuine to each other. This is validated with verification of the responses, they should be outputs of a cryptographic computation which uses a secret key that only *Alice* and *Bob* know. Figure 2.8 shows a two-pass mutual authentication protocol. It extends

the single authentication protocol with additional response which proves that *Alice* is legitimate as well.

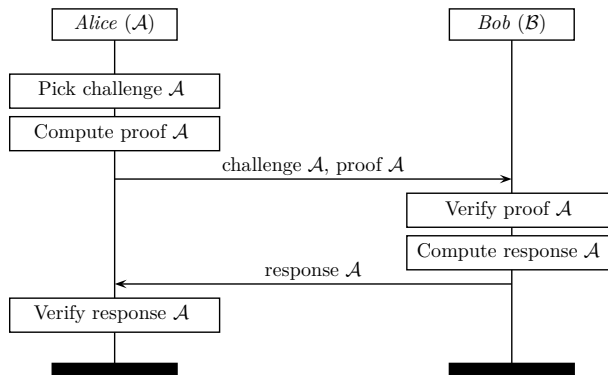


Figure 2.8: Two-pass mutual authentication protocol

Figure 2.9 presents a simplified three-pass mutual authentication scheme. After a successful mutual authentication, *Alice* and *Bob* can trust each other and start a communication session. As mentioned in Section 2.3.2, it is common to cryptographically chain all further communication to the original session authentication credentials. A session should be initialized by unique (fresh) random challenges that contains a large enough entropy to avoid predictable values or future collisions.

A more refined, yet slightly complexer, example of the three-pass mutual authentication protocol is standardized in ISO/IEC 9798-2 [ISO99]. Various improvements to this particular authentication scheme are proposed in the literature [FDW04, DDMP04, BCM12].

To strengthen an authentication protocol, there are ways to reduce the use of the global shared secret in a symmetric cryptosystem. One example is to introduce diversified cryptographic keys. A diversified key is a secret per entity that is derived from a master secret in combination with the identification information of the involved entity. In this way a different diversified key is generated for each entity. When such an entity key is recovered, only this particular entity secret is compromised and not the master secret.

2.4.2 Authorization

Authorization is sometimes confused with authentication. An authorized state represents an entity with a certain set of acquired rights, which are specified in a list of access conditions. The authentication process can be used as the underlying verification method to reach an authorized state, yet its main objective is to identify and authenticate the entity.

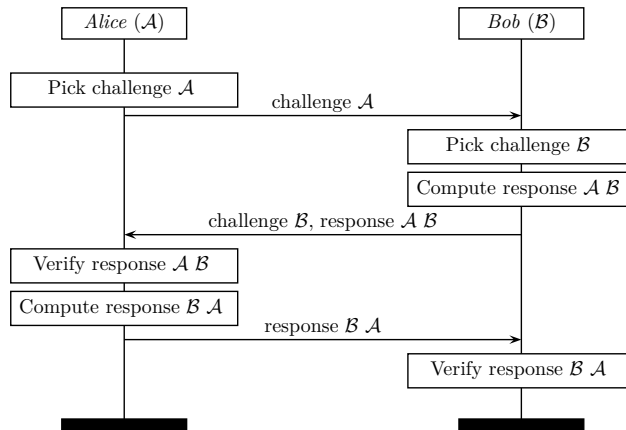


Figure 2.9: Three-pass mutual authentication protocol

The access conditions in a cryptosystem must be defined very strictly in an authorization model. Such a model includes a detailed specification of the access conditions which are valid after proving ownership of a specific set of credentials. After successful verification of the credentials a state transition takes place. The entity acquires the rights specified in the access conditions of the authorization model.

Each state transition in an authorization model must be included in the predefined list of access conditions. Unspecified events could otherwise initiate states with undefined (and potential harmful) behaviour. Furthermore, access to certain actions should be restricted by default and only be allowed when the corresponding credentials are authenticated. In a smaller system such rules are easy to define. However, the complexity of these definitions grows exponentially when more features and authentication methods are added. For instance, allowing different secret keys, remote credential alterations such as key-updates or specific authorized operations like memory locking and data manipulation.

2.5 Weaknesses in symmetric cryptosystems

This section introduces eight different types of weaknesses in cryptosystems based on symmetric cryptography. Starting with the cryptographic topics which embed weaknesses in cipher design, authentication protocol, cipher initialization and encryption oracle. Followed by communication oriented weaknesses which occur in the authorization model and communication protocol. Finally, a description of weaknesses in the implementation and deployment of a cryptosystem are given.

2.5.1 Cipher design

A secure cipher design should offer a high degree of computational complexity which provides the necessary cryptographic strength as specified in Section 2.2.2. Such a design should protect against any attack that could be mounted with fewer operations than an exhaustive search over the full key space. In practice however, this is very hard to achieve.

Serious weaknesses that compromise the computational complexity of a stream cipher often thrive on insecure cipher designs. Their internal state often relies on individual linear components that are in low cohesion with respect to the other cipher components. The components are combined together with a complex non-linear filter function $f(\cdot)$ to avoid a direct relation or partial correlation between the internal state and the produced ciphertext. However, since these components often operate independently from each other and only have modest direct influence on the produced encryption bit, an adversary could guess only a few components and learn the value of the other components by statistically inverting $f(\cdot)$. Although the purpose of a non-linear function is to counter inversion, it might be possible to use techniques like guess-and-determine to eliminate incorrect candidate internal states, for more details on this subject see Section 3.2.4. A clear example of inverting the non-linear cipher component is shown in Chapter 6 where a cryptanalytic attack is proposed against the MIFARE Classic encryption to invert the algorithm. The cipher uses a non-linear filter function, which is a specific component used in many stream ciphers, see [Gol67, Gro71, Key76, Ple77, Kuh88, And91].

Another vulnerability in stream cipher designs is persistent and overlapping cipher bits between internal successor states. Chapter 6 shows that such a weakness can enable a very optimized way to invert the non-linear function. More specifically, each second successor state overlaps with 19 of the 20 related input bits to the function. This means that only one bit needs to be guessed for each elimination iteration. This makes the computational complexity that is required to recover the secret key in this example negligible. The details of such guess-and-determine-attack is further explained in Section 3.2.4.

2.5.2 Authentication protocol

There are several symmetric authentication protocols proposed in the ISO/IEC 9798 standard [ISO99]. Some proprietary cryptosystems derived their authentication protocol from one of the standardized protocols. However, a slightly modified version could greatly reduce the security strength of the authentication protocol.

The two-pass mutual authentication protocol presented in Figure 2.8 is vulnerable to a replay attack. In this scheme both responses are derived from challenge \mathcal{A} , which was merely introduced by *Alice*. With only one authentication attempt from *Alice*, consisting of a legitimate challenge-response pair, it is possible to impersonate *Alice*

and authenticate again with *Bob* without any knowledge of the secret key. A complete example of such an attack is presented in Section 3.1.2.

The three-pass mutual authentication similar to the one illustrated in Figure 2.9 is generally a secure authentication solution. However, there are specific requirements to avoid authentication vulnerabilities. The encrypted messages should be properly chained to each other as described in Section 2.3.2. When not properly handled it allows an adversary to mount a blocking (Section 3.1.6), injection (Section 3.1.5), reflection (Section 3.1.4) attack.

A well-designed three-pass mutual authentication is by definition much more secure than a single or two-pass mutual authentication. However, some issues could still undermine the security when not properly handled. A well-known example is the way transmission timing restrictions are defined and enforced. When these are not properly restrained, it allows an adversary to mount a relay attack, see Section 3.1.3 for more details.

An authentication protocol with serious weaknesses is embedded in the MIFARE Classic cryptosystem that is described in Chapter 6. According to the specification, the authentication protocol is compliant to the ISO/IEC 9798-2 standard [ISO99]. However, the random challenges were substituted by easy-to-predict values with very low entropy. This modification undermines the protection against the reuse of authentication attempts. Attacks that exploit such a weakness are further described in Section 3.1.2.

It is important to carefully select the function which is responsible for computing a diversified key, which is introduced in Section 2.4.1. For instance an algorithm which is invertible allows an adversary to reconstruct the master key by simply deriving it from the diversified key. This particular problem is the case in the iClass cryptosystem which is introduced in Chapter 9. With only a limited computational complexity the cryptosystem allows the adversary to recover the master key from the diversified key.

2.5.3 Cipher initialization

The initialization of the internal state in a stream cipher is a crucial part in terms of its security. It should allow session-independent initialization of all the components using the full entropy of random challenges. When internal state bits are persistent between sessions, it allows an adversary to link these bits between various sessions and thereby compromise its independency. Although a cipher could be secure in terms of component design and ciphertext production, the extra dimension of linking the internal states of multiple sessions could completely undermine the protection of the cipher design.

Such a serious initialization weakness is identified in Chapter 7 for the Hitag2 cipher. It shows that one third of the internal cipher state is exactly the same after each cipher initialization. The chapter demonstrates it is not very difficult to mount a practical attack when such a weakness is present.

In an unfortunate case when a full or partial internal cipher state is recovered, it fully depends on the cipher initialization if the process can be inverted to expose the original secret key. When non-linear steps are involved in the computation of a successor state, it might be so that it is not trivial to undo or roll-back the initialization. For example, when each cipher state has a second state that merges to the same successor state, there must be two predecessor candidate states for each internal cipher state. In such a situation the number of possible candidates grows with a power of two for each previous step. In practice, most ciphers limit such non-linearity in the computation of the successor state. This is to avoid drastic reduction of possible cipher states after computing several successors. Sometimes a balance can be struck between those, for instance the CryptoMemory cipher introduced in Chapter 8 uses modular addition to achieve controlled state merging.

2.5.4 Encryption oracle

Many proprietary cryptosystems provide the encryption functionality through a black box interface. This allows the system to keep the cryptographic algorithm secret, yet still available to encrypt user-defined plaintext. However, a cipher that allows invocation of its encryption functionality can be used as an encryption oracle. It allows encryptions of arbitrary plaintexts at the adversary's request.

Especially in the case of a stream cipher this is a serious cryptographic weakness. The produced keystream is computed independently from the plaintext and delivered one bit at the time. Therefore, it allows a precise and controlled invocation of its encryption functionality.

Chapter 7 describes the Hitag2 cryptosystem where an adversary, which has access to genuine authentication attempts, can invoke the encryption functionality at will. It allows influence over the cipher initialization process and exposes the encryption oracle vulnerability to invoke and observe differences in the output. The exploit of such vulnerability is called a differential attack, see Section 3.2.5 for more details on this subject.

2.5.5 Authorization model

To compromise the authorization model, an adversary often tries to mount a privilege escalation attack. It exploits a weak spot in the authorization model that allows the adversary to perform an operation with no (or insufficient) credentials. It often involves a carefully crafted path that exists of many operations combined in such a way that the final state is undefined by the authorization model. Some systems show an error or throw an exception without resetting the already achieved authorization. For example, chapter 9 shows a situation where the adversary can access the key-update functionality on a protected area. It first performs an authentication with default credentials on a different non-initialized area and then forces a context change to the protected area.

Some authorization models define a maximum number of failed authentications and locks down when this number is exceeded. It restricts the initiating party to try to learn from the receiver responses to the chosen authentication attempts. Such a mitigation requires a solid and trustworthy communication channel. The usability of such a system is affected when the system is locked when the failed authentications were only triggered by a set of transmission errors. It would not be difficult for an adversary to mount a Denial-of-Service (DOS) attack, see Section 3.1.6. By remotely initiating a sequence of incorrect authentication attempts the receiver immediately locks down any further legitimate access.

Chapter 8 describes a system where such a security mechanism is enabled. It is remarkably easy to bypass the mitigation in this particular example. The decrementing counter that stores the count of failed authentications is only updated after the authentication is checked. By observing the power traces of these chips it is possible to detect if the counter is going to be written to, before it is actually stored. This means that when disabling the power supply just before such a write is performed, it is still possible to initiate unlimited authentication attempts.

Weaknesses in an authorization model are often difficult to detect automatically. There are ways to examine a design automatically by exploring the state machine of a system [Tre08]. However, in complex cryptosystems the combinations of inputs could lead to a state explosion which exceeds the practicality of performing such a test. A better alternative is to formalize the complete state machine and implement it accordingly. This way the implemented state transitions are verifiable and provably correspond to the original formalized design.

2.5.6 Communication protocol

To use and deploy a cryptosystem, it is necessary that the means of communication are formally specified in a communication protocol. Such a protocol defines the physical requirements of the communication channel and the message specification. The protocol features must be adjusted to the encryption techniques of the cryptosystem. When a communication protocol is designed without taking the cryptographic properties into account, weaknesses could arise when they are plainly used together without any adjustment to each other.

Chapter 7 shows that the communication protocol of Hitag2 allows adversaries to extend a very small piece of a known plaintext with an unlimited set of redundant messages. While such a feature adds support to the reliability of the communication channel, it undermines the security of the cryptosystem by providing a remotely operational encryption oracle. More details about attacks with an encryption oracle are explained in Section 2.5.4.

Another technique that improves communication integrity is to send additional verifiable information about the actual message content, which are called integrity checks. Well-known examples are parity bits and Cyclic Redundancy Check (CRC)

checksums. These integrity checks contain redundant bits which summarize the content of the message. The message is communicated together with these integrity checks, which allows the receiving party to verify (to some extent) if the received message was delivered in the correct manner. When an anomaly is detected, the receiving party could immediately discard the message and request a re-transmission. Such a technique greatly improves the integrity of the communication channel.

However, it is a complex task to combine integrity checks together with an encryption layer. For instance, it is convenient to wrap the encryption completely around the communication protocol. This inherently adds information-leaking weaknesses to the system [Kra01]. However, an adversary may not have any knowledge about the plaintext, the integrity checks still educate the adversary about bit-dependencies in the plaintext message. In some systems like Global System for Mobile Communications (GSM), the content bits are almost transmitted twice within one message. Such redundancy allows an adversary to mount a ciphertext-only attack, since the recovered secret key is just verifiable on the message structure without requiring any knowledge of the plaintext itself.

There are several systems where parity bits are encrypted in an insecure way. One of the most prominent examples is the MIFARE Classic cryptosystem described in Chapter 6. It uses a stream cipher that produces a pseudo one-time-pad, see Section 2.3.2 for more details. The messages of the system are encrypted with unique bits from the one-time-pad. However, the parity bits are encrypted with the same keystream bit that was already used to encrypt the previous bit. In short, the one-time-pad is used twice, which goes directly against the basic security principles of a one-time-pad.

Adding encryption or integrity checks at a later stage is not just adding an additional layer that wraps around an existing communication protocol. The protocol needs to be redesigned carefully by considering all security and integrity requirements.

2.5.7 Implementation

Apart from the complex design issues, the security of a cryptosystem strongly relies on the way it is implemented. Only when the implementation strictly follows all facets of the design specification could the target security strength could be achieved. Besides the design specification, the implementation should implement the best-practices in secure software and hardware development. Examples of such are, input validation, verifiable formal function annotations, thorough inspection for state transitions with undefined behaviour, extensive adversary model with attack trees and countermeasures against invasive hardware attacks.

A cryptographic algorithm or authentication protocol often relies on a specific input format. When the cryptosystem allows arbitrary length messages or messages with invalid and malicious content, it could lead to a cipher weakness and trigger unexpected and security compromising behaviour. An interesting example that uses

such a weakness is described in Chapter 6, where it is shown that the internal state of the cipher can be updated using a malicious authentication message of arbitrary length (shorter or longer).

The design of a cryptosystem can be very strong. However, when its implementation somehow leaks secret internal information to an adversary, the security of the complete system is considered compromised. There are various weaknesses in a hardware implementation that could lead to such information leakage. To avoid such vulnerabilities, shielded and tamper resistant hardware can help in reducing the leakage of side-channel information. More extensive information about (non)invasive attacks is presented in Section 3.3.

2.5.8 Deployment

The system integrator that parametrizes, configures and deploys a cryptosystem has to act very carefully and focus on a secure deployment. This requires special attention for secret key-management, quality of the pseudo-random number generator, configuration that relates to access conditions and authorization, compiling a fall-back scenario when security is compromised.

The key-management is of essence in a cryptosystem. A secret key should be a set of randomly distributed bits (ones and zeros). There must be absolutely no relation between different secret keys or any of the bits within these secret keys. In some systems (part of) the secret key is derived from another value. This must be a secure transition and produce a practically unique derivation. If such a derivation process can be reverted or allows an adversary to link various secret keys to each other, the security strength is affected. Some systems even specify a set of weak keys which should be avoided at all costs. Several examples of widely deployed systems that use weak keys are presented in Chapter 7.

Secure cryptographic challenges that are used by a cryptosystem highly depend on uniformly distributed random numbers. Any prediction of the challenge or part of it could lead to serious weaknesses in the corresponding authentication protocol. This could allow easy-to-exploit replay attacks as described in Section 3.1.2. An example of mounting such attack is shown in Chapter 7.

An inadequately configured cryptosystem could allow an adversary to gain access to functionality or memory that was designed and intended to be inaccessible. A clear example of such a weakness is shown in the Hitag2 cryptosystem described in Chapter 7, where incorrectly configured write-only flags could lead to serious authorization weakness which allows an adversary to just read out the secret key.

Furthermore, a poorly designed communication protocol that defines the messages between a cryptographically enabled device and an external system component can completely undermine the comprehensive security of a system. For instance, the cryptographic strength does not matter if the result of an authentication attempt is transmitted as a boolean value (*true* or *false*) to the external component. It allows

an adversary to enforce a successful authentication by simply always transmitting a *true* value.

A cryptosystem is often carefully designed to operate in a very secure way. However, it could happen that one of the cryptographic components is less secure than originally specified. It has to be an integral part of the deployment process to define a list of actions that specify migration steps and alternate solutions for each component of the cryptosystem.

Chapter 3

Attack scenarios

An adversary can mount a variety of attacks on a cryptosystem. This chapter introduces a few elementary versions of the most common complex digital attack scenarios. There are several attack categories where each specifies an adversary with a set of predefined capabilities. Certain attacks require only passive observation of the communication channel, while other scenarios require complete control of the communication channel. To clarify the different type of attacks, each section refers to concrete examples that were encountered in the course of this study.

This chapter contains three main attack categories. Section 3.1 first explores the attacks an adversary could mount on the authentication protocol and communication channel. Those attacks hardly depend on the cryptographic algorithm that is used, as they exploit insecure protocols. Next, a set of fundamental cryptographic attack techniques are explained in Section 3.2. Those are later used to greatly reduce the computational complexity of the ciphers explained in Part II. Finally, it briefly addresses the possibilities, impact and severity of hardware attacks. The latter shows that even when the authentication protocol and cryptographic algorithm are secure, a cryptosystem still might be compromised because of weaknesses in its hardware implementation.

3.1 Authentication and communication attacks

This section addresses six attack scenarios that compromise the authentication protocol and communication channel. The first two attacks do not alter the communication itself. The adversary just passively observes or actively forwards all original messages. This is followed by four attacks that deliberately violate the transmission integrity by retransmitting, reflecting, injecting or blocking messages.

To illustrate the attacks, the paragraphs include message sequence charts which elucidate the applicability and severity of the attacks. In addition to *Alice* and *Bob* a new communicating entity is introduced: the adversary named *Eve*. It is assumed that she takes control of the communication channel that exists between *Alice* and *Bob*. Control of this channel allows her to mount all attacks mentioned in this section. For each attack, the specific actions that *Eve* has to take are mentioned in the corresponding message sequence chart.

3.1.1 Passive eavesdropping

One of the most simple, yet effective ways to learn information about a cryptosystem is to observe and record the communicated messages. Such a gathering of data is called passive eavesdropping and the data that is recorded is referred to as a trace. The right hardware tools enable intercepting of the communicated ones and zeros. This does not immediately imply that a cryptosystem is broken when the communication is recorded. Contrarily, a good cryptosystem is mainly designed to defend against such an intruder. The messages should be encrypted in such a way that there is no way for an adversary to distinguish the trace from random noise.

Severe weaknesses in the cipher, authentication or communication protocol allow an adversary to compromise the security of the complete cryptosystem. There are numerous examples which show that an attack can be mounted with only passively eavesdropped communication traces. However, this often requires very serious weaknesses in the cryptography or an extremely large set of traces. The practicality of an attack that requires many traces drops significantly when they have to be eavesdropped from a remote cryptosystem. Nevertheless, a well-cited publication [SIR02] showed in 2002 that it is not so difficult to recover the secret key from a wireless network system by using only passive eavesdropping. This would take between four and six million packets. Considering the activity of the users and the network load, it could very well take up to a month to gather the required packets.

In practice it is very difficult to defend against a passive eavesdropper in terms of avoiding it. In a wired system, the electrical signals and load of observers could be monitored. However, a malicious entity that only receives wireless communication is almost infeasible to detect. For instance, a third party (e.g. the adversary) can silently decode the same Radio Frequency (RF) signals that are transmitted. Therefore, it is wise to assume that in a wireless set-up the messages can be intercepted by a malicious party at all times.

There are many secure systems that are resilient against passive eavesdroppers. In such a system, an adversary may gather all the communicated messages, yet gain no advantage in learning them. To achieve this, the systems must rely on a secure combination of a strong cryptographic algorithm (Section 2.2.3), formalized and proven authentication protocol (Section 2.4.1), a well-designed communication protocol (Section 2.5.6) and a correct and well-protected implementation (Section 2.5.7).

3.1.2 Relay attack

A relay attack is a kind of man-in-the-middle attack, where an adversary only forwards the communicated messages both ways. The goal of such an attack is to transparently relay the messages between an initiator and the device of a victim to unknowingly initiate a (different) communication channel. An adversary could use a relay attack to significantly increase the communication distance and thereby change the properties of the original communication channel. An access control system that is designed to work

in proximity distance does not necessarily provide the same security properties when it is available from a larger distance. Even when the used encryption is strong enough to resist cryptography attacks, the lack of physical distance boundaries could still compromise the security of the system as a whole. For instance, some access control systems allow authentication signals to be relayed undetectably over a larger distance. A relay attack breaks their security features without requiring any knowledge of the used cryptography or authentication protocol. The concept of a relay attack was introduced decades ago in [DGB88], when radio transceivers like cell phones were not so common to use. The authors explained the concept with a practical example that involves collaborating criminals that use a Mafia-owned restaurant. Therefore, they introduced a fitting name and referred to such an attack with the term *Mafia fraud*.

The original transmission is intercepted at the side of the sender and transparently mirrored at the side of the receiver. To achieve this, two adversaries (Eve_1 and Eve_2) are required. They both use a specific hardware set-up which enables them to communicate with the genuine parties ($Alice$ and Bob). Furthermore, the adversaries need on both sides a wireless transceiver which provides a fast and reliable bidirectional communication channel with each other. The transceiver acts as a relay station, which simply forwards the communication between $Alice$ and Bob . Figure 3.1 shows a schematic overview of a relay attack that is mounted on the three-pass mutual authentication protocol which was introduced in Section 2.4.1.

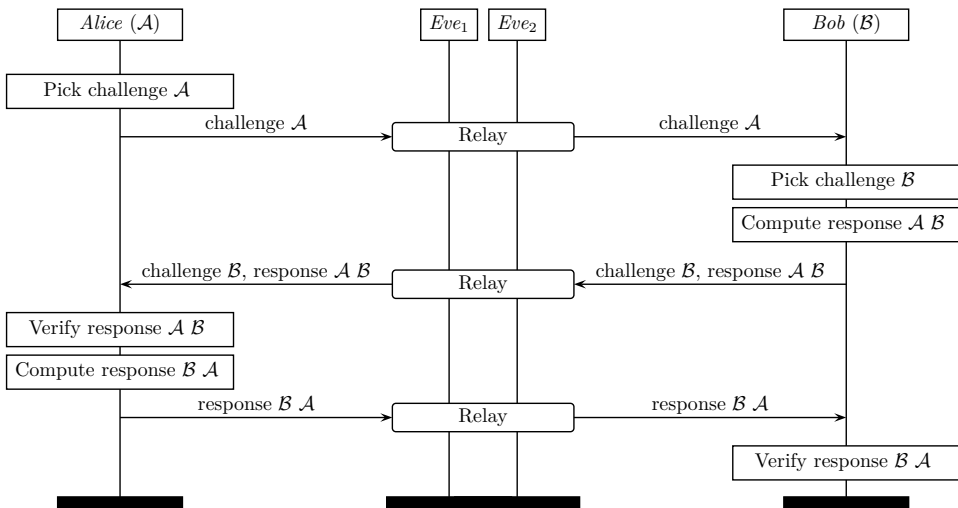


Figure 3.1: Relay attack on a three-pass mutual authentication protocol

A relay station might consist of simple consumer hardware that supports wireless communication, like cell-phones or laptops with wifi-networks [LCA⁺04, FHMM10, FHMM11]. However, most communication protocols require a robust hardware set-up which achieves fast and reliable communication. Several practical implementations are proposed in the literature that rely on optimized and dedicated embedded hard-

ware [KW05, Han05, KW06, HMM09, FDv11].

Figure 3.2 illustrates a practical relay attack [VdKGG12] that is performed on an access control system that requires authentication with an Near Field Communication (NFC) contactless smartcard. The adversary emulates such a smartcard at the access control gate which receives the instructions that need to be relayed to the genuine smartcard. On the other side, a second adversary impersonates with a malicious NFC reader the access control gate and interrogates the genuine smartcard with the relayed instructions. The relay stations in the middle transmit the signals over a larger distance to vastly extend the original proximity communication range. More detailed information about the mentioned technologies are available in Section 5.1.4 and Section 5.1.3.

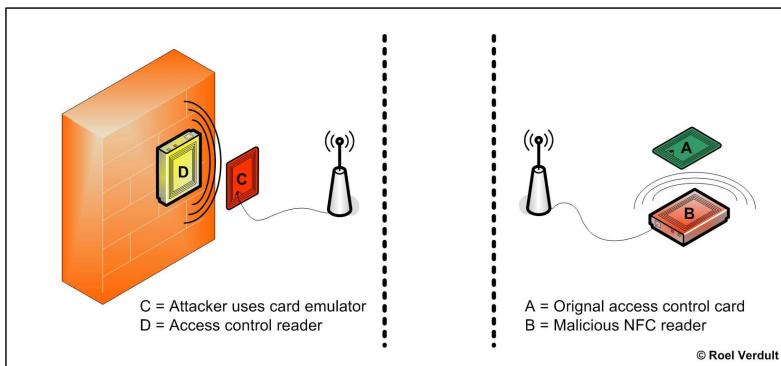


Figure 3.2: Relay attack on an access control system that uses an NFC smart card

In an advanced relay attack the complete communication is remotely reproduced. This includes the complete messages, header bits, similar modulation and encoding techniques, signal strength, angle of emission and other physically detectable properties of the transmission. In theory, there is only the time that elapsed between two transmissions that cannot be reproduced. This follows from the assumption that information cannot travel at a speed greater than the speed of light [HK05].

In practice however, is it very hard to measure this with the required accuracy and detect relayed transmissions. This is especially the case when a transmission between the relay stations completes within a smaller time-frame than it takes to transmit the original data bits. For example, when the period of the original transmission carrier wave takes more time than needed to relay the data bits. Such a set-up could be easily achieved by using a substantially higher transmission frequency between the relay stations. A well-known example in the literature [FDv11] demonstrates a practical relay attack on Passive Keyless Entry and Start (PKES) systems, which are embedded into most modern vehicles. The study shows that it is feasible to relay the signal within nanoseconds, while the original transmission takes at least a few microseconds per bit that is communicated. The accuracy of the receiving hardware in cars is unfit to detect a transmission delay of only a few nanoseconds.

In response to these kinds of attacks, several mitigating measures were proposed in the literature [BC94, HK05, KAK⁺09, KA09, Rv10]. The common ground of these publications is the extension of the authentication protocol that keeps track of the communication timings. Such schemes are referred to as distance bounding protocols, a term which was introduced by [BC94] in the early nineties.

3.1.3 Replay attack

One of the most basic attacks that an adversary can mount on a vulnerable cryptosystem is a replay attack. In such an attack, the adversary reuses ciphertext, recorded from an early eavesdropped session, and replays (retransmits) it to the cryptosystem. Protection against replay attacks is a very elementary requirement for a secure cryptosystem. The security of a system is seriously compromised when a replayed message is not properly handled. For instance, when there is no counter-measure in the authentication protocol that protects against replay attacks, it allows (re)authentication with the previous intercepted credentials as shown in Figure 3.3.

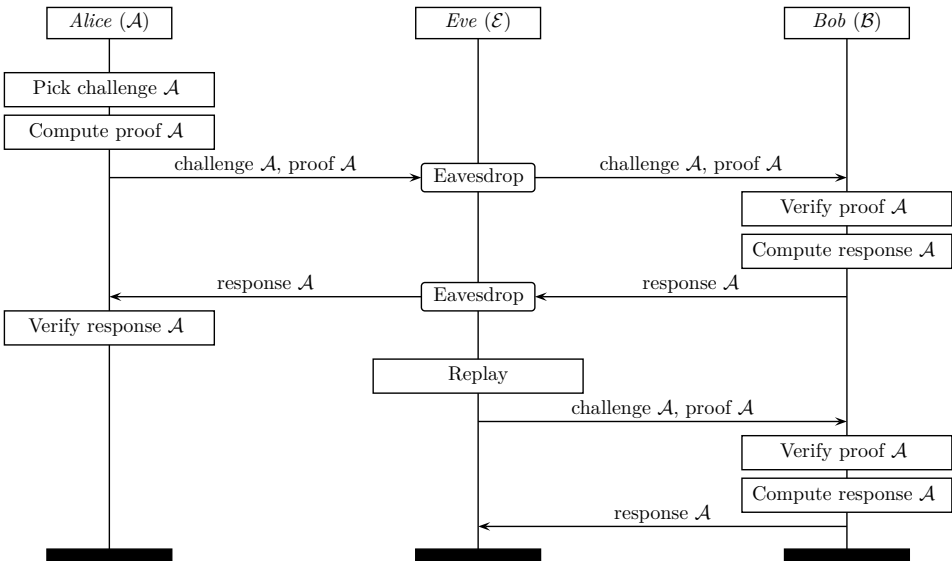


Figure 3.3: Replay attack on a two-pass mutual authentication protocol

The first authentication protocol, presented in Section 2.4.1, is vulnerable to a replay attack on *Bob* by an adversary that impersonates *Alice* and replays the same challenge \mathcal{A} which *Alice* used to authenticate before. This problem is mainly introduced by the fact that *Bob* does not send a challenge to *Alice*. Therefore, the cryptographic session that is created after authentication is not unique. It needs to be properly randomized by both parties every time they negotiate for a new authentication. Figure 3.3 shows a replay attack performed on the two-pass mutual authen-

tication protocol that was introduced in Section 2.4.1. The adversary *Eve* eavesdrops the first message that was transmitted by *Alice* and uses it later to re-authenticate with *Bob*. Notice that *Eve* did not learn the secret key and therefore cannot verify the response \mathcal{A} from *Bob*. However, assuming this message was valid, by using the same challenge, *Eve* can perform a successful authentication with *Bob* at any moment in time.

Several decades ago, various mitigation techniques were proposed in the literature to protect against replay attacks [FNS75, Ken77, PK79, Den82, BBF83]. However, there are numerous examples of cryptosystems introduced after this discussion that are still vulnerable. Chapters 7, 8 and 9 show the workings of widely-used cryptosystems that contain serious weaknesses which allow an adversary to mount various replay attacks.

3.1.4 Reflection attack

The parties in a symmetric authentication protocol prove their legitimacy to each other sequentially. However, the proofs are computed independently from each other, using the two-pass single authentication protocol from Figure 2.7 twice to validate both sides. In such a protocol the challenge-response pairs from *Alice* to *Bob* are computed in exactly in the same way as those from *Bob* to *Alice*. Because both proofs are computed from each other independently, it allows an adversary to mount a reflection attack as shown in Figure 3.4.

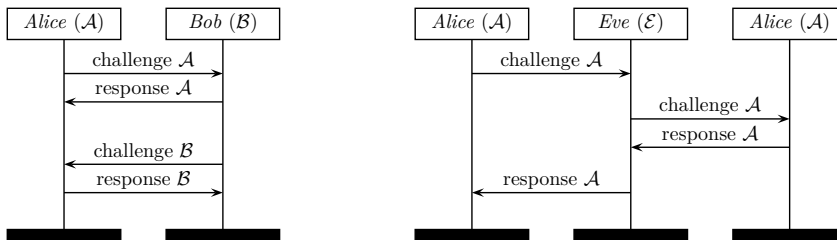


Figure 3.4: Left: Using twice a two-pass single authentication protocol
Right: Reflection attack by redirecting challenge \mathcal{A} back to *Alice*

In the reflection attack that is illustrated in Figure 3.4, the adversary *Eve* redirects (reflects) the challenge \mathcal{A} , which *Alice* picked as a challenge, back to *Alice* in another communication channel. When *Alice* responds to the challenge in the second channel, it is used by *Eve* to prove legitimacy to *Alice* in the first channel. Since *Eve* is fully authenticated in the first communication channel, the second channel can be disregarded. A reflection attack does not require knowledge about the secret key or the cryptography that is used. It exploits a weakness in the authentication protocol, as shown in Figure 3.4, which does not include countermeasures against reflection attacks. Examples of these countermeasures are the use of cryptographically linked

messages as mentioned in Section 2.3.3 or the inclusion of the sender’s identity in each communicated message.

3.1.5 Reorder and inject messages

The communication flow of the session can be compromised when it does not properly chain the messages and lacks integrity checks that verify the complete transmission. The adversary tries to tamper with the communication without directly attacking the cryptography itself. This can particularly be a problem for cryptosystems that use block ciphers, since they do not inherently keep track of the previous encryption state, see Section 2.3.3 for more details. The technique explained in Section 2.3.4 can be used to add integrity checks within a message itself. This should keep the adversary from forging new messages. However, it does not guarantee that a message with correct integrity is refused when it is reused in the same cryptographic session. Such a reuse is commonly referred to as an injection attack.

A session that contains messages which are encrypted without cryptographic chaining allows an adversary to reorder, reuse and sometimes even combine multiple messages. Such a cryptosystem allows an adversary to inject messages into the communication channel because it has no way to invalidate them. Figure 3.5 demonstrates an attack performed by *Eve* to increase the funds of *Bob* by simply reordering and injecting earlier transmitted messages by *Alice*.

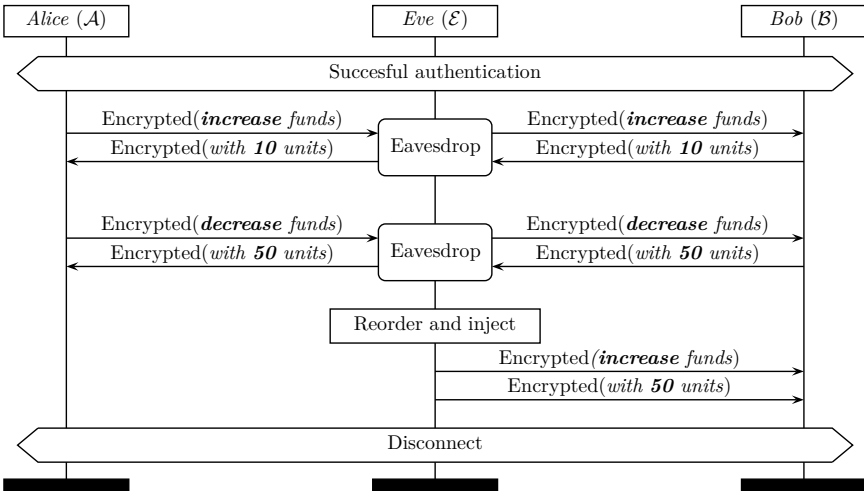


Figure 3.5: Injection attack on a protocol without proper session integrity checks

The protocol used in Figure 3.5 to increase the funds does not use any cryptographic means to ensure the session integrity. In this example the Authentication and Key Agreement (AKA) protocol is assumed to be secure and communication after the secret agreement is fully encrypted. However, the communication protocol contains

no counter-measures against adversaries that replay encrypted messages after successful authentication. Since it lacks cryptographic chaining or other measures that ensure session integrity, *Eve* is allowed to refund *Bob* without the necessity to break the encryption.

The iClass access control system presented in Chapter 9 contains a comparable integrity vulnerability. However, the integrity issues arise from unchained cryptographic messages, which are stored in memory instead of being communicated. The weakness allows an adversary to inject and interchange memory blocks within the smart card memory itself after a successful authentication between the genuine devices. Similarly, such an attack can be mounted without any knowledge of the secret key. The memory is encrypted by a secure block cipher, albeit without proper cryptographic chaining. This issue, in combination with several other weaknesses, presented in the literature [Mer10, GdKGV11, GdKGV12, KJL+13], allows an adversary to compromise the security of the system. In contrast, a well-designed cryptosystem ensures the integrity of individual messages and the whole cryptographic session, see Section 2.3.2 and Section 2.4.1 for more details.

3.1.6 Block the communication

As introduced in Section 1.1, availability is one of the key features of a secure cryptosystem. Solid reliability of a system preserves the continuity and usability. To subvert such reliability an adversary may choose to block all communication. Additionally, to alter the workings of a system an adversary might block only certain messages. The example illustrated in Figure 3.6 shows that *Eve* blocks the second pair of messages, which alters the communication flow to her advantage.

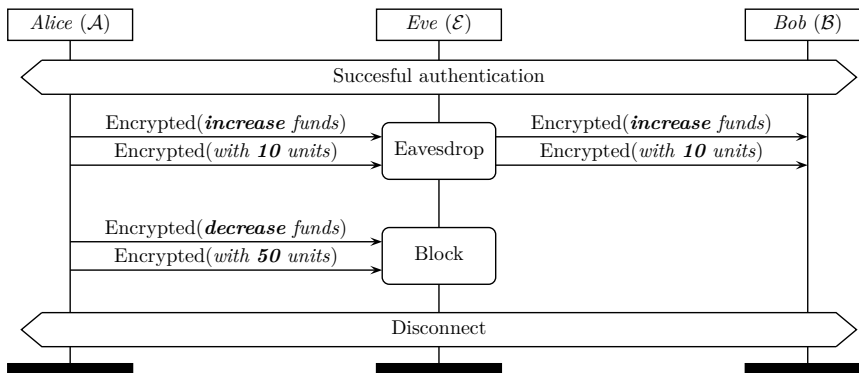


Figure 3.6: Blocking attack on the second part of the communication

The most simple form of a communication blocking attack is a complete Denial-of-Service (DOS) attack. In such an attack all the communication is blocked and remote access to the cryptosystem becomes completely unavailable. Unregulated and

professionally organized attacks on high-profile services at large institutions like electronic banking, infrastructural control centres and governmental portals have rapidly increased over the last years [SKK⁺97, WS02, MDDR04, MSB⁺06, CKBR06].

Many DOS attacks simply try to make a service unavailable, which could be motivated by several reasons such as manifesting intentional reputation damage or to blackmail the owner of the service. It differs from the previously described attacks which mainly focus on security breaches to gain access to protected information. There are, however, a few clever blocking attacks which also try to compromise the authentication or communication protocol. Although these attacks seem to be mounted much less in the wild, a protocol that is vulnerable to it might be compromised in many ways. For example, a transaction can be interrupted just before a crucial action is performed or certain remote instructions are filtered out to avoid the decrement of purchased credits like shown in Figure 3.6.

3.2 Cryptographic attacks

This section introduces seven fundamental cryptanalytic techniques which are used in cryptographic attacks, also referred to as cryptanalysis. There are many more advanced and complex cryptographic attack methodologies and techniques proposed in the literature [DM95, DKR97, BDL97, BS97, Wag99, KSW99, BW99, DS09, BKR11]. However, to maintain readability, only very rudimentary versions of the fundamental techniques are introduced. The type of cryptosystem that is mainly used to demonstrate the attack techniques is a variant of the non-linear stream cipher system, introduced in Section 2.3.2.

To mount a cryptographic attack, it sometimes requires significant computational effort to recover the secret key. The computing power that is required reflects the attack complexity, see Section 2.2.2. It is possible to generalize the computations that are required for a cryptographic attack in such a way that they can be (partially) pre-computed. Such technique is commonly referred to as Time-Memory Trade-Off (TMTO). The general idea is to split a cryptographic attack into two phases, a pre-computation phase (off-line) and active attack phase (on-line). For more information on this topic, please refer to the TMTO methods and efficient search techniques proposed in the literature [Hel80, AH88, FN91, Bab95, BPVV98, VOW99, BS00, Oec03, BBS06, BMS06, AJO08, HM13, vdBP13].

Most cryptographic algorithms described in Part II, i.e. the technical part of this study, are variations of similar stream ciphers. The introduced attack techniques are utilized in Part II to exploit various cipher weaknesses. It shows that the cryptographic strength of those cryptosystems is far from what was initially assumed.

3.2.1 Malleability attack

The specifics of a synchronous stream cipher that produces a non-linear binary sequence are explained in Section 2.3.2. It shows an illustration of a typical cryptosystem that uses such a cipher. The generated binary sequence serves as the keystream and is combined with the plaintext by applying the exclusive-or (XOR) operator. Such a cryptosystem could in principle provide a secure channel which protects the confidentiality of the data transmission. However, without further protection, the integrity of the data is not guaranteed. Additional countermeasures, such as a Message Authentication Code (MAC), can protect the authenticity of the data. Without supplementary cryptographic techniques a stream cipher system is vulnerable to a malleability attack.

During a malleability attack the ciphertext is transformed in such a way that it still decrypts to bona fide plaintext, yet satisfies the attacker's purpose. Note, that the goal of a malleability attack is not to recover the secret key. In fact, it tries to undermine the security of the cryptosystem without having any knowledge of the secret key. Figure 3.7 demonstrates a data transmission tampering of a banking application that is vulnerable to a malleability attack. A fairly small money transfer is altered by a single bit-flip and suddenly represents a very large money transfer. Even if the keystream is produced by an extremely secure stream cipher, other components of the cryptosystem might still be vulnerable. It is the strength of the entire cryptosystem that defines the actual security.

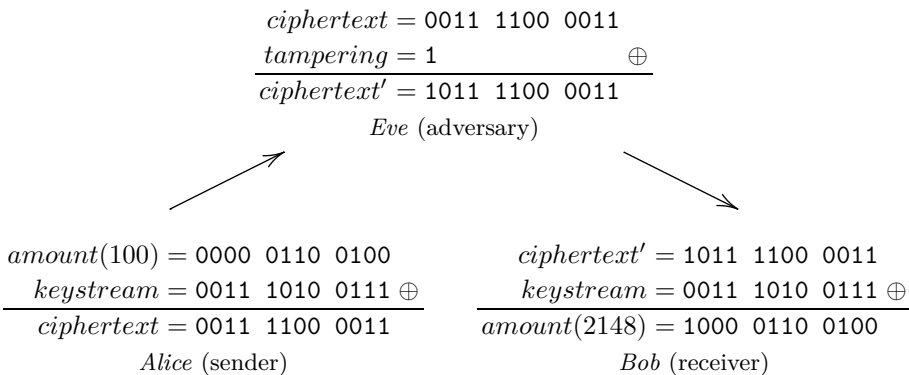


Figure 3.7: Malleability attack alters the value of a money transfer

Although a malleability attack appears to be powerful, it is not that straightforward to mount. The adversary needs to know exactly where and when the amount is transmitted to be able to tamper with it. A bit-flip at an incorrect position is meaningless and will most likely result in a corrupted transmission. Knowledge about the keystream allows an adversary to prepare more carefully crafted tampering. However, this involves reuse of the keystream, a situation which a stream cipher should avoid at all costs. As Section 2.3.2 states, it is important that the binary sequence is used

only once. To enforce this, the internal state should be initialized with a unique value each time the stream cipher is used. For instance, such a value can be derived from a secret key in combination with fresh random challenges. Section 2.4.1 shows the exchange process of random challenges during the authentication phase.

A secure exchange of fresh random challenges and a protected Initialization Vector (IV) of the cipher is not a trivial task [BG07]. In fact, several proprietary stream ciphers allow an adversary to influence the initialization of the internal state in such a way that exactly the same keystream is produced as it was generated in a previous session. A comprehensive, yet practical, example of a stream cipher malleability attack is given in [dKGHG08]. It shows how the cryptosystem, introduced in Chapter 6, is exploited in such a way that the contents of a protected smartcard can be revealed even without any knowledge of the secret key.

3.2.2 Divide-and-conquer attack

A very powerful way to reduce the complexity of a computation is to divide one big problem into two separate small problems, often referred to as divide-and-conquer. It is used in various fields within computer science to optimize the solving of hard computational problems [BS76, Ben80, Dwy87, ACG89]. Its main purpose to reduce computational complexity and generic applicability makes this technique ideally suitable to attack certain types of cryptosystems [DC94].

As mentioned in Section 2.2.2, a cryptographically secure cipher with a secret key of significant length, for instance 80 bits, takes a lot of time and resources to solve. Dividing the exhaustive search to find the secret key for such a cryptosystem would not reduce the complexity itself. It only allows the adversary to parallelize two searches, each of exactly half the complexity of the main problem 2^{79} . However, when the adversary finds a way to divide the big problem into two smaller problems, the complexity is significantly reduced.

The cipher schematic, illustrated in Figure 3.8, is used to demonstrate how to mount a divide-and-conquer attack. The 8-bit secret key $k = k_0 \dots k_7 \in \mathbb{F}_2^8$ is loaded into the internal state during initialization. Each encryption round the internal state rotates one step to the left after producing a keystream bit as output. The key space of the cipher that consists of only $2^8 = 256$ keys and keystream repeats itself after eight rotations. Therefore, this algorithm should be considered extremely insecure. However, its simplicity suits its purpose to demonstrate a divide-and-conquer attack.

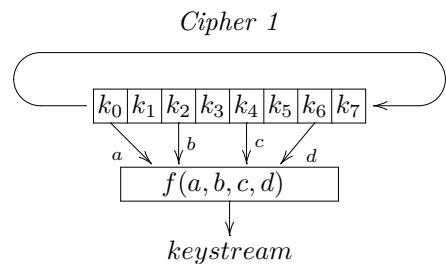


Figure 3.8: Divide-and-conquer cipher

The eight squares in Figure 3.8 represent the rotate register of the internal state. It is loaded with the ones and zeros of the secret key during initialization. The $f(\cdot)$

component is a non-linear function which uses 4 input bits (arrows from above) and produces 1 output bit, which is referred to as keystream ks . The keystream bits are defined by ks_i where i represents the number of performed cipher rotations.

Consider the first three encryption rounds as illustrated in Figure 3.9. The input to the $f(\cdot)$ function is different for the first three keystream bits ks_0 , ks_1 and ks_2 . However, the secret key bits required to produce ks_0 and ks_2 are the same, although they differ in order. To be precise, the first keystream bit is given by $ks_0 = f(k_0, k_2, k_4, k_6)$ and the third keystream bit by $ks_2 = f(k_2, k_4, k_6, k_0)$.

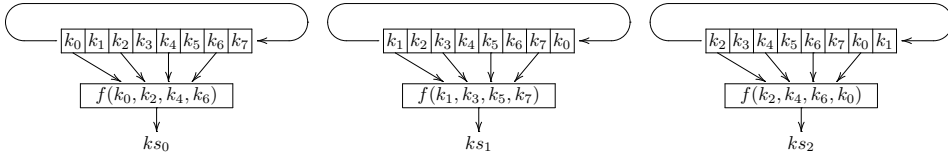


Figure 3.9: Computation of the first three encryption rounds and the derivation of ks_0 , ks_1 and ks_2

The odd and even patterns in the keystream are properties of this particular divide-and-conquer cipher. Only the even key bits k_0 , k_2 , k_4 and k_6 are required to compute the even keystream bits ks_x where $x \in 2\mathbb{N}$ defines the even positions, while only the odd key bits k_1 , k_3 , k_5 and k_7 are used to compute the odd keystream bits ks_y where $y \in 2\mathbb{N} + 1$ defines the odd positions.

Cipher 1 was designed as one big algorithm, but it can easily be divided into smaller algorithms. Figure 3.10 demonstrates how to divide the cipher, both having a key-size of exactly half compared to the original cipher.

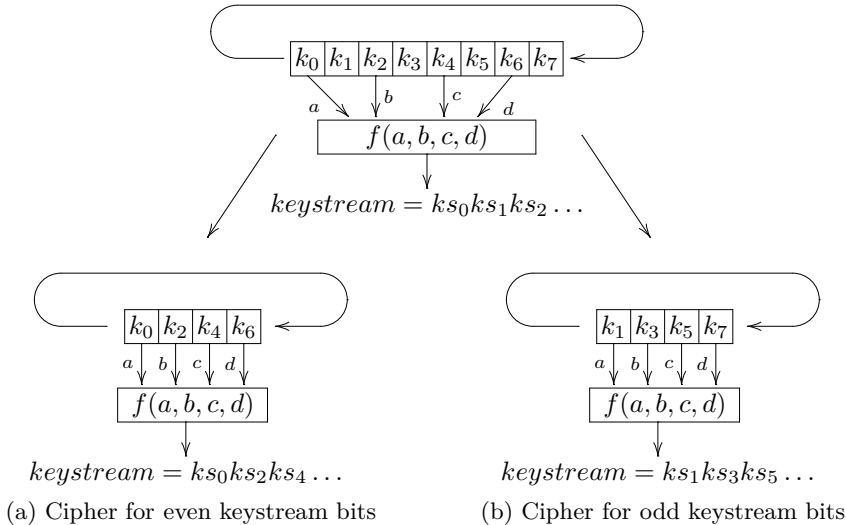


Figure 3.10: Divide-and-conquer attack by dividing odd and even keystream bits

With only half of the internal cipher state, the computational complexity for

the odd and even cipher drops to 2^4 . Consequently, the total complexity for both together is reduced to only $2 \times 2^4 = 2^5 = 32$, which is considerably less than the initial computational complexity of $2^8 = 256$. Note, that a divide-and-conquer attack does not just halve the entropy of the key space. Such division would only yield a relative small complexity reduction of $\frac{2^8}{2} = 2^7 = 128$. It is far more powerful to separate the cipher into two smaller algorithms which rely on two independent secret keys, where the two keys represent both halves of the original secret key.

3.2.3 Correlation attack

The cipher weakness that allows a correlation attack is a statically biased encryption output that is highly-influenced by certain internal state bits which are used as input. Therefore, a guess for these input bits would most likely directly influence the output bits. With the use of statistical analysis the adversary can learn information about these internal state bits.

The advantage of a correlation attack is that the adversary can significantly shrink the set of likely secret key candidates. The stronger the bias, the more information is leaked when analyzing the input and output relations. Stream ciphers in particular are often susceptible to correlation attacks. There have been numerous proposals in the literature to define fast and optimized stream cipher correlation attacks [MS88, MS89, CS91, CCCS92, MDO94, And95, CGD96, Pen96, JJ00, CJS01, JJ02, Cou03b].

In the case of a stream cipher, a correlation attack technique is often applicable on multiple sequential encryption outputs. After a successor state is reached within a stream cipher, several previously chosen bits might overlap with the input to the next encryption. Furthermore, the bias is verifiable on every produced output, which allows an adversary to correlate previous and successor states. The combination of multiple biased encryption outputs leads to more information of the complete internal state. The set with the most probable candidates is likely to contain the internal state that is derived from the correct secret key.

To explain the basic correlation attack [Sie84, Sie85], a vulnerable stream cipher is introduced in Figure 3.11. *Cipher 2* is a simple rotating stream cipher with an output component that relies on the non-linear filter function $f(\cdot)$. To increase readability the function definition is also given in the corresponding component of the figure.

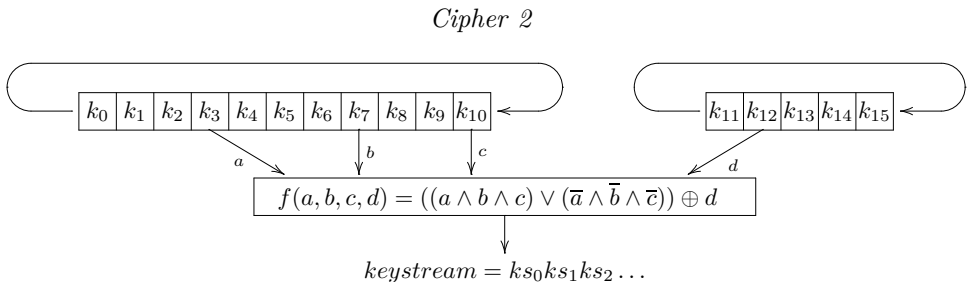


Figure 3.11: Stream cipher with correlation weakness, initialized by secret key k

The cipher illustrated in Figure 3.11 is similar to the cipher introduced in Section 3.2.2 in terms of computing its successor and the way it produces encryption bits. However, it consists of two separate rotating registers and has a larger internal state that is initialized by 16 secret key bits. Furthermore, it does not suffer from the same weakness that allows separation between odd and even bits output. Instead, it employs a non-linear filter function which generates output that is statistically biased in correlation to the input. The function $f(\cdot)$ is given by Definition 3.2.1.

Definition 3.2.1. Define the statistically biased non-linear filter function $f: \mathbb{F}_2^4 \rightarrow \mathbb{F}_2$ as

$$f(a, b, c, d) = ((a \wedge b \wedge c) \vee (\bar{a} \wedge \bar{b} \wedge \bar{c})) \oplus d$$

The filter function takes four input bits (a , b , c and d) and produces one output bit by evaluating the specified non-linear equation. The boolean table which represents the input-output relation of $f(\cdot)$ is illustrated in Figure 3.12.

$abcd$	$f(a, b, c, d)$	$abcd$	$f(a, b, c, d)$	$abcd$	$f(a, b, c, d)$	$abcd$	$f(a, b, c, d)$
0000	1	0100	0	1000	0	1100	0
0001	0	0101	1	1001	1	1101	1
0010	0	0110	0	1010	0	1110	1
0011	1	0111	1	1011	1	1111	0

Figure 3.12: Boolean input-output table that corresponds to Definition 3.2.1

The bias of the function is easily determined by looking at its input-output relation showed in Figure 3.12. Although $f(\cdot)$ produces a balanced output, which means that half of the produced output bits are zero and half of them are one, input bit d has significantly more influence on the output than the input bits a , b and c . Note the four marked rows in the input-output table illustrated in Figure 3.12 which point out where the input bits abc are equal to each other.

In 12 out of the 16 inputs for $abcd$, the output bit ks is exactly the same as the input bit d . Therefore, it is valid to conclude that on average with randomly distributed input bits, in $\frac{3}{4}$ of the cases $ks = d$, while only in $\frac{1}{4}$ of the cases $ks \neq d$. This property makes the second rotation register much more influential to the value of the produced keystream bits.

Assume an adversary recovers the first keystream bit ks_0 . She knows that ks_0 was computed by the function $f(\cdot)$ with the arguments $f(k_3, k_7, k_{10}, k_{12})$. Therefore, she can determine with probability $\frac{3}{4}$ the correct value for input d , which is for ks_0 the thirteenth secret key bit k_{12} . Although this reduces the set of likely candidate keys significantly, the power of a correlation attack is best demonstrated when it is applied multiple consecutive times.

Subsequently, consider a set of 16 contiguous keystream bits $ks_0ks_1 \dots ks_{15}$. The secret key bits $k_3k_7k_{10}k_{12}$ are used to compute ks_0 . Likewise, after five rotations the second register is completely rotated and the secret key bits $k_8k_1k_4k_{12}$ are used for

ks_5 . According to Definition 3.2.1, the fourth bit d is the most influential input bit. For both cases, ks_1 and ks_5 , the fourth input bit to $f(\cdot)$ is the secret key bit k_{12} .

Let us combine the probability that $\frac{1}{4}$ of the time $k_{12} \neq ks_0$ and that also $\frac{1}{4}$ of the time $k_{12} \neq ks_5$. If $ks_0 = ks_5$, the adversary can assume that the average probability of $ks_0 = ks_5 \neq k_{12}$ significantly drops to $\frac{1}{4} \times \frac{1}{4} = \frac{1}{16}$. This is because it is highly unlikely that in both cases the (independent) input bits a , b and c are equal to each other. A correlation attack that focusses on a combination of keystream bits provides a much greater advantage. The probability of a correct key-bit guess can be derived from the product of all probabilities that correspond to an independent verifiable statistical bias in the cipher.

The correlation attack, mounted on the output ks_0 and ks_5 , is also valid for many more bits from the recovered keystream set. For every pair of keystream bits which are five positions apart ($ks_i = ks_{i+5}$ where $i \in \{0, \dots, 10\}$), the average probability that they are exactly the same as the corresponding bit from the secret key is $\frac{15}{16}$. The properties of this cipher allow an even better (but a more complex) correlation attack. However, the purpose of the previous description is to present a simple and explanatory example.

Chapter 8 introduces a widely deployed cipher that is vulnerable to a correlation attack. It shows how to recover the secret key using a statistical bias in the output bits selection function that is part of the cryptographic algorithm. Although it is slightly different from the example presented here, the attack also exploits a correlation weakness to learn information about the internal state bits.

3.2.4 Guess-and-determine attack

Despite several well-known historical recommendations in the literature [Kuh88, And91, Gol96], many proprietary stream ciphers do not use their complete internal state to compute a keystream bit. Such cipher design allows an adversary to mount a guess-and-determine attack. This attack abuses the fact that only a few internal state bits are defined as input to the filter function. Therefore, only these bits determine the value of the computed keystream bit.

To mount such an attack, an adversary only guesses *used* bits, computes the output and evaluates it against the corresponding keystream bit that was recovered from an eavesdropped trace. The evaluation immediately leads to a contradiction for many of the guessed candidates. After evaluation, the internal state is updated accordingly and only additional required bits are guessed.

There are several impressive, yet slightly complex, examples in the literature [HR00, HR03, GM04, ZF06, Bog07a, Pas09, FLZ⁺10] that show the feasibility of guess-and-determine attacks on various cryptographic algorithms. To avoid the details of complex and optimized techniques another elementary stream cipher is introduced, see illustration in Figure 3.13.

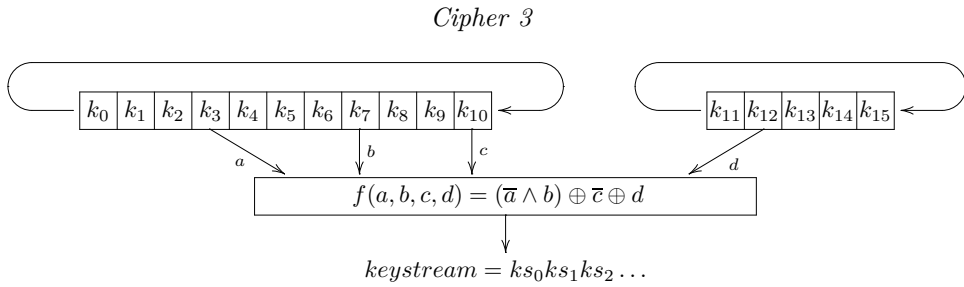


Figure 3.13: Non-linear stream cipher, initialized by secret key k

Cipher 3, illustrated in Figure 3.13, is a clear example of a stream cipher that is vulnerable to a guess-and-determine attack. The output component embeds a balanced non-linear filter function which is given by Definition 3.2.2.

Definition 3.2.2. *Redefine the filter function $f: \mathbb{F}_2^4 \rightarrow \mathbb{F}_2$ as*

$$f(a, b, c, d) = (\bar{a} \wedge b) \oplus \bar{c} \oplus d$$

The boolean table that represents the input-output relation of $f(\cdot)$ from Definition 3.2.2 is shown in Figure 3.14. Note, that c and d have always influence on the computed keystream bit, while the influence of a and b depend on one each other.

$abcd$	$f(a, b, c, d)$	$abcd$	$f(a, b, c, d)$	$abcd$	$f(a, b, c, d)$	$abcd$	$f(a, b, c, d)$
0000	1	0100	0	1000	1	1100	1
0001	0	0101	1	1001	0	1101	0
0010	0	0110	1	1010	0	1110	0
0011	1	0111	0	1011	1	1111	1

Figure 3.14: Boolean input-output table that corresponds to Definition 3.2.2

The adversary searches for a weakness in the cipher state transitions that allows a guess-and-determine attack. She starts by examining the corresponding cipher structure and the specified input bits which are required to compute the non-linear function. As mentioned in Section 3.2.3, the cipher illustrated in Figure 3.11 enables the adversary to guess only a few bits of secret key per computed output.

The cipher requires only four inputs bits at a time to compute one keystream bit. The 16 bit long secret key initializes an internal state. Therefore, the internal state has an entropy of 16 bits. Nevertheless, if only four bits of the internal state are used each time to compute a keystream bit, it makes no sense to increase complexity and guess all the bits at once. Especially, when the guessed values are not used to compute the next output bit. The power of a guess-and-determine attack rests itself in guessing each time only the bits that are actually used and make sure their output do not contradict with the keystream.

An adversary can easily perform an exhaustive-search over all 16 different input bits $abcd$ which are shown in Figure 3.14. Next, she computes the result for each

$f(a, b, c, d)$ and tests the output against one of the recovered keystream bits. This would allow her to distinguish good and bad candidates.

Function $f(\cdot)$ is balanced, so it evaluates for all 16 different values of $abcd$ to eight times to a zero and eight times to a one. Therefore, when the keystream bit is a zero, the eight $abcd$ candidates, which lead to a one, are disqualified (also referred to as eliminated).

After computing $f(\cdot)$ and performing one rotation on both registers, the cipher successor state is reached. The next keystream bit is computed with four different secret key bits. The adversary again uses the same technique and eliminates incorrect $abcd$ candidates for the second keystream bit. The computational complexity drops significantly when it is possible to evaluate the input for each consecutive keystream bit independently from the others.

Assume an adversary recovered 16 consecutive keystream bits $ks_0ks_1\dots ks_{15}$, let us examine how she can mount a guess-and-determine attack on the cipher illustrated in Figure 3.11. To clarify the secret key bits which are used to compute a keystream bit, the first 5 evaluations of the filter function $f(\cdot)$ are specified in Figure 3.15.

The four secret key bits $k_3k_7k_{10}k_{12}$ are used to produce the first keystream bit ks_0 . The adversary needs to guess all $2^4 = 16$ candidates for the 4 secret key bits $k_3k_7k_{10}k_{12}$. After elimination, only half of the candidate set survives the test. So, each guess of 4 bits allows the adversary to eliminate 1 bit of entropy. This leaves the adversary after 2^4 computations with a smaller set of $\frac{2^4}{2} = 2^3$ possible candidates.

Evaluation of the second keystream bit ks_1 requires the adversary to guess another 4 bits of the secret key, this time for $k_4k_8k_0k_{13}$. Again, only half of the candidate set survives. These four secret key bits are completely different from those used to compute ks_0 . In fact, the adversary guessed now 8 independent secret key bits, namely $k_0k_3k_4k_7k_8k_{10}k_{12}k_{13}$. Even so, she only has $2^3 \times 2^4 = 2^7$ candidates after guessing the bits that compute ks_1 . In contrast to the regular 2^8 candidates one should consider when 8 bits are unknown. It becomes even better, after computing the keystream bit for all 2^7 candidates, half of them are once again eliminated. It leaves a set of only 2^6 possible candidates with values that represent 8 bits of the secret.

The adversary applies the same technique on the third keystream bit ks_2 . She guesses 12 secret key bits and after 2^{10} computations only 2^9 possible candidates survive. The fourth keystream bit ks_3 is computed with the secret key bits $k_6k_{10}k_2k_{15}$. Note that the secret key bit k_{10} was already guessed to compute the first keystream bit ks_0 . It introduces two (partially) independent problems, a property which is similar to the divide-and-conquer attack described in Section 3.2.2.

Evaluating of ks_3 is rather special, this time the adversary only has to guess three bits instead of four. After guessing the bits and before evaluating the candidate set

$$\begin{aligned} ks_0 &= f(k_3, k_7, k_{10}, k_{12}) \\ ks_1 &= f(k_4, k_8, k_0, k_{13}) \\ ks_2 &= f(k_5, k_9, k_1, k_{14}) \\ ks_3 &= f(k_6, k_{10}, k_2, k_{15}) \\ ks_4 &= f(k_7, k_0, k_3, k_{11}) \end{aligned}$$

Figure 3.15: First five evaluations of $f(\cdot)$

against the keystream, the size of the candidate set is 2^{12} . Since a computation has to be performed for each of the candidates in this set, the total attack complexity grows to 2^{12} computations. Nevertheless, after evaluating ks_3 , the adversary guessed 15 secret key bits and compiled a set of only 2^{11} candidates.

The last secret key bit k_{11} , which is still not considered, is required to compute the fifth keystream bit ks_4 . Guessing k_{11} only mildly increases the computational complexity of the attack. The candidate set is first doubled (again) to 2^{12} and after 2^{12} computations immediately halved again to 2^{11} candidates. Testing the candidates requires in total two times the largest computation of 2^{12} plus some insignificant and negligible smaller computations which were performed on the smaller candidate sets. This results in a total attack complexity of $(2 \times 2^{12}) + 2^{10} + 2^7 + 2^4 = 9360 \approx 2^{13}$ instead of the supposed $2^{16} = 65536$ computations.

Although the computational complexity is reduced significantly in this example, further (more complex) optimizations of the guess-and-determine strategy could improve this attack even more. Such techniques are not further explored in this section, although they are further addressed in the cryptographic attacks described in Chapter 6 and Chapter 7.

3.2.5 Differential cryptanalysis

The Data Encryption Standard (DES), introduced in 1977, was considered theoretically secure for many years. This confidence lasted until 1991 when Biham and Shamir showed in [BS91] that it is possible to cryptographically attack a cipher that is similar to DES. They used a new attack technique which they introduced as *differential cryptanalysis*.

The company IBM, responsible for the initial design of DES, claimed in [Cop94] that they were already familiar with this particular attack technique for more than 15 years. Furthermore, they explained to have used their knowledge to prevent differential cryptanalysis on DES as much as possible.

Nonetheless, two years later, Biham and Shamir published another article [BS93] in which the authors specifically attacked the cryptographic algorithm of DES. As mentioned in Section 2.2.3, the attack requires an enormous amount of gathered data and is therefore considered purely theoretical. Their publications inspired many fellow academics to further explore and optimize the differential attack technique both for block ciphers [LMM91, CV95, AC09] and for stream ciphers [Din94, Mul04, WP07].

The applicability of differential cryptanalysis highly depends on the possibility to gather a set of similar encryptions which differ only to a certain extent. A straightforward approach would be to find a way that directly influences and only slightly changes the internal state of the cipher. To apply such a technique in practice, often additional components of the cryptosystems are used to intentionally create the desired difference in the internal state. Examples of such components are the internal state initialization procedure, key diversification schemes and random number

generators. With control over these components an adversary can often predict and pre-compute the desired changes.

The initialization procedure of the cipher might allow an attacker to specifically change one internal state bit at a certain position. Such a minor change could lead directly to a different output which indicates the changed bit is a significant input to the filter function. Likewise, when the change does not influence the corresponding keystream bit, it reveals that the bit is an insignificant input to filter function.

Consider the non-linear stream *Cipher 4*, illustrated in Figure 3.16. It uses the non-linear filter function from Definition 3.2.2. Although the workings of the cipher are similar to that of *Cipher 3*, the internal state is initialized in a different way.

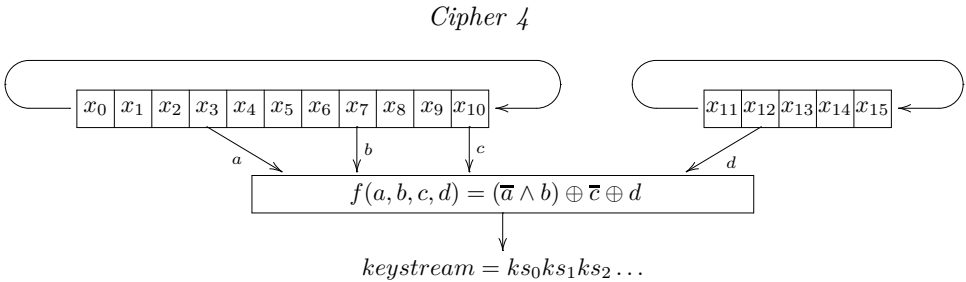


Figure 3.16: Non-linear stream cipher that initializes a fresh internal state with the key k and nonce n by $x_i \leftarrow k_i \oplus n_i$ where $i \in \{0, \dots, 15\}$

Section 2.4.1 shows that many cryptosystems use random challenges to add freshness to the encryption. In this example the challenges consist of a 16-bit random value which is called the nonce n . Furthermore, the bits of k are not just placed into the internal state of the cipher like *Cipher 3*. Instead, they are loaded with the exclusive-or (XOR) of the secret key k and the nonce n . To be precise, the initialization is performed by $x_i \leftarrow k_i \oplus n_i$ where $i \in \{0, \dots, 15\}$.

Assume two additional attack vectors on the cryptosystem. The adversary has complete control over the nonce n and secondly, she can invoke and observe plain keystream from an authentication attempt as often as she likes. This allows her to perform differential cryptanalysis in the following way. First, she authenticates using a nonce n and stores the first computed keystream bit ks_0 . Next, she flips the fourth bit n_3 from the nonce n which gives her an alternative nonce n' . Then, she re-authenticates with the nonce n' and compares the computed bit ks'_0 against the previously stored keystream bit ks_0 .

After the two authentications, the adversary did not recover the actual value of x_3 . However, she does know that the difference between the internal states of the two authentication sessions is a negation of the bit x_3 . Furthermore, the bit x_3 represents input a to the filter function, which in turn is used to compute the first keystream bit for both authentications ks_0 and ks'_0 . Note, that input b , represented by x_7 , was not altered by the nonce since $n_7 = n'_7$. The same holds for input c and d , since the fourth bit of the nonce was changed.

A closer look at the input-output relation of the filter function (see Figure 3.14) reveals that the value of input a only matters if input b is *true*. The first keystream bit is computed with x_3 as input a and x_7 as input b . Consequently, x_3 only influences the keystream bit ks_0 when $x_7 = 1$. Therefore, if $ks_0 = ks'_0$ the adversary can conclude that $x_7 = 0$. Likewise, the contrapositive is valid as well, if $ks_0 \neq ks'_0$ she can conclude that $x_7 = 1$.

In conclusion, a differential attack can be mounted on this cipher simply by flipping the fourth bit of the nonce and uses it to re-authenticate. The observed differences between the first computed keystream bit in both authentications reveals the value of x_7 . With the knowledge of nonce bit n_7 and the determination of x_7 , the secret key bit k_7 can be simply recovered by computing $k_7 = x_7 \oplus n_7$.

This particular example shows how to recover one bit of the secret key by performing only two authentications and with negligible computational complexity. Since the stream cipher is rotating regularly, the same technique could be applied to recover more bits of the secret key. Likewise, flipping nonce bit n_4 and observation of a difference in ks_1 reveals x_8 .

The ciphers presented in Chapter 6 and Chapter 9 are to some extent vulnerable to differential cryptanalysis. Both ciphers allow an adversary to manipulate the input in such a way that the output leaks information about the secret key.

Differential cryptanalysis is a powerful technique to recover a secret key. Some carelessly designed cryptosystems are vulnerable to such an extent that it is even possible to mount a differential attack in practice. For instance, the MIFARE Classic and iClass ciphers, presented in Chapter 6 and Chapter 9, are vulnerable to a practical differential attack. Both ciphers allow manipulation of the input in such a way that the computed output leaks information about the secret key.

A differential attack often requires specific input-output differences. The initialization of a cryptosystem might not always allow an adversary to enforce such differences in the internal state. Nonetheless, there is always the possibility to gather many traces and filter out the ones that satisfy the predetermined conditions. Moreover, according to the birthday paradox, the number of traces the adversary needs to gather is much smaller than the number usually suggested by intuition, see [Sim64] for more details.

3.2.6 Algebraic attacks

Several cryptographic attack methodologies aim to reduce the computational complexity of a stream cipher by attacking the non-linear function. However, some cipher designs consist of one or more linear components. Such ciphers are vulnerable to algebraic attacks. The literature includes several historical contributions concerning algebraic attacks [Hil29, Hil31, Ree77, Rub79]. Throughout the last decade, several attack generalizations and optimizations were proposed [CP02, AK03, FJ03, CM03, Cou03a, Arm04, MPC04, Cou05, CF08]. This section, however, only defines the basic

principles of algebraic attacks. Note that algebraic attacks should not be confused with linear cryptanalysis, which was introduced in [Mat94] to attack the block cipher DES.

A property of a linear boolean function is the possibility to postpone an evaluation. Computational problems which are formalized during a cryptographic attack can be written as a system of boolean equations [TT80]. Instead of computing the outcome directly, a combination of these equations can be solved by well-known techniques such as Gaussian elimination [Hil29, Mul56, Mar57, Str69, WBD74, Rub79].

In boolean algebra, a function f is \mathbb{F}_2 -linear if it satisfies $f(x \oplus y) = f(x) \oplus f(y)$ for any pair (x, y) of elements in its domain. Moreover, a boolean linear function defines that the input variables either always or never influence the output. Consequently, an observation of influenced output bits, that were computed by a linear function, leak a deterministic linear relation about the corresponding changes of the input bits. When a sequence of influenced output bits exceeds the number of input bits, it is just a matter of solving the system of equations without the need for expensive computations. Contrarily, a non-linear function makes the inversion process more difficult, since the influence of the individual input bits depends on the actual value of these bits. In contrast to a linear function, it is much more difficult to generalize an equation for non-linear functions which holds for every input.

Consider the cipher design with *linear* output as illustrated in Figure 3.17. It consist of two very small bitwise rotating registers which both deliver one input bit to the *linear* filter function $f(\cdot)$ in order to produce the next keystream bit.

Cipher 5

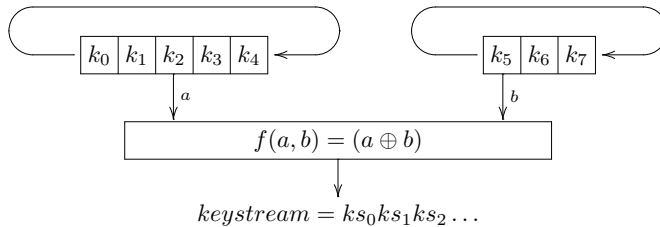


Figure 3.17: A linear stream cipher, initialized by secret key k

The linear filter function $f(\cdot)$ consists of only one exclusive-or (XOR) operation and is redefined in Definition 3.2.3. The XOR operation, denoted by the symbol \oplus , represents a mathematical addition modulo 2.

Definition 3.2.3. *Redefine the filter function $f: \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$ as*

$$f(a, b) = (a \oplus b)$$

The XOR operator is a bitwise linear operation and allows an adversary to compile a system of equations. Consider the following keystream bits $ks_0ks_1 \dots ks_7 = 10110101$. Figure 3.18 consists of four columns, each of which represents a write-up

or evaluation step. After three steps, eight equalities are formed which correspond to the considered keystream. A combination of those equalities determines two equations which should be satisfied when guessing the actual secret key bits.

$ks_0 = k_2 \oplus k_5$	$k_5 = k_2 \oplus ks_0$	$k_5 = k_2 \oplus 1$	$k_5 = \overline{k_2}$
$ks_1 = k_3 \oplus k_6$	$k_6 = k_3 \oplus ks_1$	$k_6 = k_3 \oplus 0$	$k_6 = k_3$
$ks_2 = k_4 \oplus k_7$	$k_7 = k_4 \oplus ks_2$	$k_7 = k_4 \oplus 1$	$k_7 = \overline{k_4}$
$ks_3 = k_0 \oplus k_5$	$k_5 = k_0 \oplus ks_3$	$k_5 = k_0 \oplus 1$	$k_5 = \overline{k_0}$
$ks_4 = k_1 \oplus k_6$	$k_6 = k_1 \oplus ks_4$	$k_6 = k_1 \oplus 0$	$k_6 = k_1$
$ks_5 = k_2 \oplus k_7$	$k_7 = k_2 \oplus ks_5$	$k_7 = k_2 \oplus 1$	$k_7 = \overline{k_2}$
$ks_6 = k_3 \oplus k_5$	$k_5 = k_3 \oplus ks_6$	$k_5 = k_3 \oplus 0$	$k_5 = k_3$
$ks_7 = k_4 \oplus k_6$	$k_6 = k_4 \oplus ks_7$	$k_6 = k_4 \oplus 1$	$k_6 = \overline{k_4}$
(a)	(b)	(c)	(d)

Figure 3.18: Evaluation of $ks_0ks_1\dots ks_7 = 10110101$ leads to the equalities of (d)

Given the equalities denoted in Figure 3.18d it is feasible to deduce two equations, which are given below in Equation (3.1) and Equation (3.2). The equations specify that there are two sets of secret key bits. All elements within one set carry the same value (either zero or one). However, both sets represent the opposite value to each other, which is confirmed by the fact that k_6 is an element of the first set and its complement $\overline{k_6}$ is an element of the second set.

$$k_6 = k_3 = k_1 = k_5 = k_7 \quad (3.1)$$

$$\overline{k_6} = k_4 = k_2 = k_0 \quad (3.2)$$

According to Equation (3.1) and Equation (3.2) there could be only two solutions, either $k_0k_1\dots k_7 = 10101000$ or its bitwise complement $k_0k_1\dots k_7 = 01010111$. The computational complexity to evaluate the equation is negligible. However, the actual work that is required also includes the evaluation of the keystream bits, deducing the equalities and formulating the final equation. It is much harder to generalize such tasks in terms of computational complexity, especially since several tricks to efficiently optimize such effort were proposed in the literature [CKY89, Arm04, MPC04, LA⁺08].

Techniques that find their origin in the previously explained algebraic attacks are applied in practice to the proprietary Hitag2 cipher introduced in Chapter 7. At first it considers two linearly combined inputs as one element of the initialization. This allows an adversary to drastically reduce the list of possible internal states by avoiding independent evaluation of both inputs.

3.2.7 Meet-in-the-middle attack

It is not trivial to increase the computational complexity of a cryptosystem. General reasoning could steer a designer to apply inefficient enhancements [MH81, EG85]. For

instance, it is misleading to think that applying a cipher twice, using a completely different secret key, would double the total computational complexity of a cryptographic algorithm. At first sight it suggests that an adversary needs to perform an exhaustive search twice, to independently determine the first and second secret key. In practice however, it is unlikely that an adversary would use such a method to attack a cryptosystem that applies multiple encryptions. This section introduces a powerful cryptanalytic technique that attacks such a cryptosystem from two sides, the input and the output, and tries to correlate the results in the middle. The technique is commonly referred to as a meet-in-the-middle attack. It can be seen as a special form a divide-and-conquer attack, since it generally tries to solve two independent problems concurrently.

Doubling the computational complexity means that the entropy of both secret keys are multiplied with each other, resulting in the product of both secret keys. However, a scheme where two independent ciphers are applied in a consecutive manner is vulnerable to a meet-in-the-middle attack. Such an attack reduces the computational complexity to the sum of both entropies instead of to the product.

A meet-in-the-middle attack requires knowledge of at least one plaintext-ciphertext pair. With such a pair, the adversary can attack both independent ciphers at the same time. The plaintext is used as input to the first cipher, while the ciphertext is considered as output of the second cipher. The idea is to find a list of candidate outputs (ciphertexts) of the first cipher and a list of candidate inputs (plaintexts) for the second cipher. The intersection of both lists reveals a relation between the two independent secret keys. A plaintext-ciphertext pair that consists of a long enough bitstring singles out only one possible candidate. When multiple candidates are found, more plaintext-ciphertext pairs can be utilized as test vectors to recover the valid secret keys.

For instance, Figure 3.19 shows a cryptosystem that utilizes two different stream ciphers with completely independent secret keys. The ciphers are applied consecutively after each other to encrypt the plaintext. After encryption with the first cipher, the plaintext is transformed into an intermediate state which represents the output of the first cipher as well as the input to the second cipher.

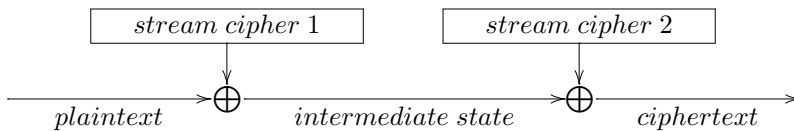


Figure 3.19: cryptosystem which is vulnerable to a meet-in-the-middle attack

Assume that both stream ciphers in Figure 3.19 are not susceptible to a known cryptographic weakness. This leaves the adversary with no other option than to perform an exhaustive search over the complete key space of both ciphers. Instead of trying for each first cipher key guess all possible second cipher keys, the adversary

performs the following attack steps.

1. She first recovers a plaintext-ciphertext pair of moderate length. It is preferably a longer bitstring than the secret keys length of both stream ciphers together. More bits, means more information and avoids multiple (false) candidates when correlating the output of the first cipher with the input of the second cipher.
2. Then, she encrypts the plaintext with the first cipher using all possible secret keys and stores the (encrypted) output in one big list. This output list represents the intermediate state candidates.
3. Next, she decrypts the ciphertext with the second cipher using all possible secret keys and looks if the (decrypted) input entry exists in the output list of the first cipher.
4. Finally, the decrypted input entry that exists in the encrypted output list is the value where both ciphers *meet in the middle*. The secret keys used to encrypt and decrypt to this value are the secret keys used by the original system to produce the plaintext-ciphertext pair.

Instead of performing a brute-force attack on the second cipher for each first cipher secret key guess, the adversary only has to perform two independent brute-force attacks and correlate the intermediate results. This makes the cryptographic problem only twice as big, meaning the sum of twice the original (*computational complexity*) $\times 2$, instead of the product (*computational complexity*)².

The meet-in-the-middle attack is a well-known strategy to defeat weak components and schemes embodied in various cryptosystem. Several improvements and practical implementations of meet-in-the-middle attacks are given in the literature [VOW96, DST04, DSP07, DS08, AS09, DTÇB09, GLRW10]. Likewise, Chapter 8 introduces a variant of the meet-in-the-middle attack that recovers the secret key during the cipher initialization phase. It first shows how to recover the internal state of the CryptoMemory cipher. Then, it starts from the default initialization and guesses the first half of the secret key while running forward and guesses the other half running backwards. During the intersection of both internal state candidate lists, the overlapping state represents the secret key.

Some cryptosystems explicitly define encryption schemes to mitigate meet-in-the-middle attacks. For instance, the Triple DES (3DES) specification defines the use of multiple consecutive utilizations of the Data Encryption Standard (DES). To mitigate a meet-in-the-middle attack, the crypto operation is performed three times instead of two. Note, that the increase in computational complexity is limited to two times the original $2^{56 \times 2} = 2^{112}$ bits, instead of the intuitively expected three times. Since the achieved complexity is only two single DES keys, the 3DES scheme defines a way to securely perform three times single DES using exactly two secret keys of 2^{56} bits. It involves a sequence of encryption, followed by a decryption and encryption again.

Multiple encryption schemes are useful to increase the computational complexity of block ciphers, yet they do not provide the same security features as stream ciphers. Combination of various stream cipher outputs means that multiple keystreams are all combined together with the plaintext using an XOR operation. This inherently defines a direct linear relation between the computed output bits of each utilized stream cipher. In fact, [MM93] shows that the keystream produced by a combination of two additive stream ciphers is as secure as the strongest of the two. Although this feature does not increase the cryptographic strength, it could be used to spread the potential risk of cipher weaknesses over multiple independent ciphers.

3.3 Physical attacks

The types of attacks mentioned in Section 3.1 and Section 3.2 rely on control over the communication channel and mathematical properties of the cryptosystem. This section addresses a third scenario where an adversary has physical access to a cryptographic hardware device. Many examples of such devices are used in everyday life. For instance, access control cards, car keys, credit cards, mobile phones and other electronically secured tools. These products often contain micro-controllers which have limited protection against malicious memory access. It may implement some countermeasures against simply reading out the secret key (e.g. using a debugger interface). However, only the high-end (and often much more expensive) devices offer protection against advanced physical attacks on the hardware itself.

Physical hardware attacks are categorized into three different classes. They range from passive measurements of the characteristics of the chip to actively injecting error states which alter the execution flow of a cryptographic computation. This section first introduces non-invasive hardware attacks, which do not directly tamper with the chip itself. Following are invasive hardware attacks, a technique where the components are altered or dismantled to directly extract secret information. In some cases the chip internals are even completely destroyed during the process. To conclude, the semi-invasive attack method is also addressed. It is similar to invasive attacks, although this time only environmental changes are applied to the chip which subtly alter the workings of the device.

3.3.1 Non-invasive attacks

There are many variants of non-invasive hardware attacks proposed in the literature. The attack types that are addressed in this section are often referred to as side-channel attacks, since they try to derive information leakage from side-effects created by the main application. The first attack technique that is addressed is a timing attack, followed by power analysis and finally Electromagnetic (EM) analysis.

Timing attacks [Koc96, SWT01, BB05] correlate the execution time of a crypto algorithm with the computations that are performed. With extensive knowledge of the

instruction timing information of the target, the execution path may leak information about the cryptographic computations. Using these computations a list of suitable candidate secret keys can be estimated.

Power analysis attacks are very effective on devices that do not implement countermeasures against them. There are two main variants: Simple Power Analysis (SPA) and Differential Power Analysis (DPA). Since its introduction in the nineties [KJJ99], several optimizations and improvements have been proposed in the literature [Cor99, Mes00, BCO04, GBPV10, KJJR11]. Mounting these attack techniques is quite affordable with equipment of only moderate costs. Various research institutes have performed independent studies [MDS99, MS00, Man03, Gou02, OMHT06, MPO05] to explore the practical resistance against power analysis on secure hardware devices that are used by the industry. Chapter 8 describes how the observation of the power levels could lead to the recovery of a secret key from a CryptoMemory chip.

Electromagnetic (EM) analysis is based on the information leaked through observing electromagnetic emission. Studies in the literature [QS01, GMO01, DMÖPV07] show that it is practical to learn secret information from a cryptosystem through EM attacks. The technique is very powerful since it can be applied from a distance without changing the original working environment.

Academics recently showed that it is even practical to extract secret information from a laptop, just by listening to the sound it makes during cryptographic computations [GST13].

3.3.2 Invasive attacks

Invasive hardware attacks start with the decapsulation of the Integrated Circuit (IC). By removing the packaging the internals of the chip are exposed. Decapsulation of an IC can be achieved by etching, drilling, laser cutting and by use of more advanced tools like a Focused Ion Beam (FIB). The technique has almost unlimited capabilities to extract information from chips and understand their functionality. However, it is time-consuming, depends on expensive equipment and requires a very knowledgeable adversary.

After the decapsulation process invasive attacks can be mounted on the surface of the chip. With electrical probing it is possible to reveal the inner circuit of a silicon chip [Wie90, BFSB06]. More advanced techniques like micro-probing [AK96, Wei00] allow needles to be attached on exactly those wires that exchange sensitive data. Observing the communication on those wires allows an adversary to learn information like hidden proprietary commands and secret cryptographic keys.

3.3.3 Semi-invasive attacks

The category with semi-invasive attacks is introduced [Sko05] to bridge the gap between non-invasive and invasive attacks. It still requires decapsulation of the chip,

similar to invasive attacks, yet it applies only non-invasive techniques with less expensive equipment. Furthermore, these attacks can be performed in a reasonably short time and are easily repeatable since the chip is not irreversibly altered in the process.

For instance, an optical microscope with a Charge-Coupled Device (CCD) camera can make an image of a chip circuit layout. There are several studies that show it is not hard to reconstruct the workings of a chip just by observing pictures of its internal components [BFL⁺93, WCAA00, NESP08, PN12].

Semi-invasive attack techniques adjust a combination of various environmental configurations and observe if the workings of the chip are influenced. The literature [SSAQ02, SA03, BBD06, HSH⁺08, HSH⁺09, Sko09, MSSS11] demonstrates practical attack results by exposing a decapsulated chip to variations in temperature, UV light, X-rays and other sources of ionizing radiation, lasers and electromagnetic fields.

Chapter 4

Scientific security assessments of proprietary cryptosystems

Besides the security assessments of proprietary cryptosystems presented in Part II, there are several examples in the academic literature which expose the insecurity of widely used proprietary cryptosystems. The majority of the evaluated systems are attacked to such an extent that they become practically exploitable. This chapter presents the most prominent scientific security assessments which are relevant to this dissertation. The assessments are categorized in four types of systems.

First, the use of access control systems is addressed in Section 4.1. Next, attacks on vehicle immobilizer systems are presented in Section 4.2. Then, various kinds of proprietary communication encryption used in the telephone industry is described in Section 4.3. Finally, in Section 4.4 various other widely used proprietary encryption techniques are addressed.

4.1 Access control systems and electronic locks

Purely mechanical keys and locks are replaced by Radio Frequency (RF) enabled devices for securing access to buildings, rooms, cars, and other property. While mechanical keys and locks are still widely used, they suffer from several disadvantages. For instance, when a mechanical key is lost, stolen or copied, all affected door locks have to be replaced to preserve a certain level of security.

Electronic access control systems can cope much better with such events. In case of loss or theft, administrators can simply lock out the affected electronic keys in a database. Furthermore, the use of cryptography can mitigate the problem of copied keys. However, in practice, the major system integrators and suppliers of electronic locks have been using weak and proprietary cryptography in their systems as will be shown next.

There are three well-known examples in the literature. Section 4.1.1 addresses the weaknesses access control systems which use Legic Prime cards. Attacks on two widely deployed versions of SimonsVoss electronic lock systems are presented in Section 4.1.2 and Section 4.1.3.

4.1.1 Legic

LEGIC Ident systems is a major supplier of access control systems that uses RFID technology for identification. In 2006, the manufacturer claims that it sold more than 70 million access control tokens, which are used in at least 50,000 installations world-wide [LEG06]. The Legic Prime system uses a proprietary RFID card to secure building access and micropayment applications [LEG03]. It is widely deployed throughout Europe and used in critical infrastructure such as military installations, governmental departments, power plants, hospitals and airports.

Despite its use in high-security installations, the literature [PN12] shows that access cards can be instantly cloned from a distance. The security only relies on a secret algorithm that combines the output of a custom CRC with the user's credentials by applying the XOR operation. There is no secret key involved in this transformation, only a secret algorithm. The authors of [PN12] reverse-engineered the secret algorithm by using a method called chip-slicing. This is a technique that sequentially removes the layers of a silicon chip, photographs the internal structure and reconstructs the operations that are performed. The method is further explained in [BFL⁺93, NESP08].

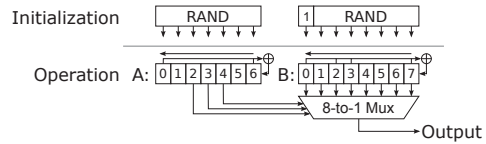


Figure 4.1: Legic obfuscation [PN12]

4.1.2 SimonsVoss G1

The scientific article [WMT⁺13] assesses the security features of electronic locks which are based on the SimonsVoss Digital Locking System 3060 Generation 1 (G1). The system is a relatively old design which was introduced in 1997. Since the successor of this electronic lock (G2) was introduced in 2007, it is discouraged to use the first generation (G1). However, it is still an actively used and widely deployed system and replacement parts are still being sold for legacy installations.

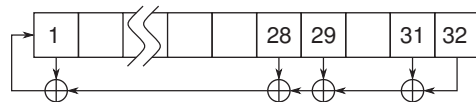


Figure 4.2: Weak PRNG implementation allowing only $2^{31} - 1$ states [WMT⁺13]

In [WMT⁺13], the radio protocol and cryptographic primitives of the system are reverse-engineered. They discovered some implementation flaws which allow two ways to extract the system-wide master secret. Either the adversary mounts a brute-force attack on gathered authentication attempts or performs a Differential Power Analysis (DPA) attack [KJJ99] on an electronic key.

In addition to this, the authors of [WMT⁺13] discovered several weaknesses in the Pseudo Random Number Generator (PRNG) implementation, which allows a

perpetrator to open a door without the necessity of recovering the secret key. Finally, they suggest some countermeasures to mitigate their attacks.

4.1.3 SimonsVoss G2

The literature [SDK⁺13, OSS⁺13] also contains an analysis of the newer, widespread SimonsVoss Digital Locking System 3060 Generation 2 (G2). With respect to Simonsvoss G1, introduced in Section 4.1.2, the new system also relies on an undisclosed, proprietary protocol to mutually authenticate transponders and locks.

The authors of [SDK⁺13, OSS⁺13] explain that their analysis reveals several security vulnerabilities that enable practical key-recovery attacks. They present two different approaches to gain unauthorized access to installations. Firstly, an attacker with physical access to a door lock can extract a master key by dumping the firmware of the chip within a time-frame of only 30 minutes.

A second cryptanalytic attack exploits an implementation flaw in the protocol and depends only on wireless communication with the electronic door lock. The system uses a modified DES algorithm and a challenge-response authentication protocol that re-uses internal bits of the previous session as challenge for the next session. This allows an adversary to derive the transponder key from a few consecutive authentication attempts gathered wirelessly. By being only a few seconds in vicinity distance of a door lock, an adversary can recover all the transponder credentials within only a few seconds of computation.

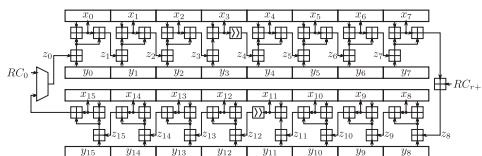


Figure 4.3: Obscurity function [SDK⁺13]

4.2 Electronic vehicle immobilizers

Electronic vehicle immobilizers have been very effective at reducing car theft. Such an immobilizer is an electronic device that prevents the engine of a vehicle from starting when the corresponding transponder is not present. This transponder is a low-frequency Radio Frequency Identification (RFID) chip which is typically embedded in the vehicle's key. When the driver starts the vehicle, the car authenticates the transponder before starting the engine, thus preventing hot-wiring. In newer vehicles the mechanical ignition key has often been removed and replaced by a start button. In such vehicles the immobilizer transponder is the only anti-theft mechanism that prevents a hijacker from driving away with the car.

In Section 4.2.1 an immobilizer cryptosystem is reverse-engineered and shown to be insecure because it utilizes an extremely weak secret key. Section 4.2.2 shows

several attacks on a cryptographic algorithm that is used in immobilizers and Remote Keyless Entry (RKE) systems.

4.2.1 DST

The security algorithms of the vehicle immobilizer cryptosystem known as a Digital Signature Transponder (DST) are reversed-engineered and attacked in [BGS⁺05]. The authors used the transponder as an encryption oracle to perform a black-box analysis and find the cryptographic relation between the input and the output. Their methodology is thoroughly explained in the paper and encourages other researches to explore similar research.

The DST cryptosystem uses a fairly small secret key of only 40 bits. They show that it is straightforward to recover the proprietary algorithms used in this cryptosystem. With knowledge of these proprietary computations an adversary could simply find the secret key by performing an exhaustive-search over the complete key space. With the storage capabilities of current hard drives, it is possible to pre-compute all possible encryptions (cipertexts) given a few chosen plaintexts. After pre-computing and sorting these ciphertexts, an adversary can quickly lookup the corresponding secret-key given a plaintext-ciphertext pair.

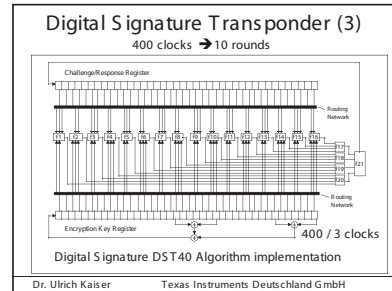


Figure 4.4: DST cipher [BGS⁺05]

4.2.2 KeeLoq

The proprietary KeeLoq cipher is used in Remote Keyless Entry (RKE) systems and remotely operated locks for garage doors. It attracted the attention of cryptographers when the cipher was published¹ in 2006. The first proposed attack [Bog07b] used a combination of correlation and linear attacks to recover the secret key with a computational complexity of 2^{52} encryptions.

The attack was later optimized in [Bog07c, Bog07a] to a complexity of $2^{50.6}$ encryptions. An algebraic slide attack was proposed in [CBW08]. Although the computational complexity of this attack is only 2^{27} , it requires a special case of an encrypted slid-pair sample. Since there is no easy

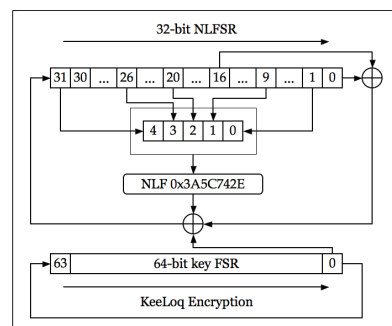


Figure 4.5: KeeLoq encryption [CBW08]

¹<http://en.wikipedia.org/wiki/KeeLoq>

way to identify such a pair, the attack is simply repeated 2^{32} times for each 2^{16} pairs, which increases the total attack complexity to at least 2^{48} .

In [IKD⁺08] a new method is proposed that uses a meet-in-the-middle attack to reduce complexity to only $2^{44.5}$ encryptions. Finally, in [EKM⁺08, PEK⁺09, KKMP09] several hardware attacks were proposed that show it is practical to recover a device specific key as well as the secret master key which allows an adversary to clone credentials from gathering only a few wireless traces.

4.3 Cellular-, cordless- and sat-phones

In the last decades several types of wireless phones were introduced. These communication devices have changed the way we interact with each other. Instead of using only wired landlines, the general public migrated to cellular- and cordless-phones. Additionally, the financial barrier to use satellite-phones is lowered by the increase of satellite providers. Compared to wired connections, it is much easier to eavesdrop a wireless device. It is challenging to achieve the same protection that secures the confidentiality of a telephone conversation.

Since the 90's most wireless communication is digitalized. One of the advantages of digitalization is the ability to transparently encrypt the communication channel. This could in principle protect the confidentiality of all telephone conversations. However, this section shows that several widely deployed proprietary cryptosystems used in cellular phones (Sections 4.3.2–Section 4.3.5), cordless phones (Section 4.3.6 and Section 4.3.7) and satellite phones (Section 4.3.8 and Section 4.3.9) fail to provide secure and adequate ways of encryption.

4.3.1 ORYX

The Telecommunications Industry Association (TIA) is the North American equivalent to European Telecommunications Standards Institute (ETSI). It designs specifications and digital cellular standards which are comparable to those used in GSM communication. One of the selected encryption algorithms to protect wireless data services is the stream cipher ORYX. The implementation of the proprietary cipher is controversial. A few years after its introduction in 1995 it was first publicly reviewed in the literature [WSD⁺99]. Although the initial security of the cipher should provide a computational complexity of 2^{96} bits, the article shows that the actual attack complexity is only 2^{16} . The authors of [WSD⁺99] used a divide-and-conquer attack, as described in Section 3.2.2, to drastically reduce the computational complexity of the cipher. After

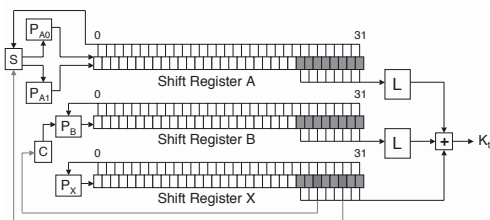


Figure 4.6: ORYX cipher [WSD⁺99]

publication the cipher was considered completely broken and shown to be insecure. The TIA updated their specification accordingly and removed ORYX from the list of proposed cryptographic primitives.

4.3.2 A3, A8 and COMP128

The Global System for Mobile Communications (GSM) specification that was published by the European Telecommunications Standards Institute (ETSI) allows Telecom operators to use their own proprietary algorithms for network authentication. As an example, the specification mentions an authentication method based on the so-called COMP128 implementation. COMP128 is an example instantiation of the two proprietary algorithms A3 and A8.

The first algorithm A3 is used to authenticate the mobile station to the network, while the second one, A8 is used to generate the session key. The session key is input to one of three additional proprietary A5 ciphers which are addressed later in Section 4.3.3, Section 4.3.4 and Section 4.3.5. Although the COMP128 implementation was just a reference example, it has been literally copied and deployed by different service providers in several types of Subscriber Identity Module (SIM) cards. After public exposure of the proprietary algorithms in 1998², the cryptographic security was thoroughly assessed in the literature.

The first cryptanalytic attack was published in [WGB98]. The attack was later extended and generalized in [HP00]. An effective hardware attack was later proposed in [RRST02], where the authors retrieve the 128-bit key by using as few as 8 chosen plaintexts. This article inspired many attack optimizations [GMO01, Nov03, Cla07, ZYSQ13] on similar algorithms.

4.3.3 A5/1

The first generations of GSM transmissions use the secret and proprietary A5 stream cipher design. This algorithm uses a 64-bit secret key and has two main variants. The A5 ciphers are responsible for encryption of the data transmitted between the mobile device and the Base Transceiver Station (BTS).

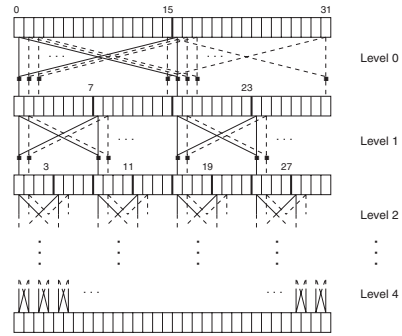


Figure 4.7: COMP128 [RRST02]

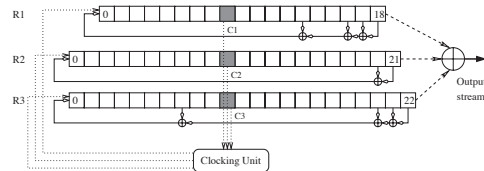


Figure 4.8: A5/1 cipher [BB06]

²<http://www.scard.org/gsm/a3a8.txt>

The original GSM cipher (which was export-restricted) embeds the stronger A5/1 version and is used mainly in Europe while the less secure A5/2 version is used in other countries. The approximate design of A5/1 became publicly known³ in 1994. Furthermore, it was concurrently published in the literature at the CHINACRYPT conference in China [XHW94]. Historical records show that there were some serious cryptographic weaknesses found in 1994 [She94a, She94b]. However, for several years no cryptographic attacks on A5/1 were published in the literature. Finally, in 1997 the first article [Gol97a] showed that the required computational complexity to mount an attack only required solving 2^{41} linear equations.

In the years after that many followed [BS99, BD00, BS00, PS00, BSW01, BBK03, EJ03a, GKVW05, MJB05, BB06, GNR08, BBK08, KPPM12, PPPM13]. The first papers purely exploit cryptographic weaknesses in A5/1 and optimize the previously proposed attacks. The latter publications propose combinations of cryptanalytic and hardware attacks to demonstrate that the theoretical attacks are practically applicable. The specific internal structure of the GSM communication protocol allows the most powerful cryptographic attack, a ciphertext only attack. An adversary that eavesdrops a communication and gathers a sequence of consecutive encrypted frames, can recover the secret key with moderate computational power.

4.3.4 A5/2

The A5/2 cipher is a weaker version of the A5/1 encryption algorithm. The main reason is that the designers moved part of the algorithm to an additional register. Thereby they increased the complexity of the algorithm, although they drastically decreased the computational complexity to attack the cipher. As mentioned in Section 4.3.3, the stronger GSM cipher A5/1 was export-restricted. It could not be used in countries that were not a member of the Organization for Economic Co-operation and Development (OECD). There are several theories, yet the actual reason why the non-OECD countries should use weaker encryption was never officially explained to the general public.

The cipher A5/2 was published in 1999 after it was reverse-engineered from a mobile cell phone⁴. Several cryptographic weaknesses were identified in the literature [BBK03, BER07, BBK08] shortly after its publication. The cryptographic strength is considered extremely low, the cipher allows key recovery from only a few consecutive ciphertext frames and negligible computational complexity.

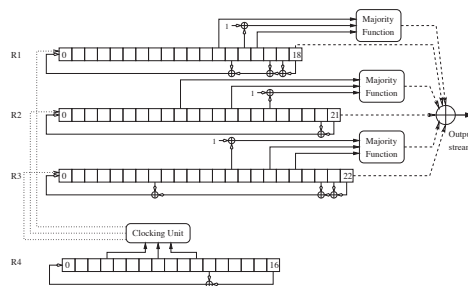


Figure 4.9: A5/2 cipher [BBK03]

³<http://cryptome.org/jya/crack-a5.htm>

⁴<http://cryptome.org/gsm-a512.htm>

4.3.5 A5/3 (KASUMI)

KASUMI is a proprietary block cipher that was designed to form the basis of the 3rd Generation Partnership Project (3GPP) confidentiality and integrity algorithms. The parties collaborate in 3GPP to standardize the next generation of cellular telephony, commonly referred to as 3G. In contrast with A5/1 and A5/2 this encryption algorithm is proprietary, yet it is not secret. The full cipher description is available in one of the 3G standardization documents [ETS12], which was published by the European Telecommunications Standards Institute (ETSI) and is freely available on their website.

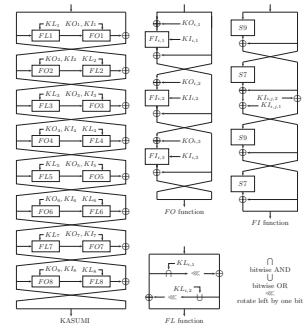


Figure 4.10:
KASUMI [DKS10]

The customly designed block cipher KASUMI is based on the publicly proposed block cipher MISTY, which was introduced in the literature [Mat97] in 1997. The 3G session key, which encrypts the communication, is regularly refreshed and temporarily stored in untrusted memory of the phone. It can be argued that the initialization of the MISTY cipher in a 3G context is unnecessarily thorough. Therefore, the designers of KASUMI decided to make MISTY faster and more hardware-friendly by simplifying its key schedule and modifying some of its components. It is important to note that if KASUMI is used in another context, where the cipher initialization is trusted to be secure, the changes can have serious impact on its security strength. Such utilization is possibly vulnerable to related key attacks [BE02, BDK05, DKS10] and higher order differential attacks [SAHK07, Sai11, JLR+13].

4.3.6 DSAA

Digital Enhanced Cordless Telecommunications (DECT) is a standard for connecting cordless telephones to a local Base Transceiver Station (BTS) over a maximum range of few hundred meters. Cordless phones using the DECT are among the most widely deployed wireless telephone security technologies. Furthermore, several other applications with even greater security requirements depend on its technology. For instance machine automation, building access control, alarm systems, and wireless credit card terminals. To date over 800 million DECT devices have been sold in more than 100 countries [WTHH11].

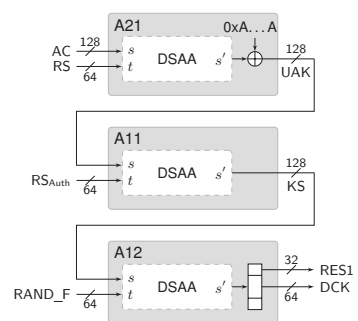


Figure 4.11: DSAA
init [MOTW09]

To achieve confidentiality and authenticity, two proprietary cryptographic algorithms are used. The DECT Standard Cipher (DSC) encrypts the communication

and protects the confidentiality of the conversation. The details of this cipher are addressed in Section 4.3.7. The second algorithm, DECT Standard Authentication Algorithm (DSAA), authenticates a DECT phone and is designed to protect against an adversary that tries to piggyback on someone else's telephone connection. The authentication algorithm was first published in [LST⁺09]. Until then the DSAA was only available under a Non-Disclosure Agreement (NDA) with the European Telecommunications Standards Institute (ETSI), which thwarted an objective security assessment through academic scrutiny.

The article [LST⁺09] gives a very detailed security analysis of the DSAA. It includes some very effective attacks on the building blocks used for DSAA. Furthermore, it points out a commonly made implementation mistake that can practically lead to a total break of DECT security. The authors show an attack which allows an adversary to impersonate a BTS and perform a relay attack as described in Section 3.1.3. The attacks were later optimized in [MOTW09] and generalized in [Tew12].

4.3.7 DSC

The encryption algorithm DECT Standard Cipher (DSC) is responsible for the confidentiality and protects the data transmitted by a cordless DECT device. Similar to DSAA, the DSC algorithm was not available to the general public. The inner workings of the algorithm were published one year later than DSAA by (partially) the same authors [NTW10]. The article proposes a correlation attack on the cipher which allows the decryption of an intercepted call. It only relies on passive eavesdropping, so a victim is not able to detect the presence of an adversary. The attack was later improved in [WTHH11] by reducing the amount of required keystream and increasing the attack speed with special-purpose hardware.

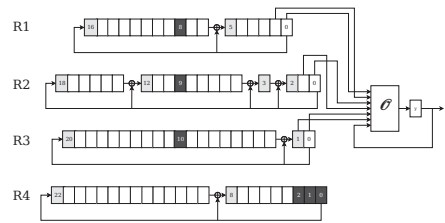


Figure 4.12: DSC cipher [Tew12]

4.3.8 GMR-1

Satellite phones use the communications satellites that are positioned in a Geostationary Earth Orbit (GEO). They are commonly referred to as a GEO Mobile Radio Interface (GMR). Although satellite phones are in use for several years, the first scientific article that independently assessed the security of sat-phone encryption was published only recently [DHW⁺12].

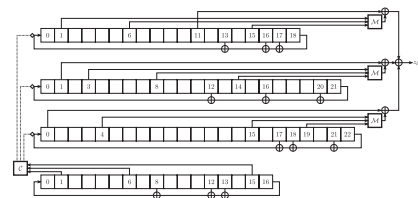


Figure 4.13: GMR-1 cipher [DHW⁺12]

According to [DHW⁺12], there are at least 100 000 sat-phone subscribers worldwide. Protecting the confidentiality is an important feature for telephones to ensure the privacy of a conversation. This is particularly true for satellite phones, since they are used for sensitive matters like intercontinental operations and disaster response. There are two providers of sat-phones, each using their own proprietary encryption technique to protect the communication channel. The first is referred to as GMR-1, which is discussed in this section, while the second GMR-2 is discussed in Section 4.3.9.

The proprietary cryptosystem GMR-1 uses a stream cipher that is a modified version of the A5/2 cipher used in GSM. The article [DHW⁺12] proposes an attack that is similar to those introduced for A5/2 presented in [PFS00, BBK08]. It enables a ciphertext-only attack, which allows an adversary to passively eavesdrop and decrypt a GMR-1 conversation.

4.3.9 GMR-2

The proprietary cryptosystem GMR-2 is addressed in the same article which exposed and identified weaknesses in the GMR-1 encryption algorithm [DHW⁺12]. The article proposes a known plaintext attack against the GMR-2 cipher by using a Time-Memory Trade-Off (TMTO). The method allows pre-computation of certain steps so that it reduces the on-line attack complexity when it is later mounted by an adversary.

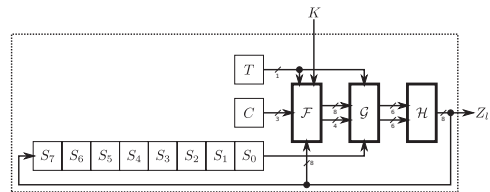


Figure 4.14: GMR-2 cipher [DHW⁺12]

The attack on GMR-2 is less practical than the one proposed for GMR-1, since it requires approximately 50 to 65 bytes of known plaintext. However, knowledge of these bytes allows an adversary to practically recover the secret key of a conversation with only moderate computational complexity.

4.4 Various encryption techniques

The previous sections addressed proprietary ciphers which were used by three specific fields of industry. This section assesses three encryption techniques which are only used by a specific application and two general ciphers which are not designed for a single purpose or particular industry.

Section 4.4.1 addresses the security of the proprietary cipher used to encrypt information stored on a Digital Versatile Disc (DVD). Next, Section 4.4.2 shows the cryptographic assessment of the proprietary encryption technique used in the communication between two Bluetooth devices. Then, the ciphers addressed in Section 4.4.3 and Section 4.4.4 demonstrate that proprietary ciphers are not necessarily insecure

by definition. Both ciphers, developed by experts, are still considered moderately secure and, if used correctly, not practically broken. However, in contrast, Section 4.4.5 shows that incorrect usage of a cipher can completely undermine the security of a cryptosystem [BGW01].

4.4.1 CSS

Content Scramble System (CSS) is a proprietary cryptosystem used to encrypt information stored on a Digital Versatile Disc (DVD). A DVD is the widely adopted successor of high capacity Compact Disc (CD) storage system. It is a Digital Rights Management (DRM) system which is designed to prevent unauthorized duplication.

The working of the CSS cryptosystem was first published in [Ste99]. The same article identifies serious weaknesses in the system and shows that the 40-bit secret key can be recovered with a computational complexity of only 2^{25} operations. A more detailed overview of the weaknesses and attacks are presented in the slides [Kes00] of a lecture about CSS.

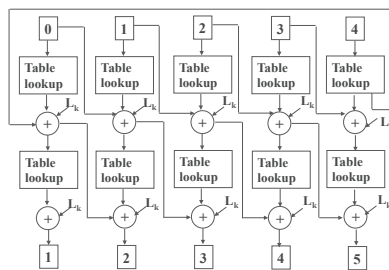


Figure 4.15: CCS decryption [Kes00]

4.4.2 E0

Bluetooth is a wireless communication standard designed in 1999 by the Bluetooth Special Interest Group (SIG). The specification defines the use of a proprietary stream cipher algorithm E0 to encrypt communication between Bluetooth devices. The first attack on the E0 stream cipher was published in the literature shortly after its introduction [HN00]. The attack is based on a general correlation attack technique as described in Section 3.2.3.

The article drew the attention of the cryptographic community which generated many improvements and theoretical attacks over the last decade [DCJP01, FL01, Flu02, GBM02, Kra02, AK03, Cou03a, LV04a, LV04b, LMV05, LW05, SW06, LV08, PDMS09]. Finally, a recently published article [ZXF13] identifies serious weaknesses in the cipher which can be exploited in a practical attack that recovers the complete 128-bit secret key.

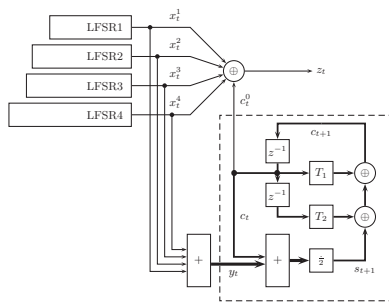


Figure 4.16: E0 cipher [DCJP01]

4.4.3 Skipjack

Skipjack is a proprietary block cipher that depends on an 80-bit secret key. It was designed by the National Security Agency (NSA) of the United States (US). Its design was classified, implemented and distributed only in a closed hardware device, called the Clipper chip. The NSA started to promote the use of the chip to the industry in 1993. For instance, Telecom providers could use the encryption to protect the confidentiality of telephone conversations. To be able to use the chip, every secret key had to be shared with the US government. This raised a lot of opposition by the general public.

The Electronic Frontier Foundation (EFF) and Electronic Privacy Information Center (EPIC) defended the public's privacy rights by opposing the use of the Clipper chip. It would subject citizens to increased and possibly illegal government surveillance. Furthermore, since the design was classified, the actual strength of the Clipper chip's encryption could not be evaluated. In spite of the NSA's effort, the Clipper chip was not embraced by consumers or manufacturers. Encouragement for its usage eventually ceased completely around 1996.

The design of Skipjack was declassified in 1998 and published by the US government. Publication of the design finally allowed public scrutiny of its security strength. Shortly after publication several minor weaknesses were identified and attacks on reduced variants of the cipher were proposed [BBD⁺99, BBS99, KRW99, Gra02]. However, an attack on the full cipher which allows recovery of the complete 80-bit secret key was not identified. In 2002, a cryptographic attack on the full cipher was proposed in [Pha02]. Interestingly, several years later the same author described a much weaker attack on Skipjack in [KP09] and completely ignored the previous publication [Pha02].

Although the secret key distribution and the secretly designed Clipper chip was controversial, the design of the proprietary cipher Skipjack does not seem to be very insecure. It is carefully designed and uses better-evaluated cipher components compared to most proprietary algorithms. Judging by the publications in the literature there are no practical attacks available.

However, many cryptographers quickly lost interest after the initial cryptanalytic assessment. The deployment of the Clipper chip had failed and with its declassification it became clear that the algorithm would actually never be deployed. Furthermore, there were alternative block ciphers proposed in the literature [DR98] with better security properties, such as the use of a 128-bit secret key.

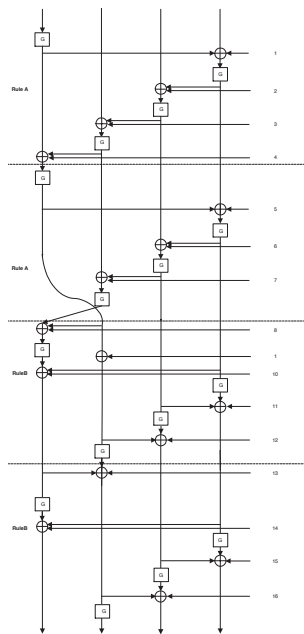


Figure 4.17: Skipjack cipher [BBS99]

4.4.5 WEP

Wired Equivalent Privacy (WEP) is a protocol for encrypting wireless network communication. It was officially introduced as part of the IEEE 802.11 network standard [C⁺07] in 1999. All packets in a WEP network are encrypted with the proprietary stream cipher RC4, see Section 4.4.4. The WEP protocol was quickly adopted by the industry and embedded in millions of devices [Mil01].

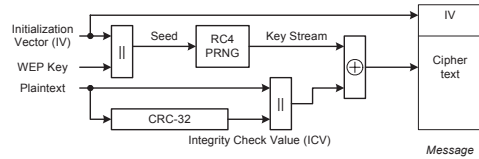


Figure 4.19: WEP encryption protocol [C⁺07]

Shortly after the introduction of the WEP protocol, several serious weaknesses were identified in the loading and scheduling of the secret key. First the attack methodology was published in the literature [FMS01], swiftly followed by an experiment that demonstrated the feasibility of the proposed attack [SIR02]. It requires negligible computational power and depends on cipher text only, which can easily be intercepted by an adversary without being detected [BGW01]. Initially, the amount of network traffic that should be intercepted was substantial. However, the fact that a complete 104-bit key could be recovered with a passive attack, clearly demonstrated that WEP could not provide adequate security. In response, the IEEE in collaboration with the Wi-Fi Alliance announced in 2003 [WIF03] that Wired Equivalent Privacy (WEP) had been deprecated and superseded by Wi-Fi Protected Access (WPA).

Although WEP was superseded by WPA, the network protocol was still the default protocol in most wireless consumer networks. The industry did not understand the seriousness and practical impact of the in their view “sophisticat attacks” [Mil01]. In the years after more optimized and practical attacks were proposed in the literature [CWHWW03, HA03, SIR04, Man05a, BHL06, C⁺06, TWP07, TB09]. An attack that was recently published in the literature [SSVV13] shows that the full attack only requires 5 seconds on an off-the-shelf Personal Computer (PC).

Chapter 5

Introduction to Radio Frequency Identification (RFID)

Radio Frequency Identification (RFID) is a digital identification technology that uses the modulation of Radio Frequency (RF) waves as communication channel. RFID devices transmit small amounts of data wirelessly over a short distance. The technique is introduced as successor of the legacy identification systems such as bar codes, entrance tickets and personal passes. Figure 5.1 shows a contactless smartcard that embeds an RFID chip and antenna. It uses Radio Frequency (RF) waves to communicate messages. Initially these messages consisted of just a Unique Identifier (UID) which were just used to identify one side to another. In later developments more messages and features were added.

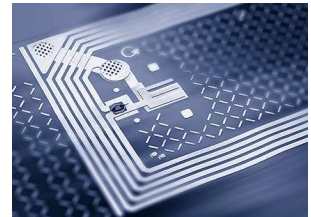


Figure 5.1: RFID chip¹

An RFID system consists of two communicating components, an interrogator and a responder. The interrogator is referred to as *reader*. It is in charge of the “conversation” and asks the “questions”. The responder is referred to as *tag* or *transponder*, as it simply “answers” within a predefined time-frame to the interrogator’s questions.

The chapter is divided into two main sections. First, Section 5.1 presents the technology details and their application in security products, followed by an overview of RFID research tools in Section 5.2.

5.1 Technology and application

There is a huge variety of tags on the market. They differ in size, casing, memory, computing power and the various security features they provide. Despite the difference in appearance and functionality, there are two main categories of RFID tags: passive and active ones. The passive tags operate without a battery and completely rely on power drained from the electro-magnetic field which is generated by the reader. Passive tags are often much cheaper to deploy. However, they have only limited computational capabilities and a small communication distance. Active tags possess

¹<http://rfid-handbook.de/downloads/images/hf-kommunikationsprinzip.png>

a small battery as a power supply. They are slightly more expensive, yet do not suffer from the constraints caused by limited power.

The cryptosystems that are addressed in Part II of this dissertation are found in passive RFID tags. Therefore, the scope of this section is limited to related techniques. A summary of RFID signal processing is given in Section 5.1.1. The communication protocols of RFID tags are specified in several standards published by the International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). An overview of the standards involved is presented in Section 5.1.2. The successor technique, which is heavily dependent on RFID communication techniques and protocols is the Near Field Communication (NFC) technology. Some cryptosystems that are assessed in Part II use NFC-compatible tags. The technical relation and security implication are addressed in Section 5.1.3.

5.1.1 Signal modulation and encoding

The carrier that provides power to a tag is also used for communication. At the physical layer of the communication, the sender has to transform the information to an analog signal. This analog signal is captured at the receiver side and transformed back to the original digital message that was 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.

The reader-to-tag communication is in most cases achieved by interrupting the carrier wave for a couple of microseconds. Because the field is dropped for only a very short period, a simple capacitor in the tag can overcome the interruption of power. The tag-to-reader communication is different, since the tag cannot interrupt the carrier wave that is generated by the reader. Instead, it modulates the signal by putting some resistance on the (sub)carrier wave of the reader.

There are many radio frequency signal modulation and encoding schemes. RFID systems often use one of the following modulation techniques: Amplitude-Shift Keying (ASK), Frequency-Shift Keying (FSK), On-Off Keying (OOK), Phase-shift keying (PSK). Additionally, transmission and framing of the actual bits is done by one of the following encoding schemes: Bi-Phase (BP), Binary Pulse Length Modulation (BPLM), Non-Return-to-Zero (NRZ) and Quad Pulse Length Modulation (QPLM). More details about the workings of these techniques are available in the doctoral dissertation of de Koning Gans [dKG13].

5.1.2 Communication standards

There are many standards for RFID technology and a wide variety of different frequencies are available for RFID applications. The most-often used frequencies are in the Low Frequency (LF) band at 125-134 kHz and in the High Frequency (HF) band at 13.56 MHz. RFID communication schemes often focus on the three lowest layers of

the Open Systems Interconnection (OSI) model, namely the *network layer*, *data link layer* and the *physical layer*. These layers facilitate the communication up to the bit level of messages that are sent over the air. The most relevant ISO/IEC documents that fall within the scope of this study are the low frequency standards [ISO94, ISO96], the high frequency standards for proximity [ISO01] and vicinity [ISO00].

5.1.3 Near Field Communication (NFC)

The abbreviation RFID is not often used as terminology for consumer products, but the more catchy term Near Field Communication (NFC) is. The NFC Forum², a consortium of more than 170 multinationals, designs and standardizes RFID-based technology under the comprehensive term NFC. To summarize the relation between NFC and RFID technology, the next paragraphs present the main objectives, capabilities and technical aspects of NFC technology.

Near Field Communication (NFC) technology is an extension of several RFID communication standards [ISO01, ISO00, JIC05]. It combines the high frequency (13.56 MHz) proximity standards and reformulates them with some additional features into two new communication standards [ISO04, ISO05]. The two main new features added in these standards are peer-to-peer connections between two active NFC devices, also referred to as NFC Interface and Protocol (NFCIP), and the emulation of a passive proximity RFID tag. The initial goal behind NFC technology is to establish more complex wireless channels that operate at a proximity distance. This makes NFC much more ambitious than RFID systems. The latter is limited to plain identification, tracking of unique card numbers or storing small monetary values in the memory of an RFID tag. Because NFC is backwards compatible with RFID systems, several deployed devices could be accessed by an NFC-enabled device. These include electronic passports and identity cards based on the International Civil Aviation Organization (ICAO) standard [ICA03], most contactless public transport tickets and access control tags that operate at 13.56 MHz.

While RFID standards merely focused on the specification of the modulation, encoding, and start and stop conditions of the communication, NFC extends its specification by adding application formats [NFC11, NFC06, NFC10, NFC13] and integration of (multiple) secure elements [ETS11, ETS08, ISO11, ISO07]. Secure elements are comparable with regular smart cards and are available in many forms, e.g. contactless smart card, Universal Integrated Circuit Card (UICC), MicroSD card with RF interface or an internal embedded chip which is integrated into an NFC controller chip. The latter one is used in the popular Google Nexus phones together with an UICC that contains the SIM and is supplied by the user's telephone company.

Many security aspects of NFC depend on the security of the underlying RFID technology. If the lowest level of communication is proven to be insecure, all technology that depends on its security is considered compromised. Additionally, various

²<http://www.nfc-forum.org>

othe NFC security issues are identified in the literature [Mul09, VK11, RLS11]; they target specific NFC standards and NFC-enabled devices.

5.1.4 Contactless smart cards

The usage of RFID tags increased considerably in the last decade. The industry promoted the technology for various applications, including solutions for security issues such as counterfeiting, impersonation and vehicle theft. RFID technology is now widely adopted and used for several security applications.

A well-known example is the usage of RFID technology in several billion actively used contactless smart cards [Jue06]. Over the last few years, more and more systems adopted contactless smart cards as a replacement for bar codes, 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, although unlike historical 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 access control to the smart card's memory.

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 card³ in London and the OV chipkaart⁴ in The Netherlands, among others. Many countries have already incorporated a contactless smart card in their electronic passports [HHJ⁺06]. Many office buildings and even secured facilities such as airports and military bases use contactless smart cards for access control.

5.1.5 Proprietary cryptography in RFID devices

Since the first generation of widely deployed RFID tags in the 90s, it has been a tradition for the industry to design proprietary RFID products. Such a proprietary design often contains customly defined modulation/encoding schemes, packets, checksums, instruction sets and in some cases even custom made cryptographic algorithms and authentication protocols.

There is not much wrong with designing a custom way of RFID communication. It allows the industry to optimize products and boost their performance for specific applications. However, this argument certainly does not hold for the proprietary cryptosystems. Designing secure algorithms is proven to be a difficult task without feedback from the scientific community [Ker83, JS97, fSN97].

The semiconductor industry advocates that proprietary cryptography prevents counterfeiting [CS05, GB06, GS07]. Unfortunately, it usually leads to a vendor lock-in situation [And03, LS04]. It is conceivable that a customer is unaware that the additional proprietary cryptography hardly increases the security strength. In fact, it

³<http://oyster.tfl.gov.uk>

⁴<http://www.ov-chipkaart.nl>

mostly just constrains their possibility to migrate to compatible products from other suppliers in the future.

Almost all major semiconductor companies that produce RFID devices are inclined to create ad hoc RFID designs that use proprietary protocols and custom-made cryptographic algorithms [PHI98, MC01, ST02, EM02, TI04, IC04, AT06, INF07, HID08, AT09, NXP10]. Such designs are often kept secret to provide security-through-obscurity. As mentioned in Section 1.2, several manufacturers claim their products provide ‘state-of-the-art’ [IC04], ‘field-proven’ [PHI98, HID08], ‘high-level’ [ST02, MC01, EM02, KM05, AT06] or ‘unbreakable’⁵ security. However, it is hard to know what this means and how much security you actually get. As long as RFID products do not comply with open and community-reviewed encryption standards, the security properties of such tags cannot be objectively assessed.

5.2 Research tools

There is cheap and off-the-shelf hardware available to communicate with RFID tags, contactless smart cards and NFC enabled devices. The design and usage of such hardware is often undocumented and uses a proprietary (and often secret) instruction set. Moreover, this hardware is typically limited to support only one vendor-specific RFID tag and does not allow a user to control the raw modulated and encoded signals. This can be convenient for an engineer who does not want to spend time on such elementary building blocks of RFID technology. However, for a researcher who wants to have full control over the modulation depth, encoding variation, protocol timing, parity bits and checksums, this is not very useful.

There are several open-design hardware projects that aim to overcome this limitation by introducing a tool that allows more access to raw Radio Frequency (RF) signals [Blo04, RCT05, Ver08a, Ver08b, dKG08, KSB08, dKGV09, KSP10, FAH+10, GdKGV12, VdKGG12]. However, most of these tools are created in an ad hoc manner and are limited to the investigation of specific technology or demonstrate only a particular functionality. For instance, they are used for quick prototyping and initial testing of the protocols proposed in [VDWKP09, FGMR10, dKGG10, GvR10, GFMR11, ABV12].

This section limits its attention to two generic hardware tools which proved themselves as best practices in scientific research, the GNURadio⁶ and the Proxmark⁷. Both tools allow full access to raw RF signals and have support for various modulation and encoding techniques. Moreover, these tools are supported by a very active development community and are used by many research institutes, industries and individuals. More functional details are presented for the GNURadio in Section 5.2.1 and the Proxmark in Section 5.2.2.

⁵http://www.nxp.com/products/automotive/car_access_immobilizers/immobilizer/

⁶<http://gnuradio.org>

⁷<http://www.proxmark.org>

5.2.1 GNURadio framework

A GNURadio device is a generic Software Defined Radio (SDR) that supports Radio Frequency (RF) signals on many frequencies, ranging from 0Hz to 6GHz. The GNURadio framework is fully open-source and named after the recursive acronym GNU's Not Unix (GNU), to specify it is Unix compatible, yet it contains only free software. The device comes with a large Field Programmable Gate Array (FPGA) that allows the user to virtually build a hardware scheme using a software-based gate configuration. The GNURadio software framework is not specifically made for RFID communication schemes. However, it supports almost all relevant modulation and encoding schemes.

The most often used GNURadio device is the Universal Software Radio Peripheral (USRP), which is made by the company Ettus Research⁸. The USRP connects to a host computer through a high-speed Universal Serial Bus (USB) or Gigabit Ethernet link, which the host-based software uses to control the USRP hardware. It is possible to integrate the general functionality of a host computer within the FPGA, which allows the USRP to operate in a standalone fashion.

The USRP motherboard provides the following features: custom clock frequency, FPGA, Analog-to-Digital Converter (ADC), Digital-to-Analog Converter (DAC), USB interface, and variable power regulation. These are the basic components that are required for baseband processing of signals. A modular front-end, called a daughter-board, is used for analog operations such as up/down-scaling, filtering, and other signal conditioning. For compatibility with the RFID frequency, two extra hardware modules (LFRX and LFTX daughter-boards) are required.

The default operation of the FPGA is to perform several Digital Signal Processing (DSP) operations, which translate real signals in the analog domain to lower-rate, complex, baseband signals in the digital domain. In most use-cases, these complex samples are transferred to the host that performs decoding operations. The code for the FPGA is open-source and can be optimized to allow high-speed, low-latency operations to occur in the FPGA.

There is no predefined frequency, fixed power regulation and specified clock; all components are configurable. Such flexibility comes with a price tag. The tool is quite expensive, it currently costs around USD 2000.

The working environment is extremely powerful, which can be a bit confusing for an inexperienced user. The main reason is that there are many configuration options which relate to various well-known communication protocols (e.g. GSM, Dect and Bluetooth).

There are a few examples in the literature [CCC⁺09, CSY⁺10, CGE12, CG12] of RFID security research that is performed with a GNURadio USRP device. However, the more ground-breaking security research that experimented with the GNURadio, used the tool to evaluate various other radio frequency enabled devices. Well-known

⁸<http://www.ettus.com/>

examples are the studies on pacemakers [HHBR⁺08], DECT cordless phones [LST⁺09] and wireless monitor systems embedded into modern vehicles [IRMTT⁺10].

5.2.2 Proxmark hardware device

The Proxmark tool is similar to a USRP device, yet it is more optimized for (and limited to) the 125-134 kHz and 13.56 MHz frequency bands, which are used in almost all low and high frequency RFID tags. Because the Proxmark is made specifically to support RFID technology, it already supports many RFID communication techniques. This includes a wide range of modulation and encoding schemes, communication protocols and most standardized contactless smart card instruction sets.

The Proxmark hardware has been developed by Jonathan Westhues⁹. The early versions of this device were able to clone unsecured tags [GR05, HJSW06], although they were not as powerful as the third version, which is currently the most recent version of the Proxmark. For instance, the initial ‘prox card cloner’ could only handle one type of modulation and was merely designed to clone UID-only tags.

The latest hardware design and firmware is in the public domain since May 2007 under the General Public License (GPL). The device costs around USD 150 and since the schematics are online, it can be ordered through any local printed circuit board (PCB) supplier. Alternatively, a completely assembled Proxmark device is available from two commercial suppliers: Rysc Corp.¹⁰ and GeZhi Electronic Corp. Ltd.¹¹. The Proxmark community¹² provides all the information that is required to assemble, compile, flash, use and develop new features for the Proxmark.

The Proxmark supports both low (125-134 kHz) and high frequency (13.56 MHz) signal processing. This is achieved by two parallel antenna circuits that can be used independently. Both circuits are connected to a 4-pin Hirose connector to connect an external loop antenna. When the Proxmark is in *reader mode* it drives the antenna coils with the appropriate frequency. This is unnecessary when the Proxmark works in *eavesdropping mode* or in *tag emulation mode* because then the electromagnetic field is generated by the reader. The signal from the antenna is filtered and routed through the FPGA after it has been digitized by an 8-bit Analog-to-Digital Converter (ADC). The FPGA relays the necessary information to perform the decoding of the signal to the micro-controller. This prevents the micro-controller from being overloaded with signal data. An FPGA has a great advantage over a normal micro-controller in the sense that it emulates hardware. A hardware description can be compiled and flashed into an FPGA. Basic arithmetic operations can be performed in parallel and faster than in a micro-controller. An FPGA is slightly slower than an Application-Specific Integrated Circuit (ASIC) implementation, however, pure hardware lacks flexibility.

⁹<http://cq.cx/proxmark3.pl>

¹⁰<http://www.proxmark3.com>

¹¹<http://www.xfpga.com>

¹²<http://www.proxmark.org>

There are several schematics of hardware improvements contributed to the Proxmark project. However, most of them focus on usability like an external battery, LCD screen or additional input buttons. The third hardware revision of the Proxmark, published in may 2007 by Jonathan Westhues¹³, is still the most used and best supported hardware version.

Since the focus of the Proxmark is merely on RFID technology, it supports a wide range of techniques that are used in this field. Most RFID communication schemes are working out-of-the-box using a command line interface. The Proxmark can work standalone which offers the functionality to perform a sequence of operations very quickly after each other. This is required for time-dependent operations like anti-collision procedures and distance bounding protocols [FGMR10, GFMR11, HK05, Han05]. The development of the Proxmark software is completely driven by open-source contributors who share their code freely in the public domain.

The user interface is controlled by predefined commands through a command line interface. These commands can be used to configure the hardware, run a specific test or to analyze the captured communication data. Almost all types of RFID tags that are used worldwide are supported. Some RFID tag-specific communication protocols are only partially implemented. However, the open source nature of the Proxmark enables a user to rapidly implement, prototype and contribute new features.

The Proxmark device is currently the most-often used tool for RFID security research. There are many examples in the literature [dKGHG08, GdKGM+08, GvRVWS09, GvRVWS10, GdKGV11, Hen11, HHJK12, GdKGV12, VGB12, PN12, VGE13, LXZ13, DvE14] that have prototyped, analyzed and verified their findings with the Proxmark tool. Besides security research, the Proxmark lends itself very well for experimenting, rapid prototyping, validation and certification of new communication techniques like the various anti-collision algorithms that were proposed in the academic community [FGMR10, GFMR11].

Despite the broad support of communication techniques, several custom and unsupported communication schemes were encountered in the course of this study. Several improvements and features were introduced to support additional techniques. Implementation details are available in the literature [VdKGG12] and the source code is embedded into the public repository¹⁴.

The Proxmark is used to experiment and verify the theoretical findings during the study and is presented in Part II (the technical section). The Proxmark proved itself to be versatile, reliable, fast and easy to extend. During the course of our research we extended the code base of the Proxmark device.

¹³<http://cq.cx/dl/proxmark3-may23-2007.zip>

¹⁴<https://github.com/Proxmark>

Part II

Technical section

Chapter 6

MIFARE Classic

The Mifare Classic is the most widely used contactless smartcard on the market. We reverse engineered the security mechanisms of this chip: the authentication protocol, the stream cipher CRYPTO1, and the initialization mechanism [GdKGM⁺08]. We show several security vulnerabilities in these mechanisms and exploit these vulnerabilities with two attacks; both are capable of retrieving the secret key from a genuine reader.

The most serious attack described in [GdKGM⁺08] retrieves a secret key in under a second. In order to clone a card, these attacks require that the adversary either has access to an eavesdropped communication session or executes a message-by-message man-in-the-middle attack between the victim and a legitimate reader. Although this is already disastrous from a cryptographic point of view, system integrators maintain that these attacks cannot be performed without being detected.

This chapter proposes four attacks that can be executed by an adversary having only wireless access to just a card (and not to a legitimate reader). The most serious of them recovers a secret key in less than a second on ordinary hardware. Besides the cryptographic weaknesses, we exploit other weaknesses in the protocol stack. A vulnerability in the computation of parity bits allows an adversary to establish a side channel. Another vulnerability regarding nested authentications provides enough plaintext for a speedy known-plaintext attack.

6.1 Introduction

With more than one billion cards sold, the Mifare Classic covers more than 70% of the contactless smartcard market¹. Such cards contain a slightly more powerful IC than classical RFID chips (developed for identification only), equipping them with modest computational power and making them suitable for applications beyond identification, such as access control and ticketing systems.

The Mifare Classic is widely used in public transport payment systems such as the Oyster card² in London, the Charlie Card in Boston³, the SmartRider in Australia⁴,

¹<http://www.nxp.com>

²<http://oyster.tfl.gov.uk>

³http://www.mbtta.com/fares_and_passes/charlie

⁴<http://www.transperth.wa.gov.au>

EasyCard in Taiwan⁵, and the OV-chipkaart⁶ in The Netherlands. It is also widely used for access control in office and governmental buildings and entrance to military bases.

According to [PHI98] the Mifare Classic complies with Parts 1 to 3 of the ISO standard 14443-A [ISO01], specifying the physical characteristics, the radio frequency interface, and the anti-collision protocol. The Mifare Classic does not implement Part 4 of the standard, describing the transmission protocol, but instead uses its own secure communication layer. In this layer, the Mifare Classic uses the proprietary stream cipher CRYPTO1 to provide data confidentiality and mutual authentication between card and reader.

Our contribution. In this chapter, we show serious vulnerabilities of the Mifare Classic that enable an attacker to retrieve all cryptographic keys of a card, just by wirelessly communicating with it. Thus, the potential impact is much larger than that of the problems previously reported in [WSvRG⁺08, NESP08, Noh08, CNO08, dKGGH08], where the attacker either needs to have access to a legitimate reader or an eavesdropped communication session. The attacks described in this chapter are fast enough to allow an attacker to wirelessly ‘pickpocket’ a victim’s Mifare Classic card, i.e., to clone it immediately.

Vulnerabilities

The vulnerabilities we discovered concern the handling of parity bits and nested authentications.

- The Mifare Classic sends a parity bit for each byte that is transmitted. Violating the standard, the Mifare Classic mixes the data link layer and secure communication layer: parity bits are computed over the plaintext instead of over the bits that are actually sent, i.e., the ciphertext. This is, in fact, authenticate-then-encrypt which is generically insecure [Kra01].

Furthermore, parity bits are encrypted with the same bit of keystream that encrypts the first bit of the next byte of plaintext. During the authentication protocol, if the reader sends wrong parity bits, the card stops communicating. However, if the reader sends correct parity bits, but wrong authentication data, the card responds with an (encrypted) error code. This breaks the confidentiality of the cipher, enabling an attacker to establish a side channel.

- The memory of the Mifare Classic is divided into sectors, each of them having its own 48-bit secret key. To perform an operation on a specific sector, the reader must first authenticate using the corresponding key. When an attacker has already authenticated for one sector (knowing the key for that sector) and

⁵<http://www.easycard.com.tw>

⁶<http://www.ov-chipkaart.nl>

subsequently attempts to authenticate for another sector (without knowing the key for this sector), that attempt leaks 32 bits of information about the secret key of that sector.

Attacks

We describe four attacks exploiting these vulnerabilities to recover the cryptographic keys from a Mifare Classic card having only contactless communication with it (and not with a legitimate reader). These attacks make different trade-offs between online communication time (the time an attacker needs to communicate with a card), offline computation time (the time it takes to compute the cryptographic key using the data gathered from the card), precomputation time (one-time generation time of static tables), disk space usage (of the static tables) and special assumptions (whether the attacker has already one sector key or not).

- The first attack exploits the weakness of the parity bits to mount an offline brute-force attack on the 48-bit key space. The attacker only needs to try to authenticate approximately 1500 times (which takes under a second).
- The second attack also exploits the weakness of the parity bits but this time the attacker mounts an adaptive chosen ciphertext attack. The attacker needs approximately 28 500 authentication attempts. In this attack, she needs to make sure that the challenge nonce of the card is constant, which is why this takes approximately fifteen minutes. During these authentication attempts, the attacker adaptively chooses her challenge to the card, ultimately obtaining a challenge that guarantees that there are only 436 possibilities for the odd-numbered bits of the internal state of the cipher. This reduces the offline search space to approximately 33 bits. On a standard desktop computer, with a single core running at 2 GHz, this search takes about one minute.
- In the third attack the attacker keeps her own challenge constant, but varies the challenge of the tag, again ultimately obtaining a special internal state of the cipher. These special states have to be precomputed and stored in a 384 GB table. This attack requires on average $2^{12} = 4096$ authentication attempts, which could in principle be done in about two minutes. A few extra authentication attempts allow efficient lookup in the table.
- The fourth attack assumes that the attacker has already recovered at least one sector key. When the attacker first authenticates for this sector and then for another sector, the authentication protocol is slightly different, viz., the challenge nonce of the tag is not sent in the clear, but encrypted with the key of the new sector. Because the random number generator has only a 16-bit state, because parity bits leak three bits of information, and because the tag's random number generator runs in sync with the communication timing, this allows an

attacker to guess the plaintext tag nonce and hence 32 bits of keystream. Due to weaknesses described in Section 6.3, we can use these 32 bits of keystream to compute approximately 2^{16} candidate keys. These can then be checked offline using another authentication attempt. Since this attack only requires three authentication attempts, the online time is negligible. The offline search takes under a second on ordinary hardware.

Related work

De Koning Gans et al. [dKGG08] have proposed an attack on a Mifare Classic tag that exploits the malleability of the CRYPTO1 stream cipher to read partial information from a tag, without even knowing the encryption algorithm. By slicing a Mifare Classic chip and taking pictures with a microscope, the cipher was reverse engineered by Nohl et al. [NESP08]. Courtois et al. claim in [CNO08] that the CRYPTO1 cipher is susceptible to algebraic attacks and Nohl shows a statistical weakness of the cipher in [Noh08].

Impact

The implications of the attacks described in this chapter are vast.

Many ticketing and payment systems using the Mifare Classic sequentially authenticate for several sectors verifying the data in the card. In the case of invalid data, the protocol aborts. With previous attacks, this means that an attacker has to either eavesdrop a full trace or walk from the reader to the card holder several times, executing a message-by-message man-in-the-middle attack. In practice, both options are hard to accomplish without being detected. Furthermore, there is no guarantee that this allows an attacker to recover all useful data in the card, since some sectors might not be read in this particular instance. Our attacks always enable an attacker to retrieve all data from the card.

Our fourth attack, where the attacker already knows a single key, is extremely fast (less than one second per key on ordinary hardware). The first key can be retrieved using one of our first three attacks, but in many situations this is not even necessary. Most deployed systems leave default keys for unused sectors or do not diversify keys at all. Nearly all deployed systems that do diversify have at least one sector key that is not diversified, namely for storing the diversification information. This is even specified in NXP's guideline for system integrators [NXP13]. This means that it is possible for an adversary to recover all keys necessary to read and write the sixteen sectors of a Mifare Classic 1k tag in less than sixteen seconds.

Overview

We start by gathering the relevant information that is already known about the Mifare Classic in Section 6.2: its logical structure, the encryption algorithm, the

authentication protocol and the initialization of the stream cipher, how to undo the initialization of the stream cipher, and information about how the tag generates its random numbers. The complete process of reverse-engineering the Mifare classic security mechanisms is addressed in Section 6.3. In Section 6.4, we continue with a precise description of the discovered weaknesses in the handling of the parity bits and nested authentications. In Section 6.5, we show how these weaknesses can be exploited to recover a sector key by communication with just a card. Section 6.6 gives some concluding remarks.

6.2 Background

6.2.1 Hardware setup

For this experiment we designed and built a custom device for tag emulation and eavesdropping. This device, called Ghost, is able to communicate with a contactless smart card reader, emulating a tag, and eavesdrop communication between a genuine tag and reader. The Ghost is completely programmable and is able to send arbitrary messages. We can also set the uid of the Ghost which is not possible with manufacturer tags. The hardware cost of the Ghost is approximately 40 USD. We also used a Proxmark⁷, a generic device for communication with RFID tags and readers, and programmed it to handle the ISO14443-A standard. As it provides similar functionality to the Ghost, we do not make a distinction between these devices in the remainder of the chapter.

On the reader side we used an OpenPCD reader⁸ and an Omnikey reader⁹. These readers contain a *MIFARE Classic* chip implementing the CRYPTO1 cipher and are fully programmable.

6.2.2 Communication

The physical layer and data link layer of the Mifare family of cards are described in the ISO standard 14443-A. We have used the Proxmark¹⁰ for communication; this device implements, among others, these two layers of this standard and can emulate both a card and a reader.

Using information from [dKGHG08] about the command codes of the Mifare Classic and through reverse-engineering the cryptographic aspects of the Mifare Classic, we implemented the functionality of a Mifare Classic reader on the Proxmark. Note that we can observe a tag's communication at the data link level, implying that we can observe the parity bits as well. Furthermore, we have the freedom to send arbitrary parity bits, which is not possible using stock commercial Mifare Classic readers.

⁷<http://cq.cx/proxmark3.pl>, <http://www.proxmark.org>

⁸<http://www.openpcd.org>

⁹<http://omnikey.aaitg.com>

¹⁰<http://www.proxmark.org/>

However, many newer NFC readers can be used to communicate with a Mifare Classic card as well and are capable of sending and receiving arbitrary parity bits.¹¹ We have also executed the attacks described in this chapter using an inexpensive (USD 30) stock commercial NFC reader. However, these readers are typically connected to a host PC using USB and it is harder to obtain accurate communication timing.

6.2.3 Memory structure of the Mifare Classic

The Mifare Classic tag is a memory chip with secure wireless communication capabilities. The memory of the tag is divided into sectors, each of which is further divided into blocks of sixteen bytes each. The last block of each sector is the sector trailer; it stores two secret keys and the access conditions for that sector.

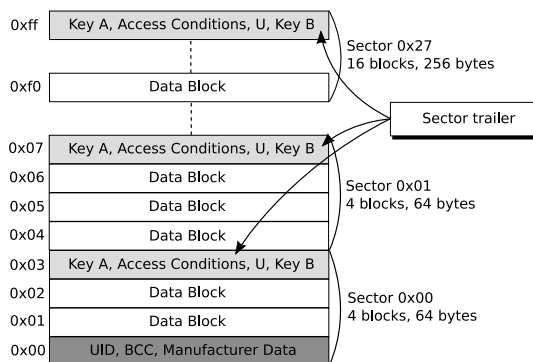


Figure 6.1: Memory layout of the Mifare Classic

To perform an operation on a specific block, the reader must first authenticate for the sector containing that block. The access conditions determine which of the two keys must be used. See Figure 6.1 for an overview of the memory of a Mifare Classic tag.

6.3 Reverse-engineering MIFARE Classic

This section describes the reverse engineering of the *MIFARE Classic* Classic chip. We do so by recording and studying traces from communication between tags and readers. We recover the encryption algorithm and the authentication protocol.

When the tag enters the electromagnetic field of the reader and powers up, it immediately starts the anti-collision protocol by sending its uid. The reader then selects this tag as specified in ISO14443-A [ISO01].

According to the manufacturer's documentation, the reader then sends an authentication request for a specific block. Next, the tag picks a challenge nonce n_T

¹¹<http://www.libnfc.org/>

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 tag. The tag finishes authentication 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 ks_1, ks_2, ks_3 . Figure 6.2 shows an example.

Step	Sender	Hex	Abstract
01	Reader	26	req type A
02	Tag	04 00	answer req
03	Reader	93 20	select
04	Tag	c2 a8 2d f4 b3	uid,bcc
05	Reader	93 70 c2 a8 2d f4 b3 ba a3	select(uid)
06	Tag	08 b6 dd	Mifare 1k
07	Reader	60 30 76 4a	auth(block 30)
08	Tag	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	Tag	8b d4 10 08	$a_T \oplus ks_3$

Figure 6.2: Authentication trace of the messages communicated between a genuine Mifare Classic card and an RFID reader

We started experimenting with the Ghost and an OpenPCD reader which we control. The pseudo-random generator in the tag 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 tag nonce every time. With the Ghost 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 it is restarted. Unlike in the tag, the state of the pseudo-random generator in the reader does not update every clock tick but with every invocation.

The pseudo-random generator in the tag used to generate n_T is a 16-bit LFSR with generating polynomial $x^{16} + x^{14} + x^{13} + x^{11} + 1$. 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 $n_k \oplus n_{k+2} \oplus n_{k+3} \oplus n_{k+5} \oplus n_{k+16} = 0$ for all $k \in \{0, 1, \dots, 15\}$. Every clock tick the LFSR shifts to the left and the feedback bit is computed using L_{16} .

Definition 6.3.1. *The feedback function $L_{16}: \mathbb{F}_2^{16} \rightarrow \mathbb{F}_2$ of the pseudo-random generator is defined by*

$$L_{16}(x_0x_1 \dots x_{15}) := x_0 \oplus x_2 \oplus x_3 \oplus x_5.$$

Let us define the function `suc` that computes the next 32-bit LFSR sequence of the 16-bit LFSR. This function is used later on in Section 6.3.1 in the authentication protocol.

Definition 6.3.2. *The successor function $\text{suc}: \mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$ is defined by*

$$\text{suc}(x_0x_1 \dots x_{31}) := x_1x_2 \dots x_{31}L_{16}(x_{16}x_{17} \dots x_{31}).$$

Because the period of the pseudo-random generator is only 65 535 and because it shifts every 9.44 μs , it cycles in 618 ms . Under similar physical conditions (i.e., do not move the tag or the reader), the challenge nonce that the tag generates only depends on the time between the moment the reader switches on the electromagnetic field and the moment it sends the authentication request. In practice, this means that an attacker who has physical control of the tag, can get the tag to send the same nonce every time. To do so, the attacker just has to drop the field (for approximately 30 μs) to discharge all capacitors in the tag, switch the field back on, and wait for a constant amount of time before authenticating.

Alternatively, by waiting exactly the right amount of time before authenticating again, the attacker can control the challenge nonce that the tag will send. This works whenever the tag does not leave the electromagnetic field in the mean time. On average, this takes $618 \text{ ms}/2 = 309 \text{ ms}$.

The Ghost can send arbitrary values as nonces and is not restricted to sending proper tag nonces. Experimenting with authentication sessions with various uids and tag nonces, we noticed that if $n_T \oplus \text{uid}$ remains constant, then the ciphertext of the encrypted reader nonce also remains constant. The answers a_T and a_R , however, have different ciphertexts in the two sessions. For example, in Figure 6.2 the uid is `0xc2a82df4` and n_T is `0x4297c0a4`, therefore $n_T \oplus \text{uid}$ is `0x803fed50`. If we instead take uid to be `0x1dfbe033` and n_T to be `0x9dc40d63`, then $n_T \oplus \text{uid}$ still equals `0x803fed50`. In both cases, the encrypted reader nonce $n_R \oplus \text{ks}_1$ is `0x7ddb9b83`. However, in Figure 6.2, $a_R \oplus \text{ks}_2$ is `0x67eb5d83` and $a_T \oplus \text{ks}_3$ is `0x8bd41008`, while with the modified uid and n_T they are, respectively, `0x4295c446` and `0xeb3ef7da`.

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 $a_R \oplus \text{ks}_2$ and $a'_R \oplus \text{ks}_2$ 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 $a'_T = \text{suc}^3(n'_T)$.

6.3.1 Tag and reader authentication protocol

Summarizing, the authentication protocol can be described as follows; see Figure 6.3. 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 $suc^2(n_T)$ encrypted, i.e., $suc^2(n_T) \oplus ks_2$. 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 $suc^3(n_T) \oplus ks_3$. Now the reader is able to verify the authenticity of the tag.

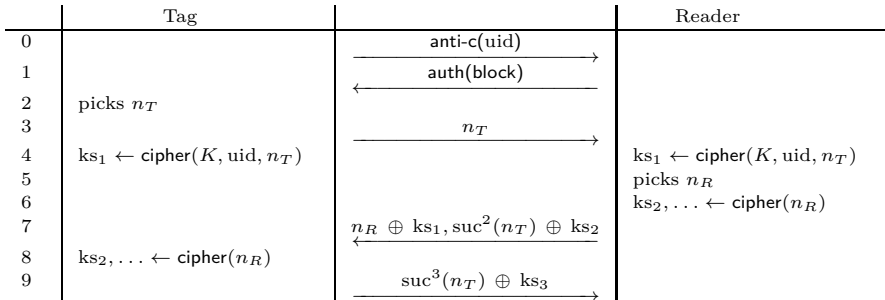


Figure 6.3: Mutual authentication protocol between tag and reader

6.3.2 Initialization

The LFSR is initialized during the authentication protocol. As before, we experimented running several authentication sessions with varying parameters. As we mention in Section 6.3.1, if $n_T \oplus \text{uid}$ remains constant, then the encrypted reader nonce also remains constant. This suggests that $n_T \oplus \text{uid}$ is first fed into the LFSR. 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. 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 the feedback polynomial of the LFSR. This suggests that the secret key K is the initial state of the LFSR.

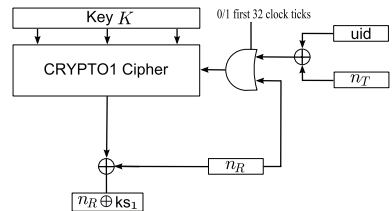


Figure 6.4: Initialization Diagram.

Proceeding to the next step in the authentication protocol, the reader nonce n_R is fed into the LFSR as well. 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 6.4 shows the initialization diagram for both reader and tag. The only difference between both sides is that the reader generates

n_R and then computes and sends $n_R \oplus ks_1$, while the tag receives $n_R \oplus ks_1$ and then computes n_R .

Note that we can, by selecting an appropriate key K , uid, and tag nonce n_T , completely control the 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 α , we often set the key to 0, let the Ghost select a uid of 0 and compute which n_T we should let the Ghost send to reach the state α . Now, because n_T is only 32 bits long and α is 48 bits long, this does not seem to allow us to control the leftmost 16 bits of α : 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 .

6.3.3 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 α of the LFSR by setting the uid, the key, and the tag nonce. As before, we use the Ghost 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 α of our choice, we can observe $n_{R,0} \oplus f(\alpha)$, since that is what is being sent by the reader. Since we power up the reader every time, the generated reader nonce is the same every time. 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 α and observe $n_{R,0} \oplus f(\alpha)$. We then vary a single bit in α , say the i th, giving state α' , 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 α , it is likely that the i th bit is not an input to f .

Sender	Hex	Hex	
Reader	26	26	req type A
Ghost	04 00	04 00	answer req
Reader	93 20	93 20	select
Ghost	00 00 00 00 00	00 00 00 00 00	uid,bcc
Reader	93 70 00 00 00 00 9c d9	93 70 00 00 00 00 9c d9	select(uid)
Ghost	08 b6 dd	08 b6 dd	Mifare 1k
Reader	60 00 f5 7b	60 00 F5 7B	auth(block 0)
Ghost	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	$n_R \oplus ks_1, \text{suc}^2(n_T) \oplus ks_2$

Figure 6.5: Nearly equal LFSR states

Figure 6.5 shows an example. The key in the reader (for block 0) is set to 0 and the Ghost sends a uid of 0. On the left hand side, the Ghost 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 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, \dots , 45, 47 are input to the filter function f . 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 6.7 for the structure of CRYPTO1). Note, that all these circuits implement balanced functions.

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

Figure 6.6: First bit of encrypted reader nonce

To verify our guess and to determine f , we again take a random state α 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). Figure 6.6 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 $f_a(x_3, x_2, x_1, x_0)$ represented by the boolean table 0x26c7 and the other three compute $f_b(x_3, x_2, x_1, x_0)$ represented by the boolean table 0x0dd3. I.e., from left to right the bits of 0x26c7 are the values of $f_a(1, 1, 1, 1), f_a(1, 1, 1, 0), \dots, f_a(0, 0, 0, 0)$ 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 $f_c(x_4, x_3, x_2, x_1, x_0)$ represented by the boolean table 0x4457c3b3. In this way we recovered the filter function f depicted in Figure 6.7.

6.3.4 CRYPTO1

After authentication, the communication between tag and reader is encrypted with the CRYPTO1 stream cipher. This cipher consists of a 48-bit linear feedback shift register (LFSR) with generating polynomial $x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} +$

$x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^9 + x^7 + x^6 + x^5 + 1$ and a non-linear filter function f [NESP08]. Each clock tick, twenty bits of the LFSR are put through the filter function, generating one bit of keystream. Then the LFSR shifts one bit to the left, using the generating polynomial to generate a new bit on the right. See Figure 6.7 for a schematic representation.

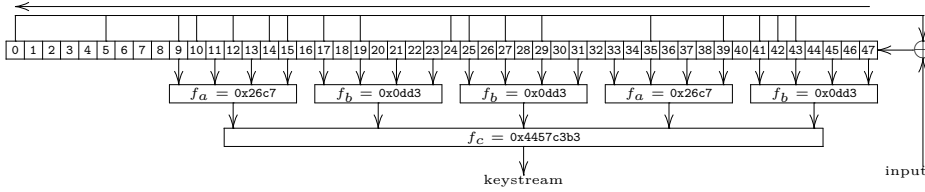


Figure 6.7: Structure of the CRYPTO1 stream cipher

Definition 6.3.3. *The feedback function $L: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$ is defined by*

$$L(x_0x_1 \dots x_{47}) := x_0 \oplus x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43}.$$

The specifics of the filter function are taken from [GdKGM+08].

Definition 6.3.4. *The filter function $f: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$ is defined by*

$$f(x_0x_1 \dots x_{47}) := f_c(f_a(x_9, x_{11}, x_{13}, x_{15}), f_b(x_{17}, x_{19}, x_{21}, x_{23}), f_b(x_{25}, x_{27}, x_{29}, x_{31}), f_a(x_{33}, x_{35}, x_{37}, x_{39}), f_b(x_{41}, x_{43}, x_{45}, x_{47})).$$

Here $f_a, f_b: \mathbb{F}_2^4 \rightarrow \mathbb{F}_2$ and $f_c: \mathbb{F}_2^5 \rightarrow \mathbb{F}_2$ are defined by

$$\begin{aligned} f_a(y_0, y_1, y_2, y_3) &:= ((y_0 \vee y_1) \oplus (y_0 \wedge y_3)) \oplus (y_2 \wedge ((y_0 \oplus y_1) \vee y_3)) \\ f_b(y_0, y_1, y_2, y_3) &:= ((y_0 \wedge y_1) \vee y_2) \oplus ((y_0 \oplus y_1) \wedge (y_2 \vee y_3)) \\ f_c(y_0, y_1, y_2, y_3, y_4) &:= (y_0 \vee ((y_1 \vee y_4) \wedge (y_3 \oplus y_4))) \\ &\quad \oplus (((y_0 \oplus (y_1 \wedge y_3)) \wedge ((y_2 \oplus y_3) \vee (y_1 \wedge y_4))). \end{aligned}$$

Because $f(x_0x_1 \dots x_{47})$ only depends on $x_9, x_{11}, \dots, x_{47}$, we shall overload notation and see f as a function $\mathbb{F}_2^{20} \rightarrow \mathbb{F}_2$, writing $f(x_0x_1 \dots x_{47})$ as $f(x_9, x_{11}, \dots, x_{47})$.

Note that f_a and f_b here are negated when compared to [GdKGM+08] and f_c is changed accordingly. The expressions for f_a , f_b , and f_c given here have the minimal number of logical operators in $\{\wedge, \vee, \oplus, \neg\}$; in practice, this allows for a fast bitsliced implementation of f [Bih97].

For future reference, note that each of the building blocks of f (and hence f itself) have the property that it is balanced.

Property 6.3.1. *Let Y_0, Y_1, \dots, Y_4 be independent uniformly distributed variables over \mathbb{F}_2 . Then*

$$\begin{aligned} P[f_a(Y_0, Y_1, Y_2, Y_3) = 0] &= 1/2 \\ P[f_b(Y_0, Y_1, Y_2, Y_3) = 0] &= 1/2 \\ P[f_c(Y_0, Y_1, Y_2, Y_3, Y_4) = 0] &= 1/2. \end{aligned}$$

Proof. By inspection. □

The following precisely defines the initialization of the cipher and the generation of the LFSR-stream $a_0a_1\dots$ and the keystream $b_0b_1\dots$.

Definition 6.3.5. *Given a key $k = k_0k_1\dots k_{47} \in \mathbb{F}_2^{48}$, a tag nonce $n_T = n_{T,0}n_{T,1}\dots n_{T,31} \in \mathbb{F}_2^{32}$, a uid $u = u_0u_1\dots u_{31} \in \mathbb{F}_2^{32}$, and a reader nonce $n_R = n_{R,0}n_{R,1}\dots n_{R,31} \in \mathbb{F}_2^{32}$, the internal state of the cipher at time i is $\alpha_i := a_i a_{i+1} \dots a_{i+47} \in \mathbb{F}_2^{48}$. Here the $a_i \in \mathbb{F}_2$ are given by*

$$\begin{aligned} a_i &:= k_i & \forall i \in [0, 47] \\ a_{48+i} &:= L(a_i, \dots, a_{47+i}) \oplus n_{T,i} \oplus u_i & \forall i \in [0, 31] \\ a_{80+i} &:= L(a_{32+i}, \dots, a_{79+i}) \oplus n_{R,i} & \forall i \in [0, 31] \\ a_{112+i} &:= L(a_{64+i}, \dots, a_{111+i}) & \forall i \in \mathbb{N}. \end{aligned}$$

Furthermore, we define the keystream bit $b_i \in \mathbb{F}_2$ at time i by

$$b_i := f(a_i a_{i+1} \dots a_{47+i}) \quad \forall i \in \mathbb{N}.$$

We denote encryptions by $\{-\}$ and define $\{n_{R,i}\}, \{a_{R,i}\} \in \mathbb{F}_2$ by

$$\begin{aligned} \{n_{R,i}\} &:= n_{R,i} \oplus b_{32+i} & \forall i \in [0, 31] \\ \{a_{R,i}\} &:= a_{R,i} \oplus b_{64+i} & \forall i \in [0, 31]. \end{aligned}$$

Note that the a_i , α_i , b_i , $\{n_{R,i}\}$, and $\{a_{R,i}\}$ are formally functions of k , n_T , u , and n_R . Instead of making this explicit by writing, e.g., $a_i(k, n_T, u, n_R)$, we just write a_i where k , n_T , u , and n_R are clear from the context.

6.3.5 Rollback

For our attacks it is important to realize that to recover the key, it is sufficient to learn the internal state of the cipher α_i at any point i in time. Since an attacker knows u , n_T , and $\{n_R\}$, the LFSR can then be rolled back to time zero [Rue86]. This is explained in detail in Section 6.2 of [GdKGM⁺08]; below we show their method translated into our notation.

Definition 6.3.6. *The rollback function $R: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$ is defined by*

$$R(x_1 x_2 \dots x_{48}) := x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{24} \oplus \\ x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43} \oplus x_{48}.$$

If one first shifts the LFSR left using L to generate a new bit on the right, then R recovers the bit that dropped out on the left, i.e.,

$$R(x_1 x_2 \dots x_{47} L(x_0 x_1 \dots x_{47})) = x_0. \quad (6.1)$$

Theorem 6.3.1. *In the situation from Definition 6.3.5, we have*

$$\begin{aligned} a_{64+i} &= R(a_{65+i} \dots a_{112+i}) && \forall i \in \mathbb{N} \\ a_{32+i} &= R(a_{33+i} \dots a_{80+i}) \oplus \{n_{R,i}\} \oplus \\ & f(0 a_{33+i} \dots a_{79+i}) && \forall i \in [0, 31] \\ a_i &= R(a_{1+i} \dots a_{48+i}) \oplus n_{T,i} \oplus u_i && \forall i \in [0, 31]. \end{aligned}$$

Proof. Straightforward, using Definition 6.3.5 and Equation (6.1). For the second equation, note that f does not depend on its leftmost input. Therefore

$$\begin{aligned} f(0 a_{33+i} \dots a_{79+i}) &= f(a_{32+i} \dots a_{79+i}) = b_{32+i} \\ \text{and hence } \{n_{R,i}\} \oplus f(0 a_{33+i} \dots a_{79+i}) &= n_{R,i}. \end{aligned}$$

□

Consequently, if an attacker somehow recovers the internal state of the LFSR $a_i = a_i a_{i+1} \dots a_{i+47}$ at some time i , then she can repeatedly apply Theorem 6.3.1 to recover $a_0 = a_0 a_1 \dots a_{47}$, which is the sector key.

6.4 Weaknesses

This section describes weaknesses in the design of the Mifare Classic. We first show that the filter function f is invertible in less than one second on ordinary hardware without the need for any precomputed tables. Next, we treat weaknesses in the way the Mifare Classic handles parity bits and then the ones concerning nested authentications. Finally, we show how to extract pure keystream from a reader without requiring any access to a genuine card. These weaknesses will be exploited in Section 6.5.

6.4.1 Odd Inputs to the Filter Function

The inputs to the filter function f are only on odd-numbered places. 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 into 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 $b_0 b_1 \dots b_{n-1}$ 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 $\bar{r} = r_0 r_1 \dots r_{46+n}$ of bits such that

$$\begin{aligned} r_k \oplus r_{k+5} \oplus r_{k+9} \oplus r_{k+10} \oplus r_{k+12} \oplus r_{k+14} \oplus r_{k+15} \oplus r_{k+17} \\ \oplus r_{k+19} \oplus r_{k+24} \oplus r_{k+25} \oplus r_{k+27} \oplus r_{k+29} \oplus r_{k+35} \oplus r_{k+39} \oplus r_{k+41} \\ \oplus r_{k+42} \oplus r_{k+43} \oplus r_{k+48} = 0, \text{ for all } k \in \{0, \dots, n-2\}, \end{aligned} \quad (6.2)$$

and such that

$$f(r_k \dots r_{k+47}) = b_k, \text{ for all } k \in \{0, \dots, n-1\}. \quad (6.3)$$

Condition (6.2) says that \bar{r} is generated by the LFSR, i.e., that $r_0 r_1 \dots r_{47}$, $r_1 r_2 \dots r_{48}$, \dots are successive states of the LFSR; Condition (6.3) 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 (6.2) and (6.3), 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 $s_0 s_1 \dots s_{19}$ 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 to verify 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 (6.2), i.e., the 49th must be the feedback generated by the previous 48 bits.

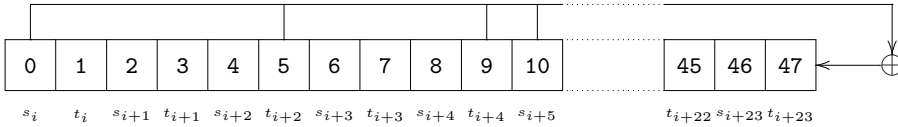


Figure 6.8: Subsequences \bar{s} and \bar{t} .

So, for every subsequence $s_i s_{i+1} \dots s_{i+24}$ of 25 consecutive bits of \bar{s} we compute its contribution $b_i^{1,\bar{s}} = s_k \oplus s_{i+5} \oplus s_{i+6} \oplus s_{i+7} \oplus s_{i+12} \oplus s_{i+21} \oplus s_{i+24}$ 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 6.8). If $s_0 t_0 s_1 \dots t_{n/2}$ is indeed generated by the LFSR, then

$$b_i^{1,\bar{s}} = b_i^{2,\bar{t}} \text{ for all } i \in \{0, \dots, n/2 - 5\}. \tag{6.4}$$

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

$$\tilde{b}_i^{1,\bar{s}} = \tilde{b}_i^{2,\bar{t}} \text{ for all } i \in \{0, \dots, n/2 - 5\}. \tag{6.5}$$

One readily sees that together, conditions (6.4) and (6.5) are equivalent to equation (6.2).

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 $b^{1,\bar{s}} \tilde{b}^{1,\bar{s}} = b^{2,\bar{t}} \tilde{b}^{2,\bar{t}}$ 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 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* Classic [JWS11].

6.4.2 Parity weaknesses

The ISO standard 14443-A [ISO01] specifies that every byte sent is followed by a parity bit. The Mifare Classic computes parity bits over the plaintext instead of over the ciphertext. Additionally, the bit of keystream used to encrypt the parity bits is reused to encrypt the next bit of plaintext.

This already breaks the confidentiality of the encryption scheme. In this chapter we shall only be concerned with the four parity bits of n_T , n_R , and a_R . The ISO standard specifies odd parity, hence the “ $\oplus 1$ ” in the definition below.

Definition 6.4.1. *In the situation from Definition 6.3.5, we define the parity bits $p_j \in \mathbb{F}_2$ by*

$$\begin{aligned} p_j &:= n_{T,8j} \oplus n_{T,8j+1} \oplus \dots \oplus n_{T,8j+7} \oplus 1 & \forall j \in [0, 3] \\ p_{j+4} &:= n_{R,8j} \oplus n_{R,8j+1} \oplus \dots \oplus n_{R,8j+7} \oplus 1 & \forall j \in [0, 3] \\ p_{j+8} &:= a_{R,8j} \oplus a_{R,8j+1} \oplus \dots \oplus a_{R,8j+7} \oplus 1 & \forall j \in [0, 3] \end{aligned}$$

and the encryptions $\{p_j\}$ of these by

$$\{p_j\} := p_j \oplus b_{8+8j} \quad \forall j \in [0, 11].$$

There is a further weakness concerning the parity bits. During the authentication protocol, when the reader sends $\{n_R\}$ and $\{a_R\}$, the tag checks the parity bits before the answer of the reader. If at least one of the eight parity bits is wrong, the tag does not respond. If all eight parity bits are correct, but the answer a_R is wrong, the tag responds with the 4-bit error code 0x5 signifying failed authentication, called ‘transmission error’ in [dKGHG08]. If all eight parity bits are correct and the answer a_R is also correct, the tag responds, of course, with its answer a_T . Furthermore, in

case the reader sends the correct parity, but the wrong answer, the 4-bit error code 0x5 is sent encrypted. This happens even though the reader has not authenticated itself and hence cannot be assumed to be able to decrypt.

Figure 6.9 shows an authentication trace where the attacker sends incorrect authentication data but correct parity bits. The exclamation marks represent parity bits that deviate from what is specified in the standard. The final message of this trace is the encrypted error message 0x5.

Reader	26	req type A
Tag	02 00	answer req
Reader	93 20	select
Tag	c1 08 41 6a e2	uid, bcc
Reader	93 70 c1 08 41 6a e2 e4 7c	select(uid)
Tag	18 37 cd	Mifare Classic 4k
Reader	60 00 f5 7b	auth(block 0)
Tag	ab cd 19 49	n_T
Reader	59! d5 92 0f! 15 b9 d5! 53!	$\{n_R\}\{a_R\}$
Tag	a	$\{5\}$

Figure 6.9: Trace of a failed authentication attempt

6.4.3 Nested authentications

Once an attacker knows a single sector key of a Mifare Classic, there is a vulnerability that allows an adversary to recover more keys. When a reader is already communicating (encrypted) with a tag, a subsequent authentication command for a new sector also has to be sent encrypted. After this authentication command, the internal state of the cipher is set to the key for the new sector and the authentication protocol from Section 6.3.1 starts again. This time, however, the challenge of the tag is also sent encrypted. Because there are only 2^{16} possible nonces, an attacker can simply try to guess a nonce to recover 32 bits of keystream.

Also here, the information that leaks through the parity bits can be used to speed up the attack. Although there are 2^{16} tag nonces, we show below that the parity bits sent with the encrypted tag nonce leak three bits of information, so that there are only 2^{13} tag nonces possible.

Definition 6.4.2. *In the situation from Definition 6.3.5, we define $\{n_{T,i}\} \in \mathbb{F}_2$ by*

$$\{n_{T,i}\} := n_{T,i} \oplus b_i \quad \forall i \in [0, 31].$$

Theorem 6.4.1. *For every $j \in \{0, 1, 2\}$ we have*

$$n_{T,8j} \oplus n_{T,8j+1} \oplus \cdots \oplus n_{T,8j+7} \oplus n_{T,8j+8} = \{p_j\} \oplus \{n_{T,8j+8}\} \oplus 1$$

Proof. We compute as follows.

$$\begin{aligned}
& n_{T,8j} \oplus n_{T,8j+1} \oplus \cdots \oplus n_{T,8j+7} \oplus n_{T,8j+8} \\
&= p_j \oplus \mathbf{1} \oplus n_{T,8j+8} && \text{(by Dfn. 6.4.1)} \\
&= p_j \oplus b_{8+8j} \oplus n_{T,8j+8} \oplus b_{8+8j} \oplus \mathbf{1} \\
&= \{p_j\} \oplus \{n_{T,8j+8}\} \oplus \mathbf{1} && \text{(by Dfns. 6.4.1 and 6.4.2)}
\end{aligned}$$

□

Since the attacker can observe $\{p_j\}$ and $\{n_{T,8j+8}\}$, this theorem gives an attacker three bits of information about n_T .

In practice, timing information between the first and second authentication attempt leaks so much additional information that the attacker can accurately predict what the challenge nonce will be.

It turns out that the distance between the tag nonces used in consecutive authentication attempts strongly depends on the time between those attempts. Here distance is defined as follows.

Definition 6.4.3. Let n_T and n'_T be two tag nonces. We define the distance between n_T and n'_T as

$$d(n_T, n'_T) := \min_{i \in \mathbb{N}} \text{suc}^i(n_T) = n'_T.$$

6.4.4 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 ks_2 (i.e., 32 bits of keystream) by computing $\text{suc}^2(n_T)$ and XOR-ing.

Moreover, experiments show that if the tag does not send anything, then most readers will time out and send a `halt` command. Since communication is encrypted it actually sends `halt` \oplus ks_3 . Knowing the byte code of the `halt` command (0x500057cd [ISO01]) we recover ks_3 .

Some readers do not send a `halt` command but instead continue as if authentication succeeded. This typically means that it sends an encrypted `read` command. As the byte code of the `read` command is also known [dKGGH08], this also enables us to recover ks_3 by guessing the block number.

It is important to note that one can obtain such an authentication session (or rather, a partial authentication session, as the Ghost never authenticates itself) from a reader (and hence ks_2 , ks_3) without knowing the secret key and, in fact, without using a tag.

If an attacker does have access to both a tag and a reader and can eavesdrop a successful (complete) authentication session, then both ks_2 and ks_3 can be recovered from the answers $\text{suc}^2(n_T) \oplus \text{ks}_2$ and $\text{suc}^3(n_T) \oplus \text{ks}_3$ of the tag and the reader. This works even if the reader does not send `halt` or `read` after timeout.

6.5 Attacks

This section shows how the weaknesses described in the previous section can be exploited.

6.5.1 Brute-force attack

The attacker plays the role of a reader and tries to authenticate for a sector of her choice. She answers the challenge of the tag with eight random bytes (and eight random parity bits) for $\{n_R\}$ and $\{a_R\}$. With probability $1/256$, the 8 parity bits are correct and the tag responds with the encrypted 4-bit error code. A success leaks 12 bits of entropy (out of 48).

Repeating the above procedure sufficiently at least four times (in practice six is enough) uniquely determines the key. Since the key length is only 48 bits, the attacker can now brute force the key: she can just check which of the 2^{48} keys produces all six times the correct parity bits and received response. In practice, gathering those six authentication sessions with correct parity bits only takes on average $6 \cdot 256 = 1536$ authentication attempts which can be done in less than one second. The time it takes to perform the offline brute-force attack of course is strongly dependent on the resources the attacker has at her disposal. We give an estimate based on the performance of COPACOBANA [KPP⁺06]; this is a code-cracker built from off-the-shelf hardware costing approximately USD 10 000. Based on the fact that COPACOBANA finds a 56-bit DES key in on average 6.4 days, pessimistically assuming that one can fit the same number of CRYPTO1 checks on an FPGA as DES-decryptions, and realizing that the search space is a factor of 256 smaller, we estimate that this takes on average $6.4 \text{ days}/256 = 36 \text{ min}$.

In Sections 6.5.2 and 6.5.3 the same idea is exploited in a different way, trading online communication for computation time.

6.5.2 Varying the reader nonce

This section shows how an attacker can mount a chosen ciphertext attack by adaptively varying the encryption of n_R . We assume that the attacker can control the power up timing of the tag, thereby causing the tag to produce the same n_T every time.

We first give the idea of the attack. The attacker runs authentication sessions until she guesses the correct parity bits. The internal state of the stream cipher just after feeding in n_R is α_{64} . She then runs another authentication session, keeping the first 31 bits of $\{n_R\}$ (and the three parity bits) the same, flipping the last bit of $\{n_R\}$ (and randomly picking the rest until the parity is ok). Now the state of the stream cipher just after feeding in the reader nonce is $\alpha_{64} \oplus 1$, i.e., α_{64} with the last bit flipped. Since the parity of the last byte of n_R changed (since the attacker flipped

just the last bit), and since its parity in the first run is encrypted with $f(\alpha_{64})$ and in the second run with $f(\alpha_{64} \oplus 1)$, she can deduce whether or not the last bit of n_R influences the encryption of the next bit, i.e., whether or not $f(\alpha_{64}) = f(\alpha_{64} \oplus 1)$. Approx. 9.4% of the possible α_{64} 's has $f(\alpha_{64}) \neq f(\alpha_{64} \oplus 1)$ and they can easily be generated since only the twenty bits that are input to f are relevant. By repeating this, the attacker eventually (on average after 10.6 tries) finds an instance in which α_{64} is in those 9.4% and then she only has to search, offline, 9.4% of all possible states.

We now make this idea precise and at the same time generalize it to the last bit of each of the four bytes in the reader nonce. The following definition says that a reader nonce has property F_j (for $j \in \{0, 1, 2, 3\}$) if flipping the last bit of the $(j + 1)$ th byte of the reader nonce changes the encryption of the next bit.

Definition 6.5.1. *Let $j \in \{0, 1, 2, 3\}$ and let n_R and n'_R be reader nonces with the property that $n'_{R,8j+7} = \overline{n_{R,8j+7}}$ and $n'_{R,i} = n_{R,i}$ for all $i < 8j + 7$ (and no restrictions on $n_{R,i}$ and $n'_{R,i}$ for $i > 8j + 7$). We say that n_R has property F_j if $b_{8j+40} \neq b'_{8j+40}$.*

Formally this is not just a property of n_R , but also of k , n_T , and u . Now k and u of course do not vary, so we ignore that here. Furthermore, when deciding whether or not n_R has property F_j in Protocol 6.5.1 below, the attacker also keeps n_T constant.

The attacker does change the reader nonce. We use a'_i to refer to the bits of the LFSR-stream where the reader nonce n'_R is used and similarly for α'_i, b'_i , etc. I.e., a'_i denotes $a_i(k, n_T, u, n'_R)$.

Note that α_{8j+40} (resp. α'_{8j+40}) is the internal state of the cipher just after feeding in the $(j + 1)$ th byte of n_R (resp. n'_R) and $b_{8j+40} = f(\alpha_{8j+40})$ (resp. $b'_{8j+40} = f(\alpha'_{8j+40})$), so that F_j does not depend on $n_{R,i}$ and $n'_{R,i}$ for $i > 8j + 7$. Also observe that $\alpha'_{8j+40} = a_{8j+40} \dots a_{8j+86} a'_{8j+87}$, i.e., α_{8j+40} and α'_{8j+40} only differ in the last position.

The crucial idea is that an attacker can decide whether or not n_R has property F_j , only knowing $\{n_R\}$. (In practice, the attacker of course *chooses* $\{n_R\}$.)

Protocol 6.5.1. *Given $\{n_R\}$, an attacker can decide as follows whether or not n_R has property F_j . She first chooses $\{a_R\}$ arbitrary. She then starts, consecutively, several authentication sessions with the tag. After the tags sends its challenge n_T , the attacker answers $\{n_R\}, \{a_R\}$. Inside this answer, the attacker also has to send the (encryptions of) the parity bits: $\{p_4\}, \dots, \{p_{11}\}$. For these, she tries all 256 possibilities. After on average 128 authentication sessions, and after at most 256, with different choices for the $\{p_i\}$, the parity bits are correct and the attacker recognizes this because the tag responds with an error code.*

Now the attacker defines $\{n'_{R,8j+7}\} := \overline{\{n_{R,8j+7}\}}$, i.e., she changes the last bit of the j th byte of $\{n_R\}$. The earlier bits of $\{n'_R\}$ she chooses the same as those of $\{n_R\}$; the later bits of $\{n'_R\}$ and $\{a'_R\}$ the attacker chooses arbitrarily. Again, the attacker repeatedly tries to authenticate to find the correct parity bits $\{p'_i\}$ to send.

Note that necessarily $\{p'_i\} = \{p_i\}$ for $i \in \{4, \dots, j+3\}$, so this takes on average 2^{7-j} authentication attempts and at most 2^{8-j} .

Now n_R has property F_j if and only if $\{p_{j+4}\} \neq \{p'_{j+4}\}$.

Proof. Because the attacker modified the ciphertext of the last bit of the j th byte of n_R , the last bit of the plaintext of this byte also changes: $n'_{R,8j+7} = \{n'_{R,8j+7}\} \oplus b'_{8j+39} = \{n'_{R,8j+7}\} \oplus b'_{8j+39} = \overline{\{n_{R,8j+7}\}} \oplus b_{8j+39} = \overline{n_{R,8j+7}} \oplus b_{8j+39} \oplus b_{8j+39} = \overline{n_{R,8j+7}}$. Hence, the parity of this byte changes: $p'_{j+4} = n'_{R,8j} \oplus \dots \oplus n'_{R,8j+6} \oplus n'_{R,8j+7} \oplus 1 = n_{R,8j} \oplus \dots \oplus n_{R,8j+6} \oplus \overline{n_{R,8j+7}} \oplus 1 = \overline{p_{j+4}}$.

Now $\{p_{j+4}\} \oplus \{p'_{j+4}\} = p_{j+4} \oplus b_{8j+40} \oplus p'_{j+4} \oplus b'_{8j+40} = p_{j+4} \oplus b_{8j+40} \oplus \overline{p_{j+4}} \oplus b'_{8j+40} = b_{8j+40} \oplus b'_{8j+40}$. Hence $\{p_{j+4}\} = \{p'_{j+4}\}$ if and only if $b_{8j+40} = b'_{8j+40}$, i.e., $\{p_{j+4}\} \neq \{p'_{j+4}\}$ if and only if n_R has property F_j . \square

The theorem below shows that the probability that n_R has the property F_j is approximately 9.4%.

Lemma 6.5.2. *Let Y_0, \dots, Y_4 be independent uniformly distributed random variables over \mathbb{F}_2 . Then*

$$\begin{aligned} P[f_b(Y_0, Y_1, Y_2, Y_3) \neq f_b(Y_0, Y_1, Y_2, \overline{Y_3})] &= \frac{1}{4} \\ P[f_c(Y_0, Y_1, Y_2, Y_3, Y_4) \neq f_c(Y_0, Y_1, Y_2, Y_3, \overline{Y_4})] &= \frac{3}{8}. \end{aligned}$$

Proof. By inspection. \square

Theorem 6.5.3. *Let $Y_0, Y_1, \dots, Y_{18}, Y_{19}$ be independent uniformly distributed random variables over \mathbb{F}_2 . Then*

$$P[f(Y_0, Y_1, \dots, Y_{18}, Y_{19}) \neq f(Y_0, Y_1, \dots, Y_{18}, \overline{Y_{19}})] = \frac{3}{32}.$$

Proof. Write $Z_0 := f_a(Y_0, \dots, Y_3)$, $Z_1 := f_b(Y_4, \dots, Y_7)$, $Z_2 := f_b(Y_8, \dots, Y_{11})$, $Z_3 := f_a(Y_{12}, \dots, Y_{15})$, and $Z_4 := f_b(Y_{16}, \dots, Y_{19})$. Furthermore, write $Z'_4 := f_b(Y_{16}, \dots, Y_{18}, \overline{Y_{19}})$. Note that Z_0, \dots, Z_4 are independent and, by Theorem 6.3.1, uniformly distributed over \mathbb{F}_2 . Then

$$\begin{aligned} &P[f(Y_0, Y_1, \dots, Y_{18}, Y_{19}) \neq f(Y_0, Y_1, \dots, Y_{18}, \overline{Y_{19}})] \\ &= P[f_c(Z_0, \dots, Z_4) \neq f_c(Z_0, \dots, Z_3, Z'_4)] \\ &= P[f_c(Z_0, \dots, Z_4) \neq f_c(Z_0, \dots, Z'_4) | Z_4 \neq Z'_4] \\ &\quad \cdot P[Z_4 \neq Z'_4] \\ &= P[f_c(Z_0, \dots, Z_3, 0) \neq f_c(Z_0, \dots, Z_3, 1)] \\ &\quad \cdot P[f_a(Y_{16}, \dots, Y_{18}, 0) \neq f_a(Y_{16}, \dots, Y_{18}, 1)] \\ &= \frac{3}{8} \cdot \frac{1}{4} \quad \text{(by Lemma 6.5.2)} \\ &= \frac{3}{32}. \end{aligned}$$

Alternatively, one can also obtain this result by simply checking all 2^{20} possibilities. \square

We now describe how an attacker can find an $\{n_R\}$ such that n_R has all four properties F_j . Recall that these properties also depend on n_T and it is possible that for a fixed n_T no n_R has all four properties. In that case, as is explained in the protocol below, the attacker makes the tag generate a different n_T and starts the search again.

Protocol 6.5.4. *An attacker can find $\{n_R\}$ such that n_R has properties F_0, F_1, F_2, F_3 in a backtracking fashion. She first loops over all possibilities for the first byte of $\{n_R\}$ (taking the other bytes of $\{n_R\}$ arbitrary). Using Protocol 6.5.1, the attacker decides if n_R has property F_0 (which only depends on the first byte). If it has, she continues with the second byte of $\{n_R\}$, looping over all possibilities for the second byte of $\{n_R\}$ while keeping the first byte fixed, trying to find $\{n_R\}$ such that n_R also has property F_1 . She repeats this for the third and fourth byte of $\{n_R\}$. If at some stage no possible byte has property F_j , the search backtracks to the previous stage. It fails at the first stage, the attacker has to try a different tag nonce.*

By simulating this protocol (for a random key and random uid, and a random tag nonce in every outer loop of the search), we can estimate the number of authentication attempts needed to find a reader nonce having all four properties F_j .

Observation 6.5.5. *The expected number of authentication attempts needed to find an n_R which has all four properties F_j is approximately 28 500.*

Table 6.1: Odd bits of α_{64} ending in 0 when n_R has all properties F_j

```

0x000041414110 0x000041414140 0x000141414110 0x000141414140 0x000441414110 0x000441414140 0x000441414140 0x001441414110 0x001441414140 0x001541414110 0x001541414140 0x004441414110 0x004441414140 0x005141414110 0x005141414140 0x010041414110 0x010041414140
0x010141414110 0x010141414140 0x010441414110 0x010441414140 0x011441414110 0x011441414140 0x011541414110 0x011541414140 0x014141414110 0x014141414140 0x014441414110 0x014441414140 0x015141414110 0x015141414140 0x040010414110 0x040010414140
0x040040414110 0x040040414140 0x040041414110 0x040041414140 0x040110414110 0x040110414140 0x040111414110 0x040111414140 0x0401141414110 0x040114141440 0x040114141410 0x040114141440 0x0401151414110 0x040115141440 0x0404151414110 0x040415141440
0x041440414110 0x041440414140 0x041441414110 0x041441414140 0x041510414110 0x041510414140 0x041511414110 0x041511414140 0x0415141414110 0x041514141440 0x041541414110 0x041541414140 0x044141414110 0x044141414140 0x044410414110 0x044410414140
0x044440414110 0x044440414140 0x044441414110 0x044441414140 0x045141414110 0x045141414140 0x045141414140 0x045141414140 0x045141414140 0x045141414140 0x045141414140 0x045141414140 0x045141414140 0x045141414140 0x045141414140 0x045141414140
0x140141414140 0x140441414110 0x140441414140 0x141441414110 0x141441414140 0x141541414110 0x141541414140 0x141541414140 0x141541414140 0x141541414140 0x141541414140 0x141541414140 0x141541414140 0x141541414140 0x141541414140 0x141541414140
0x144441414110 0x144441414140 0x145141414110 0x145141414140 0x150041414110 0x150041414140 0x1504141414110 0x150414141440 0x150414141410 0x150414141440 0x150414141410 0x150414141440 0x150414141410 0x150414141440 0x150414141410 0x150414141440
0x155141414110 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140 0x155141414140
0x410041414140 0x410110414110 0x410110414140 0x410111414110 0x410111414140 0x4101141414110 0x410114141440 0x410114141410 0x410114141440 0x410114141410 0x410114141440 0x410114141410 0x410114141440 0x410114141410 0x410114141440 0x410114141410
0x410441414110 0x410441414140 0x411410414110 0x411410414140 0x411141414110 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140
0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140 0x411141414140
0x414141414110 0x414141414140 0x414410414110 0x414410414140 0x414411414110 0x414411414140 0x414411414140 0x414411414140 0x414411414140 0x414411414140 0x414411414140 0x414411414140 0x414411414140 0x414411414140 0x414411414140 0x414411414140
0x414411414140 0x415141414110 0x415141414140 0x440041414110 0x440041414140 0x440141414110 0x440141414140 0x440441414110 0x440441414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140
0x44141414110 0x44141414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140 0x44141414110 0x44141414140
0x445141414140 0x510010414110 0x510010414140 0x510011414110 0x510011414140 0x510040414110 0x510040414140 0x510041414110 0x510041414140 0x51041414110 0x51041414140 0x51041414110 0x51041414140 0x51041414110 0x51041414140 0x51041414110 0x51041414140
0x51041414140 0x511141041410 0x511141041440 0x511141414110 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140 0x511141414140
0x511510414110 0x511510414140 0x511511414110 0x511511414140 0x511540414110 0x511540414140 0x511541414110 0x511541414140 0x511541414140 0x511541414140 0x511541414140 0x511541414140 0x511541414140 0x511541414140 0x511541414140 0x511541414140 0x511541414140
0x514141414140 0x514410414110 0x514410414140 0x514411414110 0x514411414140 0x514411414140 0x514440414110 0x514440414140 0x514441414110 0x514441414140 0x514441414140 0x514441414140 0x514441414140 0x514441414140 0x514441414140 0x514441414140
0x515141414110 0x515141414140

```

Once the attacker has found an n_R having all four properties F_j , the number of possibilities for the internal state of the cipher after feeding in this particular n_R is seriously restricted. The following theorem states how many possibilities there still are.

Theorem 6.5.6. *Suppose that n_R has properties $F_0, F_1, F_2,$ and F_3 . Then there are only 436 possibilities for the odd-numbered bits of α_{64} . Table 6.1 lists (in hexadecimal,*

Table 6.2: Excerpt from table T_{0xa04} of internal cipher states α_{32} at index $0xa04$

```

0x000004d4d1f 0x000001247b8b 0x000001513ca3 0x0000049e0e78 0x000004cafec1 0x000006f945be 0x000007089ea5 0x0000072b67df 0x000008e7948e
0x00000a137cd9 0x00000aed7467 0x00000b92342b 0x00000c6db6a0 0x00000cbd2daa 0x00000cda7817 0x00000d0c8bd27 0x00000e98af03 0x00001089393d
0x0000129478db 0x000012f4cde6 0x000015382c19 0x000016a7a95c 0x0000172bbeb6 0x0000173f2299 0x00001821aa0a 0x000018769666 0x00001a6d513e
0x00001b1c2ff7 0x00001c259261 0x00001c46edf7 0x00001c5a3fde 0x00001c97ee44 0x00001f19da5e 0x00001fef9ec2 0x000022ce6797 0x000023a396ce
0x000023a92baa 0x000026bc6e18 0x0000278a7954 ...

```

with zeros on the places of the even-numbered bits) the 218 of those possibilities that have the last bit a_{111} equal to 0; the other 218 are the same except that they have a_{111} equal to 1.

Proof. By explicit computation. For each of the 2^{24} elements $y_0y_1 \dots y_{23}$ of \mathbb{F}_2^{24} , one checks if

$$\begin{aligned}
 f(y_4, y_5, \dots, y_{23}) &\neq f(y_4, y_5, \dots, \overline{y_{23}}), \\
 f(y_0, y_1, \dots, y_{19}) &\neq f(y_0, y_1, \dots, \overline{y_{19}}), \\
 &\text{and there exist } y_{-8}, y_{-7}, \dots, y_{-1} \in \mathbb{F}_2 \text{ such that} \\
 f(y_{-4}, y_{-3}, \dots, y_{15}) &\neq f(y_{-4}, f_{-3}, \dots, \overline{y_{15}}) \\
 &\text{and} \\
 f(y_{-8}, y_{-7}, \dots, y_{11}) &\neq f(y_{-8}, f_{-7}, \dots, \overline{y_{11}}).
 \end{aligned}$$

□

Consequently, when the attacker has found a reader nonce n_R that has properties F_0 , F_1 , F_2 , and F_3 , there are only $436 \cdot 2^{24} \approx 2^{32.8} \approx 7.3 \cdot 10^9$ possibilities for the internal state α_{64} of the cipher just after shifting in the reader nonce. Using Theorem 6.3.1, these can be used to compute $7.3 \cdot 10^9$ candidate keys. The attacker can then check these candidate keys by trying to decrypt the received 4-bit error messages.

6.5.3 Varying the tag nonce

In the previous approach, the attacker kept n_T constant and tried to find a special $\{n_R\}$ such that she gained knowledge about the internal cipher state. Now the attacker does the opposite: she keeps $\{n_R\}$ (and $\{a_R\}$ and the $\{p_i\}$ as well) constant, but varies n_T instead. As before, the attacker waits for the tag to respond; when this happens, she gains knowledge about the internal state of the cipher.

Protocol 6.5.7. *The attacker repeatedly tries to authenticate to the tag, every time with a different tag nonce n_T and sending all zeros as its response (including the encrypted parity bits), i.e., $\{n_R\} = 0$, $\{a_R\} = 0$, $\{p_4\} = \dots = \{p_{11}\} = 0$. She waits for an n_T such that the tag actually responds (i.e., the parity bits are the correct parity bits) and where the encrypted error code is $0x5$ (i.e., $b_{96} = b_{97} = b_{98} = b_{99} = 0$).*

Note that twelve bits have to be ‘correct’ (the eight parity bits and the four keystream bits), so this will take on average $2^{12} = 4096$ authentication attempts.

The following defines a large table that needs to be precomputed.

Definition 6.5.2.

$$T := \{\alpha_{32} \in \mathbb{F}_2^{48} \mid \{n_R\} = \{a_R\} = 0 \Rightarrow \{p_4\} = \dots = \{p_{11}\} = b_{96} = \dots = b_{99} = 0\}.$$

So the attacker knows that after the tag sends the challenge n_T found in Protocol 6.5.7, the current state of the cipher, α_{32} , appears in T . Now T can be precomputed; one would expect it to contain $2^{48}/2^{12} = 2^{36}$ elements; in fact, it contains 0.82% fewer elements due to a small bias in the cipher. In principle, the attacker could now use Theorem 6.3.1 to roll back each of the LFSRs in the table to find candidate keys and check each of these keys against a few other attempted authentication sessions.

In practice, searching through T takes about one day, which is undesirable. The attacker can shrink the search space by splitting T as follows.

Protocol 6.5.8. *After finding n_T in Protocol 6.5.7, the attacker again repeatedly tries to authenticate to the tag, every time with the tag nonce n_T she just found. Instead of zeros, she now sends ones for the response and this time she tries all possibilities for the encrypted parity bits until the tag responds with an encrypted error code. I.e., $\{n_R\} = 0\mathbf{x}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}$ and $\{a_R\} = 0\mathbf{x}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}$ and successively tries all possibilities for $\{p_4\}, \dots, \{p_{11}\}$ until one is correct.*

This time, because eight bits have to be ‘correct’, on average 128 authentication attempts are needed.

The table T can be split into $2^{12} = 4096$ parts indexed by the eight encrypted parity bits and four keystream bits that encrypt the error code.

Definition 6.5.3. *For every $\gamma = \gamma_0 \dots \gamma_{11} \in \mathbb{F}_2^{12}$ we define*

$$T_\gamma := \{\alpha_{32} \in T \mid \{n_R\} = \{a_R\} = 0\mathbf{x}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f}\mathbf{f} \Rightarrow \{p_4\} = \gamma_0 \wedge \dots \wedge \{p_{11}\} = \gamma_7 \wedge b_{96} = \gamma_8 \wedge \dots \wedge b_{99} = \gamma_{11}\}.$$

So instead of storing T as one big table, during precomputation the attacker creates the 4096 tables T_γ . Taking $\gamma := \{p_4\} \dots \{p_{11}\}b_{96} \dots b_{99}$ at the end of Protocol 6.5.8, the attacker knows that α_{32} must be an element of T_γ . Now T_γ contains only approximately 2^{24} entries, so this can easily be read from disk to generate 2^{24} candidate keys and check them against a few other authentication sessions. Table 6.2 shows, as an example, the first part of T_γ for $\gamma = 0\mathbf{x}\mathbf{a}04 = 1010\ 0000\ 0100$.

6.5.4 Nested authentication attack

We now assume that the attacker already knows at least one sector key; let us call this sector the exploit sector.

The time between two consecutive authentication attempts might vary from card to card, although it is quite constant for a specific card. Therefore, an attacker can first estimate this time by authenticating two times for the exploit sector. In this way the attacker can estimate the distance δ between the first and the second tag nonce.

As explained in Section 6.4.3, the attacker can now authenticate for the exploit sector and subsequently for another sector. In the authentication for the exploit sector the tag nonce n_T^0 is sent in the clear; during the second authentication the tag nonce n_T is sent encrypted as $\{n_T\}$. By computing $\text{suc}^i(n_T^0)$ for i close to δ , the adversary has a small number of guesses for n_T . The adversary can further narrow the possibilities for n_T using the three bits of information from the parity bits (Theorem 6.4.1). In this way the adversary can accurately guess n_T and hence recover the first 32 bits of keystream, $b_0b_1 \dots b_{31}$.

We shall show how a variant of the attack of Section 6.3 of [GdKGM⁺08] can be used to recover approximately 2^{16} possible candidate keys. By performing this procedure two or three times, the attacker can recover the key for the second sector as well by taking the intersection of the two or three sets of candidate keys.

The crucial ingredient in the attack is the fact that the inputs to the filter function are only on odd-numbered places of the LFSR. This makes it possible to compute separately all possibilities for the odd-numbered bits of the LFSR-stream and the even-numbered bits of the LFSR-stream that are compatible with the keystream.

Definition 6.5.4. *We define the odd tables T_i^O by*

$$T_0^O := \{x_9x_{11} \dots x_{45}x_{47} \in \mathbb{F}_2^{20} \mid f(x_9x_{11} \dots x_{45}x_{47}) = b_0\}$$

and for $i \in \{1, \dots, 15\}$

$$T_i^O := \{x_9x_{11} \dots x_{45+2i}x_{47+2i} \in \mathbb{F}_2^{20+i} \mid x_9 \dots x_{45+2i} \in T_{i-1}^O \wedge f(x_{9+2i}x_{11+2i} \dots x_{45+2i}x_{47+2i}) = b_{2i}\}.$$

Symmetrically, we define the even tables T_i^E by

$$T_0^E := \{x_{10}x_{12} \dots x_{46}x_{48} \in \mathbb{F}_2^{20} \mid f(x_{10}x_{12} \dots x_{46}x_{48}) = b_1\}$$

and for $i \in \{1, \dots, 15\}$

$$T_i^E := \{x_{10}x_{12} \dots x_{46+2i}x_{48+2i} \in \mathbb{F}_2^{20+i} \mid x_{10} \dots x_{46+2i} \in T_{i-1}^E \wedge f(x_{10+2i}x_{12+2i} \dots x_{46+2i}x_{48+2i}) = b_{2i+1}\}.$$

We write $T^O := T_{15}^O$ and $T^E := T_{15}^E$.

Because of the structure of the filter function f , T_0^O and T_0^E are exactly of size 2^{19} (Theorem 6.3.1). The other tables are approximately of this size as well. An entry $x_9x_{11} \dots x_{45+2i}$ of T_{i-1}^O leads to four different possibilities in T_i^O : it can appear in T_i^O extended with 0 and with 1; it can appear extended only with 0; it can appear extended only with 1; or it cannot appear at all. Overall, these possibilities are equally likely, and hence T_i^O has, on average, the same size as T_{i-1}^O (and similarly for T^E).

The feedback function L can also be split into an even and an odd part.

Definition 6.5.5. We define the odd part of the feedback function, $L^O: \mathbb{F}_2^{24} \rightarrow \mathbb{F}_2$, by

$$L^O(x_1x_3 \dots x_{47}) := x_5 \oplus x_9 \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{25} \oplus \\ x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{43}$$

and the even part of the feedback function, $L^E: \mathbb{F}_2^{24} \rightarrow \mathbb{F}_2$, by $L^E(x_0x_2 \dots x_{46}) := x_0 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{24} \oplus x_{42}$.

Note that L^E and L^O combine to give L , in the sense that

$$L(x_0x_1x_2 \dots x_{47}) = L^E(x_0x_2 \dots x_{46}) \oplus L^O(x_1x_2 \dots x_{47}). \quad (6.6)$$

As the $a_9a_{10} \dots a_{77}a_{78}$ are being shifted through the LFSR, the uid u and the tag nonce n_T are shifted in as well. In the following definition we compute the 22 bits of feedback from the LFSR from time 9 to time 31, taking care of the shifting in of $u \oplus n_T$, and also splitting the contribution from the odd- and even-numbered bits of the LFSR. At this point, the situation in [GdKGM⁺08] is slightly simpler. There, the attacker tries to find the state of the LFSR after initialization, so nothing is being shifted in.

Definition 6.5.6.

$$\psi^O(x_9x_{11} \dots x_{77}) := (L^E(x_{9+2i}x_{11+2i} \dots x_{55+2i}) \oplus n_{T,9+2i} \oplus u_{9+2i}, \\ L^O(x_{11+2i}x_{13+2i} \dots x_{57+2i}) \oplus n_{T,10+2i} \oplus u_{10+2i})_{i \in \{0, \dots, 10\}}$$

and

$$\psi^E(x_{10}x_{12} \dots x_{78}) := (L^O(x_{10+2i}x_{12+2i} \dots x_{56+2i}) \oplus x_{57+2i}, \\ L^E(x_{10+2i}x_{12+2i} \dots x_{56+2i}) \oplus x_{58+2i})_{i \in [0, 10]}.$$

Definition 6.5.7. We define the combined table T^C as follows.

$$T^C := \{x_9x_{10}x_{11} \dots x_{78} \mid x_9x_{11} \dots x_{77} \in T^O \wedge x_{10}x_{12} \dots x_{78} \in T^E \wedge \\ \psi^O(x_9x_{11} \dots x_{77}) = \psi^E(x_{10}x_{12} \dots x_{78})\}.$$

Note that T^C can easily be computed by first sorting T^O by ψ^O and T^E by ψ^E .

The crucial point is the following theorem; it shows that the actual LFSR-stream of the tag under attack is in the table T^C .

Theorem 6.5.9. $a_9a_{10}a_{11} \dots a_{78} \in T^C$.

Proof. By definition of T^O and T^E , $a_9a_{11} \dots a_{77} \in T^O$ and $a_{10}a_{12} \dots a_{78} \in T^E$. We only have to check that the sequence $a_9a_{10}a_{11} \dots a_{78}$ satisfies the constraint defining T^C . For this, we have

$$\begin{aligned}
& \psi^O(a_9a_{11} \dots a_{77}) \oplus \psi^E(a_{10}a_{12} \dots a_{78}) \\
&= (L^E(x_{9+2i}x_{11+2i} \dots x_{55+2i}) \oplus n_{T,9+2i} \oplus u_{9+2i} \oplus \\
&\quad L^O(x_{10+2i}x_{12+2i} \dots x_{56+2i}) \oplus x_{57+2i}, \\
&\quad L^O(x_{11+2i}x_{13+2i} \dots x_{57+2i}) \oplus n_{T,10+2i} \oplus u_{10+2i} \oplus \\
&\quad L^E(x_{10+2i}x_{12+2i} \dots x_{56+2i}) \oplus x_{58+2i})_{i \in \{0, \dots, 10\}} \\
&\hspace{15em} \text{(by Dfn. 6.5.6)} \\
&= (L(x_{9+2i}x_{10+2i} \dots x_{56+2i}) \oplus n_{T,9+2i} \oplus u_{9+2i} \oplus x_{57+2i}, \\
&\quad L(x_{10+2i}x_{11+2i} \dots x_{57+2i}) \oplus n_{T,10+2i} \oplus u_{10+2i} \oplus x_{58+2i})_{i \in \{0, \dots, 10\}} \\
&\hspace{15em} \text{(by Eqn. (6.6))} \\
&= (0, 0)_{i \in \{0, \dots, 10\}}, \\
&\hspace{15em} \text{(by Dfn. 6.3.5)}
\end{aligned}$$

as required. □

Taking the first 48 bits of every entry of T^C , the attacker can apply Theorem 6.3.1 nine times for every entry, obtaining one candidate key for every entry of T^C . Because we have used 32 bits of keystream and the key is 48 bits, on average there will be 2^{16} candidate keys. Doing this procedure once more gives another set of approximately 2^{16} candidate keys; the actual key must be in the intersection. In practice, most of the time the intersection only contains a single key; occasionally it contains two keys and then a third run of this whole procedure can be used to determine the key (or both candidate keys can just be tested online, of course).

6.6 Conclusions

We have found serious ‘textbook’ vulnerabilities in the Mifare Classic tag. In particular, the Mifare Classic mixes two layers of the protocol stack and reuses a one-time pad for the encryption of the parity bits. It also sends encrypted error messages before a successful authentication. These weaknesses allow an adversary to recover a secret key within seconds. Moreover, tag nonces are predictable which, besides allowing replays, provides known plaintext for our nested authentication attack. We have executed these attacks in practice and retrieved all secret keys from a number of cards, including cards used in large access control and public transport ticketing systems.

To slightly hamper an adversary, system integrators could consider the following countermeasures:

- diversify all keys in the card;
- cryptographically bind the memory contents of the card to the UID, for instance by including a MAC;
- perform regular integrity checks in the back office.

For the time being, the second countermeasure prevents an attacker from cloning a card onto a blank one. However, this does not stop an attacker from emulating that card with an emulator like the Proxmark.

Early on we have notified the manufacturer NXP of these vulnerabilities. Since the protocol is implemented in hardware, we do not foresee any definitive countermeasure to these attacks that does not require replacing the entire infrastructure. However, NXP has developed and deployed a backwards compatible successor to the Mifare Classic, the Mifare Plus. We are collaborating with NXP, providing feedback to help them improving the security of their new prototypes, given the limitations of the backwards compatibility mode.

6.7 Acknowledgements

We are grateful to our faculty's computer department (C&CZ) for providing us with computing power and to Ben Polman in particular for his assistance.

An electronic vehicle immobilizer is an anti-theft device that prevents the engine of the vehicle from starting unless the corresponding transponder is present. Such a transponder is a passive RFID tag which is embedded in the car key and wirelessly authenticates to the vehicle. It prevents a perpetrator from hot-wiring the vehicle or starting the car by forcing the mechanical lock. Having such an immobilizer is required by law in several countries. *Hitag2*, introduced in 1996, is currently the most widely used transponder in the car immobilizer industry. It is used by at least 34 car makes and fitted in more than 200 different car models. *Hitag2* uses a proprietary stream cipher with 48-bit keys for authentication and confidentiality. This chapter reveals several weaknesses in the design of the cipher and presents three practical attacks that recover the secret key using only wireless communication. The most serious attack recovers the secret key from a car in less than six minutes using ordinary hardware. This attack allows an adversary to bypass the cryptographic authentication, leaving only the mechanical key as safeguard. This is even more sensitive on vehicles where the physical key has been replaced by a keyless entry system based on *Hitag2*. During our experiments we managed to recover the secret key and start the engine of many vehicles from various makes using our transponder emulating device. These experiments also revealed several implementation weaknesses in the immobilizer units.

7.1 Introduction

In the past, most cars relied only on mechanical keys to prevent a hijacker from stealing the vehicle. Since the '90s most car manufacturers incorporated an electronic car immobilizer as an extra security mechanism in their vehicles. From 1995 it is mandatory that all cars sold in the EU are fitted with such an immobilizer device, according to European directive 95/56/EC. Similar regulations apply to other countries like Australia, New Zealand (AS/NZS 4601:1999) and Canada (CAN/ULC S338-98). An electronic car immobilizer consists of two main components: a small transponder chip which is embedded in (the plastic part of) the car key, see Figure 7.1; and a reader which is located somewhere in the dashboard of the vehicle and has an antenna coil around the ignition, see Figure 7.2.

The transponder is a passive RFID tag that operates at a Low Frequency (LF) of 125 kHz. It is powered up when it comes in proximity range of the electronic field



Figure 7.1: Car keys with a *Hitag2* transponder/chip

of the reader. When the transponder is absent, the immobilizer unit prevents the vehicle from starting the engine.

A distinction needs to be made with remotely operated central locking system, which opens the doors, is battery powered, operates at a Ultra-High Frequency (UHF) of 433 MHz, and only activates when the user pushes a button on the remote key. More recent car keys are often deployed with a hybrid chip that supports the battery powered ultra-high frequency as well as the passive low frequency communication interface.

With the *Hitag2* family of transponders, its manufacturer NXP Semiconductors (formerly Philips Semiconductors) leads the immobilizer market [Noh10]. Table 7.1 shows a list containing some of the vehicles that are deployed with a *Hitag2* transponder. Even though NXP boasts “Unbreakable security levels using mutual authentication, challenge-response and encrypted data communication”¹, it uses a shared key of only 48 bits.

Since 1988, the automotive industry has moved towards the so-called keyless ignition or keyless entry in their high-end vehicles [HTTN88]. In such a vehicle the mechanical key is no longer present and it has been replaced by a start button like the one shown in Figure 7.3. The only anti-theft mechanism left in these vehicles is the immobilizer. Startlingly, many keyless ignition or entry vehicles sold nowadays are still based on the *Hitag2* cipher. In some keyless entry cars *Hitag2* is also used as a backup mechanism for opening the doors, e.g., when the battery of the remote is depleted.

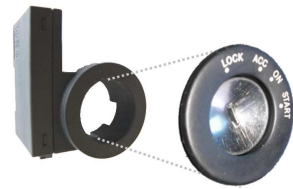


Figure 7.2: Immobilizer unit around the ignition barrel



Figure 7.3: Keyless hybrid transponder and engine start/stop button

¹http://www.nxp.com/products/automotive/car_access_immobilizers/immobilizer/

Table 7.1: Vehicles using *Hitag2* [Key12] – boldface indicates vehicles we tested

Make	Models
Acura	CSX, MDX, RDX, TL, TSX
Alfa Romeo	156, 159, 166, Brera, Giulietta, Mito, Spider
Audi	A8
Bentley	Continental
BMW	Serie 1 , 5, 6, 7, all bikes
Buick	Enclave, Lucerne
Cadillac	BLS, DTS, Escalade, SRX, STS, XLR
Chevrolet	Avanlache, Caprice, Captiva, Cobalt, Equinox, Express, HHR Impala, Malibu, Montecarlo, Silverado, Suburban, Tahoe Trailblazer, Uplander
Chrysler	300C, Aspen, Grand Voyager, Pacifica, Pt Cruiser, Sebring Town Country, Voyager
Citroen	Berlingo , C-Crosser, C2, C3 , C4 , C4 Picasso, C5 , C6, C8 Nemo, Saxo, Xsara, Xsara Picasso
Dacia	Duster, Logan , Sandero
Daewoo	Captiva, Windstorm
Dodge	Avenger, Caliber, Caravan, Charger, Dakota, Durango Grand Caravan, Journey, Magnum, Nitro, Ram
Fiat	500, Bravo, Croma, Daily, Doblo, Fiorino, Grande Punto Panda, Phedra, Ulysse, Scudo
GMC	Acadia, Denali, Envoy, Savana, Siera, Terrain, Volt, Yukon
Honda	Accord, Civic , CR-V, Element, Fit, Insight, Stream, Jazz, Odyssey, Pilot, Ridgeline, most bikes
Hummer	H2, H3
	Grandeur, I30 , Matrix, Santafe, Sonata, Terracan, Tiburon Tucoson, Tuscanti
Isuzu	D-Max
Iveco	35C11, Eurostar, New Daily, S-2000
Jeep	Commander, Compass, Grand Cherokee, Liberty, Patriot Wrangler
Kia	Carens, Carnival, Ceed, Cerato, Magentis, Mentor, Optima Picanto, Rio, Sephia, Sorento, Spectra, Sportage
Lancia	Delta, Musa, Phedra
Mini	Cooper
Mitsubishi	380, Colt, Eclipse, Endeavor, Galant, Grandis, L200 Lancer, Magna, Outlander, Outlander, Pajero, Raider
Nissan	Almera, Juke , Micra , Pathfinder, Primera, Qashqai, Interstar Note, Xterra
Opel	Agila, Antara, Astra, Corsa, Movano, Signum, Vectra Vivaro, Zafira
Peugeot	106 , 206 , 207, 307 , 406, 407, 607, 807, 1007, 3008, 5008 Beeper, Partner, Boxer , RCZ
Pontiac	G5, G6, Pursuit, Solstice, Torrent
Porsche	Cayenne
Renault	Clio , Duster, Kangoo , Laguna II , Logan, Master Megane , Modus, Sandero, Traffic , Twingo
Saturn	Aura, Outlook, Sky, Vue
Suzuki	Alto, Grand Vitara, Splash, Swift, Vitara, XL-7
Volkswagen	Touareg, Phaeton

Related work

A similar immobilizer transponder is produced by Texas Instruments under the name Digital Signature Transponder (DST). It is protected by a different proprietary cryptographic algorithm that uses a secret key of only 40 bits. The workings of these algorithms are reversed engineered by Bono et al. in [BGS⁺05]. Francillon et al. demonstrated in [FDv11] that it is possible to relay in real-time the (encrypted) communication of several keyless entry systems. The chapter shows that in some cases such a communication can be intercepted over a distance of at least 100 meters.

The history of the NXP *Hitag2* family of transponders overlaps with that of other security products designed and deployed in the late nineties, see Chapter 4. Originally, information on *Hitag2* transponders was limited to data sheets with high level descriptions of the chip's functionality [NXP10], while details on the proprietary cryptographic algorithms were kept secret by the manufacturer. This phase, in which security was strongly based on obscurity, lasted until in 2007 when the *Hitag2* inner workings were reverse engineered [Wie07]. Similarly to its predecessor Crypto1 (used in MIFARE Classic), the *Hitag2* cipher consists of a 48-bit Linear Feedback Shift Register (LFSR) and a non-linear filter function used to output keystream.

The publication of the *Hitag2* cipher attracted the interest of the scientific community. Courtois et al. [COQ09] were the first to study the vulnerabilities of the *Hitag2* stream cipher to algebraic attacks by transforming the cipher state into a system of equations and using SAT solvers to perform key recovery attacks. Their most practical attack requires two days computation and a total of four eavesdropped authentication attempts to extract the secret key. A more efficient attack, requiring 16 chosen initialization vectors (IV) and six hours of computations, was also proposed. However, and as noted by the authors themselves, chosen-IV attacks are prevented by the *Hitag2* authentication protocol (see Sect. 7.3.5), thus making this attack infeasible in practice.

In [SNC09], Soos et al. introduced a series of optimizations on SAT solvers that made it possible to reduce the attack time of Courtois et al. to less than 7 hours. More recently, Štembera and Novotný [vN11] implemented a brute-force attack that could be carried out in less than two hours by using the COPACOBANA² high-performance cluster of FPGAs. Note however, that such attack would require about 4 years if carried out on a standard Personal Computer (PC).

Finally, Sun et al. [SHXZ11] tested the security of the *Hitag2* cipher against cube attacks. Although according to their results the key can be recovered in less than a minute, this attack requires chosen initialization vectors and thus should be regarded as strictly theoretical.

²<http://www.copacobana.org>

Our contribution

In this chapter, we show a number of vulnerabilities in the *Hitag2* transponders that enable an adversary to retrieve the secret key. We propose three attacks that extract the secret key under different scenarios. We have implemented and successfully executed these attacks in practice on more than 20 vehicles of various make and model. On all these vehicles we were able to use an emulating device to bypass the immobilizer and start the vehicle.

Concretely, we found the following vulnerabilities in *Hitag2*.

- The transponder lacks a pseudo-random number generator, which makes the authentication procedure vulnerable to replay attacks. Moreover, the transponder provides known data when a read command is issued on the block where the transponder's identity is stored, allowing to recover keystream. Redundancy in the commands allow an adversary to expand this keystream to arbitrary lengths. This means that the transponder provides an arbitrary length keystream oracle.
- With probability $1/4$ the output bit of the cipher is determined by only 34 bits of the internal state. As a consequence, (on average) one out of four authentication attempts leaks one bit of information about the secret key.
- The 48-bit internal state of the cipher is only randomized by a nonce of 32 bits. This means that 16 bits of information on the secret key are persistent throughout different sessions.

We exploit these vulnerabilities in the following three practical attacks.

- The first attack exploits the malleability of the cipher and the fact that the transponder does not have a pseudo-random number generator. It uses a keystream shifting attack following the lines of [dKGGH08]. This allows an adversary to first get an authentication attempt from the reader which can later be replayed to the transponder. Exploiting the malleability of the cipher, this can be used to read known plaintext (the identity of the transponder) and recover keystream. In a new session the adversary can use this keystream to read any other memory block (with exception of the secret key when configured correctly) within milliseconds. When the key is not read protected, this attack can also be used to read the secret key. This was in fact the case for most vehicles we tested from a French car make.
- The second attack is slower but more general in the sense that the same attack strategy can be applied to other LFSR based ciphers. The attack uses a time/memory tradeoff as proposed in [Hel80, Bab95, BPVV98, BS00, Oec03, BMS06]. Exploiting the linear properties of the LFSR, we are able to efficiently generate the lookup table, reducing the complexity from 2^{48} to 2^{37} encryptions.

This attack recovers the secret key regardless of the read protection configuration of the transponder. It requires 30 seconds of communication with the transponder and another 30 seconds to perform 2000 table lookups.

- The third attack is also the most powerful, as it only requires a few authentication attempts from the car immobilizer to recover the secret key (assuming that the adversary knows a valid transponder *id*). This cryptanalytic attack exploits dependencies among different sessions and a low degree determination of the filter function used in the cipher. In order to execute this attack, an adversary first gathers 136 partial authentication attempts from the car. This can be done within one minute. Then, the adversary needs to perform 2^{35} operations to recover the secret key. This takes less than five minutes on a laptop with a dual core running at 2.4 GHz.

Furthermore, besides looking into the security aspects of *Hitag2* we also study how it is deployed and integrated in car immobilizer systems by different manufacturers. Our study reveals that in many vehicles the transponder is misconfigured by having readable or default keys, and predictable passwords, whereas the immobilizer unit employs weak pseudo-random number generators. All cars we tested use identifier white-listing as an additional security mechanism. This means that in order to use our third attack to hijack a car, an adversary first needs to eavesdrop, guess or wirelessly pickpocket a legitimate transponder *id*, see Section 7.7.5.

Following the principle of responsible disclosure, we have contacted the manufacturer NXP and informed them of our findings six months ahead of publication. We have also provided our assistance in compiling a document to inform their customers about these vulnerabilities. The communication with NXP has been friendly and constructive. NXP encourages the automotive industry for years to migrate to more secure products that incorporate strong and community-reviewed ciphers like AES [DR02]. It is surprising that the automotive industry is reluctant to migrate to secure products given the cost difference of a better chip (\leq USD 1) in relation to the prices of high-end car models (\geq USD 50 000).

7.2 Hardware setup

Before diving into details about *Hitag2*, this section introduces the experimental platform we have developed in order to carry out attacks in real-life deployments of car immobilizer systems. In particular, we have built a portable and highly flexible setup allowing us to i) eavesdrop communications between *Hitag2* readers and transponders, ii) emulate a *Hitag2* reader, and iii) emulate a *Hitag2* transponder. Figure 7.4 depicts our setup in the setting of eavesdropping communications between a reader and a transponder.

The central element of our experimental platform is the Proxmark board³, originally developed by Jonathan Westhues⁴, and designed to work with RFID transponders ranging from low frequency (125 kHz) to high frequency (13.56 MHz).

The Proxmark board cost around 200 United States Dollar (USD) and comes equipped with a FPGA and an ARM microcontroller. Low-level RF operations such as modulation/demodulation are carried out by the FPGA, whereas high-level operations such as encoding/decoding of frames are performed in the microcontroller.

Hitag2 tags are low frequency transponders used in proximity area RFID applications [NXP10]. Communication from reader to transponder is encoded using Binary Pulse Length Modulation (BPLM), whereas from transponder to reader it can be encoded using either Manchester or Biphase coding. In order to eavesdrop, generate, and read communications from reader to transponder, we added support for encoding/decoding BPLM signals, see Figure 7.5.



Figure 7.4: Experimental setup for eavesdropping

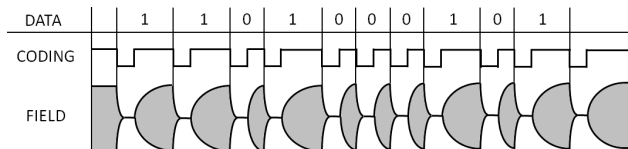


Figure 7.5: Reader modulation of a *read* command

For the transponder side, we have also added the functionalities to support the Manchester coding scheme as shown in Figure 7.6.

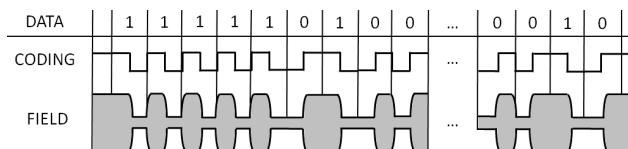


Figure 7.6: Communication from transponder to reader

³<http://www.proxmark.org>

⁴<http://cq.cx/proxmark3.pl>

7.3 Hitag2

This section describes *Hitag2* in detail. Most of this information is in the public domain. We first describe the *Hitag2* functionality, memory structure, and communication protocols, this comes mostly from the product data sheet [NXP10]. Then we describe the cipher and the authentication protocol which was previously reverse engineered in [Wie07]. In Section 7.3.7 we show that it is possible to run the cipher backwards which we use in our attacks.

7.3.1 Functionality

Access to the *Hitag2* memory contents is determined by pre-configured security policies. *Hitag2* transponders offer up to three different modes of operation:

1. In *public mode* the contents of the user data pages are simply broadcast by the transponder once it is powered up.
2. In *password mode* reader and transponder authenticate each other by interchanging their passwords. Communication is carried out in the clear, therefore this authentication procedure is vulnerable to replay attacks.
3. In *crypto mode* the reader and the transponder perform a mutual authentication by means of a 48-bit shared key. Communication between reader and transponder is encrypted using a proprietary stream cipher. This mode is used in car immobilizer systems and will be the focus of this study.

7.3.2 Memory

Hitag2 transponders have a total of 256 bits of non-volatile memory (EEPROM) organized in 8 blocks of 4 bytes each. Table 7.2 illustrates the memory contents of a transponder configured in crypto mode. Block 0 stores the read-only transponder identifier; the secret key is stored in blocks 1 and 2; the password and configuration bits in block 3; blocks 4 till 7 store user defined memory. Access to any of the memory blocks in crypto mode is only granted to a reader after a successful mutual authentication.

7.3.3 Communication

The communication protocol between the reader and transponder is based on the master-slave principle. The reader sends a command to the transponder, which then responds after a predefined period of time. There are five different commands: *authenticate*, *read*, *read*, *write* and *halt*. As shown in Table 7.3, the *authenticate* command has a fixed length of 5 bits, whereas the others have a length of at least 10 bits. Optionally, these 10 bits can be extended with a redundancy message of size

Table 7.2: *Hitag2* memory map in crypto mode [NXP10]

Block	Contents
0	transponder identifier id
1	secret key low $k_0 \dots k_{31}$
2	secret key high $k_{32} \dots k_{47}$ reserved
3	configuration password
4 – 7	user defined memory

multiple of 5 bits. A redundancy message is composed of the bit-complement of the last five bits of the command. According to the datasheet [NXP10] this feature is introduced to “achieve a higher confidence level”.

In crypto mode the transponder starts in a halted state and is activated by the *authenticate* command. After a successful authentication, the transponder enters the active state in which it only accepts active commands which are encrypted. Every encrypted bit that is transferred consists of a plaintext bit XOR’ed with one bit of the keystream. The active commands have a 3-bit argument n which represents the offset (block number) in memory. From this point we address *Hitag2* active commands by referring to *commands* and explicitly mention authentication otherwise.

Table 7.3: *Hitag2* commands using block number n

Command	Bits	State
<i>authenticate</i>	11000	halted
<i>read</i>	$11n_0n_1n_200\overline{n_0n_1n_2} \dots$	active
<i>read</i>	$01n_0n_1n_210\overline{n_0n_1n_2} \dots$	active
<i>write</i>	$10n_0n_1n_201\overline{n_0n_1n_2} \dots$	active
<i>halt</i>	$00n_0n_1n_211\overline{n_0n_1n_2} \dots$	active

Next we define the function *cmd* which constructs a bit string that represents a command c on block n with r redundancy messages.

Definition 7.3.1. *Let c be the first 2-bit command as defined in Table 7.3, n be a 3-bit memory block number and r be the number of redundancy messages. Then, the function $cmd: \mathbb{F}_2^2 \times \mathbb{F}_2^3 \times \mathbb{N} \rightarrow \mathbb{F}_2^{(10+5r)}$ is defined by*

$$cmd(c, n, 0) = cn\overline{cn}$$

$$cmd(c, n, r + 1) = \begin{cases} cmd(c, n, r)cn, & r + 1 \text{ is odd;} \\ cmd(c, n, r)\overline{cn}, & \text{otherwise.} \end{cases}$$

For example, the command to read block 0 with two redundancy messages results in the following bit string.

$$cmd(11, 0, 2) = 11000 \ 00111 \ 11000 \ 00111$$

The encrypted messages between reader and transponder are transmitted without any parity bits. The transponder response always starts with a prefix of five ones, see Figure 7.7. In the remainder of this chapter we will omit this prefix. A typical forward and backwards communication takes about 12 ms.

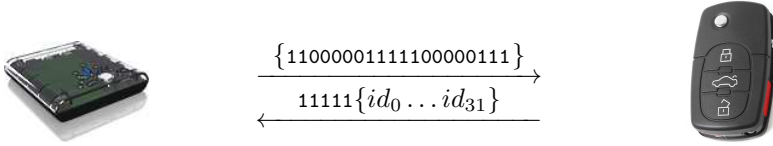


Figure 7.7: Message flow for reading memory block 0

7.3.4 Cipher

In crypto mode, the communication between transponder and reader (after a successful authentication) is encrypted with the *Hitag2* stream cipher. This cipher has been reverse engineered in [Wie07]. The cipher consists of a 48-bit linear feedback shift register (LFSR) and a non-linear filter function f . Each clock tick, twenty bits of the LFSR are put through the filter function, generating one bit of keystream. Then the LFSR shifts one bit to the left, using the generating polynomial to generate a new bit on the right. See Figure 7.8 for a schematic representation.

Definition 7.3.2. *The feedback function $L: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$ is defined by $L(x_0 \dots x_{47}) := x_0 \oplus x_2 \oplus x_3 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_{16} \oplus x_{22} \oplus x_{23} \oplus x_{26} \oplus x_{30} \oplus x_{41} \oplus x_{42} \oplus x_{43} \oplus x_{46} \oplus x_{47}$.*

The filter function f consists of three different circuits f_a, f_b and f_c which output one bit each. The circuits f_a and f_b are employed more than once, using a total of twenty input bits from the LFSR. Their resulting bits are used as input for f_c . The circuits are represented by three boolean tables that contain the resulting bit for each input.

Definition 7.3.3 (Filter function). *The filter function $f: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$ is defined by*

$$f(x_0 \dots x_{47}) = f_c(f_a(x_2x_3x_5x_6), f_b(x_8x_{12}x_{14}x_{15}), \\ f_b(x_{17}x_{21}x_{23}x_{26}), f_b(x_{28}x_{29}x_{31}x_{33}), \\ f_a(x_{34}x_{43}x_{44}x_{46})),$$

where $f_a, f_b: \mathbb{F}_2^4 \rightarrow \mathbb{F}_2$ and $f_c: \mathbb{F}_2^5 \rightarrow \mathbb{F}_2$ are

$$f_a(i) = (0xA63C)_i \\ f_b(i) = (0xA770)_i \\ f_c(i) = (0xD949CBB0)_i.$$

For future reference, note that each of the building blocks of f (and hence f itself) is balanced.

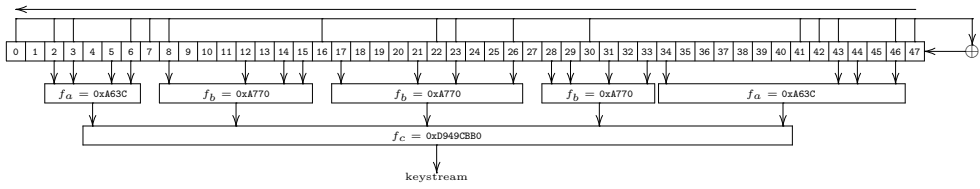


Figure 7.8: Structure of the *Hitag2* stream cipher, based on [Wie07]

Remark 7.1 (Cipher schematic). *Figure 7.8 is different from the schematic that was introduced by [Wie07] and later used by [GdKGM⁺08, COQ09, vN11, SHXZ11]. The input bits of the filter function in Figure 7.8 are shifted by one with respect to those of [Wie07]. The filter function in the old schematic represents a keystream bit at the previous state $f(x_{i-1} \dots x_{i+46})$, while the one in Figure 7.8 represents a keystream bit of the current state $f(x_i \dots x_{i+47})$. Furthermore, we have adapted the boolean tables to be consistent with our notation.*

7.3.5 Authentication protocol

The authentication protocol used in *Hitag2* in crypto mode, reversed engineered and published online in 2007 [Wie07], is depicted in Figure 7.9. The reader starts the communication by sending an authenticate command, to which the transponder answers by sending its identifier id . From this point on, communication is encrypted, i.e., XOR'ed with the keystream. The reader responds with its encrypted challenge n_R and the answer $a_R = 0xFFFFFFFF$ also encrypted to prove knowledge of the key; the transponder finishes with its encrypted answer a_T (corresponding to block 3 in Table 7.2) to the challenge of the reader.

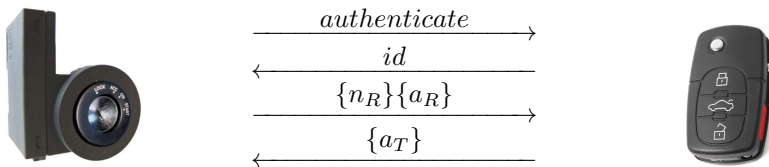


Figure 7.9: *Hitag2* authentication protocol

During the authentication protocol, the internal state of the stream cipher is initialized. The initial state consists of the 32-bits identifier concatenated with the first 16 bits of the key. Then reader nonce n_R XOR'ed with the last 32 bits of the key is shifted in. During initialization, the LFSR feedback is disabled. Since communication is encrypted from n_R onwards, the encryption of the later bits of n_R are influenced by its earlier bits. Authentication is achieved by reaching the same internal state of the cipher after shifting in n_R .

7.3.6 Cipher Initialization

The following precisely defines the initialization of the cipher and the generation of the LFSR-stream $a_0a_1\dots$ and the keystream $b_0b_1\dots$.

Definition 7.3.4. *Given a key $k = k_0\dots k_{47} \in \mathbb{F}_2^{48}$, an identifier $id = id_0\dots id_{31} \in \mathbb{F}_2^{32}$, a reader nonce $n_R = n_{R_0}\dots n_{R_{31}} \in \mathbb{F}_2^{32}$, a reader answer $a_R = a_{R_0}\dots a_{R_{31}} \in \mathbb{F}_2^{32}$, and a transponder answer $a_T = a_{T_0}\dots a_{T_{31}} \in \mathbb{F}_2^{32}$, the internal state of the cipher at time i is $\alpha_i := a_i\dots a_{47+i} \in \mathbb{F}_2^{48}$. Here the $a_i \in \mathbb{F}_2$ are given by*

$$\begin{aligned} a_i &:= id_i & \forall i \in [0, 31] \\ a_{32+i} &:= k_i & \forall i \in [0, 15] \\ a_{48+i} &:= k_{16+i} \oplus n_{R_i} & \forall i \in [0, 31] \\ a_{80+i} &:= L(a_{32+i}\dots a_{79+i}) & \forall i \in \mathbb{N} . \end{aligned}$$

Furthermore, we define the keystream bit $b_i \in \mathbb{F}_2$ at time i by

$$b_i := f(a_i\dots a_{47+i}) \quad \forall i \in \mathbb{N} .$$

Define $\{n_R\}, \{a_R\}_i, \{a_T\}_i \in \mathbb{F}_2$ by

$$\begin{aligned} \{n_R\}_i &:= n_{R_i} \oplus b_i & \forall i \in [0, 31] \\ \{a_R\}_i &:= a_{R_i} \oplus b_{32+i} & \forall i \in [0, 31] \\ \{a_T\}_i &:= a_{T_i} \oplus b_{64+i} & \forall i \in [0, 31] . \end{aligned}$$

Note that the $a_i, \alpha_i, b_i, \{n_R\}_i, \{a_R\}_i,$ and $\{a_T\}_i$ are formally functions of $k, id,$ and n_R . Instead of making this explicit by writing, e.g., $a_i(k, id, n_R)$, we just write a_i where $k, id,$ and n_R are clear from the context.

7.3.7 Rollback

To recover the key it is sufficient to learn the internal state of the cipher α_i at any point i in time. Since an attacker knows id and $\{n_R\}$, the LFSR can then be rolled back to time zero [Rue86].

Definition 7.3.5. *The rollback function $R: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$ is defined by $R(x_1\dots x_{48}) := x_2 \oplus x_3 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_{16} \oplus x_{22} \oplus x_{23} \oplus x_{26} \oplus x_{30} \oplus x_{41} \oplus x_{42} \oplus x_{43} \oplus x_{46} \oplus x_{47} \oplus x_{48}$.*

If one first shifts the LFSR left using L to generate a new bit on the right, then R recovers the bit that dropped out on the left, i.e.,

$$R(x_1\dots x_{47} L(x_0\dots x_{47})) = x_0 . \quad (7.1)$$

Theorem 7.3.1. *In the situation from Definition 7.3.4, we have*

$$\begin{aligned} a_{32+i} &= R(a_{33+i}\dots a_{80+i}) & \forall i \in \mathbb{N} \\ a_i &= id_i & \forall i \in [0, 31] . \end{aligned}$$

Proof. Straightforward, using Definition 7.3.4 and Equation (7.1). \square

If an attacker manages to recover the internal state of the LFSR $\alpha_i = a_i a_{i+1} \dots a_{i+47}$ at some time i , then she can repeatedly apply Theorem 7.3.1 to recover $a_0 a_1 \dots a_{79}$ and, consequently, the keystream $b_0 b_1 b_2 \dots$. By having eavesdropped $\{n_R\}$ from the authentication protocol, the adversary can further calculate

$$n_{R_i} = \{n_R\}_i \oplus b_i \quad \forall i \in [0, 31] .$$

Finally, the adversary can compute the secret key as follows

$$\begin{aligned} k_i &= a_{32+i} & \forall i \in [0, 15] \\ k_{16+i} &= a_{48+i} \oplus n_{R_i} & \forall i \in [0, 31] . \end{aligned}$$

7.4 Hitag2 weaknesses

This section describes three weaknesses in the design of *Hitag2*. The first one is a protocol flaw while the last two concern the cipher's design. These weaknesses will later be exploited in Section 7.5.

7.4.1 Arbitrary length keystream oracle

This weakness describes that without knowledge of the secret key, but by having only one authentication attempt, it is possible to gather an arbitrary length of keystream bits from the transponder. Section 7.3.3 describes the reader commands that can modify or halt a *Hitag2* transponder. As mentioned in Definition 7.3.1 it is possible to extend the length of such a command with a multiple of five bits. A 10-bit command can have an optional number of redundancy messages r so that the total bit count of the message is $10 + 5r$ bits. Due to power and memory constraints, *Hitag2* seems to be designed to communicate without a send/receive buffer. Therefore, all cipher operations are performed directly at arrival or transmission of bits. Experiments show that a *Hitag2* transponder successfully accepts encrypted commands from the reader which are sent with 1000 redundancy messages. The size of such a command consists of $10 + 5 \times 1000 = 5010$ bits.

Since there is no challenge from the transponder it is possible to replay any valid $\{n_R\}\{a_R\}$ pair to the transponder to achieve a successful authentication. After sending a_T , the internal state of the transponder is initialized and it waits for an encrypted command from the reader as defined in Table 7.3. Without knowledge of the keystream bits $b_{96} b_{97} \dots$ and onwards, all possible combinations need to be evaluated. A command consist of at least 10 bits, therefore there are 2^{10} possibilities. Each command requires a 3-bit parameter containing the block number. Both *read* and *read* receive a 32-bit response, while the write and halt have a different response length. Hence, when searching for 10-bit encrypted commands that get a 32-bit response there

are exactly 16 out of the 2^{10} values that match. On average the first *read* command is found after 32 attempts, the complement of this *read* and its parameters are a linear difference and therefore take only 15 attempts more.

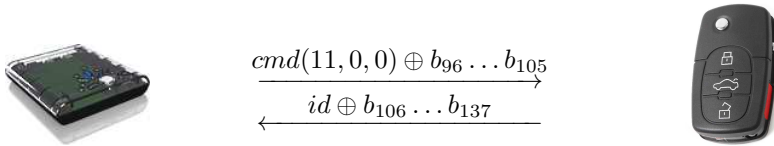


Figure 7.10: Read *id* without redundancy messages

One of the 16 guesses represents the encrypted bits of the read command on the first memory block. This block contains the *id* which is known plaintext since it is transmitted in the clear during the authentication. Therefore, there is a guess such that the communicated bits are equal to the messages in Figure 7.10.

With the correct guess, 40 keystream bits can be recovered. This keystream is then used to encrypt a slightly modified read command on block 0 with six redundancy messages, as explained in Section 7.3.3. The transponder responds with the next 32-bit of keystream which are used to encrypt the identifier as shown in Figure 7.11. Hence the next 30 keystream bits were retrieved using previously recovered keystream and by extending the *read* command.

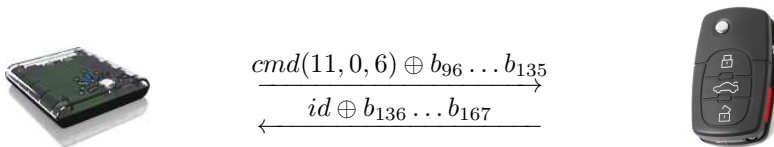


Figure 7.11: Read *id* using 6 redundancy messages

This operation can be repeated many times. For example, using the recovered keystream bits $b_{96} \dots b_{167}$ it is possible to construct a 70-bit *read* command with 12 redundancy messages etc. In practice it takes less than 30 seconds to recover 2048 bits of contiguous keystream.

7.4.2 Dependencies between sessions

Section 7.3.6 shows that at cipher state α_{79} the cipher is fully initialized and from there on the cipher only produces keystream. This shows that the 48-bit internal state of the cipher is randomized by a reader nonce n_R of only 32 bits. Consequently, at state α_{79} , only LFSR bits 16 to 47 are affected by the reader nonce. Therefore LFSR bits 0 to 15 remain constant throughout different session which gives a strong dependency between them. These 16 session persistent bits correspond to bits $k_0 \dots k_{15}$ of the secret key.

7.4.3 Approximation of the filter function

The filter function $f: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$ consists of three building blocks f_a, f_b and f_c arranged in a two layer structure, see Figure 7.8. Due to this particular structure, input bits $a_{34} \dots a_{47}$ only affect the rightmost input bit of f_c . Furthermore, simple inspection of f_c shows that in 8 out of 32 configurations of the input bits, the rightmost input bit has no influence on the output of f_c . In those cases the output of f_c is determined by its 4-leftmost input bits. Furthermore, this means that with probability 1/4 the filter function f is determined by the 34-leftmost bits of the internal state. The following theorem states this precisely.

Theorem 7.4.1. *Let X be a uniformly distributed variable over \mathbb{F}_2^{34} . Then*

$$\mathbb{P}[\forall Y, Y' \in \mathbb{F}_2^{14} : f(XY) = f(XY')] = 1/4.$$

Proof. By inspection. □

Definition 7.4.1. *The function that checks if the rightmost input bit to f_c has no influence $P: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$ is defined by*

$$P(x_0 \dots x_{47}) = (0x84D7)_i$$

where

$$i = f_a(x_2x_3x_5x_6)f_b(x_8x_{12}x_{14}x_{15}) \\ f_b(x_{17}x_{21}x_{23}x_{26})f_b(x_{28}x_{29}x_{31}x_{33}).$$

Because $P(x_0 \dots x_{47})$ only depends on $x_0 \dots x_{33}$ we shall overload notation and see $P(\cdot)$ as a function $\mathbb{F}_2^{34} \rightarrow \mathbb{F}_2$, writing $P(x_0 \dots x_{47})$ as $P(x_0 \dots x_{33}0^{14})$.

7.5 Attacks

This section describes three attacks against *Hitag2*. The first attack is straightforward and grants an adversary read and write access to the memory of the transponder. The cryptanalysis described in the second attack recovers the secret key after briefly communicating with the car and the transponder. This attack uses a general technique that can be applied to other LFSR-like stream ciphers. The third attack describes a custom cryptanalysis of the *Hitag2* cipher. It only requires a few authentication attempts from the car and allows an adversary to recover the secret key with a computational complexity of 2^{35} operations. The last two attacks allow a trade-off between time/memory/data and time/traces respectively. For the sake of simplicity we describe these attacks with concrete values that are either optimal or what we consider ‘sensible’ in view of currently available hardware.

7.5.1 Malleability attack

This attack exploits the arbitrary length keystream oracle weakness described in Section 7.4.1, and the fact that during the authentication algorithm the transponder does not provide any challenge to the reader. These well-known weaknesses allow an adversary to first acquire keystream and then use it to read or write any block on the card with constant communication and computational complexity. After the recovery of the keystream bits $b_{96} \dots b_{137}$ as shown in Figure 7.10 an adversary can dump the complete memory of the transponder which includes its password. Recovery of the keystream and creating a memory dump from the transponder takes in total less than one second and requires only to be in proximity distance of the victim. This shows a similar scenario to the attack described in Chapter 6 which shows how to wirelessly pickpocket a MIFARE Classic card from the victim.

The memory blocks where the cryptographic key is stored have an extra optional protection mechanism. There is a one time programmable configuration bit which determines whether these blocks are readable or not. If the reader tries to read a protected block, then the transponder does not respond. In that case the adversary can still use the attacks presented in Section 7.5.2 and Section 7.5.3. If the transponder is not correctly configured, it enables an adversary to read all necessary data to start the car.

7.5.2 Time/memory tradeoff attack

This attack is very general and it can be applied to any LFSR-based stream cipher as long as enough contiguous keystream is available. This is in fact the case with *Hitag2* due to the weakness described in Section 7.4.1. It extends the methods of similar time/memory tradeoffs articles published over the last decades [Hel80, Bab95, BPVV98, BS00, Oec03, BMS06]. This attack requires communication with the reader and the transponder. The next proposition introduces a small trick that makes it possible to quickly perform n cipher steps at once. Intuitively, this proposition states that the linear difference between a state s and its n -th successor is a combination of the linear differences generated by each bit. This will be later used in the attack.

Proposition 7.5.1. *Let s be an LFSR state and $n \in \mathbb{N}$. Furthermore, let $d_i = \text{suc}^n(2^i)$ i.e., the LFSR state that results from running the cipher n steps from the state 2^i . Then*

$$\text{suc}^n(s) = \bigoplus_{i=0}^{47} (d_i \cdot s_i).$$

To perform the attack the adversary A proceeds as follows:

1. Only once, A builds a table containing 2^{37} entries. Each entry in the table is of the form $\langle ks, s \rangle$ where $s \in \mathbb{F}_2^{48}$ is an LFSR state and $ks \in \mathbb{F}_2^{48}$ are 48 bits of keystream produced by the cipher when running from s . Starting from some

state where $s \neq 0$, the adversary generates 48 bits of keystream and stores it. Then it uses Proposition 7.5.1 to quickly jump $n = 2^{11}$ cipher states to the next entry in the table. This reduces the computational complexity of building the table from 2^{48} to $48 \times 2^{37} = 2^{42.5}$ cipher ticks. Moreover, in order to improve lookup time the table is sorted on ks and divided into 2^{24} sub-tables encoded in the directory structure like `/ks_byte1/ks_byte2/ks_byte3.bin` where each `ks_byte3.bin` file has only 8 KB. The total size of this table amounts 1.2 TB.

2. A emulates a transponder and runs an authentication attempt with the target car. Following the authentication protocol, the car answers with a message $\{n_R\}\{a_R\}$.
3. Next, the attacker wirelessly replays this message to the legitimate transponder and uses the weakness described in Section 7.4.1 to obtain 256 bytes of keystream $ks_0 \dots ks_{2048}$. Note that this might be done while the key is in the victim's bag or pocket.
4. The adversary sets $i = 0$.
5. Then it looks up (in logarithmic time) the keystream $ks_i \dots ks_{i+47}$ in the table from step 1.
6. If the keystream is not in the table then it increments i and goes back to step 5. If there is a match, then the corresponding state is a candidate internal state. A uses the rest of the keystream to confirm whether this is the internal state of the cipher.
7. Finally, the adversary uses Theorem 7.3.1 to rollback the cipher state and recover the secret key.

Complexity and time. In step 1 the adversary needs to pre-compute a 1.2 TB table which requires $2^{42.5}$ cipher ticks, which is equal to 2^{37} encryptions. During generation, each entry is stored directly in the corresponding `.bin` file as mentioned before. Each of these 8 KB files also needs to be sorted but it only takes a few minutes to sort them all. Computing and sorting the whole table takes less than one day on a dual core laptop running at 2.4 GHz. Steps 2-3 take about 30 seconds to gather the 256 bytes of keystream from the transponder. Steps 4-6 require (in worst case) 2000 table lookups which take less than 30 seconds on a standard laptop. This adds to a total of one minute to execute the attack from begin to end.

7.5.3 Cryptanalytic attack

A combination of the weaknesses described in Section 7.4.2 and 7.4.3 enable an attacker to recover the secret key after gathering a few authentication attempts from a car. In case that identifier white-listing is used as a secondary security measure,

which is in fact the case for all the cars we tested, the adversary first needs to obtain a valid transponder *id*, see Section 7.7.5.

The intuition behind the attack is simple. Suppose that an adversary has a guess for the first 34 bits of the key. One out of four traces is expected to have the property from Theorem 7.4.1 which enables the adversary to perform a test on the first bit of $\{a_R\}$. The dependencies between sessions described in Section 7.4.2 allow the attacker to perform this test many times decreasing drastically the amount of candidate (partial) keys. If an attacker gathers 136 traces this allows her (on average) to perform $136/4 = 34$ bit tests, i.e. just as much as key bits were guessed. For the small amount of candidate keys that pass these tests (typically 2 or 3), the adversary performs an exhaustive search for the remaining 14 bits of the key. A precise description of this attack follows.

1. The attacker uses a transponder emulator (like the Proxmark) to initiate 136 authentication attempts with the car using a fixed transponder *id*. In this way the attacker gathers 136 traces of the form $\{n_R\}\{a_R\}$. Next the attacker starts searching for the secret key. For this we split the key k into three parts $k = \bar{k}\hat{k}\bar{k}$ where $\bar{k} = k_0 \dots k_{15}$, $\hat{k} = k_{16} \dots k_{33}$, and $\bar{k} = k_{34} \dots k_{47}$.
2. for each $\bar{k} = k_0 \dots k_{15} \in \mathbb{F}_2^{16}$ the attacker builds a table $T_{\bar{k}}$ containing entries

$$\langle y \oplus b_0 \dots b_{17}, \overline{b_{32}}, \bar{k}y \rangle$$

for all $y \in \mathbb{F}_2^{18}$ such that $P(\bar{k}y0^{14}) = 1$. Note that the expected size of this table is $2^{18} \times 1/4 = 2^{16}$ which easily fits in memory.

3. For each $\hat{k} = k_{16} \dots k_{33} \in \mathbb{F}_2^{18}$ and for each trace $\{n_R\}\{a_R\}$, the attacker sets $z := \hat{k} \oplus \{n_R\}_0 \dots \{n_R\}_{17}$. If there is an entry in $T_{\bar{k}}$ for which $y \oplus b_0 \dots b_{17}$ equals z but $\overline{b_{32}} \neq \{a_R\}_0$ then the attacker learns that \hat{k} is a bad guess, so he tries the next one. Otherwise, if $\overline{b_{32}} = \{a_R\}_0$ then \hat{k} is still a viable guess and therefore the adversary tries the next trace.
4. Each $\bar{k}\hat{k}$ that passed the test for all traces is a partial candidate key. For each such candidate (typically 2 or 3), the adversary performs an exhaustive search for the remaining key bits $\bar{k} = k_{34} \dots k_{47}$. For each full candidate key, the adversary decrypts two traces and checks whether both $\{a_R\}$ decrypt to all ones as specified in the authentication protocol. If a candidate passes this test then it is the secret key. If none of them passes then the adversary goes back to Step 2 and tries the next \bar{k} .

Complexity and time. In step 1, the adversary needs to gather 136 partial authentication traces. This can be done within 1 minute using the Proxmark. In steps 2 and 3, the adversary needs to build 2^{16} tables. For each of these tables the adversary needs to compute 2^{18} encryptions plus 2^{18} table lookups. Step 4 has negligible complexity thus we ignore it. This adds to a total complexity of $2^{16} \times (2^{18} + 2^{18}) = 2^{35}$

encryptions/lookups. Note that it is straightforward to split up the search space of \bar{k} in as many processes as you wish. On an dual core laptop, running at 2.4 GHz, this computation takes less than five minutes. Therefore, the whole attack can be performed in less than 360 seconds which explains the title of the chapter.

This attack is faster than other practical attacks proposed in [COQ09, vN11]. The following table shows a comparison between this attack and other attacks from the literature.

Table 7.4: Comparison of attack times and requirements

Attack	Description	Practical	Computation	Traces	Time
[vN11]	brute-force	yes	2 102 400 min	2	4 years
[COQ09]	sat-solver	yes	2 880 min	4	2 days
[SNC09]	sat-solver	no ¹	386 min	N/A	N/A
[SHXZ11]	cube	no ²	1 min	500	N/A
Our	cryptanalytic	yes	5 min	136	6 min

¹Soos et al. require 50 bits of contiguous keystream.

²Sun et al. require control over the encrypted reader nonce $\{n_R\}$

7.6 Starting a car

In order to elaborate on the practicality of our attacks, this section describes our experience with one concrete vehicle. For this we have chosen a German car, mainly because it has keyless ignition. Instead of the typical mechanical key, this car has a hybrid remote control which contains a *Hitag2* transponder. In the dashboard of the car there is a slot to insert the remote and a button to start the engine. When a piece of plastic of suitable size is inserted in this slot the car repeatedly attempts to authenticate the transponder (and fails). This car uses an identifier white-list as described in Section 7.7.5. The same section explains how to wirelessly pickpocket a valid identifier from the victim's remote. As soon as the car receives a valid identifier, the dashboard lights up and the LCD screen pops-up displaying the message shown in Figure 7.12-Left. Note also the sign on the dashboard.

At this point we used the Proxmark to quickly gather enough traces and execute the attack from Section 7.5.3 to recover the secret key. This car is one of the few that we tested that does not have a predictable password so we wirelessly read it from the victim's remote. Then we use the Proxmark to emulate the transponder. Figure 7.12-Right shows that the car accepts the Proxmark as if it was the legitimate transponder. The same picture shows (by looking at the tachometer) that at this stage it is possible to start the engine.

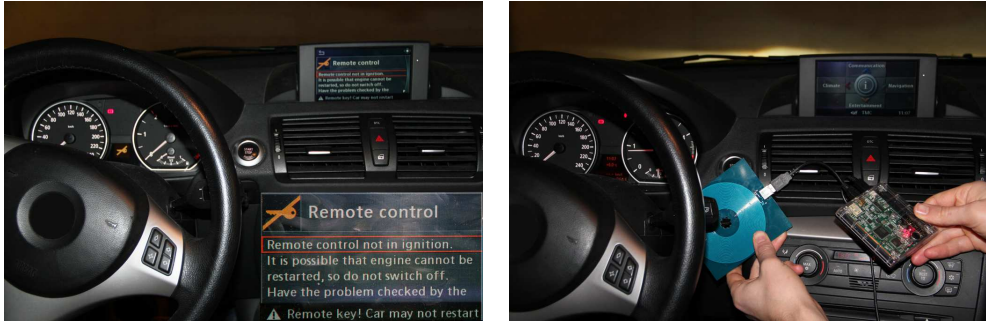


Figure 7.12: Left: Authentication failure message
Right: Successful authentication using a Proxmark

7.7 Implementation weaknesses

To verify the practicality of our attacks, we have tested all three of them on at least 20 different car models from various makes. During our experiments we found that, besides the weaknesses in cipher and protocol, the transponder is often misconfigured and poorly integrated in the cars. Most of the cars we tested use a default or predictable transponder password. Some generate nonces with a very low entropy. Most car keys have vehicle-dependent information stored in the user defined memory of the transponder, but none of the tested cars actually check this data. Some cars use *Hitag2* for key-less ignition systems, which are more vulnerable because they lack a physical key. This section summarizes some of the weaknesses we found during our practical experiments. Especially, Section 7.7.4 shows the implications of the attack described in Section 7.5.3 when the transponder uses a predictable password. Section 7.7.5 describes how to circumvent identifier white-listing. This is an additional security mechanism which is often used in vehicle immobilizers.

7.7.1 Weak random number generators

From the cars we tested, most pseudo-random number generators (PRNG) use the time as a seed. The time intervals do not have enough precision. Multiple authentication attempts within a time frame of one second get the same random number. Even worse, we came across two cars which have a PRNG with dangerously low entropy. The first one, a French car (A), produces nonces with only 8 bits of entropy, by setting 24 of the 32 bits always to zero as shown in Table 7.5.

Another French car (B), produced random looking nonces, but in fact, the last nibble of each byte was determined by the last nibble of the first byte. A subset of these nonces are shown shown in Table 7.6.

Table 7.5: Random numbers generated by car A

Origin	Message	Description
CAR	18	authenticate
TAG	39 0F 20 10	<i>id</i>
CAR	0A 00 00 00 23 71 90 14	$\{\mathbf{n}_R\}\{a_R\}$
TAG	27 23 F8 AF	$\{a_T\}$
CAR	18	authenticate
TAG	39 0F 20 10	<i>id</i>
CAR	56 00 00 00 85 CA 95 BA	$\{\mathbf{n}_R\}\{a_R\}$
TAG	38 07 50 C5	$\{a_T\}$

Table 7.6: Random numbers generated by car B

$\{\mathbf{n}_R\}$	$\{a_R\}$
20 D1 0B 08	56 36 F3 66
70 61 1B 58	1B 18 F3 38
B0 A1 5B 98	1E 94 62 3A
D0 41 FB B8	01 3B 54 10
25 1A 3C AD	15 88 5E 19
05 7A 9C 8D	F7 4D F7 70
C5 3A 5C 4D	30 B1 4A D4
E5 DA FC 6D	D8 BD 79 C3

7.7.2 Low entropy keys

Some cars have repetitive patterns in their keys which makes them vulnerable to dictionary attacks. Recent models of a Korean car (C) use the key with the lowest entropy we came across. It tries to access the transponder in password mode as well as in crypto mode. For this it uses the default password MIKR and a key of the form 0xFFFF*****FF as shown in Table 7.7.

Table 7.7: Car C authenticates using the default password and secret key
0xFFFF814632FF

Origin	Message	Description
CAR	18	authenticate
TAG	E4 13 05 1A	<i>id</i>
CAR	4D 49 4B 52	password = MIKR
CAR	18	authenticate
TAG	E4 13 05 1A	<i>id</i>
CAR	DA 63 3D 24 A7 19 07 12	$\{n_R\}\{a_R\}$
TAG	EC 2A 4B 58	$\{a_T\}$

7.7.3 Readable keys

Section 7.5.1 shows how to recover the memory dump of a *Hitag2* transponder. Almost all makes protect the secret key against read operations by setting the bits of the configuration in such a way that block one and two are not readable, but there are some exceptions. For example, experiments show that most cars from a French manufacturer have *not* set this protection bit. This enables an attacker to recover the secret key in an instant. Even more worrying, many of these cars have the optional feature to use a remote key-less entry system which have a much wider range and are therefore more vulnerable to wireless attacks. The combination of a transponder that is wirelessly accessible over a distance of several meters and a unprotected readable key is most worrying.

7.7.4 Predictable transponder passwords

The transponder password is encrypted and sent in the transponder answer a_T of the authentication protocol. This is an additional security mechanism of the *Hitag2* protocol apart from the cryptographic algorithm. Besides the fact that the transponder proves knowledge of the secret key, it sends its password encrypted. In general it is good to have some fallback scenario and countermeasure if the used cryptosystem gets broken. Section 7.5.3 demonstrates how to recover the secret key from a vehicle. But to start the engine, it is necessary to know the transponder password as well. Experiments show that at least half of the cars we tested on use default or predictable passwords.

7.7.5 Identifier pickpocketing

The first generation of vehicle immobilizers were not able to compute any cryptographic operations. These transponders were simply transmitting a constant (unique) identifier over the RF channel. Legitimate transponder identifiers were white-listed by the vehicle and only those transponders in the white-list would enable the engine to start. Most immobilizer units in cars still use such white-listing mechanism, which is actually encouraged by NXP. These cars would only attempt to authenticate transponders in their white-list. This is an extra obstacle for an attacker, namely recovering a genuine identifier from the victim before being able to execute any attack. There are (at least) two ways for an adversary to wirelessly pickpocket a *Hitag2* identifier:

- One option is to use the low-frequency (LF) interface to wirelessly pickpocket the identifier from the victim's key. This can be done within proximity distance and takes only a few milliseconds. According to the *Hitag2* datasheet [NXP10], the communication range of a transponder is up to one meter. However, *Hitag2* transponders embedded into car keys are optimized for size and do not achieve

such a communication distance. Nevertheless, an adversary can use tuned equipment with big antennas that ignore radiation regulations (e.g., [FCC09]) in order to reach a larger reading distance. Many examples in the literature show the simplicity and low-cost of such a setup [SA00, KW05, KW06, Han06].

- Another option is to use the wide range Ultra-High Frequency (UHF) interface. For this an adversary needs to eavesdrop the transmission of a hybrid *Hitag2* transponder [PHI99] when the victim presses a button on the remote (e.g. to close the doors). Most keyless entry transponders broadcast their identifier in the clear on request (see for example [PHI99]).

With respect to the LF interface, the UHF interface has a much wider transmission range. As shown in [FDv11] it is not hard to eavesdrop such a transmission from a distance of 100 meters. From a security perspective, the first generation *Hitag2* transponders have a physical advantage over the hybrid transponders since they only support the LF interface.

7.8 Mitigation

This section briefly discusses a simple but effective authentication protocol for car immobilizers and it also describes a number of mitigating measures for the attacks proposed in Section 7.5. For more details we refer the reader to [BP08, And10].

First of all we emphasize that it is important for the automotive industry to migrate from weak proprietary ciphers to a peer-reviewed one such as AES [DR02], used in Counter with CBC-MAC (CCM) mode. A straightforward mutual authentication protocol is sketched in Figure 7.13. The random nonces n_R , n_T , secret key k and transponder password PWD_T should be at least 128 bits long. Comparable schemes are proposed in the literature [LSS05, LSS06, WHWC07, WWW07, WKR+08].

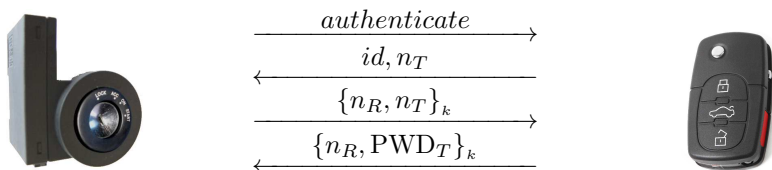


Figure 7.13: Immobilizer authentication protocol using AES

There are already immobilizer transponders on the market that implement AES like the ATA5795[AT11] from Atmel and the Hitag AES / Pro [NXP11] from NXP. It should be noted that, although they use a peer-reviewed encryption algorithm, their authentication protocol is still proprietary and therefore lacks public and academic scrutiny.

In order to reduce the applicability of our cryptographic attack, the automotive industry could consider the following measures. This attack is the most sensitive as it

does not require access to the car key. These countermeasures should be interpreted as palliating (but not a solution) before migrating to a more secure and openly designed product.

- **Extend the transponder password**

The transponder password is an important part of the authentication protocol but grievously it has only an entropy of 24 bits. Such a password is easy to find via exhaustive search. Furthermore, as we mentioned in Section 7.7.4, manufacturers often deployed their cars with predictable transponder passwords. As shown in Table 7.2, there are four pages of user defined memory available in a *Hitag2* transponder. These could be used to extend the transponder password with 128 bits of random data to increase its entropy. This implies that an adversary needs to get access to the transponder's memory before being able to steal a car.

- **Delay authentication after failure**

The cryptographic car-only attack explained in Section 7.5.3 requires several authentication attempts to reduce the computational complexity. Extending the time an adversary needs to gather these traces increases the risk of being caught. To achieve this, the immobilizer introduces a pause before re-authenticating that grows incrementally or exponentially with the number of sequential incorrect authentications. An interesting technique to implement such a countermeasure is proposed in [RSH+12]. The robustness, availability and usability of the product is affected by this delay, but it increases the attack time considerably and therefore reduces the risk of car theft.

Besides these measures, it is important to improve the pseudo-random number generator in the vehicles which is used to generate reader nonces. Needless to say, the same applies to cryptographic keys and transponder passwords. NIST has proposed a statistical test suite which can be used to give an approximation about the quality of a pseudo-random number generator [RSN+01].

7.9 Conclusions

We have found many serious vulnerabilities in the *Hitag2* and its usage in the automotive industry. In particular, *Hitag2* allows replaying reader data to the transponder; provides an unlimited keystream oracle and uses only one low-entropy nonce to randomize a session. These weaknesses allow an adversary to recover the secret key within seconds when wireless access to the car and key is available. When only communication with the car is possible, the adversary needs less than six minutes to recover the secret key. The cars we tested use identifier white-listing. To circumvent this, the adversary first needs to obtain a valid transponder *id* by other means e.g., eavesdrop it when the victim locks the doors. This UHF transmission can be intercepted from a

distance of 100 meters [FDv11]. We have executed all our attacks (from Section 7.5) in practice within the claimed attack times. We have experimented with more than 20 vehicles of various makes and models and found also several implementation weaknesses.

In line with the principle of responsible disclosure, we have notified the manufacturer NXP six months before disclosure. We have constructively collaborated with NXP, discussing mitigating measures and giving them feedback to help improve the security of their products.

7.10 Acknowledgments

The authors would like to thank Bart Jacobs for his firm support in the background. We are also thankful to E. Barendsen, L. van den Broek, J. de Bue, Y. van Dalen, E. Gouwens, R. Habraken, I. Haerkens, S. Hoppenbrouwers, K. Koster, S. Meeuwsen, J. Reule, J. Reule, I. Roggema, L. Spix, C. Terheggen, M. Vaal, S. Vernooij, U. Zeitler, B. Zwanenburg, and those who prefer to remain anonymous for (bravely) volunteering their cars for our experiments.

Chapter 8

SecureMemory, CryptoMemory and CryptoRF

The Atmel chip families SecureMemory, CryptoMemory, and CryptoRF use a proprietary stream cipher to guarantee authenticity, confidentiality, and integrity. This chapter describes the cipher in detail and points out several weaknesses. One is the fact that the three components of the cipher operate largely independently; another is that the intermediate output generated by two of those components is strongly correlated with the generated keystream. For SecureMemory, a single eavesdropped trace is enough to recover the secret key with probability 0.57 in 2^{39} cipher ticks. This is a factor of $2^{31.5}$ faster than a brute force attack. On a single core 2 GHz laptop, this takes around 10 minutes. With more traces, the secret key can be recovered with virtual certainty without significant additional cost in time. For CryptoMemory and CryptoRF, if one has 2640 traces it is possible to recover the key in 2^{52} cipher ticks, which is 2^{19} times faster than brute force. On a 50 machine cluster of 2 GHz quad core machines this would take less than 2 days.

8.1 Introduction

This chapter addresses the (in)security of the cryptographic mechanisms used in the Atmel product families SecureMemory, CryptoMemory, and CryptoRF.

These products are integrated circuits, consisting essentially of a small piece of memory and have cryptographic capabilities to authenticate and encrypt. They have two main application areas: in smart cards (ID and access cards, health care cards, loyalty cards, internet kiosk, energy meters, and e-government); and embedded, to authenticate one piece of hardware to another or for the secure storing of cryptographic data (printers and print cartridges, removable storage devices, set top boxes, access control systems, subassembly authentication, counterfeit protection, networked systems) [BJ04, CGY08, ZJS⁺09]. A concrete example is the widely sold NVIDIA graphics cards, which uses CryptoMemory as secure storage to protect the HDCP license keys¹, required to play high-definition video². It is also used in media players

¹http://www.expreview.com/review/2007-10-17/1192604816d5951_2.html

²http://download.nvidia.com/downloads/pvzone/Checklist_for_Building_a_HDPC.pdf

such as Microsoft's Zune Player[Dip09] and SanDisk's Sansa Connect³, for instance to securely store the device id and device certificate.

The Atmel chips AT88SC153 and AT88SC1608, which we call here the SecureMemory family, were introduced in 1999. A newer version, the CryptoMemory family, which includes the AT88SCxxxxC chips, with more advanced cryptographic features, was introduced in early 2002. These two families only have contact interfaces; in late 2003, the CryptoRF family, which is a variant of the CryptoMemory family with the AT88SCxxxxCRF chips, was introduced also has a Radio Frequency (RF) interface [Jar04].

The technology used in the chips is covered by US Patent 7395435 B2 [BCM08]. Details on the communication protocol used between a card (or slave device) and a reader (or master device) can be found in the documentation [AT07, AT09]; although this does include details on how the cryptographic algorithms are called and on the message flow between card and reader, it does not provide detailed information on the cipher itself. As experience has shown with other chips from the same era, the secrecy of such a proprietary cipher does not inspire confidence in its cryptographic strength, see Chapter 4. The cipher used in CryptoMemory, however, boasts a security stronger than DES; a security evaluation [Tec05] claims that the observed bias in the output is "trivial enough to confirm that CryptoMemory is impermeable to shortcut attacks". Admittedly, sophisticated attacks were not in the scope of this evaluation. We should note at this point that a backdoor has been found in the SecureMemory chips AT88SC153/1608 [Fly08]: using ultra-violet light, an EEPROM fuse can be restored to its original value and the contents of the memory can be read un-encrypted. A relay attack has been executed against the newer chips [KCP07].

Our contribution

As part of our research we have reverse engineered the security mechanisms of the SecureMemory and CryptoMemory families (the CryptoRF family uses exactly the same cipher as the CryptoMemory family, so we ignore that distinction from now on). This includes the ciphers, the authentication and encryption mechanisms, and the password mechanisms. This was accomplished by disassembling various executable applications from Atmel (e.g., ECESstudio), that implement the ciphers in software. We wrote a prototype implementation and then we compared its output with the packages we observed from the cards.

We have implemented the full functionality of these families in software and released this under GPL licence⁴. Our implementation can communicate with genuine Atmel products using commercial smartcards and readers.

³<http://www.rockbox.org/wiki/SansaConnect>

⁴Sample code available at <http://www.libnfc.org/documentation/examples/nfc-cryptorf>

In this chapter, we focus exclusively on the ciphers and the authentication mechanism; that is all that is needed to recover the keys.

SecureMemory and CryptoMemory use a stream cipher. It has an internal state of 109 bits (SecureMemory) or 117 bits (CryptoMemory), organized in three shift registers with a non-linear feedback function. Every clock tick, two of the registers produce 4 bits of output; the output of the third is used to select bits from one of the two other registers, producing 4 bits of keystream. The difference between SecureMemory and CryptoMemory, as far as the cipher is concerned, is that in CryptoMemory the last 8 bits of the output are remembered and are fed back into the shift registers every tick. Details are described in Section 8.3.

During initialization, 64-bit nonces from reader and card and a 64-bit shared key are used to set the initial state of the cipher; reader and tag subsequently authenticate by exchanging part of the following keystream. See Section 8.3.1 for details.

Next, we describe passive attacks against SecureMemory and CryptoMemory. We assume that an attacker has managed to eavesdrop a number of authentication sessions. For SecureMemory, we use a combination of a correlation attack (see for example [Rue92]) and a meet-in-the-middle attack to recover a significant part of the internal state of the cipher. Meet-in-the-middle attacks were also showed to be successful against other stream ciphers like A51 [BBK08] and KeeLoq [IKD⁺08]. Once we have the internal state of the cipher, we use a meet-in-the-middle attack to recover the shared secret key. This attack costs 2^{39} time, taking ticks of the cipher as unit. On a single core 2 GHz laptop, the attack takes approximately 10 minutes. Even though this attack is rather straightforward, it serves as an intermediate, didactic step to the more involved attack on CryptoMemory.

For CryptoMemory, the principal idea behind the attack is the same, but it is much more complicated because of the presence of the feedback of the output into the cipher state. We still use a correlation attack to recover a large part of the internal state (two of the three registers). Unlike with SecureMemory, we do not recover this with certainty, but obtain a few hundred to a few thousand candidates. Because of the feedback, it is also not directly possible to roll back these partial states, so we have to search for the correct remaining part of the internal state (the third register). After this, we can again unroll and use a meet-in-the-middle attack to recover the shared secret key. Using approximately 2640 traces, this attack has a time complexity of 2^{52} cipher ticks; it would take less than 2 days on a 50 machine cluster of 2 GHz quad core machines. Details on these attacks are described in Section 8.4 (for SecureMemory) and Section 8.5 (for CryptoMemory).

We have discussed these vulnerabilities with Atmel in October 2009. To give the manufacturer ample time to take appropriate measures, we have not disclosed them until March 2010.

8.2 Background

SecureMemory and CryptoMemory are ISO/IEC 7816 smartcards that communicate through a contact interface. CryptoRF is a ISO/IEC 14443-B smartcard that uses a contactless interface. CryptoMemory is also available in the form of an embedded EEPROM IC with a serial interface. The available memory ranges from 128 bytes to 32 KB. The memory is split into multiple user zones and one system zone which stores the configuration of the smartcard.

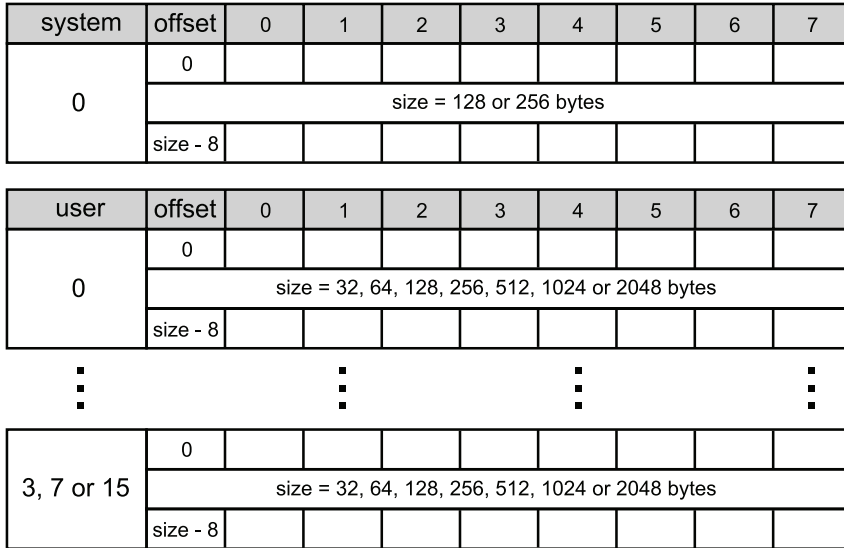


Figure 8.1: Logical memory structure

The system zone is divided into six sections: card identification, access control, cryptography, secret and password. Blank cards operate with a user-defined identifier which is customized by the system integrator. To achieve key diversification, the documentation recommends that the authentication keys should be derived from this identifier. The access control section is used to restrict read and write operations per user zone and define the requirements of successful Authentication and Key Agreement (AKA) and encrypted communication. To verify the legitimacy and prevent eavesdropping of data and passwords a mutual authentication protocol with encryption is available.

8.3 The ciphers

This section describes the stream ciphers that are used in SecureMemory and CryptoMemory. As mentioned in the introduction, the internal state of the cipher consists of three shift registers; for CryptoMemory, an additional register remembers the last 8 bits of output.

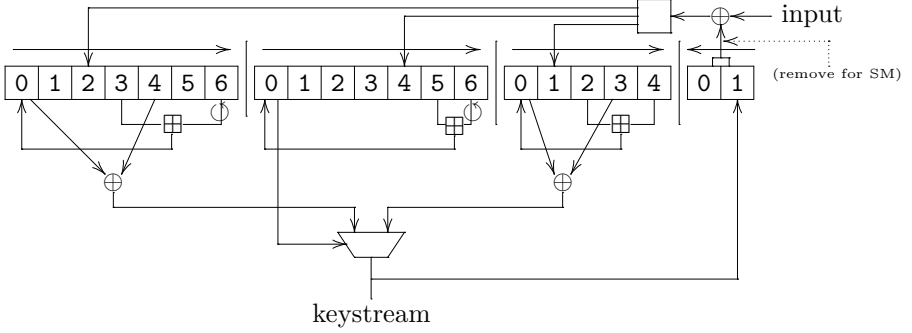


Figure 8.2: The schematic of the SecureMemory and CryptoMemory cipher

Definition 8.3.1. A cipher state of *SecureMemory* s is an element of \mathbb{F}_2^{109} consisting of the following components:

1. the left-hand segment $l = (l_0, \dots, l_6) \in (\mathbb{F}_2^5)^7$;
2. the middle segment $m = (m_0, \dots, m_6) \in (\mathbb{F}_2^7)^7$;
3. the right-hand segment $r = (r_0, \dots, r_4) \in (\mathbb{F}_2^5)^5$.

The cipher state of *CryptoMemory* is an element of \mathbb{F}_2^{117} . In addition to the three segments above, it also has the following component:

4. the feedback segment $f = (f_0, f_1) \in (\mathbb{F}_2^4)^2$.

Every tick, a cipher state s together with an input $a \in \mathbb{F}_2^8$ evolves in a successor state s' . First, the input a (and in case of *CryptoMemory* also the feedback segment f) are XORed into s at several places, giving an intermediate state \hat{s} . Second, the left, right and middle segments are shifted to the right and new 0th entries are computed using a bitwise rotate and a modular addition. Thirdly and finally, the output segment is shifted to the left and a new 1st entry is computed using a non-linear function of the other segments, giving the successor state s' . The following sequence of definitions gives the details of this construction; see also Figure 8.2.

Definition 8.3.2. Define the bitwise left rotate operator $L: \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ by

$$L(x_0x_1 \dots x_{n-1}) = (x_1 \dots x_{n-1}x_0).$$

Definition 8.3.3. Define the modified modular addition operator $\boxplus: \mathbb{F}_2^n \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ (identifying elements of \mathbb{F}_2^n with elements of $\{0, 1, \dots, 2^n - 1\}$) by

$$x \boxplus y = \begin{cases} x + y \pmod{2^n - 1} & \text{if } x = y = 0 \text{ or } x + y \neq 0 \\ 2^n - 1 & \text{otherwise,} \end{cases}$$

where $+ \pmod{2^n - 1}$ denotes integer addition modulo $2^n - 1$.

Note that this operation is not injective when fixing one of the arguments: $x \boxplus 0$ and $x \boxplus (2^n - 1)$ both equal x (unless $x = 0$, because $0 \boxplus 0 = 0$ and $0 \boxplus (2^n - 1) = 2^n - 1$). Also observe that the result of \boxplus is never 0 unless both operands are 0.

Definition 8.3.4. Let $s = (l, m, r, f) \in \mathbb{F}_2^{117}$ be a cipher state (for CryptoMemory; for SecureMemory, ignore f) and let $a \in \mathbb{F}_2^8$ be an input. We define the intermediate state $\hat{s} = (\hat{l}, \hat{m}, \hat{r}, \hat{f})$ and the successor state $s' = (l', m', r', f')$ as follows. For CryptoMemory, define $b := a \oplus f_0 f_1$; for SecureMemory, $b := a$. The intermediate state \hat{s} is given by

$$\begin{aligned}\hat{l}_2 &:= l_2 \oplus b_3 b_4 b_5 b_6 b_7 \\ \hat{m}_4 &:= m_4 \oplus b_4 b_5 b_6 b_7 b_0 b_1 b_2 \\ \hat{r}_1 &:= r_1 \oplus b_0 b_1 b_2 b_3 b_4\end{aligned}\tag{8.1}$$

and all other entries are copied from l , m , r , and f . The successor state s' is given by

$$\begin{aligned}l'_{i+1} &:= \hat{l}_i & i \in \{0, \dots, 5\} \\ m'_{i+1} &:= \hat{m}_i & i \in \{0, \dots, 5\} \\ r'_{i+1} &:= \hat{r}_i & i \in \{0, \dots, 3\} \\ l'_0 &:= \hat{l}_3 \boxplus L(\hat{l}_6) \\ m'_0 &:= \hat{m}_5 \boxplus L(\hat{m}_6) \\ r'_0 &:= \hat{r}_2 \boxplus \hat{r}_4.\end{aligned}$$

We call the rightmost 4 bits of $l'_0 \oplus l'_4$ the output of the left-hand segment (i.e., $l'_{0,1} \oplus l'_{4,1} \ l'_{0,2} \oplus l'_{4,2} \ \dots \ l'_{0,4} \oplus l'_{4,4}$, a bitwise XOR) and denote it by $outputl(l')$. We call the rightmost 4 bits of $r'_0 \oplus r'_3$ the output of the right-hand segment r' (i.e., $r'_{0,1} \oplus r'_{3,1} \ r'_{0,2} \oplus r'_{3,2} \ \dots \ r'_{0,4} \oplus r'_{3,4}$) and denote it by $outputr(r')$. The output of the cipher state s' , denoted by $output(s')$ is defined by

$$output(s')_i = \begin{cases} outputl(l')_i, & \text{if } m'_{0,i+3} = 0; \\ outputr(r')_i, & \text{if } m'_{0,i+3} = 1 \end{cases} \quad i \in \{0, \dots, 3\}.$$

Note how the rightmost 4 bits of the middle segment acts as a selector; it selects either a bit from the left-hand segment or a bit from the right-hand segment to be included in the output. For CryptoMemory, we define

$$\begin{aligned}f'_0 &:= \hat{f}_1 (= f_1) \\ f'_1 &:= output(s').\end{aligned}$$

Define the transition function suc that takes an input a and a state s and outputs the successor state s' . We write $suc^n(a, s)$ denoting $suc(a, s)$ when $n = 1$ and $suc^{n-1}(a, suc(a, s))$ when $n > 1$.

Note that the whole feedback segment can be reconstructed from the other three segments; $f_1 = \text{output}(s)$, but also f_0 can be reconstructed: shift all segments in s one to the left and take output. To be precise, we have

$$f_{0,i} = \begin{cases} l_{1,i+1} \oplus l_{5,i+1} & \text{if } m_{1,i+3} = 0 \\ r_{1,i+1} \oplus r_{4,i+1} & \text{if } m_{1,i+3} = 1. \end{cases} \quad (8.2)$$

(So the rightmost 4 bits of m_1 act as a selector between the rightmost 4 bits of $l_1 \oplus l_5$ and $r_1 \oplus r_4$.)

8.3.1 Initialization and authentication

The cipher is initialized during the authentication protocol. At the beginning of this protocol, tag and reader exchange nonces as depicted in Figure 8.3. The tag nonce is scrambled together with the first half of the reader nonce and fed into the cipher. Subsequently, the shared key scrambled together with the second half of the reader nonce is fed in. Then, the cipher produces keystream that will be used as authenticator for both reader and tag. The precise definitions follow.

Let $nt \in (\mathbb{F}_2^8)^8$ be a tag nonce, $nr \in (\mathbb{F}_2^8)^8$ a reader nonce, and $k \in (\mathbb{F}_2^8)^8$ the shared key between the tag and the reader.

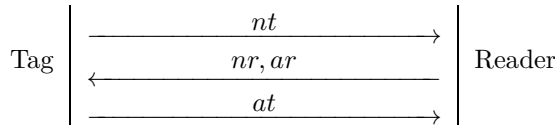


Figure 8.3: Authentication protocol

The initial cipher state has all components l , m , r (for SecureMemory and CryptoMemory) and f (for CryptoMemory) equal to zero. Then the cipher is evolved through the states s_0, s_1, \dots defined as

$$\begin{aligned} s_0 &:= 0 \\ s_{i+1} &:= \text{suc}(nr_i, \text{suc}^v(nt_{2i+1}, \text{suc}^v(nt_{2i}, s_i))) & i \in \{0, \dots, 3\} \\ s_{i+5} &:= \text{suc}(nr_{i+4}, \text{suc}^v(k_{2i+1}, \text{suc}^v(k_{2i}, s_{i+4}))) & i \in \{0, \dots, 3\} \end{aligned}$$

where $v = 1$ for SecureMemory and $v = 3$ for CryptoMemory. The following is an overview of the input during the initialisation phase of SecureMemory (left) and CryptoMemory (right).

(s_0)	nt_0	nt_1	nr_0	(s_0)	nt_0	nt_0	nt_0	nt_1	nt_1	nt_1	nr_0
(s_1)	nt_2	nt_3	nr_1	(s_1)	nt_2	nt_2	nt_2	nt_3	nt_3	nt_3	nr_1
(s_2)	nt_4	nt_5	nr_2	(s_2)	nt_4	nt_4	nt_4	nt_5	nt_5	nt_5	nr_2
(s_3)	nt_6	nt_7	nr_3	(s_3)	nt_6	nt_6	nt_6	nt_7	nt_7	nt_7	nr_3
(s_4)	k_0	k_1	nr_4	(s_4)	k_0	k_0	k_0	k_1	k_1	k_1	nr_4
(s_5)	k_2	k_3	nr_5	(s_5)	k_2	k_2	k_2	k_3	k_3	k_3	nr_5
(s_6)	k_4	k_5	nr_6	(s_6)	k_4	k_4	k_4	k_5	k_5	k_5	nr_6
(s_7)	k_6	k_7	nr_7	(s_7)	k_6	k_6	k_6	k_7	k_7	k_7	nr_7
(s_8)				(s_8)							

Starting from the state called s_0 , we feed in nt_0 (three times for CryptoMemory), then nt_1 (three times for CryptoMemory), then nr_0 , and arrive at the state called s_1 , etc. Note that these states are non-consecutive, e.g., s_1 is not the successor of s_0 ; we have only named those states that are needed for the description.

The authentication protocols for CryptoMemory and SecureMemory are similar, although the keystream bits used as authenticators are different in both cards. The precise definitions follow.

SecureMemory Authentication

After initialization, the cipher produces keystream that will be used for mutual authentication of tag and reader. In SecureMemory, every second output nibble is discarded. The keystream bits used as authenticators for the tag $at \in (\mathbb{F}_2^4)^{16}$ and for the reader $ar \in (\mathbb{F}_2^4)^{16}$ are interleaved. We define the following relevant states:

$$s_i := \text{suc}^2(0, s_{i-1}) \quad i \in \{9, \dots, 40\}.$$

The authenticator nibbles for the tag are

$$\begin{aligned} at_i &:= \text{output}(s_{2i+9}) \\ at_{i+1} &:= \text{output}(s_{2i+10}) \end{aligned} \quad i \in \{0, 2, \dots, 14\}$$

and the authenticator nibbles for the reader are

$$\begin{aligned} ar_i &:= \text{output}(s_{2i+11}) \\ ar_{i+1} &:= \text{output}(s_{2i+12}) \end{aligned} \quad i \in \{0, 2, \dots, 14\}.$$

The overview of the output during the authentication phase of SecureMemory is as follows.

$$\begin{aligned}
(s_8) & - at_0 - at_1 - ar_0 - ar_1 \\
(s_{12}) & - at_2 - at_3 - ar_2 - ar_3 \\
(s_{16}) & - at_4 - at_5 - ar_4 - ar_5 \\
(s_{20}) & - at_6 - at_7 - ar_6 - ar_7 \\
(s_{24}) & - at_8 - at_9 - ar_8 - ar_9 \\
(s_{28}) & - at_{10} - at_{11} - ar_{10} - ar_{11} \\
(s_{32}) & - at_{12} - at_{13} - ar_{12} - ar_{13} \\
(s_{36}) & - at_{14} - at_{15} - ar_{14} - ar_{15} \\
(s_{40}) &
\end{aligned}$$

Starting from the state called s_8 , one output nibble is discarded, the following is called at_0 , the next one is discarded, etc., until ar_1 . The state then reached is called s_{12} , etc.

CryptoMemory Authentication

After initialization, CryptoMemory generates the reader authenticator first and then the tag authenticator. While generating the reader authenticator $ar \in (\mathbb{F}_2^4)^{16}$, five keystream nibbles are discarded and then two nibbles (one byte) are used, with the exception of the first byte where only 4 nibbles are discarded. The tag authenticator $at \in (\mathbb{F}_2^4)^{16}$ consists of the bitstring `0xff` followed by the next 14 (consecutive) keystream nibbles produced. To be precise, define the following sequence of states. Again, note that these states are not consecutive.

$$\begin{aligned}
s_9 & := \text{suc}^5(0, s_8) \\
s_{10} & := \text{suc}(0, s_9) \\
s_i & := \text{suc}^6(0, s_{i-1}) & i \in \{11, 13, \dots, 23\} \\
s_i & := \text{suc}(0, s_{i-1}) & i \in \{12, 14, \dots, 24\} \\
s_i & := \text{suc}(0, s_{i-1}) & i \in \{25, \dots, 38\}.
\end{aligned}$$

Then the reader authenticator is defined as

$$ar_i := \text{output}(s_{i+9}) \quad i \in \{0, \dots, 15\}$$

and the tag authenticator is defined as

$$\begin{aligned}
at_0 & := \text{0xf} \\
at_1 & := \text{0xf} \\
at_i & := \text{output}(s_{i+23}) & i \in \{2, \dots, 15\}.
\end{aligned}$$

The following shows a schematic overview of the output during the authentication phase of CryptoMemory.

$$\begin{array}{l}
(s_8) \quad 4\times - \quad ar_0 \quad ar_1 \quad 5\times - \quad ar_2 \quad ar_3 \\
(s_{12}) \quad 5\times - \quad ar_4 \quad ar_5 \quad 5\times - \quad ar_6 \quad ar_7 \\
(s_{16}) \quad 5\times - \quad ar_8 \quad ar_9 \quad 5\times - \quad ar_{10} \quad ar_{11} \\
(s_{20}) \quad 5\times - \quad ar_{12} \quad ar_{13} \quad 5\times - \quad ar_{14} \quad ar_{15} \\
(s_{24}) \quad at_2 \quad at_3 \quad at_4 \quad at_5 \quad at_6 \quad at_7 \quad at_8 \\
(s_{31}) \quad at_9 \quad at_{10} \quad at_{11} \quad at_{12} \quad at_{13} \quad at_{14} \quad at_{15} \\
(s_{38})
\end{array}$$

Starting from the state called s_8 , the first 4 output nibbles generated are discarded, the next two are called ar_0 and ar_1 respectively, the next 5 are discarded again, etc. For future reference, note that an attacker who has observed an authentication trace sees 16 consecutive keystream nibbles, viz., $ar_{14}, ar_{15}, at_2, at_3, \dots, at_{15}$.

Example 8.1. *Table 8.1 shows an authentication trace using the shared key $k = 0x4f794a463ff81d81$; the first two bytes on every line are a command and the last two are a CRC.*

Table 8.1: Communication trace of the CryptoMemory authentication protocol

	Message	Interpretation
R	1600 5007 add3	Read nt
T	1600 ff81c91e11a6393e 00 1b66	$nt = ff \dots 3e$
R	1800 3d28a6ae3a767a25 d308e40bb3200ee0 a905	Auth $nr = 3d \dots 25, ar = d3 \dots e0$
T	1800 00 9b85	Ok
R	1600 5007 add3	Read at
T	1600 ff4c1c06b43cbcc2 00 440b	$at = ff \dots c2$

8.4 Attacking SecureMemory

We now turn our attention to attacking the ciphers. We start with an attack against SecureMemory. We assume that an attacker has eavesdropped a single authentication session; the attack we describe recovers the shared secret key with probability at least 0.57. By using more authentication sessions we can arbitrarily increase this probability.

The attack we describe takes place in two phases. First, we use a correlation attack to recover (the left-hand and right-hand segments of) the internal state of the cipher just after feeding in the key and just before generating the authenticators (called state s_8 in Section 8.3.1). Since the attacker knows nr and nt , he can also compute the state just before feeding in the shared secret key (called state s_4). Then, we use a meet-in-the-middle attack to recover k .

The running time of the whole attack is 2^{39} ticks of the SecureMemory cipher, which on a single core 2 GHz laptop takes about 10 minutes.

8.4.1 Recovering the internal state

There are two weaknesses in the SecureMemory cipher that make it possible to recover the internal state of the cipher. The first one is that there is a high correlation between the output of the right-hand (or left-hand) segment and the keystream itself. Consider a state s and its output nibble $\text{output}(s)$. For those bits for which the middle segment chooses the right ($m_{0,i} = 1$), the output bit is equal to the corresponding output bit of the right-hand segment ($\text{output}(s)_i = \text{output}_r(r)_i$). Assuming a uniform probability, this happens with probability $\frac{1}{2}$. Where the middle segment chooses the left ($m_{0,i} = 0$), the output bit is equal to the corresponding output bit of the left-hand segment ($\text{output}(s)_i = \text{output}_l(l)_i$), but with probability $\frac{1}{2}$ this equals the output bit of the right-hand segment ($\text{output}(r)_i$) anyway. So, with probability $\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$, an output bit of the right-hand segment equals the corresponding keystream bit (and similarly for the left-hand segment). The second weakness is that the three segments operate independently. So, knowing (or guessing) the right-hand (or left-hand or middle) segment of a state s , the attacker can compute the right-hand (or left-hand or middle) segment of the successor state s' .

Definition 8.4.1. *Consider a guess r for the right-hand segment of the internal state of the cipher at the start of the authentication phase (state s_8 in Section 8.3.1) and consider the 16 output nibbles generated by the right-hand segment that are used to compute at and ar . We define the score of r to be the number of bits that these 16 output nibbles have in common with the actual keystream (i.e., with at and ar). A similar definition applies for a guess l of the left-hand segment.*

For a wrong guess of the right-hand segment, one would expect the score to be 64 (half of 128 bits correct); for the correct guess, one would expect the score to be 96 ($\frac{3}{4}$ of the 128 bits correct). Hence, the attacker iterates over the 2^{25} possible right-hand segments and computes their score. The one with the highest score is most likely to be the correct one. As an approximation, we can assume that the score for the $2^{25} - 1$ wrong guesses is binomially distributed with parameters $p = \frac{1}{2}$ and $n = 128$ (128 bits, each of which has a probability of $\frac{1}{2}$ of being correct) and the score for the correct guess is binomially distributed with parameters $p = \frac{3}{4}$ and $n = 128$. Also assuming, again as an approximation, that all these score are independent random variables, the following proposition applies.

Proposition 8.4.1. *Let $X_1, \dots, X_{2^{25}-1} \sim \text{Binom}(128, \frac{1}{2})$ and $Y \sim \text{Binom}(128, \frac{3}{4})$ be independent random variables. Then*

$$\mathbb{P}[\forall i. X_i < Y] \approx 0.57$$

and

$$\mathbb{P}[\exists i. [Y < X_i \wedge \forall j \neq i. X_j < X_i]] \approx 0.24.$$

Proof. Since the X_i 's and Y are assumed to be independent, we get

$$\begin{aligned} \mathbb{P}[\forall i. X_i < Y] &= \sum_{k=0}^{128} \mathbb{P}[\forall i. X_i < k \wedge Y = k] \\ &= \sum_{k=0}^{128} \mathbb{P}[X_1 < k]^{2^{25}-1} \cdot \mathbb{P}[Y = k] \approx 0.57 \end{aligned}$$

and

$$\begin{aligned} \mathbb{P}[\exists i. [Y < X_i \wedge \forall j \neq i. X_j < X_i]] \\ &= \sum_{k=0}^{128} \mathbb{P}[\exists i. X_i = k \wedge \forall j \neq i. X_j < k \wedge Y < k] \\ &= \sum_{k=0}^{128} \sum_{i=1}^{2^{25}-1} \mathbb{P}[X_i = k \wedge \forall j \neq i. X_j < k \wedge Y < k] \\ &= \sum_{k=0}^{128} (2^{25} - 1) \cdot \mathbb{P}[X_1 = k] \cdot \mathbb{P}[X_2 < k]^{2^{25}-2} \cdot \mathbb{P}[Y < k] \\ &\approx 0.24 \end{aligned} \quad \square$$

So, as an approximation, with probability 0.57, the correct right-hand segment has a score that is higher than the score of all the wrong ones. With probability of only 0.24, a wrong right-hand segment has a score higher than all the others (including the correct one). Therefore we get with probability 0.81 one right-hand segment that scores higher than all the others. So we need, on average $1/0.81 \approx 1.23$ traces to obtain such a single candidate. The conditional probability that this candidate is indeed the correct right-hand segment is $0.57/0.81 \approx 0.71$. To obtain a trace in which the highest scoring right-hand segment is indeed the correct one, we need on average $1/0.57 \approx 1.75$ traces.

From now on, assume that we have a trace for which exactly one right-hand segment scores higher than all the others. For the attack we assume that it is the correct right-hand segment r .

Then we try to find possible candidates for the left-hand segment. Note that for those bits of the known keystream where the output of the right-hand segment does *not* produce the correct bit, the corresponding bit from the middle segment (the selector) *must* choose the left-hand segment and the corresponding bit from the output of the left-hand segment *must* equal that keystream bit. For instance, when r has a score of 96, this happens $128 - 96 = 32$ times. So, the attacker iterates over all 2^{35} possibilities for the left-hand segment, keeping only those that satisfy the above constraint. Experiments show that this leaves only between, approximately, 10 and 200 candidates.

At this point, one could also try to recover (candidates for) the middle segment, but as we will show in the next two sections, that is not even necessary. Later, when we are attacking CryptoMemory in Section 8.5, we do recover the middle part as well.

8.4.2 Unrolling the cipher

Whenever the input b is known, given a state s' it is possible run the cipher backwards and recover the previous state s . We start by defining an inverse to \boxplus .

Definition 8.4.2. Define $\boxminus: \mathbb{F}_2^n \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ by

$$x \boxminus y = x - y \pmod{2^n - 1}.$$

To reconstruct s , we first unshift the cipher, recovering the intermediate state \hat{s} :

$$\begin{aligned} \hat{l}_i &:= l'_{i+1} & i \in \{0, \dots, 5\} \\ \hat{m}_i &:= m'_{i+1} & i \in \{0, \dots, 5\} \\ \hat{r}_i &:= r'_{i+1} & i \in \{0, \dots, 3\} \\ \hat{l}_6 &:= L^{-1}(l'_0 \boxplus l'_4) \\ \hat{m}_6 &:= L^{-1}(m'_0 \boxplus m'_6) \\ \hat{r}_4 &:= r'_0 \boxplus r'_3. \end{aligned}$$

There is, however, an issue here that needs special care: the modified modular addition operator \boxplus is not injective when fixing one argument. When $l'_0 = l'_4 \neq 0$ there are two possible previous left-hand segments, namely, $L(l'_6)$ could be equal to either 0 or to 31. Therefore, in those cases we have to consider both possible predecessors. Similarly, splitting might happen in the middle and right-hand segments. This issue does not significantly hamper the possibility of unrolling the cipher since it only splits with probability $1/31$ for the left-hand and right-hand segments and with probability $1/127$ for the middle segment. Also, because the result of \boxplus is never 0 unless both operands are 0, there are some states that do not have a predecessor. This happens when $l'_0 = 0$, but $l'_4 \neq 0$.

Next, we simply XOR back the input to recover the previous state:

$$\begin{aligned} l_2 &:= \hat{l}_2 \oplus b_3 b_4 b_5 b_6 b_7 \\ m_4 &:= \hat{m}_4 \oplus b_4 b_5 b_6 b_7 b_0 b_1 b_2 \\ r_1 &:= \hat{r}_1 \oplus b_0 b_1 b_2 b_3 b_4. \end{aligned}$$

Also note that when unrolling the cipher, the three segments operate independently.

8.4.3 Recovering the key

In this section we use a meet-in-the-middle technique to recover the secret key. We recall from Section 8.3.1 that the cipher state s_4 , just before feeding in the key k , is known to an attacker. Observe that the transition function for the right-hand segment only uses five bits $b_0 \dots b_4$ of the input. Therefore, by guessing 20 bits of the key, it is possible to roll the cipher 6 times (feeding in nr_4 and nr_5 accordingly) and

compute 2^{20} possibilities for the right-hand segment of s_6 . Similarly starting at the end; we have just recovered the right-hand segment for the state at the start of the generation of the authenticators, i.e., just after feeding in the key. By guessing the other 20 bits of the key, it is possible to unroll the cipher 6 times (feeding in nr_7 and nr_6 accordingly) and obtain another set of 2^{20} candidates for the right-hand segment of s_6 . (Actually, rolling back there are more candidates because of the splitting; in practice this is at most a ten fold increase.) Intersecting these sets of states, since the right-hand segment has 25 bits of entropy and we have guessed 40 bits of the input, we get approximately 2^{15} candidates. In practice, because of the additional splitting when rolling back, we get between 30 000 and 66 000 candidates for the right-hand segment of s_6 . Note that we do not just get this set of candidates, but for each candidate we also have the leftmost five bits of every byte of the key. So, in fact, we get between 30 000 and 66 000 candidates for the leftmost five bits of every byte of the key.

Similarly for the left-hand segment, the transition function uses only bits $b_3 \dots b_7$. Just as before, but for each candidate segment l from Section 8.4.1, we guess two times 20 bits and we meet-in-the-middle at (the left-hand segment of) s_6 . In practice, we get between 1000 and 3000 candidates. Also here, we do not just obtain this set of candidates, but for each candidate we have the rightmost five bits of every byte of the key. So we get between 1000 and 3000 candidates for the rightmost five bits of every byte of the key.

We now combine the first set of *left-hand candidates* and the second set of *right-hand candidates*. Of course, a left-hand candidate (40 bits of the key) can only be combined with a right-hand candidate (also 40 bits of the key) if the bits of the key that they share are equal. These are bits b_3 and b_4 of every byte; $8 \cdot 2 = 16$ bits in total. So, we get between $2^{16} \cdot (30\,000/2^{16}) \cdot (1000/2^{16}) \approx 450$ and $2^{16} \cdot (66\,000/2^{16}) \cdot (3000/2^{16}) \approx 3000$ candidate keys. Simulating the whole authentication session, which now also involves running the middle segment, with each of these candidate keys reveals what the actual key is.

8.4.4 Complexity and time

As a basic unit of time, we take one tick of the SecureMemory cipher. As a reference note that an authentication attempt takes 88 ticks which means that a naive brute force attack takes $2^{64} \cdot 88 = 2^{70.5}$ cipher ticks.

The most time in this attack is spent in recovering the set of 10 to 200 candidates for the left-hand segment. Here we have to loop over 2^{35} candidates; for each of those candidates we have to compute a maximum of 64 ticks of the cipher, two for each of the nibbles of ar and at since SecureMemory skips every second output during the generation of the authenticators. (Note, we only have to compute the left-hand segment, so we count this as $\frac{1}{3}$). So this gives a complexity of 2^{39} ticks, comparable to the time it takes to simulate $2^{32.5}$ authentications. Note that for the right-hand

segment we only have to loop over 2^{25} candidates and for the meet-in-the-middle key-recovery we only have to build tables of size 2^{20} , so we can ignore that. Also note that when our authentication session does not have the desired property that a single right-hand segment has the highest score, we only have wasted the 2^{25} loop over the possible right-hand segments. Since this happens only with probability $1 - 0.81 = 0.19$ we can ignore this time. After on average $1/0.81 \approx 1.23$ traces, we do have a single candidate for the right-hand segment and with probability 0.71 we recover the key in 2^{39} ticks. This yields an average complexity of $2^{39}/0.71 = 2^{39.5}$ ticks, i.e., the complexity of simulating 2^{33} authentications. On average, this needs $1/0.57 \approx 1.75$ authentication traces. In practice, running this attack on a single core 2 GHz laptop takes about 10 minutes.

8.5 Attacking CryptoMemory

We now turn our attention to CryptoMemory. We describe an attack in three phases: recovering the internal state by means of a correlation attack; unrolling the internal state; and recovering the key by means of a meet-in-the-middle attack. There is, however, a major complication. Because the output of the cipher is fed back into the internal state, it is no longer possible to run the three segments independently, at least not under all circumstances. This seriously complicates the recovery of the internal state and also makes unrolling the cipher slightly harder. Finally, we propose a trade-off between the number of authentication traces needed and off-line computation time, similar to the one proposed by Biryukov et al. in [BMS06].

8.5.1 Recovering the internal state

The starting point for this attack is the same as the one described in Section 8.4. Although the three segments cannot be run independently as for SecureMemory, because of the feedback from the keystream, it is possible to do so when the keystream is fully known. Now most of the time, CryptoMemory discards several output nibbles when generating keystream. When it generates the tag authenticator ar , however, CryptoMemory does produce 16 consecutive keystream nibbles (namely the last 2 of at and the 14 of ar that it actually generates). Note, by the way, that SecureMemory never produces consecutive keystream nibbles, but there that is not needed for the attack anyway. Let $ks \in (\mathbb{F}_2^4)^{16}$ be those 64 bits of keystream. Knowing that the keystream bits are equal to the output of either the left-hand or the right-hand segment, we define the following score, similar to the one for SecureMemory, but using only the 64 bits of ks .

Definition 8.5.1. *Consider a guess r for the right-hand segment of the internal state of the cipher just before producing the last byte of ar (the state s_{23} in Section 8.3.1) and consider the 16 output nibbles generated by the right-hand segment that are used to compute ar_{14} , ar_{15} , and at_2 to at_{15} . We define the score of r to be the number*

of bits that these 16 output nibbles have in common with the actual keystream ks . A similar definition applies for a guess l of the left-hand segment.

A random segment has an expected score of 32. The correct segment, instead, has an expected score of 48. Although not nearly as pronounced as for SecureMemory, this correlation can be exploited to narrow our search space to segments with high score.

The attack proceeds as follows. An attacker first eavesdrops a number N_t of authentication traces. For each trace it will iterate over all 2^{35} left-hand segments and keep only those with score higher than a threshold N_l . Similarly for the right-hand segment, it will iterate over all 2^{25} right-hand segments and keep only those with score higher than a threshold N_r . We call these segments *candidate segments*, as these are our guesses for the left-hand and right-hand segments of s_{23} . Now, for each (l, r) pair of candidate segments, an adversary could try all 2^{49} middle segments, unroll the cipher as described in Section 8.5.2 and check if it produces the correct ar nibbles. It is, however, possible to do better than that; we now describe a directed search through these middle segments.

Let $ksl_0 = \text{outputl}(l)$ and $ksr_0 = \text{outputr}(r)$ be the first nibble of output generated by l and r , respectively. Then, since we know the keystream produced, for those bits where ksl_0 is different from ksr_0 , we know what the selector bits are, i.e., some bits of m_0 . On average, we know two bits out of four.

More precisely, for all $j \in \{0, 1, 2, 3\}$

$$\text{if } ksl_{0,j} \oplus ksr_{0,j} = 1, \text{ then } m_{0,j} := ksl_{0,j} \oplus ks_{0,j}. \quad (8.3)$$

Next, we compute the successor state. We have only partial information on the middle segment, viz., only a few bits of m_0 . Hence, we only get partial information on the middle segment of the successor state, viz., a few bits of m'_1 . We repeat this procedure six times, for ksl_1 and ksr_1 to ksl_6 and ksr_6 , each time obtaining more partial information about the components of the middle segment. After that there is no extra information gain as we know a number of bits of each component and they start falling off on the right.

So far we have consumed seven out of sixteen keystream nibbles. Next we start searching but keep only those states that are consistent with the remaining keystream. Precisely, we iterate over all $2^7 = 128$ values for m_5 and m_6 , but consider only those that match, respectively, the known bits of m_5 and m_6 . Then we compute the successor state. At this point $m_5 \boxplus L(m_6)$ is assigned to m_0 , m_6 falls out, and m_5 shifts to m_6 . We now check if the newly computed m_0 satisfies the condition

$$\text{if } ksl_{7,j} \oplus ksr_{7,j} = 1, \text{ then } m_{0,j} = ksl_{7,j} \oplus ks_{7,j}$$

and otherwise discard it.

Now that all bits of m_6 are set, we iterate only over all 128 values of m_5 . Again, we compute the successor state and check the corresponding condition. At this point

m_0 is shifted one position to the right and a new m_0 is computed from the sum m_5 and m_6 , i.e., we have set all bits of m_1 , m_0 and m_6 . We repeat this procedure four more times, until all seven words of the middle segment are set. This gives us a number of candidate states that depend on the overlap between ksl and ksr . These candidate states can be unrolled as we describe in Section 8.5.2 and verified against the ar nibbles.

8.5.2 Unrolling the cipher

Just as with SecureMemory, it is also possible to unroll the CryptoMemory cipher. If we are at a state s' and know the keystream f_0f_1 output by the previous state s , the procedure is merely as described in Section 8.4.2, taking care of setting the input b to $a \oplus f_0f_1$ accordingly. If we do not know the keystream f_0f_1 , we first have to reconstruct it. Of course, reconstructing f_1 is no problem, as $f'_0 = f_1$. Now, first of all, we compute \hat{s} as in Section 8.4.2 (if splitting happens, consider one possibility for \hat{s}). Write $b = b_0b_1 \dots b_7 = a \oplus f_0f_1$. By Equations (8.2) and (8.1), we have

$$\begin{aligned} f_{0,i} &= \begin{cases} l_{1,i+1} \oplus l_{5,i+1} & \text{if } m_{1,i+3} = 0 \\ r_{1,i+1} \oplus r_{4,i+1} & \text{if } m_{1,i+3} = 1 \end{cases} \\ &= \begin{cases} \hat{l}_{1,i+1} \oplus \hat{l}_{5,i+1} & \text{if } \hat{m}_{1,i+3} = 0 \\ \hat{r}_{1,i+1} \oplus b_{i+1} \oplus \hat{r}_{4,i+1} & \text{if } \hat{m}_{1,i+3} = 1. \end{cases} \end{aligned}$$

Now note that $b_4 = a_4 \oplus f_{1,0} = a_4 \oplus \hat{f}_{1,0}$, so we can use the above equation to compute $f_{0,3}$. Now $b_3 = a_3 \oplus f_{0,3}$, so we can use the above equation again to compute $f_{0,2}$. Doing this twice more, we can also compute $f_{0,1}$ and $f_{0,0}$. Using Equation (8.1), we can now compute s .

8.5.3 Recovering the key

As before, we use a meet-in-the-middle technique to recover the secret key. This time the computational complexity of the attack is higher due to the keystream feedback loop, which makes it impossible to treat the left-hand and right-hand segments separately.

Recall that the cipher state s_4 , just before feeding in the key k , is known to an adversary and assume the adversary also knows the state s_8 , e.g., by running the attack described in Section 8.5.1.

Then, the adversary simply guesses the first 32 bits of k and runs the cipher forward from s_4 , until half way the initialization protocol, i.e., to state s_6 . This produces a set S_f of 2^{32} candidate states for s_6 .

Similarly, it guesses the last 32 bits of k and runs the cipher backwards (unrolls) from s_8 . This produces a set S_b of candidate states (around $2 \cdot 2^{32}$; the additional factor is because of the states with multiple predecessors). Since we are guessing only

two times 32 bits and states consist of 117 bits, there is only one element in $S_f \cap S_b$, namely s_6 . The guessed bits that lead to it constitute the actual key k .

8.5.4 Complexity and time

As a baseline let us first establish the complexity of a naive brute force attack, again taking cipher ticks as a basic unit of time. For each of the 2^{64} possible keys, we need 125 cipher ticks to simulate an authentication. This adds to a complexity of 2^{71} cipher ticks.

For the first phase of the attack, recovering the internal state, we need to iterate over all 2^{35} left-hand segments and for each of them we need to compute the score, which takes 16/3 cipher ticks. This is of complexity 2^{38} ticks and takes about half an hour on a single core 2 GHz laptop. We also need to do the same for all 2^{25} right-hand segments, but this can be ignored.

The number of left-right candidates produced depends on the values of N_l and N_r (defined in Section 8.5.1) and there is a trade-off between the number of traces required for the attack and its computational complexity. As an example we consider two reference configurations: $C_0 := (N_l = 55, N_r = 51)$ and $C_1 := (N_l = 56, N_r = 52)$. Take for example C_0 . Our attack from Section 8.5.1 will only be able to recover the internal state for those traces with left-hand segment score higher than 55 and right-hand segment score higher than 51. The score of the left-hand and right-hand segments of the correct state are binomially distributed as $\text{Binom}(64, \frac{3}{4})$. Hence the probability that the score of the correct left-hand segment is at least 55 is approximately 0.025 and the probability that the score of the correct right-hand segment is at least 51 is approximately 0.239. Assuming these scores are independent, the probability that both scores satisfy the requirement is $0.025 \cdot 0.239 \approx 0.0060$. Experiments show that the distributions of these scores is indeed very close to binomially distributed, but they are not totally independent. The experimentally observed probability of having both scores meet the requirement is approximately 0.00185. So the expected number of authentication session needed is $1/0.0018 \approx 166$. The score of a random left-hand or right-hand segment is also binomially distributed, but with distribution $\text{Binom}(64, \frac{1}{2})$. The probability of a random segment having a score of 55 or higher is $0.177 \cdot 10^{-8}$ and the probability of a random segment having a score of 51 or higher is $0.95 \cdot 10^{-6}$. So, we expect around $0.177 \cdot 10^{-8} \cdot 2^{35} \approx 60.9$ left-hand segments to have a score of 55 or higher and around $0.95 \cdot 10^{-6} \cdot 2^{25} \approx 31.6$ right-hand segments to have a score of 51 or higher. So this gives approximately $60.9 \cdot 31.6 \approx 1920$ left-right candidates per trace. We established the number of candidates for the middle segment per left-right candidate empirically; we get on average $1.43 \cdot 10^9$ candidates for the middle segment. For each middle segment we need to unroll the cipher at most 64 times. This gives us a total complexity of 2^{55} cipher ticks, comparable to simulating 2^{48} authentications. Assuming we dispose of a cluster of 50 2 GHz quad core computers, all this computation would take about two weeks.

Considering C_1 , the expected number of traces needed to find one in which the correct segment has a score of at least 56 and the correct right-hand segment has a score of at least 52 is $1/0.000378 \approx 2640$ traces. Here we expect only $9.5 \cdot 7.7 \approx 73$ left-right candidates. Experiments show that the number of middle segments per left-right candidate is approximately $2.12 \cdot 10^{10}$. This gives us a total complexity of 2^{52} cipher ticks, i.e., the computational complexity of 2^{45} authentications. With the same computational power this would take less than 2 days of computation.

Once we have recovered the internal state, we need to do the meet-in-the-middle approach described in Section 8.5.3 to recover the key. For each of the 2^{32} guesses, we need to compute 14 cipher shifts to build the set S_f . Compared to the complexity above, this is negligible. It does require a storage space of around 16 Gb though. Using a 2 GHz computer with enough internal memory, it takes about half an hour to construct and sort this table. Similarly, from the other side we need another 2^{36} cipher ticks to unroll the cipher. Since we only need the intersection of S_f and S_b , we do not actually store S_b , but only do a logarithmic search for each element of S_b in the table for S_f . This whole computation takes another half hour.

8.6 Conclusion

In this chapter we have described the ciphers used in the product families SecureMemory, CryptoMemory, and CryptoRF. We have shown weaknesses of these ciphers, most notably the fact that the three components of the cipher operate independently (knowing the keystream, in the case of CryptoMemory and CryptoRF) and that there is a strong correlation between the intermediate output of two of those components and the generated keystream. We have shown that an attacker can use these weaknesses and eavesdropped sessions to recover the secret key. For SecureMemory, the attack has a time complexity of 2^{39} cipher ticks; in practice it takes around 10 minutes on a single core 2 GHz laptop. For CryptoMemory and CryptoRF, the attack has a time complexity of 2^{52} cipher ticks; on a cluster of 50 2 GHz quad core machines, it takes about 5 days to execute the attack. We have implemented the full functionality of the chips in software. We have also implemented the full attacks and tested this on genuine traces.

Chapter 9

iClass and iClass Elite

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 card uses 64-bit keys to provide authenticity and data integrity. The cipher and key diversification algorithms used in iClass are proprietary and little information about them is publicly available. In this chapter we have reverse engineered all security mechanisms in the card including cipher, authentication protocol and also key diversification algorithms, which we publish in full detail. Furthermore, we have found six critical weaknesses that we exploit in two attacks, one against iClass Standard and one against iClass Elite (a.k.a., iClass High Security). In order to recover a secret card key, the first attack requires one authentication attempt with a legitimate reader and 2^{22} queries to a card. This attack has a computational complexity of 2^{40} MAC computations. The whole attack can be executed within a day on a single core running at 2 GHz. Remarkably, the second attack which is against iClass Elite is significantly faster. It directly recovers the system wide 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.

9.1 Introduction

iClass is an ISO/IEC 15693 [ISO00] 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 others¹. 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 SmartCenter 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

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

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 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 used for Authentication and Key Agreement (AKA). This master key is stored unprotected in the EEPROM memory of every iClass Standard reader. 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].

9.2 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⁺08, GvRVWS10, PN12, VGB12]. Experience has shown that the secrecy of proprietary ciphers does not contribute to their cryptographic strength, see Chapter 4. HID proposes iClass as a migration option for systems using Mifare Classic, boosting that iClass provides “improved security, performance and data integrity”². 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 published a technical report [KJL⁺13] describing an independent reverse engineering approach 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

²<http://www.hidglobal.com/products/readers/iclass/high-frequency-migration-readers>

that they can be executed within a day and require only wireless communication with a genuine iClass card/reader.

9.2.1 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. 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 COPACOBANA (a generic massively parallel FPGA-computer [KPP+06]). 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 faster alternative, it is possible to emulate a predefined card identity and use it to build a pre-computed table of ciphertexts for this particular identity using a Time-Memory Trade-Off (TMTO), see Chapter 3.2. 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 [GdKGV12]. This process is explained later in Section 9.5. 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 brute-force 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 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. These results previously appeared in abbreviated form as [[GdKGV11](#), [GdKGV12](#)].

9.2.2 Outline




This chapter is organized as follows. Section 9.3 starts with a description of the iClass architecture, the functionality of the card, the cryptographic algorithms. Then, Section 9.4 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 9.6 after the reverse engineering and description of the cipher in Section 9.5. Finally, Section 9.7 describes the key diversification in iClass Elite and presents an attack against this scheme.

9.3 iClass

An HID iClass card is in fact a pre-configured and re-branded PicoPass card manufactured by Inside Secure³. 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.

Table 9.1: Memory layout of an iClass card

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

	publicly readable
	write-only after authentication
	read-write after authentication

9.3.1 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 Table 9.1. 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 \mathcal{K}_1 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

³<http://www.insidesecond.com/eng/Products/Secure-Solutions/PicoPass>

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 9.7.1. 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 ($k1$ for application 1 or $k2$ 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 $k \leftarrow k \oplus p$.

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 9.6.2.

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 9.3.2. Cryptographic keys $k1$ and $k2$ can be seen as part of application 1 and 2, respectively. This means that in order to modify a key e.g., $k1$, the reader first needs to run a successful authentication with $k1$.

9.4 iClass standard

In this chapter we first reverse engineer the iClass Standard key diversification. Then, we describe its weaknesses in Section 9.4.3. Finally, we describe the first attack against iClass Standard in Section 9.4.4.

Our first approach for reverse engineering is in line with that of [GdKGM⁺08, LST⁺09, GvRVWS10] and consists of analyzing the update card key messages sent by an iClass compatible reader while we apply small modifications to 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 [GdKGV11]. We reverse engineered the Omnikey Secure 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 in order to intercept ISO/IEC 15693 [ISO00] frames. We have released the library, firmware and an implementation of *hash0* under the GNU General Public License and they are available at <http://www.proxmark.org>.

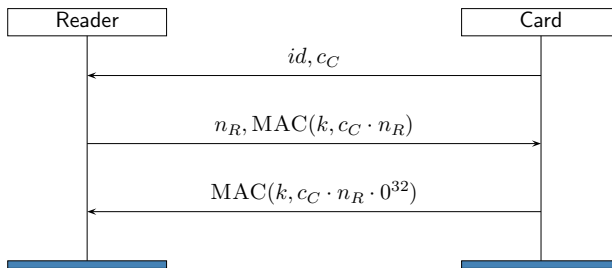


Figure 9.1: Authentication protocol

Later in Section 9.5, 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.

9.4.1 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 9.4.2. 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 .

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

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

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

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

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

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

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

The results are grouped by the position of the flipped bit m . Then, the OR and AND is computed of all the results in a group. These cumulative OR and AND values for 64 bits that were flipped on a few thousand random 64-bit bitstrings $c \in \mathcal{C}$ are presented in Figure 9.3 and 9.4. The output difference for flipping all possible bits is abbreviated as follows.

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

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 $hash0$ at 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 can even 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:

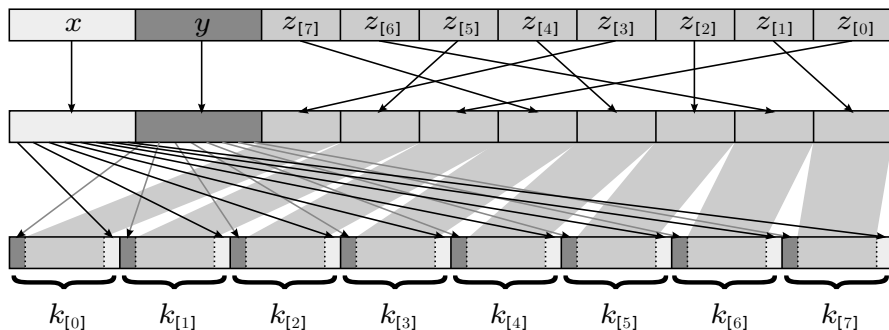
- generate a large set of random bitstrings $c \in \mathbb{F}_2^{64}$.
- for each c
 - calculate $id = \text{DES}_{\text{dec}}(c, \mathcal{K})$ and $id^m = \text{DES}_{\text{dec}}(\text{flip}(c, m), \mathcal{K})$ for $m = 0 \dots 63$
 - for each m authenticate with id , perform a key update, the reader 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 $c \in \mathcal{C}$. Experiments show that for this particular function, having more than 3000 samples does not produce any difference on the cumulative OR and AND tables. This amount of samples can be obtained within a couple of days.

Function input partitioning

Figure 9.3 shows 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 9.4. These differences reveal that the input c of hash0 is of the form $c = x \cdot y \cdot z_{[7]} \dots z_{[0]}$ 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 . We noticed that the first 16 bits of the input exhibit a different behavior than the rest and therefore decided to split the input into two parts. The structure of the mask in Figure 9.3 is computed with $x = y = 0^8$ and z a random bitstring. Whereas in Figure 9.4 we flip only bits of x and y . This leads to the following observations:

- $z_{[0]} \dots z_{[3]}$ affects $k_{[0]} \dots k_{[3]}$ and $z_{[4]} \dots z_{[7]}$ affects $k_{[4]} \dots k_{[7]}$.
- $z_{[0]} \dots z_{[3]}$ 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_{7-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 in Figure 9.4 shows that $k_{[i]_0}$ is exclusively affected by y for $i = 0 \dots 7$.
- x defines a permutation. The output is scrambled after flipping a single bit within x . The AND-mask in Figure 9.4 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 $0x7e$ (01111110 in binary representation) in Figure 9.3.

Figure 9.2: Schematic representation of the function $hash0$

The above observations suggest that we can recover different parts of the function independently. Figure 9.2 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 $z_{[i]} \dots z_{[i+5]}$ and affects $k_{[i+1]} \dots k_{[i+6]}$. Furthermore, it is observed that the leftmost bit of all output bytes $k_{[i+7]}$ and the permutation of $z_{[i]}$ to $k_{[i+1]} \dots k_{[i+6]}$ is determined by x . Finally, every input bit y_{7-i} is copied to output bit $k_{[i]0}$.

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 9.7.

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 this part of the function we keep $x = y = 0^8$ and let z vary over random bitstrings. According to Figure 9.3 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 9.3.

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

Note that $k_{[4]1} \dots k_{[4]6} = z_{[3]}$. 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 Table 9.3. 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

bit ↓	OR-mask of differences in output k	AND-mask of differences in output k
$z_{[0]}$ { 63	0x7e7e7e7e00000000	0x0400000000000000
62	0x7e7e7e7e00000000	0x0400000000000000
61	0x7a7e7e7e00000000	0x0800000000000000
60	0x727e7e7e00000000	0x1000000000000000
59	0x627e7e7e00000000	0x2000000000000000
58	0x427e7e7e00000000	0x4000000000000000
57	0x007e7e7e00000000	0x0000000000000000
$z_{[1]}$ {
52	0x007e7e7e00000000	0x0000000000000000
51	0x00007e7e00000000	0x0000000000000000
$z_{[2]}$ {
46	0x00007e7e00000000	0x0000000000000000
45	0x000007e7e0000000	0x0000000000000000
$z_{[3]}$ {
40	0x000007e7e0000000	0x0000000000000000
39	0x0000000027e7e7e	0x000000002000000
38	0x0000000047e7e7e	0x000000004000000
37	0x0000000087e7e7e	0x000000008000000
36	0x0000000107e7e7e	0x000000010000000
35	0x0000000207e7e7e	0x000000020000000
34	0x0000000407e7e7e	0x000000040000000
33	0x000000007e7e7e	0x0000000000000000
$z_{[5]}$ {
28	0x000000007e7e7e	0x0000000000000000
27	0x00000000007e7e	0x0000000000000000
$z_{[6]}$ {
22	0x00000000007e7e	0x0000000000000000
21	0x0000000000007e	0x0000000000000000
$z_{[7]}$ {
16	0x00000000000007e	0x0000000000000000

Figure 9.3: OR and AND-mask for flipping bits 16...63 of c

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 . We write z_7 when the bit flip is set to zero and \check{z}_7 when is set to one.

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

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

from which we deduce that

$$k_{[0]_1} \dots k_{[0]_6} = (z_7 \bmod 63) + 1. \quad (9.1)$$

The remaining $k_{[1]_1} \dots k_{[1]_6}$, $k_{[2]_1} \dots k_{[2]_6}$ and $k_{[3]_1} \dots k_{[3]_6}$ can be found in a similar

bit ↓	OR-mask of differences in output k	AND-mask of differences in output k	
y	15	0xfc00000000000000	0x8000000000000000
	14	0x00fc000000000000	0x0080000000000000
	13	0x0000fc0000000000	0x0000800000000000
	12	0x000000fc00000000	0x0000008000000000
	11	0x00000000fe000000	0x00000000fe000000
	10	0x0000000000fe0000	0x0000000000fe0000
	9	0x000000000000fe00	0x000000000000fe00
	8	0x00000000000000fe	0x00000000000000fe
x	7	0x7f7f7f7e7e7f7f7f	0x0101010000010101
	6	0x00007f7e7f000000	0x0000010001000000
	5	0x7f7e7e7e7f000000	0x0100000001000000
	4	0x7f7e7e7e7f0000	0x0100000000010000
	3	0x00007f7e7e7e7f00	0x0000010000000100
	2	0x7f7e7f7f7f7f00	0x0100010101010100
	1	0x7f7e7f7e7e7f00	0x0100010000010100
	0	0x7f7e7f7e7f7e00	0x0100010001000100

Figure 9.4: OR and AND-mask for flipping bits $0 \dots 15$ of c
 Table 9.3: Input-output relations for $z_{[7]} \leftrightarrow k_{[0]}$

$z_{[7]}$ 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

way by flipping bits in the input and closely looking at the input-output relations. For more details on the reverse engineering of this function see [GdKGV11]. The complete definition of the function is given in Section 9.4.2. Eventually, the modulo operations are separated from the subfunction *check* and defined in the main function *hash0*. Also, the definition in Section 9.4.2 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 9.4 clearly shows. Flipping bit y_{7-i} results in the complement of $k_{[i]_0}$ in the output, for $i = 0 \dots 7$. Note that no other input bit influences any least significant output bit of the output bytes $k_{[i]_0}$.

Furthermore, $k_{[i]_1} \dots k_{[i]_6}$ 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 $k_{[i]_7}$ 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 $y_{7-i} = 0$.

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 9.4, 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 9.4.2.

9.4.2 The function *hash0*

The following sequence of definitions precisely describe the recovered function *hash0*. The details of this construction are not necessary to understand the attacks presented in Section 9.6.5 and Section 9.7.3.

The function *hash0* first computes $x' = x \bmod 70$ which results in 70 possible permutations. Then for all z_i the modulus and additions are computed before calling the subfunction *check*.

Then, the subfunction *check* is called twice, first on input z'_0, \dots, z'_3 and then on input z'_4, \dots, z'_7 . The definition of the function *check* is as follows.

Definition 9.4.3. 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

$$\begin{aligned} ck(1, -1, z_{[0]} \dots z_{[3]}) &= z_{[0]} \dots z_{[3]} \\ ck(i, -1, z_{[0]} \dots z_{[3]}) &= ck(i-1, i-2, z_{[0]} \dots z_{[3]}) \\ ck(i, j, z_{[0]} \dots z_{[3]}) &= \\ &\begin{cases} ck(i, j-1, z_{[0]} \dots z_{[i]} \leftarrow j \dots z_{[3]}), & \text{if } z_{[i]} = z_{[j]}; \\ ck(i, j-1, z_{[0]} \dots z_{[3]}), & \text{otherwise.} \end{cases} \end{aligned}$$

Definition 9.4.4. 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

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

Definition 9.4.5. Define the bitstring $\pi \in (\mathbb{F}_2^8)^{35}$ in hexadecimal notation as

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

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 9.4.6. Let the function *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 $hash0(x, y, z_{[0]} \dots z_{[7]}) = k_{[0]} \dots k_{[7]}$ where

$$\begin{aligned} z'_{[i]} &= (z_{[i]} \bmod (63 - i)) + i & i = 0 \dots 3 \\ z'_{[i+4]} &= (z_{[i+4]} \bmod (64 - i)) + i & i = 0 \dots 3 \\ \hat{z} &= check(z') \\ p &= \begin{cases} \overline{\pi_{[x \bmod 35]}}, & \text{if } x_7 = 1; \\ \pi_{[x \bmod 35]}, & \text{otherwise.} \end{cases} \\ \tilde{z} &= permute(p, \hat{z}, 0, 4) \\ k_{[i]} &= \begin{cases} y_{(7-i)} \cdot \overline{\tilde{z}_{[i]}} \cdot p_{(7-i)} + 1, & \text{if } y_{(7-i)} = 1; \\ y_{(7-i)} \cdot \tilde{z}_{[i]} \cdot \overline{p_{(7-i)}}, & \text{otherwise.} \end{cases} \\ & & i = 0 \dots 7 \end{aligned}$$

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

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

9.4.3 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 9.4.4.

Omnikey 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_{COW} 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. The entropy of the master key $\mathcal{K}1$ is 56 bits. However, the diversified key k_{id} is a DES encryption of the identifiers id , which is a bijective mapping with an entropy of 64 bits. Contrarily, the function $hash0(k_{id})$ is not bijective, since it can produce the same output (collisions) for a different input values of k_{id} .

The modulo operations on x ($\frac{256}{70}$) and $z_{[0]}, \dots, z_{[7]}$ are responsible for collisions in the output. The modulo operation on x has the most influence to create a collision. Without looking at z , the operation on x already shows that the average number of pre-images is at least

$$\frac{256}{70} \approx 3.66$$

Furthermore, the function $hash0$ is not an One-Way Function (OWF), since it is possible to invert its operations. There are multiple pre-images because of the mentioned modulo operations, but inverting them is straightforward, see Section 9.4.3 for more details.

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 9.4.7.

Definition 9.4.7. *Let the inverted function $check^{-1}: (\mathbb{F}_2^6)^8 \rightarrow (\mathbb{F}_2^6)^8$ be defined as $check(z_{[0]} \dots z_{[7]})$ in Definition 9.4.3 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 9.4.8. 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 9.4.9. 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

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

$$x' \text{ is unique elem. in } \mathbb{F}_2^8 \text{ s.t. } \begin{cases} p = \overline{\pi_{[x' \pmod{35}]}} \Leftrightarrow x'_7 = 1 \\ p = \pi_{[x' \pmod{35}]} \Leftrightarrow x'_7 = 0 \end{cases}$$

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

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

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

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

$$p_i = \overline{k'_{[i]_7}} \quad i = 0 \dots 7$$

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

$$y_i = k_{[7-i]_0} \quad i = 0 \dots 7$$

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 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.

9.4.4 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 9.6.5 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 $\mathcal{K}1$ and $\mathcal{K}2$ that are stored in the Omnikey reader for the iClass application. The key $\mathcal{K}2$ 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 $\mathcal{K}1$ stored in software we were still able to verify it since we could dump the EEPROM of a reader where $\mathcal{K}1$ was stored, see Section 9.5.1. It would have been possible to recover $hash0$ from the EEPROM as well, although the prior knowledge about $hash0$ 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 Time-Memory Trade-Off (TMTO) that is constructed for a specific plaintext and runs through all possible encryptions of this plaintext. Note that the 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 9.5 did not only reveal the iClass security mechanisms, but also more weaknesses that are described in Section 9.6. We use some of these weaknesses to lower the requirements on the adversary and deploy a second attack on iClass Standard, when the adversary does not control the reader, in Section 9.4.4.

9.5 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 9.6 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 9.7.

9.5.1 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. 9.5), as many other embedded devices, rely on the popular PIC microcontroller (Fig. 9.5b) to perform their computations.

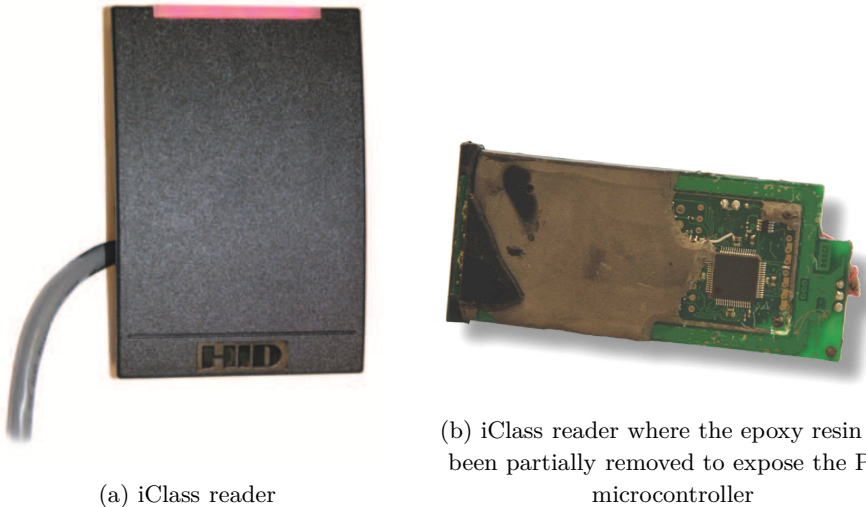


Figure 9.5: iClass readers

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 to

disassemble, debug and reverse engineer the algorithms.

9.5.2 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 9.6. 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 9.6 represent byte operations while dashed lines represent bit operations.

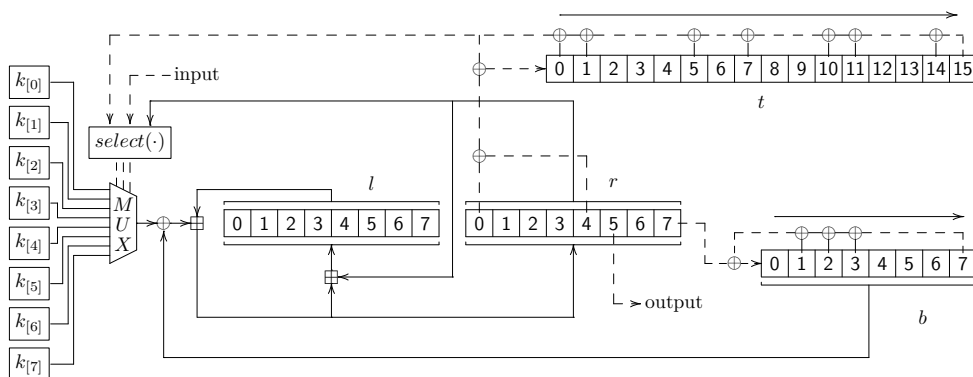


Figure 9.6: The iClass cipher

Definition 9.5.1 (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 $select(\cdot)$ 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 describes the cipher in detail; see also Figure 9.6.

Definition 9.5.2. *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 9.5.3. *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 9.5.4 (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

$$\begin{aligned} z_0 &= (r_0 \wedge r_2) \oplus (r_1 \wedge \overline{r_3}) \oplus (r_2 \vee r_4) \\ z_1 &= (r_0 \vee r_2) \oplus (r_5 \vee r_7) \oplus r_1 \oplus r_6 \oplus x \oplus y \\ z_2 &= (r_3 \wedge \overline{r_5}) \oplus (r_4 \wedge r_6) \oplus r_7 \oplus x \end{aligned}$$

Definition 9.5.5 (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. Define the successor cipher state $s' = \langle l', r', t', b' \rangle$ as*

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

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

$$\begin{aligned} suc(k, s, \epsilon) &= s \\ suc(k, s, x_0 \dots x_n) &= suc(k, suc(k, s, x_0 \dots x_{n-1}), x_n). \end{aligned}$$

Definition 9.5.6 (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*

$$\begin{aligned} output(k, s, \epsilon) &= \epsilon \\ output(k, s, x_0 \dots x_n) &= output(s) \cdot output(k, s', x_1 \dots x_n) \\ &\text{where } s' = suc(k, s, x_0). \end{aligned}$$

Definition 9.5.7 (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

$$\begin{aligned} t &\leftarrow 0\text{x}E012 & l &\leftarrow (k_{[0]} \oplus 0\text{x}4C) \boxplus 0\text{x}EC \\ b &\leftarrow 0\text{x}4C & r &\leftarrow (k_{[0]} \oplus 0\text{x}4C) \boxplus 0\text{x}21 \end{aligned}$$

Definition 9.5.8 (MAC function). Define the function $\text{MAC}: (\mathbb{F}_2^8)^8 \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^{32}$ as

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

9.6 Weakness in iClass

This section describes weaknesses in the design and implementation of iClass. We present four weaknesses that are later exploited in Section 9.6.5 to mount an attack that recovers the systems master key.

9.6.1 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 9.6.1. Let β 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_β .

This is because the output of the cipher is determined by the three rightmost (least significant) bits of register r , the three rightmost bits of l and the three rightmost bits of the selected key byte XOR b . Furthermore, only the rightmost bit of r influences register b . 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. The following table shows the corresponding MAC value for each possible value of β .

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 9.6.2 and 9.6.3 results in a vulnerability exploited in Section 9.6.5.

9.6.2 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 9.3.1). 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 (TMTO), see Section 3.2. Moreover, in case the cipher has some weak keys like the ones described in Section 9.6.1, it allows an adversary to force the usage of one of these keys.

9.6.3 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 .

9.6.4 Lower card key entropy

After careful inspection of the function `hash0` (Section 9.4.3) 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 9.4.3) that is caused by `hash0`.

9.6.5 Key recovery attack on iClass Standard

This section shows how the weaknesses described in Section 9.6 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 9.4.3 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 9.6.3, 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 default key $k2$ [HID06] 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 9.6.2, 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 property described in Section 9.6.1, after each key update A runs 8 authentication attempts, one for each possible value of β , using the MAC values shown in Table 9.4. 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 β as stated in Proposition 9.6.1. Knowing β , 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 $k1_{[7]}$ 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.

Table 9.4: Corresponding MAC for each value of β

β	$C_\beta = \text{MAC}(k, c_C \cdot n_R)$
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

This attack can be further optimized. The restriction on the last bit of each byte imposed by `hash0`, described at the end of Section 9.6.4, 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 9.6.4, recovering the master key is just as hard as breaking single DES.

9.7 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 9.7.1 we describe the weaknesses of this scheme in Section 9.7.2. Finally, the third and fastest attack of this chapter, concerning iClass Elite, is given in Section 9.7.3.

First, recall the key diversification of the iClass Standard system that we described in Section 9.4.2. 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., $k = hash0(\text{DES}_{\text{enc}}(\mathcal{K}, id))$. 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 9.2. *The DES implementation used in iClass is non-compliant with the NIST standard [FIP77]. Concretely, iClass deviates from the standard in the way of representing keys. According to the standard a DES key is of the form $\langle k_0 \dots k_6 p_0, \dots, k_{47} \dots k_{55} p_7 \rangle$ 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 $\langle k_0 \dots k_{55} p_0 \dots p_7 \rangle$.*

9.7.1 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 9.4.2 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 9.7.2 describes two weaknesses that are later exploited in Section 9.7.3.

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

Definition 9.7.1 (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 9.7.2. *Let the second hash 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

$$\begin{aligned} k_{[i]} &= k'_{[i]} \bmod 128, & i &= 0 \dots 7 \\ k'_{[0]} &= id_{[0]} \oplus \dots \oplus id_{[7]} & k'_{[4]} &= \overline{rr(id_{[4]} \boxplus k'_{[2]})} + 1 \\ k'_{[1]} &= id_{[0]} \boxplus \dots \boxplus id_{[7]} & k'_{[5]} &= \overline{rl(id_{[5]} \boxplus k'_{[3]})} + 1 \\ k'_{[2]} &= rr(swap(id_{[2]} \boxplus k'_{[1]})) & k'_{[6]} &= rr(id_{[6]} \boxplus (k'_{[4]} \oplus \mathcal{3C})) \\ k'_{[3]} &= rl(swap(id_{[3]} \boxplus k'_{[0]})) & k'_{[7]} &= rl(id_{[7]} \boxplus (k'_{[5]} \oplus \mathcal{C3})) \end{aligned}$$

Definition 9.7.3. *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 9.7.4. *Let the third hash 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 9.7.5. Let $h \in (\mathbb{F}_2^8)^{128}$. Let $k^{sel} \in (\mathbb{F}_2^8)^8$ be the Selected key defined as

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

The last step to compute the diversified card key is just like in iClass

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

9.7.2 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 9.7.3 to mount an attack against iClass Elite that recovers the custom master key.

Redundant key diversification on iClass Elite

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

Figure 9.7: Chosen card identities

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

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

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

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

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 9.7.1, 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.

9.7.3 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 9.7.2, the adversary builds a list of chosen card identities like the ones shown in Figure 9.7. This Figure 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 9.7.2, the adversary is able to recover the custom master key \mathcal{K}^{cus} with a total computational complexity of 2^{25} DES encryptions.

9.8 Conclusion

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-3. 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.

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⁺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.

A practical counter-measure until migration would be to stop using the iClass Elite diversification scheme and only use iClass Standard with customized master keys for all applications. However, such measure should only be considered as a temporary mitigation and not as a definite solution as the (more expensive) attack on iClass Standard still applies.

In line with the principles of responsible disclosure, we have notified the manu-

facturer HID Global and informed them of our findings back in November 2011. By the time of writing this chapter, HID has extended their product line with support for AES-enabled Mifare DESFire EV1 cards⁴

9.9 Acknowledgments

We are thankful to Milosch Meriac for his kind support while bypassing the PIC's read protection mechanisms which enabled the firmware recovery of the iClass reader. The authors would also like to thank the anonymous reviewers for their outstanding work. Their constructive and valuable comments helped us to substantially improve the quality of this study.

⁴www.hidglobal.com/iclass-hf-migration-reader-family-datasheet

10.1 Disclaimer

Due to an interim injunction¹, ordered by the High Court of London on Tuesday the 25th of June, 2013, the authors are restrained from publishing the technical contents of the scientific article *Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer* [VGE13] until further notice.

10.2 Historical claim

Figure 10.1 contains the cryptographic hash (SHA-512) of the original final paper [VGE13] which was scheduled to appear in the proceedings of the 22nd USENIX Security Symposium, Washington DC, August 2013.

```
9d05ba88740499eecea3d8609174b444
43683da139f78b783666954ccc605da8
4601888134bf0c23ba46fb4a88c056bf
bbb629e1ddffcf60fa91880b4d5b4aca
```

Figure 10.1: Cryptographic hash (SHA-512) of the original paper [VGE13]

¹<http://www.bailii.org/ew/cases/EWHC/Ch/2013/1832.html>

This page is intentionally left blank

Part III

Back matter

Conclusion

In computer security research it is common practice to spend a significant part of our time analyzing existing cryptosystems. In our field we cannot prove that a cryptographic technique or mechanism is secure. The best way to gain confidence that such a technique is secure is by independent peer-reviews and thorough public scrutiny. If the scientific community, industry and general public do not manage to break a cryptosystem after some time, then confidence is gained in such a technique or mechanism. In this way the security research progresses to develop the best possible protection mechanisms in the field of computer security.

Cryptosystems which are kept secret lack public scrutiny and carry the risk of being deployed in many (millions of) devices before potential cryptographic weaknesses come to light. Once a cryptosystem is proven to be insecure, it tends to quickly lose the trust of its users. It is difficult and costly to migrate widely deployed systems to more secure solutions in a reasonably fast manner. Public exposure on the other hand, allows the biggest stake-holder, the general public, to independently assess the protection that is offered by the cryptosystem.

History shows that, despite secrecy of the algorithm, most vulnerabilities of a cryptosystem eventually get exposed, privately or publicly. Furthermore, non-public exchange of cryptographic design weaknesses in a secret algorithm enables adversaries to abuse the system and potentially carry out malicious activities without being detected.

Nevertheless, there can be reasons to specifically choose to design a proprietary cryptographic algorithm. For instance, when the developer intends to acquire a patent on the design. In such a case the proprietary algorithm becomes publicly available at a later stage (published in a patent). Publishing of the patent with the algorithm still allows public and scientific scrutiny. Contrarily, a reason to design a secret algorithm is to avoid compatible or counterfeit products from competitors. However, such an approach might have a different design goal as primary target. For instance, to create a vendor-lock situation, in stead of delivering secure cryptographic algorithms and protocols.

Finally, there are sincere arguments to advocate the use of secret algorithms in a strictly controlled environment. For instance, institutes like government agencies and military establishments have the resources, knowledge and ability to extend secure cryptographic designs without directly compromising its original security strength. When physical access to a system that implements a secret design is strictly controlled, there are at least two factors that can increase its security properties. First, the secrecy adds an extra obstacle for an adversary to overcome before in-depth cryptographic analysis can be performed. Secondly, the altered parameters, additional computations and specific use might increase the overall cryptographic strength. With no limitations on resources, a different set of cryptographic operations might be a better fit than the general-purpose standardized cryptographic algorithms.

Apart from this exception, this study reinforces the point that has been made many times: secrecy of an algorithm does not add measurable security to a publicly deployed cryptosystem. Especially in the long term, it is wise to assume that the inner workings eventually get exposed to adversaries. Such metrics have a serious impact when the security of the cryptosystem mostly depends on the secrecy of the algorithm (security by obscurity). It is difficult to design a strong cryptographic algorithm and, without proper independent peer-reviews, there is a great risk that an unintended design mistake compromises the security of the cipher.

Deployment of badly designed cryptographic algorithms is undesirable, regardless of their secrecy. This dissertation shows that weak and vulnerable ciphers lead to insecure products. The ciphers addressed in this study are derived from obsolete and insecure designs. Surprisingly, most of these ciphers have been massively deployed since the nineties, while a decade before that the literature already contained a large collection of studies that prove the insecurity of similar cipher designs.

With the introduction of cheaper and more powerful hardware it is much easier to implement secure cryptographic algorithms that require a significant amount of computation. Although the ciphers addressed in this thesis are embedded into products which are still widely deployed and used by the general public, the industry is starting to offer compatible replacements which utilize openly designed and community-reviewed cryptographic algorithms. The academic community supports this trend and works closely together with several international institutions, governments and industries to achieve better and more secure cryptographic algorithms.

A lesson that can be learned from the examples illustrated in this dissertation is the security impact of combined vulnerabilities. The security of a cryptosystem depends on the system as a whole. Implementation weaknesses can be utilized by an adversary to exploit a cryptosystem more efficiently. Combining such vulnerabilities can be a serious threat, even if a cryptographic algorithm itself is not particularly weak. For instance, when a cipher specification states that an Initialization Vector (IV) should be initialized with a random bitstring, then the cryptosystem should ensure this at all times. The National Institute of Standards and Technology (NIST) introduced a statistical test suite [RSN⁺01] that can be used to automatically estimate the random-

ness of an IV. Additionally, the implementation of the component and interface to the rest of the cryptosystem should be considered. Any external influence on the IV value must be seen as a potential security compromise of the cryptosystem. In order to find such problems it is good practice to incorporate some form of formal verification in the development and implementation of security products, see for instance [FL12]. Furthermore, systematic and automated model checking techniques proposed in [Tre08] can help to detect and avoid implementation weaknesses like privilege escalations. Alternatively, formalizing the whole design in a theorem prover [Bla01, JWS11] may reveal additional weaknesses.

It remains an open question whether a comprehensive automated framework could detect the impact of combined security vulnerabilities in a cryptosystem. An interesting study is to learn the capabilities, limitations and practicality of such a framework. A quick start could be made by using the in-depth overview of vulnerabilities presented in this dissertation. Automated techniques and methodologies that generalize these weaknesses in a framework which tests for combined vulnerabilities in a cryptosystem would be extremely useful for the development of future designs.

Index

- Access condition, 31
- Advanced Encryption Standard, 23
- Adversary, 17
- Algebraic attacks, 60
- Asymmetric cryptography, 24
- Attack complexity, 21
- Authentication attempt, 33
- Authentication protocol, 29
- Authenticity, 3
- Authorization, 31
- Authorization model, 32
- Availability, 2, 3

- Balanced output, 54
- Black box, 35
- Blocking attack, 48
- Brute-force attack, 21

- Candidate disqualification, 57
- Candidate elimination, 57
- Chaining of encryption, 27
- Challenge-response, 29
- Challenge-response pair, 33
- Cipher, 17
- Cipher complexity, 21
- Cipher components, 33
- Cipher specification, 20
- Ciphertext, 17
- Communication blocking attack, 48
- Communication channel, 2
- Communication session, 31
- Complexity, 21
- Computational complexity, 21
- Computations, 22
- Confidential communication channel, 2
- Confidentiality, 2
- Credentials, 32
- Cryptanalysis, 49
- Cryptanalytic techniques, 49
- Cryptographic algorithm, 20
- Cryptographic challenge-response, 29
- Cryptographic operation, 2
- Cryptographic session, 27
- Cryptography, 4, 17
- Cryptosystem, 20
- Custom cryptography, 86

- Data Encryption Standard, 23
- Data integrity, 2
- Decryption, 17
- Differential attack, 58
- Diversified keys, 31
- Divide-and-conquer, 51

- Eavesdropping, 42
- Encryption, 2, 17
- Encryption layer, 37
- Encryption oracle, 30, 35
- Encryption state, 47
- Entity, 3, 17
- Entity authentication, 2, 3
- Entropy, 21
- Error prone, 25
- Exhaustive search, 21

- Filter function, 33

- Genuine entity, 17
- Initialization vector, 30
- Injection attack, 47
- Integrity checks, 36
- intermediate state, 63
- Internal cipher state, 25
- Internal state, 25
- Invasive attacks, 65
- Invertible function, 33
- Kerckhoffs's principle, 4, 22
- Key space, 21
- Keystream, 25
- Legitimate party, 17
- Linear boolean function, 61
- Linear cryptanalysis, 61
- Low cohesion, 33
- Mafia fraud, 43
- Malleability attack, 50
- Mathematical computation, 2
- Meet-in-the-middle attack, 63
- Message Authentication, 2
- Message authentication, 3
- Message tampering, 2
- Micro-controller, 1
- Mutual authentication, 30
- Non-invasive attacks, 65
- Non-linear filter function, 33
- Non-linear function, 33
- Non-linearity, 20
- Non-repudiation, 2, 3
- One-time pad, 24
- Parity bit, 36
- Passive eavesdropping, 42
- Perfect cipher, 21
- Perfect Secrecy, 25
- Plaintext, 17
- Predecessor state, 35
- Private key, 24
- Privilege escalation, 35
- Proprietary cryptography, 4
- Public key, 24
- Reflection attack, 46
- Relay attack, 42
- Relay station, 43
- Replay attack, 45
- Secret key, 24
- Secure channel, 2
- Secure Hash Algorithm, 23
- Secure Hash Standard, 23
- Semi-invasive attacks, 65
- Session, 27
- Side-channel attacks, 65
- Single authentication, 30
- State machine, 36
- State transitions, 36
- Stream cipher, 33
- Substitution, 18
- Successor function, 27
- Successor state, 35
- Symmetric authentication protocol, 46
- Symmetric cryptography, 24
- Tampering, 2
- Third party, 17
- Time-memory trade off, 78
- Trace, 42
- Transform, 17
- Transmission tampering, 50
- Unilateral authentication, 30

Acronyms

3DES	Triple DES	64
3GPP	3rd Generation Partnership Project	76
ADC	Analog-to-Digital Converter	88
AES	Advanced Encryption Standard	23
ASK	Amplitude-Shift Keying	84
ASIC	Application-Specific Integrated Circuit	89
AKA	Authentication and Key Agreement	47
BPLM	Binary Pulse Length Modulation	84
BP	Bi-Phase	84
BTS	Base Transceiver Station	74
CBC	Cipher Block Chaining	28
CCD	Charge-Coupled Device	67
CCM	Counter with CBC-MAC	28
CD	Compact Disc	79
CERT	Computer Emergency Response Team	8
CMAC	Cipher-based Message Authentication Code	29
CRC	Cyclic Redundancy Check	36
CSS	Content Scramble System	79
CTR	Counter	28
DAC	Digital-to-Analog Converter	88
DECT	Digital Enhanced Cordless Telecommunications	76
DES	Data Encryption Standard	23

DoD	Department of Defence	5
DOS	Denial-of-Service	36
DPA	Differential Power Analysis	66
DRM	Digital Rights Management	79
DSAA	DECT Standard Authentication Algorithm	77
DSC	DECT Standard Cipher	76
DST	Digital Signature Transponder	72
DSP	Digital Signal Processing	88
DVD	Digital Versatile Disc	78
EFF	Electronic Frontier Foundation	80
EM	Electromagnetic	65
EPIC	Electronic Privacy Information Center	80
ETSI	European Telecommunications Standards Institute	73
FIB	Focused Ion Beam	66
FPGA	Field Programmable Gate Array	88
FSK	Frequency-Shift Keying	84
GCM	Galois Counter Mode	28
GEO	Geostationary Earth Orbit	77
GMR	GEO Mobile Radio Interface	77
GNU	GNU's Not Unix	88
GPL	General Public License	89
GSM	Global System for Mobile Communications	37
HMAC	Hashed Message Authentication Code	29
HF	High Frequency	84
IC	Integrated Circuit	66
ICAO	International Civil Aviation Organization	85
ICT	Information and Communication Technology	8
IEC	International Electrotechnical Commission	84
IETF	Internet Engineering Task Force	8
IPsec	Internet Protocol Security	28
ISO	International Organization for Standardization	84
IV	Initialization Vector	30
LF	Low Frequency	84
MAC	Message Authentication Code	29

NBS	National Bureau of Standards.....	23
NCSC	National Cyber Security Centre.....	8
NDA	Non-Disclosure Agreement.....	77
NFC	Near Field Communication.....	44
NFCIP	NFC Interface and Protocol.....	85
NIST	National Institute of Standards and Technology.....	23
NRZ	Non-Return-to-Zero.....	84
NSA	National Security Agency.....	80
OOK	On-Off Keying.....	84
OSI	Open Systems Interconnection.....	85
OTP	One-time Pad.....	24
OWF	One-Way Function.....	28
PKES	Passive Keyless Entry and Start.....	44
QPLM	Quad Pulse Length Modulation.....	84
RC4	Rivest Cipher 4.....	81
RSA	Rivest, Shamir and Adleman.....	81
RF	Radio Frequency.....	42
RFID	Radio Frequency Identification.....	10
RKE	Remote Keyless Entry.....	72
OECD	Organization for Economic Co-operation and Development.....	75
PC	Personal Computer.....	82
PRNG	Pseudo Random Number Generator.....	25
PSK	Phase-shift keying.....	84
RFC	Request for Comments.....	8
SDR	Software Defined Radio.....	88
SEI	Software Engineering Institute.....	8
SHA-3	Secure Hash Algorithm-3.....	23
SHS	Secure Hash Standard.....	23
SIG	Special Interest Group.....	79
SIM	Subscriber Identity Module.....	74
SPA	Simple Power Analysis.....	66
SSH	Secure Shell.....	28
TIA	Telecommunications Industry Association.....	73
TLS	Transport Layer Security.....	28

TMTO	Time-Memory Trade-Off.....	49
UICC	Universal Integrated Circuit Card.....	85
UID	Unique Identifier	83
UHF	Ultra-High Frequency.....	124
US	United States.....	5
USB	Universal Serial Bus	88
US	United States.....	5
USD	United States Dollar.....	129
USRP	Universal Software Radio Peripheral	88
VPN	Virtual Private Network	81
WEP	Wired Equivalent Privacy.....	81
WPA	Wi-Fi Protected Access.....	81
WPA2	Wi-Fi Protected Access II	28
XOR	exclusive-or.....	14

List of Figures

1.1	General categorization of cryptography and study focus	9
2.1	Caesar substitution using a rotation of three positions	18
2.2	Substitution example	18
2.3	Scytale transposition using a split position of seven characters	19
2.4	Modified Caesar substitution using a random permutation	21
2.5	Enigma machine	26
2.6	Typical non-linear stream cipher system	26
2.7	Two-pass single authentication protocol	30
2.8	Two-pass mutual authentication protocol	31
2.9	Three-pass mutual authentication protocol	32
3.1	Replay attack on a three-pass mutual authentication protocol	43
3.2	Relay attack on an access control system that uses an NFC smart card	44
3.3	Replay attack on a two-pass mutual authentication protocol	45
3.4	Demonstration of a reflection attack	46
3.5	Injection attack on a protocol without proper session integrity checks	47
3.6	Blocking attack on the second part of the communication	48
3.7	Malleability attack alters the value of a money transfer	50
3.8	Divide-and-conquer cipher	51
3.9	Computation of the first three encryption	52
3.10	Divide-and-conquer attack by dividing odd and even keystream bits	52
3.11	Stream cipher with correlation weakness, initialized by secret key k	53
3.12	Boolean input-output table that corresponds to Definition 3.2.1	54
3.13	Non-linear stream cipher, initialized by secret key k	56
3.14	Boolean input-output table that corresponds to Definition 3.2.2	56

3.15	First five evaluations of $f(\cdot)$	57
3.16	Non-linear stream cipher initialized with key XOR nonce	59
3.17	A linear stream cipher, initialized by secret key k	61
3.18	Evaluation of $ks_0ks_1\dots ks_7 = 10110101$ leads to the equalities of (d)	62
3.19	cryptosystem which is vulnerable to a meet-in-the-middle attack	63
4.1	Logic obfuscation [PN12]	70
4.2	Weak PRNG implementation allowing only $2^{31} - 1$ states [WMT+13]	70
4.3	Obscurity function [SDK+13]	71
4.4	DST cipher [BGS+05]	72
4.5	KeeLoq encryption [CBW08]	72
4.6	ORYX cipher [WSD+99]	73
4.7	COMP128 [RRST02]	74
4.8	A5/1 cipher [BB06]	74
4.9	A5/2 cipher [BBK03]	75
4.10	KASUMI [DKS10]	76
4.11	DSAA init [MOTW09]	76
4.12	DSC cipher [Tew12]	77
4.13	GMR-1 cipher [DHW+12]	77
4.14	GMR-2 cipher [DHW+12]	78
4.15	CCS decryption [Kes00]	79
4.16	E0 cipher [DCJP01]	79
4.17	Skipjack cipher [BBS99]	80
4.18	RC4 cipher	81
4.19	WEP encryption protocol [C+07]	82
5.1	RFID chip	83
6.1	Memory layout of the Mifare Classic	98
6.2	Authentication trace	99
6.3	Authentication protocol	101
6.4	Initialization Diagram.	101
6.5	Nearly equal LFSR states	102
6.6	First bit of encrypted reader nonce	103
6.7	Structure of the CRYPTO1 stream cipher	104
6.8	Subsequences \bar{s} and \bar{t}	108
6.9	Trace of a failed authentication attempt	110
7.1	Car keys with a <i>Hitag2</i> transponder/chip	124
7.2	Immobilizer unit around the ignition barrel	124

7.3	Keyless hybrid transponder and engine start/stop button	124
7.4	Experimental setup for eavesdropping	129
7.5	Reader modulation of a <i>read</i> command	129
7.6	Communication from transponder to reader	129
7.7	Message flow for reading memory block 0	132
7.8	Structure of the <i>Hitag2</i> stream cipher, based on [Wie07]	133
7.9	<i>Hitag2</i> authentication protocol	133
7.10	Read <i>id</i> without redundancy messages	136
7.11	Read <i>id</i> using 6 redundancy messages	136
7.12	Starting the ignition of a car with the Proxmark	142
7.13	Immobilizer authentication protocol using AES	145
8.1	Logical memory structure	152
8.2	The schematic of the SecureMemory and CryptoMemory cipher	153
8.3	Authentication protocol	155
9.1	Authentication protocol	176
9.2	Schematic representation of the function <i>hash0</i>	179
9.3	OR and AND-mask for flipping bits 16...63 of <i>c</i>	180
9.4	OR and AND-mask for flipping bits 0...15 of <i>c</i>	181
9.5	iClass readers	188
9.6	The iClass cipher	189
9.7	Chosen card identities	196
10.1	Cryptographic hash (SHA-512) of the original paper [VGE13]	201

Curriculum Vitae

Roel Verdult was born on the 20th of October 1982 in Zevenaar, The Netherlands. In 2004 he obtained his BSc from the Faculty of Engineering, at the HAN University of Applied Sciences, The Netherlands. He continued his master's degree at the Radboud University Nijmegen and followed the security research track of the computer science department. In 2008 he obtained his MSc degree from the Radboud University Nijmegen, The Netherlands. His thesis, entitled "Security Analysis of RFID Tags", has been awarded with the highest possible grade.

In July 2008, he joined the Digital Security group at the Faculty of Science of the Radboud University Nijmegen as an assistant researcher. In 2011 he became a PhD student under the supervision of prof. dr. Bart Jacobs and dr. Lejla Batina. His research has been funded by a joint PhD grant between the universities Radboud University Nijmegen, The Netherlands and KU Leuven, Belgium.

His research covers a variety of security topics which include (but are not limited to) the electronic passports, contactless smartcards, Radio Frequency Identification (RFID), Near Field Communication (NFC), secure storage, authentication protocols and other types of transmission security. The relevance of his research is recognized by the significant awards received from national and international institutions:

- **International Hermesdorf Prize**

Radboud University Nijmegen award for special attention in the media during 2013, January 2014

- **Best Paper Award**

USENIX Workshop on Offensive Technologies 2011, Augustus 2011

- **Outstanding Paper Award**

IEEE Symposium on Security and Privacy 2009, May 2009

- **Most Appealing Master Thesis**

Aia Software Master Thesis Award, January 2009

- **National student of the year**
Dutch award was presented by LSVB, ISO and ScienceGuide, November 2008
- **Information Security Award**
Joop Bautz nomination, October 2008
- **NWO I/O Award**
Netherlands Organisation for Scientific Research (NWO), September 2008
- **Gratuity for the MIFARE Classic research**
Radboud University Nijmegen, September 2008

Bibliography

- [ABP⁺13] Nadhem J AlFardan, Daniel J Bernstein, Kenneth G Paterson, Bertram Poettering, and JC Schuldt. On the security of RC4 in TLS. In *22nd USENIX Security Symposium (USENIX Security 2013)*. USENIX Association, 2013.
- [ABV12] Gergely Alpár, Lejla Batina, and Roel Verdult. Using NFC phones for proving credentials. In *16th Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance (MM&DFT 2012)*, volume 7201 of *Lecture Notes in Computer Science*, pages 317–330. Springer-Verlag, 2012.
- [AC09] Martin Albrecht and Carlos Cid. Algebraic techniques in differential cryptanalysis. In *Fast Software Encryption*, pages 193–208. Springer-Verlag, 2009.
- [ACG89] Mikhail J Atallah, Richard Cole, and Michael T Goodrich. Cascading divide-and-conquer: A technique for designing parallel algorithms. *SIAM Journal on Computing*, 18(3):499–532, 1989.
- [AH88] Hamid Reza Amirazizi and Martin E Hellman. Time-memory-processor trade-offs. *IEEE Transactions on Information Theory*, 34(3):505–512, 1988.
- [AJO08] Gildas Avoine, Pascal Junod, and Philippe Oechslin. Characterization and improvement of time-memory trade-off based on perfect tables. *ACM Transactions on Information and System Security (TISSEC 2008)*, 11(4):1–22, 2008.
- [AK96] Ross J. Anderson and Markus G. Kuhn. Tamper resistance - a cautionary note. In *2nd USENIX Workshop on Electronic Commerce*, pages 1–11. USENIX Association, 1996.

- [AK03] Frederik Armknecht and Matthias Krause. Algebraic attacks on combiners with memory. In *23rd International Cryptology Conference, Advances in Cryptology (CRYPTO 2003)*, volume 2729 of *Lecture Notes in Computer Science*, pages 162–175. Springer-Verlag, 2003.
- [AM97] Ross Anderson and Charalampos Maniavas. Chameleon - a new kind of stream cipher. In *4th International Workshop on Fast Software Encryption (FSE 1997)*, volume 1267 of *Lecture Notes in Computer Science*, pages 107–113. Springer-Verlag, 1997.
- [And91] Ross J Anderson. Tree functions and cipher systems. *Cryptologia*, 15(3):194–202, 1991.
- [And95] Ross Anderson. Searching for the optimum correlation attack. In *2nd International Workshop on Fast Software Encryption (FSE 1994)*, volume 1008 of *Lecture Notes in Computer Science*, pages 137–143. Springer-Verlag, 1995.
- [And03] Ross Anderson. Cryptography and competition policy: issues with ‘trusted computing’. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 3–10. ACM, 2003.
- [And10] Ross J. Anderson. *Security Engineering: A guide to building dependable distributed systems*. Wiley, 2010.
- [Arm04] Frederik Armknecht. Improving fast algebraic attacks. In *11th International Workshop on Fast Software Encryption (FSE 2004)*, volume 3017 of *Lecture Notes in Computer Science*, pages 65–82. Springer-Verlag, 2004.
- [AS09] Kazumaro Aoki and Yu Sasaki. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In *29th International Cryptology Conference, Advances in Cryptology (CRYPTO 2009)*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer-Verlag, 2009.
- [AT06] Standard read/write crypto identificastion IC - e5561. Product Datasheet, September 2006. Atmel Corporation.
- [AT07] CryptoMemory specification. Product Datasheet, April 2007. Atmel Corporation.
- [AT09] CryptoRF specification, AT88SCxxxxCRF. Product Datasheet, March 2009. Atmel Corporation.
- [AT11] Embedded avr microcontroller including rf transmitter and immobilizer lf functionality for remote keyless entry - ATA5795C. Product Datasheet, November 2011. Atmel Corporation.

- [Bab95] Steve Babbage. A space/time tradeoff in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection*, volume 408 of *Conference Publications*, pages 161–166. IEEE Computer Society, 1995.
- [BB05] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
- [BB06] Elad Barkan and Eli Biham. Conditional estimators: An effective attack on A5/1. In *12th International Workshop on Selected Areas in Cryptography (SAC 2005)*, volume 3897 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2006.
- [BBD⁺99] Eli Biham, Alex Biryukov, Orr Dunkelman, Eran Richardson, and Adi Shamir. Initial observations on Skipjack: Cryptanalysis of Skipjack-3XOR. In *5th International Workshop on Selected Areas in Cryptography (SAC 1998)*, volume 1556 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, 1999.
- [BBD06] Alexander Becher, Zinaida Benenson, and Maximillian Dornseif. Tampering with notes: Real-world physical attacks on wireless sensor networks. In *3rd International Conference on Security in Pervasive Computing (SPC 2006)*, volume 3934 of *Lecture Notes in Computer Science*, pages 104–118. Springer-Verlag, 2006.
- [BBF83] R Kenneth Bauer, Thomas A. Berson, and Richard J. Feiertag. A key distribution protocol using event markers. *ACM Transactions on Computer Systems (TOCS)*, 1(3):249–255, 1983.
- [BBK03] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In *23rd International Cryptology Conference, Advances in Cryptology (CRYPTO 2003)*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer-Verlag, 2003.
- [BBK08] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. *Journal of Cryptology*, 21(3):392–429, 2008.
- [BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In *18th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1999)*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer-Verlag, 1999.

- [BBS06] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *26th International Cryptology Conference, Advances in Cryptology (CRYPTO 2006)*, volume 4117 of *Lecture Notes in Computer Science*, pages 1–21. Springer-Verlag, 2006.
- [BC94] Stefan Brands and David Chaum. Distance-bounding protocols. In *12th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1993)*, volume 765 of *Lecture Notes in Computer Science*, pages 344–359. Springer-Verlag, 1994.
- [BC08] Eli Biham and Yaniv Carmeli. Efficient reconstruction of RC4 keys from internal states. In *15th International Workshop on Fast Software Encryption (FSE 2008)*, volume 5086 of *Lecture Notes in Computer Science*, pages 270–288. Springer-Verlag, 2008.
- [BCM08] Jean Pierre Benhammou, Vincent C. Colnot, and David J. Moore. Secure memory device for smart cards. US Patent 7395435 B2, July 2008.
- [BCM12] David Basin, Cas Cremers, and Simon Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. In *1st International Conference on Principles of Security and Trust*, pages 129–148. Springer-Verlag, 2012.
- [BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer-Verlag, 2004.
- [BD00] Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In *1st International Conference on Cryptology in India, Progress in Cryptology (INDOCRYPT 2000)*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer-Verlag, 2000.
- [BDK05] Eli Biham, Orr Dunkelman, and Nathan Keller. A related-key rectangle attack on the full KASUMI. In *11th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2005)*, volume 3788 of *Lecture Notes in Computer Science*, pages 443–461. Springer-Verlag, 2005.
- [BdKGP⁺12] Arjan Blom, Gerhard de Koning Gans, Erik Poll, Joeri de Ruiter, and Roel 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*, pages 1–16. Springer-Verlag, 2012.

- [BDL97] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. 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 37–51. Springer-Verlag, 1997.
- [BE02] Mark Blunden and Adrian Escott. Related key attacks on reduced round KASUMI. In *9th International Workshop on Fast Software Encryption (FSE 2002)*, volume 2365 of *Lecture Notes in Computer Science*, pages 277–285. Springer-Verlag, 2002.
- [Ben80] Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.
- [BER07] Andrey Bogdanov, Thomas Eisenbarth, and Andy Rupp. A hardware-assisted realtime attack on A5/2 without precomputations. In *9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007)*, volume 4727 of *Lecture Notes in Computer Science*, pages 394–412. Springer-Verlag, 2007.
- [BFL⁺93] Simon Blythe, Beatrice Fraboni, Sanjay Lall, Haroon Ahmed, and Ugo de Riu. Layout reconstruction of complex silicon chips. *IEEE journal of solid-state circuits*, 28(2):138–145, 1993.
- [BFSB06] Michael Brutscheck, Marco Franke, Andreas Th Schwarzbacher, and Steffen Becker. Determination of pin types and minimisation of test vectors in unknown CMOS integrated circuits. In *International Conference on Electronic Devices and Systems (EDS IMAPS CS 2006)*, pages 64–69, 2006.
- [BG07] Côme Berbain and Henri Gilbert. On the security of IV dependent stream ciphers. In *14th International Workshop on Fast Software Encryption (FSE 2007)*, volume 4593 of *Lecture Notes in Computer Science*, pages 254–273. Springer-Verlag, 2007.
- [BGL05] Lawrence D Bodin, Lawrence A Gordon, and Martin P Loeb. Evaluating information security investments using the analytic hierarchy process. *Communications of the ACM*, 48(2):78–83, 2005.
- [BGN05] Eli Biham, Louis Granboulan, and Phong Q Nguyễn. Impossible fault analysis of RC4 and differential fault analysis of RC4. In *12th International Workshop on Fast Software Encryption (FSE 2005)*, volume 3557 of *Lecture Notes in Computer Science*, pages 359–367. Springer-Verlag, 2005.

- [BGS⁺05] Stephen C. Bono, Matthew Green, Adam Stubblefield, Ari Juels, Aviel D. Rubin, and Michael Szydlo. Security analysis of a cryptographically-enabled RFID device. In *14th USENIX Security Symposium (USENIX Security 2005)*, pages 1–16. USENIX Association, 2005.
- [BGV⁺12] Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, and Ingrid Verbauwhede. Power analysis of Atmel CryptoMemory - recovering keys from secure EEPROMs. In *12th Cryptographers' Track at the RSA Conference, Topics in Cryptology (CT-RSA 2012)*, volume 7178 of *Lecture Notes in Computer Science*, pages 19–34. Springer-Verlag, 2012.
- [BGW01] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: the insecurity of 802.11. In *7th International Conference on Mobile Computing and Networking (MOBICOM 2001)*, pages 180–189. ACM, 2001.
- [BHL06] Andrea Bittau, Mark Handley, and Joshua Lackey. The final nail in WEP's coffin. In *27th IEEE Symposium on Security and Privacy (S&P 2006)*, pages 386–400. IEEE Computer Society, 2006.
- [Bih97] Eli Biham. A fast new DES implementation in software. In *4th International Workshop on Fast Software Encryption (FSE 1997)*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer-Verlag, 1997.
- [Bir04] Alex Biryukov. Block ciphers and stream ciphers: The state of the art. *IACR Cryptology ePrint Archive*, 2004(94):1–22, 2004.
- [BJ04] Jean Pierre Benhammou and Mary Jarboe. Security at an affordable price. *Atmel Applications Journal*, 3:29–30, 2004.
- [BK14] Christian Brandt and Michael Kasper. Don't push it: Breaking iButton security. In *6th International Symposium on Foundations and Practice of Security*, volume 8352 of *Lecture Notes in Computer Science*, pages 369–387. Springer-Verlag, 2014.
- [BKL⁺07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte VIKKELSOE. PRESENT: An ultra-lightweight block cipher. In *9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007)*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer-Verlag, 2007.
- [BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In *17th International Conference on the Theory and Application of Cryptology and Information*

- Security, Advances in Cryptology (ASIACRYPT 2011)*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer-Verlag, 2011.
- [BKZ11] Alex Biryukov, Ilya Kizhvatov, and Bin 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] Bruno 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.
- [Blo04] Eric Blossom. Gnu radio: tools for exploring the radio frequency spectrum. *Linux journal*, 2004(122):4, 2004.
- [BM03] Colin Boyd and Anish Mathuria. *Protocols for authentication and key establishment*. Springer-Verlag, 2003.
- [BMS06] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved time-memory trade-offs with multiple data. In *12th International Workshop on Selected Areas in Cryptography (SAC 2005)*, volume 3897 of *Lecture Notes in Computer Science*, pages 110–127. Springer-Verlag, 2006.
- [Bog07a] Andrey Bogdanov. Attacks on the KeeLoq block cipher and authentication systems. In *3rd Conference on RFID Security (RFIDSec 2007)*, volume 2007, 2007.
- [Bog07b] Andrey Bogdanov. Cryptanalysis of the KeeLoq block cipher. *IACR Cryptology ePrint Archive*, 2007:55, 2007.
- [Bog07c] Andrey 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, 2007.
- [BP08] Andrey Bogdanov and Christof Paar. On the security and efficiency of real-world lightweight authentication protocols. In *1st Workshop on Secure Component and System Identification (SECSI 2008)*. ECRYPT, 2008.
- [BPVV98] Johan Borst, Bart Preneel, Joos Vandewalle, and Joos V. On the time-memory tradeoff between exhaustive key search and table precomputation. In *19th Symposium in Information Theory in the Benelux*, pages 111–118, 1998.

- [BS76] Jon Louis Bentley and Michael Ian Shamos. Divide-and-conquer in multidimensional space. In *8th ACM Symposium on Theory of Computing (STOC 2013)*, pages 220–230. ACM, 1976.
- [BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.
- [BS93] Eli Biham and Adi Shamir. *Differential cryptanalysis of the Data Encryption Standard*, volume 28. Springer-Verlag, 1993.
- [BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *17th International Cryptology Conference, Advances in Cryptology (CRYPTO 1997)*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer-Verlag, 1997.
- [BS99] Alex Biryukov and Adi Shamir. Real time cryptanalysis of the alleged A5/1 on a PC, 1999.
- [BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *6th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2000)*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2000.
- [BSW01] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In *8th International Workshop on Fast Software Encryption (FSE 2000)*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2001.
- [BvDKM13] Victor Bos, Ton van Deursen, Piotr Kordy, and Sjouke Mauw. AlateX macro package for message sequence charts. Describing MSC macro package version 2.0, April 2013. Université du Luxembourg.
- [BVvE13] Willem Burgers, Roel Verdult, and Marko van Eekelen. Prevent session hijacking by binding the session to the cryptographic network credentials. In *18th Nordic Conference on Secure IT Systems (NordSec 2013)*, volume 8208 of *Lecture Notes in Computer Science*, pages 33–50. Springer-Verlag, 2013.
- [BW99] Alex Biryukov and David Wagner. Slide attacks. In *6th International Workshop on Fast Software Encryption (FSE 1999)*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, 1999.
- [C+06] Rafik Chaabouni et al. Break WEP faster with statistical analysis. Technical report, technical report, EPFL, LASEC, 2006.

- [C⁺07] LAN/MAN Committee et al. IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications (IEEE Std 802.11-2007), 2007.
- [CBW08] Nicolas T. Courtois, Gregory V. Bard, and David Wagner. Algebraic and slide attacks on KeeLoq. In *15th International Workshop on Fast Software Encryption (FSE 2008)*, volume 5086 of *Lecture Notes in Computer Science*, pages 97–115. Springer-Verlag, 2008.
- [CC90] Elliot J Chikofsky and James H Cross. Reverse engineering and design recovery: A taxonomy. *Software, IEEE*, 7(1):13–17, 1990.
- [CCC⁺09] Chun-Chieh Chen, Inn-Tung Chen, Chen-Mou Cheng, Ming-Yang Chih, and Jie-Ren Shih. A practical experience with RFID security. In *10th International Conference on Mobile Data Management: Systems, Services and Middleware (MDM 2009)*, pages 395–396. IEEE Computer Society, 2009.
- [CCCS92] Paul Camion, Claude Carlet, Pascale Charpin, and Nicolas Sendrier. On correlation-immune functions. In *11th International Cryptology Conference, Advances in Cryptology (CRYPTO 1991)*, volume 576 of *Lecture Notes in Computer Science*, pages 86–100. Springer-Verlag, 1992.
- [CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In *25th International Cryptology Conference, Advances in Cryptology (CRYPTO 2005)*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer-Verlag, 2005.
- [CF08] Claude Carlet and Keqin Feng. An infinite class of balanced functions with optimal algebraic immunity, good immunity to fast algebraic attacks and good nonlinearity. In *14th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2008)*, volume 5350 of *Lecture Notes in Computer Science*, pages 425–440. Springer-Verlag, 2008.
- [CG95] Cristina Cifuentes and K John Gough. Decompilation of binary programs. *Software: Practice and Experience*, 25(7):811–829, 1995.
- [CG12] Qi Chai and Guang Gong. BUPLE: securing passive RFID communication through physical layer enhancements. In *7th International Workshop on RFID Security and Privacy (RFIDSec 2011)*, pages 127–146. Springer-Verlag, 2012.

- [CGD96] Andrew Clark, Jovan Dj Golić, and Ed Dawson. A comparison of fast correlation attacks. In *3rd International Workshop on Fast Software Encryption (FSE 1996)*, volume 1039 of *Lecture Notes in Computer Science*, pages 145–157. Springer-Verlag, 1996.
- [CGE12] Qi Chai, Guang Gong, and Daniel Engels. How to develop clairaudience-active eavesdropping in passive RFID systems. In *13th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2012)*, pages 1–6. IEEE Computer Society, 2012.
- [CGY08] Luo Chunyu, Deng Guishi, and Guo Yanhong. Copyright protection model of embedded systems and its application in digital tv set-top-box. In *Computational Intelligence and Design (ISCID 2008)*, volume 2, pages 130–133. IEEE Computer Society, 2008.
- [Cif94] Cristina Cifuentes. *Reverse compilation techniques*. PhD thesis, Queensland University of Technology, School of Computing Science, 1994.
- [CJS01] Vladimor V Chepyzhov, Thomas Johansson, and Ben Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In *7th International Workshop on Fast Software Encryption (FSE 2000)*, volume 1978 of *Lecture Notes in Computer Science*, pages 181–195. Springer-Verlag, 2001.
- [CKBR06] Glenn Carl, George Kesidis, Richard R Brooks, and Suresh Rai. Denial-of-Service attack-detection techniques. *Internet Computing*, 10(1):82–89, 2006.
- [CKY89] John F Canny, Erich Kaltofen, and Lakshman Yagati. Solving systems of nonlinear polynomial equations faster. In *2nd International Symposium on Symbolic and Algebraic Computation (ISSAC 1989)*, pages 121–128. ACM, 1989.
- [Cla04] Christophe Clavier. Side channel analysis for reverse engineering (scare) - an improved attack against a secret A3/A8 gsm algorithm. *IACR Cryptology ePrint Archive*, 2004, 2004.
- [Cla07] Christophe Clavier. An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm. In *3rd Conference on Information Systems Security (ICISS 2007)*, volume 4812 of *Lecture Notes in Computer Science*, pages 143–155. Springer-Verlag, 2007.
- [CM03] Nicolas T Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In *22nd International Conference on the Theory and Application of Cryptographic Techniques, Advances in*

- Cryptology (EUROCRYPT 2003)*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.
- [CMK⁺11] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX Security Symposium (USENIX Security 2011)*, pages 77–92. USENIX Association, 2011.
- [CNO08] Nicolas Courtois, Karsten Nohl, and Sean O’Neil. Algebraic attacks on the crypto-1 stream cipher in MIFARE Classic and oyster cards. *IACR Cryptology ePrint Archive*, 2008:166, 2008.
- [Cop94] Don Coppersmith. The data encryption standard (DES) and its strength against attacks. *IBM journal of research and development*, 38(3):243–250, 1994.
- [COQ09] Nicolas T. Courtois, Sean O’Neil, and Jean-Jacques 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.
- [Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In *1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES 1999)*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer-Verlag, 1999.
- [Cou03a] Nicolas T Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *23rd International Cryptology Conference, Advances in Cryptology (CRYPTO 2003)*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer-Verlag, 2003.
- [Cou03b] Nicolas T Courtois. Higher order correlation attacks, xl algorithm and cryptanalysis of toyocrypt. In *5th International Conference on Information Security and Cryptology (ICISC 2002)*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199. Springer-Verlag, 2003.
- [Cou05] Nicolas T Courtois. Algebraic attacks on combiners with memory and several outputs. In *11th International Conference on Information Security and Cryptology (ICISC 2008)*, volume 3506 of *Lecture Notes in Computer Science*, pages 3–20. Springer-Verlag, 2005.
- [Cou09] Nicolas 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.
- [CP02] Nicolas T Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *8th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2002)*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.
- [CS91] Vladimir Chepyzhov and Ben Smeets. On a fast correlation attack on certain stream ciphers. In *10th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1991)*, volume 547 of *Lecture Notes in Computer Science*, pages 176–185. Springer-Verlag, 1991.
- [CS05] David Clark and Kevin Simmons. A dynamic 2D laser mark. *Industrial Laser Solutions For Manufacturing*, 20(8):19, 2005.
- [CSY+10] Ming-Yang Chih, Jie-Ren Shih, Bo-Yin Yang, Jintai Ding, and Chen-Mou Cheng. Mifare classic: Practical attacks and defenses. In *20th Cryptology and Information Security Conference (CISC 2010)*. Chinese Cryptology and Information Security Association, 2010.
- [Cum03] Nathan Cummings. iClass levels of security, April 2003.
- [Cum06] Nathan Cummings. Sales training. Slides from HID Technologies, March 2006.
- [CV95] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. In *13th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1994)*, volume 950 of *Lecture Notes in Computer Science*, pages 356–365. Springer-Verlag, 1995.
- [CW02] Steve Christey and Chris Wysopal. Responsible vulnerability disclosure process. <http://tools.ietf.org/html/draft-christey-wysopal-vuln-disclosure-00>, 2002. RFC draft.
- [CWHWW03] Nancy Cam-Winget, Russ Housley, David Wagner, and Jesse Walker. Security flaws in 802.11 data link protocols. *Communications of the ACM*, 46(5):35–39, 2003.
- [DC94] Ed Dawson and Andrew Clark. Divide and conquer attacks on certain classes of stream ciphers. *Cryptologia*, 18(1):25–40, 1994.
- [DC06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In *9th International Conference on Information Security (ISC 2006)*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer-Verlag, 2006.

- [DCJP01] Christophe De Canniere, Thomas Johansson, and Bart Preneel. Cryptanalysis of the Bluetooth stream cipher. Technical report, COSIC internal report, 2001.
- [DDMP04] Anupam Datta, Ante Derek, John C Mitchell, and Dusko Pavlovic. Abstraction and refinement in protocol derivation. In *17th Computer Security Foundations Workshop (CSFW 2004)*, pages 30–45. IEEE Computer Society, 2004.
- [DDSW11] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel 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.
- [Den82] Dorothy Elizabeth Robling Denning. *Cryptography and data security*. Addison-Wesley Longman Publishing Co., Inc., 1982.
- [DGB88] Yvo Desmedt, Claude Goutier, and Samy Bengio. Special uses and abuses of the fiat-shamir passport protocol. In *7th International Cryptology Conference, Advances in Cryptology (CRYPTO 1987)*, volume 293 of *Lecture Notes in Computer Science*, pages 21–39. Springer-Verlag, 1988.
- [DH76] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [DH79] Whitfield Diffie and Martin E Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, 1979.
- [DHW⁺12] Benedikt Driessen, Ralf Hund, Carsten Willems, Carsten Paar, and Thorsten 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.
- [Din94] Cunsheng Ding. The differential cryptanalysis and design of natural stream ciphers. In *1st International Workshop on Fast Software Encryption (FSE 1993)*, volume 809 of *Lecture Notes in Computer Science*, pages 101–115. Springer-Verlag, 1994.
- [Dip09] Brian Dipert. The Zune HD: more than an iPod touch wanna-be? *Electrical Design News (EDN)*, page 20, October 2009.
- [dKG08] Gerhard de Koning Gans. Analysis of the MIFARE Classic used in the OV-chipkaart project. Master’s thesis, Radboud University Nijmegen, 2008.

- [dKG13] Gerhard de Koning Gans. *Outsmarting Smart Cards*. PhD thesis, Radboud Universiteit Nijmegen, Nijmegen, Netherlands, April 2013.
- [dKGdR12] Gerhard de Koning Gans and Joeri 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] Gerhard de Koning Gans and Flavio D. Garcia. Towards a practical solution to the RFID desynchronization problem. In *6th Workshop on RFID Security (RFIDSec 2010)*, volume 6370 of *Lecture Notes in Computer Science*, pages 203–219. Springer-Verlag, 2010.
- [dKGGH08] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio 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.
- [dKGV09] Gerhard de Koning Gans and Roel Verdult. Proxmark.org – A radio frequency identification tool. <http://www.proxmark.org>, 2009. *Research and Developers Community*.
- [DKR97] Joan Daemen, Lars Knudsen, and Vincent Rijmen. The block cipher SQUARE. In *4th International Workshop on Fast Software Encryption (FSE 1997)*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer-Verlag, 1997.
- [DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. In *30th International Cryptology Conference, Advances in Cryptology (CRYPTO 2010)*, volume 6223 of *Lecture Notes in Computer Science*, pages 393–410. Springer-Verlag, 2010.
- [DLMV05] Rémy Daudigny, Hervé Ledig, Frédéric Muller, and Frédéric Valette. SCARE of the DES. In *3rd International Conference on Applied Cryptography and Network Security (ACNS 2005)*, pages 393–406. Springer-Verlag, 2005.
- [DM95] Donald Davies and Sean Murphy. Pairs and triplets of DES S-boxes. *Journal of Cryptology*, 8(1):1–25, 1995.
- [DMÖPV07] Elke De Mulder, Siddika Berna Örs, Bart Preneel, and Ingrid Verbauwhede. Differential power and electromagnetic attacks on a FPGA implementation of elliptic curve cryptosystems. *Computers & Electrical Engineering*, 33(5):367–382, 2007.

- [DR98] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1998.
- [DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002.
- [DR11] Thai Duong and Juliano Rizzo. Here come the XOR Ninjas. White paper, Netifera, May 2011.
- [DS08] Hüseyin Demirci and Ali Aydın Selçuk. A meet-in-the-middle attack on 8-round AES. In *15th International Workshop on Fast Software Encryption (FSE 2008)*, volume 5086 of *Lecture Notes in Computer Science*, pages 116–126. Springer-Verlag, 2008.
- [DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In *28th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2009)*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer-Verlag, 2009.
- [DSP07] Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved meet-in-the-middle attacks on reduced-round DES. In *8th International Conference on Cryptology in India, Progress in Cryptology (INDOCRYPT 2007)*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer-Verlag, 2007.
- [DST04] Hüseyin Demirci, Ali Aydın Selçuk, and Erkan Türe. A new meet-in-the-middle attack on the IDEA block cipher. In *10th International Workshop on Selected Areas in Cryptography (SAC 2003)*, volume 3006 of *Lecture Notes in Computer Science*, pages 117–129. Springer-Verlag, 2004.
- [DTÇB09] Hüseyin Demirci, İhsan Taşkın, Mustafa Çoban, and Adnan Baysal. Improved meet-in-the-middle attacks on AES. In *10th International Conference on Cryptology in India, Progress in Cryptology (INDOCRYPT 2009)*, volume 5922 of *Lecture Notes in Computer Science*, pages 144–156. Springer-Verlag, 2009.
- [DvE14] Hristo Dimitrov and Kim van Erkelens. Evaluation of the feasible attacks against RFID tags for access control systems, 2014.
- [Dwo04] Morris Dworkin. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality. *NIST Special publication (800-38C)*, 38C:1–27, 2004.
- [Dwo05] Morris Dworkin. Recommendation for block cipher modes of operation: The CMAC mode for authentication. *NIST Special Publication (800-38B)*, 38B:1–25, 2005.

- [Dwo07] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. *NIST Special publication (800-38D)*, 38D:1–39, 2007.
- [Dwy87] Rex A Dwyer. A faster divide-and-conquer algorithm for constructing delaunay triangulations. *Algorithmica*, 2(1-4):137–151, 1987.
- [EG85] Shimon Even and Oded Goldreich. On the power of cascade ciphers. *ACM Transactions on Computer Systems (TOCS)*, 3(2):108–116, 1985.
- [EJ03a] Patrik Ekdahl and Thomas Johansson. Another attack on A5/1. *IEEE Transactions on Information Theory*, 49(1):284–289, 2003.
- [EJ03b] Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher SNOW. In *9th International Workshop on Selected Areas in Cryptography (SAC 2002)*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 2003.
- [EKM⁺08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the KeeLoq code hopping scheme. In *28th International Cryptology Conference, Advances in Cryptology (CRYPTO 2008)*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer-Verlag, 2008.
- [EM02] 125khz crypto read/write contactless identification device, EM4170. Product Datasheet, Mar 2002. EM Microelectronic-Marin SA.
- [EMST78] William F Ehrtam, Carl HW Meyer, John L Smith, and Walter L Tuchman. Message verification and transmission error detection by block chaining. US Patent 4,074,066, Feb 1978.
- [ETS08] Smart cards; UICC – contactless front-end (CLF) interface; host controller interface (hci) (ETSI TS 102 613), 2008. European Telecommunications Standards Institute (ETSI).
- [ETS11] Smart cards; UICC – contactless front-end (CLF) interface; part 1: Physical and data link layer characteristics (ETSI TS 102 613), 2011. European Telecommunications Standards Institute (ETSI).
- [ETS12] Universal mobile telecommunications system (UMTS); LTE; 3G security; specification of the 3GPP confidentiality and integrity algorithms; document 2: KASUMI specification (ETSI TS 135 202), 2012. European Telecommunications Standards Institute (ETSI).

- [FAH⁺10] Martin Feldhofer, Manfred Josef Aigner, Michael Hutter, Thomas Plos, Erich Wenger, and Thomas Baier. Semi-passive RFID development platform for implementing and attacking security tags. In *2nd International Workshop on RFID/USN Security and Cryptography (RISC 2010)*, pages 1–6. IEEE Computer Society, 2010.
- [FCC09] Guidelines for evaluating the environmental effects of radio frequency radiation, April 2009.
- [FDv11] Aurélien Francillon, Boris Danev, and Srdjan Čapkun. Relay attacks on passive keyless entry and start systems in modern cars. In *18th Network and Distributed System Security Symposium (NDSS 2011)*. The Internet Society, 2011.
- [FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong authentication for RFID systems using the aes algorithm. In *6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, pages 357–370. Springer-Verlag, 2004.
- [FGMR10] Renato Ferrero, Filippo Gandino, Bartolomeo Montrucchio, and Maurizio Rebaudengo. Fair anti-collision protocol in dense RFID networks. In *3rd International EURASIP Workshop on RFID Technology (EURASIP-RFID 2010)*, pages 101–105. IEEE Computer Society, 2010.
- [FHMM10] Lishoy Francis, Gerhard Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical nfc peer-to-peer relay attack using mobile phones. In *Radio Frequency Identification: Security and Privacy Issues*, volume 6370 of *Lecture Notes in Computer Science*, pages 35–49. Springer-Verlag, 2010.
- [FHMM11] Lishoy Francis, Gerhard P Hancke, Keith Mayes, and Konstantinos Markantonakis. Practical relay attack on contactless transactions by using nfc mobile phones. *IACR Cryptology ePrint Archive*, 2011:618, 2011.
- [FIP77] PUB FIPS. Data encryption standard (DES). *National Bureau of Standards (NBS)*, 46(0), 1977.
- [FIP93] PUB FIPS. Secure hash standard (SHS). *National Institute for Standards and Technology (NIST)*, 180(0), 1993.
- [FIP01] PUB FIPS. Advanced encryption standard (AES). *National Institute for Standards and Technology (NIST)*, 197(1), 2001.
- [FIP08] PUB FIPS. The keyed-hash message authentication code (hmac). *National Institute for Standards and Technology (NIST)*, 1:1Ö13, 2008.

- [FIP14] PUB FIPS. Secure hash algorithm-3 (SHA-3) standard: Permutation-based hash and extendable-output functions. *National Institute for Standards and Technology (NIST)*, 202(0), 2014.
- [FJ03] Jean-Charles Faugere and Antoine Joux. Algebraic cryptanalysis of hidden field equation (hfe) cryptosystems using gröbner bases. In *23rd International Cryptology Conference, Advances in Cryptology (CRYPTO 2003)*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60. Springer-Verlag, 2003.
- [FL01] Scott Fluhrer and Stefan Lucks. Analysis of the E0 encryption system. In *8th International Workshop on Selected Areas in Cryptography (SAC 2001)*, volume 2259 of *Lecture Notes in Computer Science*, pages 38–48. Springer-Verlag, 2001.
- [FL12] Riccardo Focardi and Flaminia 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.
- [Flu02] Scott Fluhrer. Improved key recovery of level 1 of the Bluetooth encryption system. *IACR Cryptology ePrint Archive*, 2002(68):1–6, 2002.
- [Fly08] Flylogic. Atmel CryptoMemory AT88SC153/1608 security alert. Retrieved from <http://www.flylogic.net/blog/?p=25> on October 13th, 2009, 2008.
- [FLZ⁺10] Xiutao Feng, Jun Liu, Zhaocun Zhou, Chuankun Wu, and Dengguo Feng. A byte-based guess and determine attack on SOSEMANUK. In *16th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2010)*, volume 6477 of *Lecture Notes in Computer Science*, pages 146–157. Springer-Verlag, 2010.
- [FMS01] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *8th International Workshop on Selected Areas in Cryptography (SAC 2001)*, volume 2259 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2001.
- [FN91] Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 534–541. ACM, 1991.
- [FNS75] Horst Feistel, William A Notz, and J Lynn Smith. Some cryptographic techniques for machine-to-machine data communications. *Proceedings of the IEEE*, 63(11):1545–1554, 1975.

- [fSN97] National Institute for Standards and Technology (NIST). Announcing request for candidate algorithm nominations for the advanced encryption standard (AES). *Federal Register*, 62(177):48051Ö–48058, 1997.
- [GB06] Bill Glover and Himanshu Bhatt. *RFID essentials*. O’Reilly Media, Inc., 2006.
- [GBM02] Jovan Dj Golić, Vittorio Bagini, and Guglielmo Morgari. Linear cryptanalysis of Bluetooth stream cipher. In *21st International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2002)*, volume 2332 of *Lecture Notes in Computer Science*, pages 238–255. Springer-Verlag, 2002.
- [GBPv10] Benedikt Gierlich, Lejla Batina, Bart Preneel, and Ingrid Verbauwhede. Revisiting higher-order DPA attacks. In *10th Cryptographers’ Track at the RSA Conference, Topics in Cryptology (CT-RSA 2010)*, volume 5985 of *Lecture Notes in Computer Science*, pages 221–234. Springer-Verlag, 2010.
- [GdKGM⁺08] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart 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] Flavio D. Garcia, Gerhard de Koning Gans, and Roel Verdult. Exposing iClass key diversification. In *5th USENIX Workshop on Offensive Technologies (WOOT 2011)*, pages 128–136. USENIX Association, 2011.
- [GdKGV12] Flavio D. Garcia, Gerhard de Koning Gans, and Roel Verdult. Tutorial: Proxmark, the swiss army knife for RFID security research. Technical report, Radboud University Nijmegen, 2012.
- [GdKGV14] Flavio D. Garcia, Gerhard de Koning Gans, and Roel Verdult. Wirelessly lockpicking a smart card reader. *International Journal of Information Security*, 13(5):403–420, 2014.
- [GdKGVm12] Flavio D. Garcia, Gerhard de Koning Gans, Roel Verdult, and Milosch Meriac. Dismantling iClass and iClass Elite. In *17th European Symposium on Research in Computer Security (ESORICS 2012)*, volume 7459 of *Lecture Notes in Computer Science*, pages 697–715. Springer-Verlag, 2012.
- [GFMR11] Filippo Gandino, Renato Ferrero, Bartolomeo Montrucchio, and Maurizio Rebaudengo. Probabilistic DCS: An RFID reader-to-reader anti-collision protocol. *Journal of Network and Computer Applications*, 34(3):821–832, 2011.

- [GGvR08] David Galindo, Flavio D. Garcia, and Peter van Rossum. Computational soundness of non-malleable commitments. In *4th Information Security Practice and Experience Conference (ISPEC 2008)*, volume 4266 of *Lecture Notes in Computer Science*, pages 361–376. Springer-Verlag, 2008.
- [GH05] Flavio D. Garcia and Jaap-Henk Hoepman. Off-line karma: A decentralized currency for peer-to-peer and grid applications. In *3th Applied Cryptography and Network Security (ACNS 2005)*, volume 3531 of *Lecture Notes in Computer Science*, pages 364–377. Springer-Verlag, 2005.
- [GKVV05] Marcin Gomul̄kiewicz, Mirosław Kutylowski, Heinrich Theodor Vierhaus, and Paweł Właż. Synchronization fault cryptanalysis for breaking A5/1. In *4th International Workshop on Experimental and Efficient Algorithms (WEA 2005)*, volume 3503 of *Lecture Notes in Computer Science*, pages 415–427. Springer-Verlag, 2005.
- [GL02] Lawrence A Gordon and Martin P Loeb. The economics of information security investment. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):438–457, 2002.
- [GLRW10] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In *16th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2010)*, volume 6477 of *Lecture Notes in Computer Science*, pages 56–75. Springer-Verlag, 2010.
- [GM04] Praveen S Gauravaram and William L Millan. Improved attack on the cellular authentication and voice encryption algorithm (CAVE). In *Cryptographic Algorithms and their Uses (CAU 2004)*, pages 1–13. Queensland University of Technology, 2004.
- [GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001)*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 2001.
- [GMPS14] Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. (non-) random sequences from (non-) random permutations \mathcal{Q} analysis of RC4 stream cipher. *Journal of Cryptology*, 27(1):67–108, 2014.
- [GNR08] Timo Gendrullis, Martin Novotný, and Andy Rupp. A real-world attack breaking A5/1 within hours. In *10th International Workshop on*

- Cryptographic Hardware and Embedded Systems (CHES 2008)*, volume 5154 of *Lecture Notes in Computer Science*, pages 266–282. Springer-Verlag, 2008.
- [Gol67] S.W. Golomb. *Shift Register Sequences*. Holden-Day Series in Information Systems. Holden-Day, 1967.
- [Gol96] Jovan Dj Golić. On the security of nonlinear filter generators. In *3rd International Workshop on Fast Software Encryption (FSE 1996)*, volume 1039 of *Lecture Notes in Computer Science*, pages 173–188. Springer-Verlag, 1996.
- [Gol97a] Jovan Dj. Golić. 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.
- [Gol97b] Jovan Dj. Golić. Linear statistical weakness of alleged RC4 keystream generator. 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 226–238. Springer-Verlag, 1997.
- [Gou02] Louis Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In *6th International Workshop on Theory and Practice in Public Key Cryptography (PKC 2003)*, volume 2567 of *Lecture Notes in Computer Science*, pages 199–211. Springer-Verlag, 2002.
- [GR05] Simson Garfinkel and Beth Rosenberg. *RFID: Applications, Security, And Privacy*. Addison-Wesley Publishing Company Incorporated, 2005.
- [Gra02] Louis Granboulan. Flaws in differential cryptanalysis of Skipjack. In *8th International Workshop on Fast Software Encryption (FSE 2001)*, volume 2355 of *Lecture Notes in Computer Science*, pages 328–335. Springer-Verlag, 2002.
- [Gro71] El Groth. Generation of binary sequences with controllable complexity. *IEEE Transactions on Information Theory*, 17(3):288–296, 1971.
- [GS07] Gary M. Gaukler and Ralf W. Seifert. Applications of RFID in supply chains. In *Trends in supply chain design and management*, pages 29–48. Springer, 2007.
- [GST13] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low - bandwidth acoustic cryptanalysis. *IACR Cryptology ePrint Archive*, 2013(857):1Q57, 2013.

- [GvR06a] Flavio D. Garcia and Peter van Rossum. Sound computational interpretation of formal hashes. Technical Report ICIS-R06001, Institute for Computing and Information Sciences, Radboud University Nijmegen, 2006.
- [GvR06b] Flavio D. Garcia and Peter van Rossum. Sound computational interpretation of symbolic hashes in the standard model. In *1st International Workshop on Security, Advances in Information and Computer Security (IWSEC 2006)*, volume 4266 of *Lecture Notes in Computer Science*, pages 33–47. Springer-Verlag, 2006.
- [GvR08] Flavio D. Garcia and Peter van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Theoretical Computer Science*, 394(1–2):112–133, 2008.
- [GvR10] Flavio D. Garcia and Peter van Rossum. Modeling privacy for off-line RFID systems. In *9th Smart Card Research and Advanced Applications (CARDIS 2010)*, volume 6035 of *Lecture Notes in Computer Science*, pages 194–208. Springer-Verlag, 2010.
- [GVRS07] Flavio D Garcia, Peter Van Rossum, and Ana Sokolova. Probabilistic anonymity and admissible schedulers. *CoRR*, abs/0706.1019, 2007.
- [GvRVWS09] Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny 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] Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Dismantling SecureMemory, CryptoMemory and CryptoRF. In *17th ACM Conference on Computer and Communications Security (CCS 2010)*, pages 250–259. ACM, 2010.
- [HA03] Russ Housley and William Arbaugh. Security problems in 802.11-based networks. *Communications of the ACM*, 46(5):31–34, 2003.
- [Han05] Gerhard P. Hancke. A practical relay attack on ISO 14443 proximity cards. Technical report, University of Cambridge Computer Laboratory, 2005.
- [Han06] Gerhard P. Hancke. Practical attacks on proximity identification systems (short paper). In *27th IEEE Symposium on Security and Privacy (S&P 2006)*, pages 328–333. IEEE Computer Society, 2006.
- [HCJ02] Shai Halevi, Don Coppersmith, and Charanjit Jutla. Scream: A software-efficient stream cipher. In *9th International Workshop on Fast Software Encryption (FSE 2002)*, volume 2365 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, 2002.

- [Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
- [Hen11] Martin Henzl. Security of contactless smart cards. In *17th annual student conference and competition STUDENT EEICT 2011*, pages 585–589. Faculty of Information Technology BUT, 2011.
- [HHBR⁺08] Daniel Halperin, Thomas S. Heydt-Benjamin, Benjamin Ransford, Shane S. Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno, and William H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *29th IEEE Symposium on Security and Privacy (S&P 2008)*, pages 129–142. IEEE Computer Society, 2008.
- [HHJ⁺06] Jaap-Henk Hoepman, Engelbert Hubbers, Bart Jacobs, Martijn Oostdijk, and Ronny Wichers Schreur. Crossing borders: Security and privacy issues of the european e-passport. In *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.
- [HHJK12] Martin Henzl, Petr Hanacek, Peter Jurnecka, and Matej Kacic. A concept of automated vulnerability search in contactless communication applications. In *46th Annual IEEE International Carnahan Conference on Security Technology (ICCST 2012)*, pages 180–186. IEEE Computer Society, 2012.
- [HID06] HID Global. HID management key letter, November 2006.
- [HID08] 13.56 MHz contactless iClass card. Product Features and Specifications, October 2008. HID Global.
- [HID09] HID Global. iClass RW100, RW150, RW300, RW400 readers, 2009.
- [Hil29] Lester S. Hill. Cryptography in an algebraic alphabet. *American Mathematical Monthly*, 36(6):306–312, 1929.
- [Hil31] Lester S Hill. Concerning certain linear transformation apparatus of cryptography. *American Mathematical Monthly*, pages 135–154, 1931.
- [HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, 2007.
- [HJSW06] John Halamka, Ari Juels, Adam Stubblefield, and Jonathan Westhues. The security implications of VeriChip cloning. *Journal of the American Medical Informatics Association*, 13(6):601–607, 2006.

- [HK05] Gerhard P Hancke and Markus G 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.
- [HM13] Jin Hong and Sunghwan Moon. A comparison of cryptanalytic tradeoff algorithms. *Journal of Cryptology*, 26(4):559–637, 2013.
- [HMM09] Gerhard P Hancke, KE Mayes, and Konstantinos Markantonakis. Confidence in smart token proximity: Relay attacks revisited. *Computers & Security*, 28(7):615–627, 2009.
- [HN00] M. Hermelin and K. Nyberg. Correlation properties of the Bluetooth combiner. In *2nd Information Security and Cryptology (ICISC 1999)*, volume 1787 of *Lecture Notes in Computer Science*, pages 17–29. Springer-Verlag, 2000.
- [Hou73] Barron Cornelius Housel. *A Study of Decompiling Machine Language into High-Level Machine Independent Languages*. PhD thesis, Purdue University, Computer Science, 1973.
- [HP00] Helena Handschuh and Pascal Paillier. Reducing the collision probability of alleged Comp128. In *3rd Smart Card Research and Advanced Applications Conference (CARDIS 1998)*, volume 1820 of *Lecture Notes in Computer Science*, pages 366–371. Springer-Verlag, 2000.
- [HR00] Philip Hawkes and Gregory G Rose. Exploiting multiples of the connection polynomial in word-oriented stream ciphers. In *6th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2000)*, volume 1976 of *Lecture Notes in Computer Science*, pages 303–316. Springer-Verlag, 2000.
- [HR03] Philip Hawkes and Gregory G Rose. Guess-and-determine attacks on SNOW. In *9th International Workshop on Selected Areas in Cryptography (SAC 2002)*, volume 2595 of *Lecture Notes in Computer Science*, pages 37–46. Springer-Verlag, 2003.
- [HSH⁺08] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. In *17th USENIX Security Symposium (USENIX Security 2008)*, pages 45–60. USENIX Association, 2008.
- [HSH⁺09] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

- [HTTN88] Motoki Hirano, Mikio Takeuchi, Takahisa Tomoda, and Kin-Ichiro Nakano. Keyless entry system with radio card transponder. *IEEE Transactions on Industrial Electronics*, 35:208–216, 1988.
- [IC04] PicoPass 2KS. Product Datasheet, Nov 2004. Inside Contactless.
- [ICA03] Machine readable travel documents, 2003. International Civil Aviation Organization (ICAO).
- [IKD⁺08] Sebastiaan Indestege, Nathan Keller, Orr Dunkelmann, Eli Biham, and Bart Preneel. A practical attack on KeeLoq. In *27th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2008)*, volume 4965 of *Lecture Notes in Computer Science*, pages 1–8. Springer-Verlag, 2008.
- [INF07] Intelligent 10 kbit EEPROM, with contactless interface compliant to ISO/IEC 15693, and ISO/IEC 18000-3 mode 1, and security logic, SRF 55V10S. Short Product Information, Jul 2007. Infineon Technologies AG.
- [IRMTT⁺10] Rob Millerb Ishtiaq Roufa, Hossen Mustafaa, Sangho Ohb Travis Taylor, Wenyan Xua, Marco Gruteserb, Wade Trappeb, and Ivan Seskarb. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *19th USENIX Security Symposium (USENIX Security 2010)*, pages 323–338. USENIX Association, 2010.
- [ISO94] Radio frequency identification of animals – code structure (ISO/IEC 11784), 1994. International Organization for Standardization (ISO).
- [ISO96] Radio frequency identification of animals – technical concept (ISO/IEC 11785), 1996. International Organization for Standardization (ISO).
- [ISO99] Information technology – security techniques – entity authentication – part 2: Mechanisms using symmetric encipherment algorithms (ISO/IEC 9798 part 2), 1999. International Organization for Standardization (ISO).
- [ISO00] Identification cards – contactless integrated circuit(s) cards – vicinity cards (ISO/IEC 15693), 2000. International Organization for Standardization (ISO).
- [ISO01] Identification cards – contactless integrated circuit cards – proximity cards (ISO/IEC 14443), 2001. International Organization for Standardization (ISO).

- [ISO04] Information technology – telecommunications and information exchange between systems – near field communication interface and protocol 1 (NFCIP-1) (ISO/IEC 18092), 2004. International Organization for Standardization (ISO).
- [ISO05] Information technology – telecommunications and information exchange between systems – near field communication interface and protocol 2 (NFCIP-2) (ISO/IEC 21481), 2005. International Organization for Standardization (ISO).
- [ISO07] Information technology – telecommunications and information exchange between systems – near field communication wired interface (NFC-WI) (ISO/IEC 28361), 2007. International Organization for Standardization (ISO).
- [ISO11] Information technology – telecommunications and information exchange between systems – front-end configuration command for (NFC-FEC) (ISO/IEC 16353), 2011. International Organization for Standardization (ISO).
- [Jar04] Mary Jarboe. Introduction to CryptoMemory. *Atmel Applications Journal*, 3:28, 2004.
- [JIC05] Specification of implementation for integrated circuit(s) cards (JIC-SAP/JSA JIS X 6319), 2005. Japan IC Card System Application Council (JICSAP).
- [JJ00] Thomas Johansson and Fredrik Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In *20th International Cryptology Conference, Advances in Cryptology (CRYPTO 2000)*, volume 1880 of *Lecture Notes in Computer Science*, pages 300–315. Springer-Verlag, 2000.
- [JJ02] Fredrik Jönsson and Thomas Johansson. A fast correlation attack on lili-128. *Information Processing Letters*, 81(3):127–132, 2002.
- [JLR⁺13] Keting Jia, Leibo Li, Christian Rechberger, Jiazhe Chen, and Xiaoyun Wang. Improved cryptanalysis of the block cipher KASUMI. In *19th International Conference on Selected Areas in Cryptography (SAC 2012)*, volume 7707 of *Lecture Notes in Computer Science*, pages 222–233. Springer-Verlag, 2013.
- [Jon03] Jakob Jonsson. On the security of CTR + CBC-MAC. In *9th International Workshop on Selected Areas in Cryptography (SAC 2002)*, volume 2595 of *Lecture Notes in Computer Science*, pages 76–93. Springer-Verlag, 2003.

- [JS97] Norman D. Jorstad and Landgrave T. Smith. Cryptographic algorithm metrics. In *20th National Information Systems Security Conference*. National Institute of Standards and Technology (NIST), 1997.
- [Jue06] Ari Juels. RFID security and privacy: A research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, 2006.
- [JWS11] Bart Jacobs and Ronny 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.
- [KA09] Chong Hee Kim and Gildas Avoine. RFID distance bounding protocol with mixed challenges to prevent relay attacks. In *8th International Conference on Cryptology and Network Security (CANS 2009)*, volume 5888 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2009.
- [KAK⁺09] Chong Hee Kim, Gildas Avoine, François Koeune, François-Xavier Standaert, and Olivier 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.
- [KCP07] Timo Kasper, Dario Curiuccio, and Christof Paar. An embedded system for practical security analysis of contactless smartcards. In *Workshop in Information Security Theory and Practice (WISTP 2007)*, volume 4462 of *Lecture Notes in Computer Science*, pages 150–160. Springer-Verlag, 2007.
- [KCR⁺10] Karl Koscher, Alexei Czeskis, Franziska Roesner, Franziska Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage. Experimental security analysis of a modern automobile. In *31st IEEE Symposium on Security and Privacy (S&P 2010)*, pages 447–462. IEEE Computer Society, 2010.
- [Ken77] Stephen Thomas Kent. Encryption-based protection for interactive user/computer communication. In *5th Symposium on Data Communications (SIGCOMM 1977)*, pages 5–7. ACM, 1977.
- [Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, 9(1):5–38, 1883.
- [Kes00] Gregory Kesden. Content scrambling system (CSS), 2000. Carnegie Mellon University.

- [Key76] Edwin Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Transactions on Information Theory*, 22(6):732–736, 1976.
- [Key12] Keyline. Transponder guide. http://www.keyline.it/files/884/transponder_guide_16729.pdf, 2012.
- [KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *19th International Cryptology Conference, Advances in Cryptology (CRYPTO 1999)*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
- [KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.
- [KJL⁺13] ChangKyun Kim, Eun-Gu Jung, Dong Hoon Lee, Chang-Ho Jung, and Daewan Han. Cryptanalysis of INCrypt32 in HID’s iClass systems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 96(1):35–41, 2013.
- [KKMP09] Markus Kasper, Timo Kasper, Amir Moradi, and Christof 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.
- [Kle08] Andreas Klein. Attacks on the RC4 stream cipher. *Designs, Codes and Cryptography*, 48(3):269–286, 2008.
- [KM05] Michael Knebelkamp and Herbert Meier. Latest generation technology for immobiliser systems. *Elektron*, 13(2):22–25, 2005.
- [Koc96] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *16th International Cryptology Conference, Advances in Cryptology (CRYPTO 1996)*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.
- [KP09] Jongsung Kim and Raphael Chung-Wei Phan. Advanced differential-style cryptanalysis of the NSA’s skipjack block cipher. *Cryptologia*, 33(3):246–270, 2009.
- [KPP⁺06] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. Breaking ciphers with COPACOBANA—a cost-optimized parallel code breaker. In *8th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4249 of *Lecture Notes in Computer Science*, pages 101–118. Springer-Verlag, 2006.

- [KPPM12] Maria Kalenderi, Dionisios Pnevmatikatos, Ioannis Papaefstathiou, and Charalampos Manifavas. Breaking the GSM A5/1 cryptography algorithm with rainbow tables and high-end FPGAS. In *22nd International Conference on Field Programmable Logic and Applications (FPL 2012)*, pages 747–753. IEEE Computer Society, 2012.
- [Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *21st International Cryptology Conference, Advances in Cryptology (CRYPTO 2001)*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer-Verlag, 2001.
- [Kra02] Matthias Krause. BDD-based cryptanalysis of keystream generators. In *21st International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2002)*, volume 2332 of *Lecture Notes in Computer Science*, pages 222–237. Springer-Verlag, 2002.
- [KRW99] Lars R Knudsen, Matthew JB Robshaw, and David Wagner. Truncated differentials and skipjack. In *19th International Cryptology Conference, Advances in Cryptology (CRYPTO 1999)*, volume 1666 of *Lecture Notes in Computer Science*, pages 165–180. Springer-Verlag, 1999.
- [KSB08] M. Ayoub Khan, Manoj Sharma, and Prabhu R. Brahmanandha. FSM based manchester encoder for UHF RFID tag emulator. In *17th International Conference on Computing, Communication and Networking (ICCCn 2008)*, pages 1–6. IEEE Computer Society, 2008.
- [KSP10] Timo Kasper, Michael Silbermann, and Christof Paar. All you can eat or breaking a real-world contactless payment system. In *14th International Conference on Financial Cryptography and Data Security (FC 2010)*, volume 6052 of *Lecture Notes in Computer Science*, pages 343–350. Springer-Verlag, 2010.
- [KSRW04] Tadayoshi Kohno, Adam Stubblefield, Aviel D. Rubin, and Dan 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.
- [KSW97] John Kelsey, Bruce Schneier, and David Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In *1st International Conference on Information and Communications Security (ICICS 1997)*, volume 1334 of *Lecture Notes in Computer Science*, pages 233–246. Springer-Verlag, 1997.

- [KSW99] John Kelsey, Bruce Schneier, and David Wagner. Mod n cryptanalysis, with applications against RC5P and M6. In *6th International Workshop on Fast Software Encryption (FSE 1999)*, volume 1636 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, 1999.
- [Kuh88] GJ Kuhn. Algorithms for self-synchronizing ciphers. In *1st Southern African Conference on Communications and Signal Processing (COMSIG 1988)*, pages 159–164. IEEE, 1988.
- [KW05] Ziv Kfir and Avishai Wool. Picking virtual pockets using relay attacks on contactless smartcard. In *1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005)*, pages 47–58. IEEE Computer Society, 2005.
- [KW06] Ilan Kirschenbaum and Avishai Wool. How to build a low-cost, extended-range RFID skimmer. In *15th USENIX Security Symposium (USENIX Security 2006)*, pages 43–57. USENIX Association, 2006.
- [KWH99] Paul D Kundarewich, Steven JE Wilton, and Alan J Hu. A CPLD-based RC4 cracking system. In *IEEE Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 397–402. IEEE Computer Society, 1999.
- [LA⁺08] Chein-Shan Liu, Satya N Atluri, et al. A novel time integration method for solving a large system of non-linear algebraic equations. *Computer Modeling in Engineering & Sciences (CMES)*, 31(2):71–83, 2008.
- [Lan01] Carl E Landwehr. Computer security. *International Journal of Information Security*, 1(1):3–13, 2001.
- [LÇA⁺04] Albert Levi, Erhan Çetintaş, Murat Aydos, Çetin Kaya Koç, and M Ufuk Çağlayan. Relay attacks on bluetooth authentication and solutions. In *19th International Symposium on Computer and Information Sciences (ISCIS 2004)*, pages 278–288. Springer-Verlag, 2004.
- [LCPW01] David P Leech, Michael W Chinworth, Gary G. Payne, and Christopher M. Waychoff. The economic impacts of NIST’s data encryption standard (DES) program. Technical report, National Institute of Standards and Technology (NIST), 2001.
- [LEG03] LEGIC prime data media MIM256 and MIM1024. Public Datasheet, Feb 2003. LEGIC Identsystems Ltd.
- [LEG06] Frequently asked questions about LEGIC. LEGIC FAQ, Nov 2006. LEGIC Identsystems Ltd.

- [LG98] Mike Loukides and John Gilmore, editors. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998.
- [LMM91] Xuejia Lai, James L Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *10th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1991)*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer-Verlag, 1991.
- [LMV05] Yi Lu, Willi Meier, and Serge Vaudenay. The conditional correlation attack: A practical attack on Bluetooth encryption. In *25th International Cryptology Conference, Advances in Cryptology (CRYPTO 2005)*, volume 3621 of *Lecture Notes in Computer Science*, pages 97–117. Springer-Verlag, 2005.
- [LS04] Tom Lookabaugh and Douglas C Sicker. Security and lock-in. In *Economics of Information Security*, pages 225–246. Springer-Verlag, 2004.
- [LSS05] Kerstin Lemke, Ahmad-Reza Sadeghi, and Christian Stübke. An open approach for designing secure electronic immobilizers. In *Information Security Practice and Experience (ISPEC 2005)*, volume 3439 of *Lecture Notes in Computer Science*, pages 230–242. Springer-Verlag, 2005.
- [LSS06] Kerstin Lemke, Ahmad-Reza Sadeghi, and Christian Stübke. Anti-theft protection: Electronic immobilizers. *Embedded Security in Cars*, pages 51–67, 2006.
- [LST⁺09] Stefan Lucks, Andreas Schuler, Erik Tews, Ralf-Philipp Weinmann, and Matthias Wenzel. Attacks on the DECT authentication mechanisms. In *9th Cryptographers' Track at the RSA Conference, Topics in Cryptology (CT-RSA 2009)*, volume 5473 of *Lecture Notes in Computer Science*, pages 48–65. Springer-Verlag, 2009.
- [LV04a] Yi Lu and Serge Vaudenay. Cryptanalysis of Bluetooth keystream generator two-level E0. In *10th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2004)*, volume 3329 of *Lecture Notes in Computer Science*, pages 483–499. Springer-Verlag, 2004.
- [LV04b] Yi Lu and Serge Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In *24th International Cryptology Conference, Advances in Cryptology (CRYPTO 2004)*, volume 3152 of *Lecture Notes in Computer Science*, pages 407–425. Springer-Verlag, 2004.

- [LV08] Yi Lu and Serge Vaudenay. Cryptanalysis of an E0-like combiner with memory. *Journal of Cryptology*, 21(3):430–457, 2008.
- [LW05] Ophir Levy and Avishai Wool. Uniform framework for cryptanalysis of the Bluetooth E0 cipher. In *1st International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm 2005)*, pages 365–373. IEEE Computer Society, 2005.
- [LXZ13] Jinlong Lu, Longsen Xu, and Maolin Zhang. Research on contactless IC card simulation technology. In *3rd International Conference on Information Engineering and Applications (IEA 2012)*, volume 216 of *Lecture Notes in Electrical Engineering*, pages 97–102. Springer-Verlag, 2013.
- [Man03] Stefan Mangard. A simple power-analysis (spa) attack on implementations of the aes key expansion. In *5th International Conference on Information Security and Cryptology (ICISC 2002)*, volume 2587 of *Lecture Notes in Computer Science*, pages 343–358. Springer-Verlag, 2003.
- [Man05a] Itsik Mantin. A practical attack on the fixed RC4 in the WEP mode. In *11th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2005)*, volume 3788 of *Lecture Notes in Computer Science*, pages 395–411. Springer-Verlag, 2005.
- [Man05b] Itsik Mantin. Predicting and distinguishing attacks on RC4 keystream generator. In *24th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2005)*, volume 3494 of *Lecture Notes in Computer Science*, pages 491–506. Springer-Verlag, 2005.
- [Mar57] Harry M Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.
- [Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *12th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1993)*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1994.
- [Mat97] Mitsuru Matsui. New block encryption algorithm MISTY. In *4th International Workshop on Fast Software Encryption (FSE 1997)*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 1997.

- [MB01] Sjouke Mauw and Victor Bos. Drawing Message Sequence Charts with \LaTeX . *TUGBoat*, 22(1-2):87–92, March/June 2001.
- [MC01] KeeLoq crypto read/write transponder module, HCS410/WM. Product Datasheet, Jan 2001. Microchip Technology Incorporated.
- [McG02] David McGrew. Counter mode security: Analysis and recommendations. *Cisco Systems, November, 2002*.
- [MDDR04] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, 2004.
- [MDO94] William Millan, EP Dawson, and LJ O'Connor. Fast attacks on tree-structured ciphers. In *1st Workshop in Selected Areas in Cryptography (SAC 1994)*, pages 146–158, 1994.
- [MDS99] Thomas S Messerges, Ezzy A Dabbish, and Robert H Sloan. Investigations of power analysis attacks on smartcards. In *USENIX workshop on Smartcard Technology*. USENIX Association, 1999.
- [Mer10] Milosch Meriac. Heart of darkness - exploring the uncharted backwaters of HID iClass security. In *27th Chaos Computer Congress (27C3)*, December 2010.
- [Mes00] Thomas S Messerges. Using second-order power analysis to attack DPA resistant software. In *2nd International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer-Verlag, 2000.
- [MH81] Ralph C Merkle and Martin E Hellman. On the security of multiple encryption. *Communications of the ACM*, 24(7):465–467, 1981.
- [Mil01] Sandra Kay Miller. Facing the challenge of wireless security. *Computer*, 34(7):16–18, 2001.
- [Mir02] Ilya Mironov. (not so) random shuffles of RC4. In *22nd International Cryptology Conference, Advances in Cryptology (CRYPTO 2002)*, volume 2442 of *Lecture Notes in Computer Science*, pages 304–319. Springer-Verlag, 2002.
- [MJB05] Alexander Maximov, Thomas Johansson, and Steve Babbage. An improved correlation attack on A5/1. In *12th International Workshop on Selected Areas in Cryptography (SAC 2002)*, volume 3897 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2005.

- [MK08] Alexander Maximov and Dmitry Khovratovich. New state recovery attack on RC4. In *28th International Cryptology Conference, Advances in Cryptology (CRYPTO 2008)*, volume 5157 of *Lecture Notes in Computer Science*, pages 297–316. Springer-Verlag, 2008.
- [MM93] Ueli M Maurer and James L Massey. Cascade ciphers: The importance of being first. *Journal of Cryptology*, 6(1):55–61, 1993.
- [MOPS13] Amir Moradi, David Oswald, Christof Paar, and Pawel Swierczynski. Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering. In *International Symposium on Field Programmable Gate Arrays (FPGA 2013)*, pages 91–100. ACM, 2013.
- [MOTW09] H Gregor Molter, Kei Ogata, Erik Tews, and Ralf-Philipp Weinmann. An efficient FPGA implementation for an DECT brute-force attacking scenario. In *5th International Conference on Wireless and Mobile Communications (ICWMC 2009)*, pages 82–86. IEEE Computer Society, 2009.
- [MPC04] Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of boolean functions. In *27th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2004)*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. Springer-Verlag, 2004.
- [MPG11] Subhamoy Maitra, Goutam Paul, and Sourav Sen Gupta. Attack on broadcast RC4 revisited. In *18th International Workshop on Fast Software Encryption (FSE 2011)*, volume 6733 of *Lecture Notes in Computer Science*, pages 199–217. Springer-Verlag, 2011.
- [MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In *7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005)*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer-Verlag, 2005.
- [MS88] Willi Meier and Othmar Staffelbach. Fast correlation attacks on stream ciphers. In *7th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1988)*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer-Verlag, 1988.
- [MS89] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

- [MS00] Rita Mayer-Sommer. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. In *2nd International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 78–92. Springer-Verlag, 2000.
- [MS02] Itsik Mantin and Adi Shamir. A practical attack on broadcast RC4. In *8th International Workshop on Fast Software Encryption (FSE 2001)*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer-Verlag, 2002.
- [MSB⁺06] David Moore, Colleen Shannon, Douglas J Brown, Geoffrey M Voelker, and Stefan Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.
- [MSSS11] Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. Semi-invasive EM attack on FPGA RO PUFs and countermeasures. In *6th Workshop on Embedded Systems Security (WESS 2011)*, pages 1–9. ACM, 2011.
- [MT99] Serge Mister and Stafford E Tavares. Cryptanalysis of RC4-like ciphers. In *5th International Workshop on Selected Areas in Cryptography (SAC 1998)*, volume 1556 of *Lecture Notes in Computer Science*, pages 131–143. Springer-Verlag, 1999.
- [Mul56] David E Muller. A method for solving algebraic equations using an automatic computer. *Mathematical Tables and Other Aids to Computation*, 10(56):208–215, 1956.
- [Mul04] Frédéric Muller. Differential attacks against the Helix stream cipher. In *11th International Workshop on Fast Software Encryption (FSE 2004)*, volume 3017 of *Lecture Notes in Computer Science*, pages 94–108. Springer-Verlag, 2004.
- [Mul09] Collin Mulliner. Vulnerability analysis and attacks on nfc-enabled mobile phones. In *1st International Workshop on Sensor Security (IWSS 2009)*, pages 695–700. IEEE Computer Society, 2009.
- [MV04] David A McGrew and John Viega. The security and performance of the galois/counter mode (GCM) of operation. In *5th International Conference on Cryptology in India, Progress in Cryptology (INDOCRYPT 2004)*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer-Verlag, 2004.
- [MVOV10] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2010.

- [NESP08] Karsten Nohl, David Evans, Starbug, and Henryk Plötz. Reverse engineering a cryptographic RFID tag. In *17th USENIX Security Symposium (USENIX Security 2008)*, pages 185–193. USENIX Association, 2008.
- [NFC06] Technical specification, NFC data exchange format (NDEF), 2006. NFC Forum - NDEF 1.0.
- [NFC10] Technical specification, connection handover, 2010. NFC Forum - Connection Handover 1.2.
- [NFC11] Technical specification, nfc activity specification, 2011. NFC Forum - ACTIVITY 1.0.
- [NFC13] Technical specification, personal health device communication, 2013. NFC Forum - PHDC 1.0.
- [Noh08] Karsten Nohl. Cryptanalysis of crypto-1. *Computer Science Department University of Virginia, White Paper*, 2008.
- [Noh10] Karsten Nohl. Immobilizer security. In *8th International Conference on Embedded Security in Cars (ESCAR 2010)*, 2010.
- [Nov03] Roman Novak. Side-channel attack on substitution blocks. In *1st International Conference on Applied Cryptography and Network Security (ACNS 2003)*, volume 2846 of *Lecture Notes in Computer Science*, pages 307–318. Springer-Verlag, 2003.
- [NP07] Karsten Nohl and Henryk Plötz. Mifare, little security, despite obscurity. *Presentation on the 24th Congress of the Chaos Computer Club in Berlin (24C3)*, December 2007.
- [NTW10] Karsten Nohl, Erik Tews, and Ralf-Philipp Weinmann. Cryptanalysis of the DECT standard cipher. In *17th International Workshop on Fast Software Encryption (FSE 2010)*, volume 6147 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2010.
- [NXP10] Transponder IC, Hitag2. Product Data Sheet, Nov 2010. NXP Semiconductors.
- [NXP11] Hitag pro. Product Data Sheet, 2011. NXP Semiconductors.
- [NXP13] Mifare application directory (MAD). Application note, Jan 2013. NXP Semiconductors.
- [Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *23rd International Cryptology Conference, Advances in Cryptology (CRYPTO 2003)*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer-Verlag, 2003.

- [OMHT06] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical second-order DPA attacks for masked smart card implementations of block ciphers. In *6th Cryptographers' Track at the RSA Conference, Topics in Cryptology (CT-RSA 2006)*, volume 3860 of *Lecture Notes in Computer Science*, pages 192–207. Springer-Verlag, 2006.
- [ORSVH95] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich Von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, 1995.
- [OSS⁺13] David Oswald, Daehyun Strobel, Falk Schellenberg, Timo Kasper, and Christof Paar. When reverse-engineering meets side-channel analysis—digital lockpicking in practice. In *20th International Conference on Selected Areas in Cryptography (SAC 2013)*, *Lecture Notes in Computer Science*. Springer-Verlag, 2013.
- [Pas09] Enes Pasalic. On guess and determine cryptanalysis of LFSR-based stream ciphers. *IEEE Transactions on Information Theory*, 55(7):3398–3406, 2009.
- [PDMS09] Nikolaos Petrakos, George W Dinolt, James Bret Michael, and Pantelimon Stanica. Cube-type algebraic attacks on wireless encryption protocols. *IEEE Computer*, 42(10):103–105, 2009.
- [PEK⁺09] Christof Paar, Thomas Eisenbarth, Markus Kasper, Timo Kasper, and Amir Moradi. Keeloq and side-channel analysis—evolution of an attack. In *6th International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2009)*, pages 65–69. IEEE Computer Society, 2009.
- [Pen96] Walter T Penzhorn. Correlation attacks on stream ciphers: Computing low-weight parity checks based on error-correcting codes. In *3rd International Workshop on Fast Software Encryption (FSE 1996)*, volume 1039 of *Lecture Notes in Computer Science*, pages 159–172. Springer-Verlag, 1996.
- [PFS00] Slobodan Petrovic and Amparo Fuster-Sabater. Cryptanalysis of the A5/2 algorithm. *IACR Cryptology ePrint Archive*, 2000(52):107, 2000.
- [PGV94] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *13th International Cryptology Conference, Advances in Cryptology (CRYPTO 1993)*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer-Verlag, 1994.

- [Pha02] Raphael Chung-Wei Phan. Cryptanalysis of full Skipjack block cipher. *Electronics Letters*, 38(2):69–71, 2002.
- [PHI98] MIFARE Classic 1k, MF1ICS50. Public product data sheet, July 1998. Philips Semiconductors.
- [PHI99] Security transponder plus remote keyless entry – HITAG2 plus, PCF7946AT. Product Profile, Jun 1999. Philips Semiconductors.
- [PK79] Gerald J Popok and Charles S Kline. Encryption and secure computer networks. *ACM Computing Surveys (CSUR)*, 11(4):331–356, 1979.
- [Ple77] Vera S Pless. Encryption schemes for computer confidentiality. *IEEE Transactions on Computers*, 100(11):1133–1136, 1977.
- [PM07] Goutam Paul and Subhamoy Maitra. Permutation after RC4 key scheduling reveals the secret key. In *14th International Workshop on Selected Areas in Cryptography (SAC 2007)*, volume 4876 of *Lecture Notes in Computer Science*, pages 360–377. Springer-Verlag, 2007.
- [PN12] Henryk Plötz and Karsten 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.
- [PP04] Souradyuti Paul and Bart Preneel. A new weakness in the RC4 key-stream generator and an approach to improve the security of the cipher. In *11th International Workshop on Fast Software Encryption (FSE 2004)*, volume 3017 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, 2004.
- [PPPM13] Panagiotis Papantonakis, Dionisios Pnevmatikatos, Ioannis Papaefstathiou, and Charalampos Manifavas. Fast, FPGA-based rainbow table creation for attacking encrypted mobile communications. In *23rd International Conference on Field Programmable Logic and Applications (FPL 2013)*, pages 1–6. IEEE Computer Society, 2013.
- [PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *15th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 1996)*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer-Verlag, 1996.
- [PS00] Thomas Pornin and Jacques Stern. Software-hardware trade-offs: Application to A5/1 cryptanalysis. In *2nd International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 318–327. Springer-Verlag, 2000.

- [QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Smart Card Programming and Security*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer-Verlag, 2001.
- [RCT05] Melanie Rieback, Bruno Crispo, and Andrew 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*, pages 184–194. Springer-Verlag, 2005.
- [Ree77] James Reeds. “Cracking” a random number generator. *Cryptologia*, 1(1):20–26, 1977.
- [Rei62] Edgar C Reinke. Classical cryptography. *The Classical Journal*, 58(3):113–121, 1962.
- [RLS11] Michael Roland, Josef Langer, and Josef Scharinger. Security vulnerabilities of the NDEF signature record type. In *3rd International Workshop on Near Field Communication (NFC 2011)*, pages 65–70. IEEE Computer Society, 2011.
- [RM99] Kristoffer H Rose and Ross Moore. Xy-pic reference manual, 1999.
- [Roo95] Andrew Roos. A class of weak keys in the RC4 stream cipher, 1995.
- [RRST02] Josyula R Rao, Pankaj Rohatgi, Helmut Scherzer, and Stephane Tinguely. Partitioning attacks: or how to rapidly clone some GSM cards. In *23rd IEEE Symposium on Security and Privacy (S&P 1997)*, pages 31–41. IEEE Computer Society, 2002.
- [RSA78] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [RSH⁺12] Amir Rahmati, Mastooreh Salajegheh, Dan Holcomb, Jacob Sorber, Wayne P. Burleson, and Kevin 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⁺01] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San 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.

- [Rub79] Frank Rubin. Decrypting a stream cipher based on J-K flip-flops. *IEEE Transactions on Computers*, 100(7):483–487, 1979.
- [Rue86] Rainer A Rueppel. Stream ciphers. In *Analysis and Design of Stream Ciphers*, pages 5–16. Springer-Verlag, 1986.
- [Rue92] R.A. Rueppel. Stream ciphers. *Contemporary cryptology: The science of information integrity*, pages 65–134, 1992.
- [Rv10] Kasper Bonne Rasmussen and Srdjan Čapkun. Realization of rf distance bounding. In *19th USENIX Security Symposium (USENIX Security 2010)*, pages 389–402. USENIX Association, 2010.
- [SA00] Frank Stajano and Ross J. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *7th International Workshop on Security Protocols (WSP 2000)*, volume 1796 of *Lecture Notes in Computer Science*, pages 172–182. Springer-Verlag, 2000.
- [SA03] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002)*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer-Verlag, 2003.
- [SAHK07] Nobuyuki Sugio, Hiroshi Aono, Sadayuki Hongo, and Toshinobu Kaneko. A study on higher order differential attack of KASUMI. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 90(1):14–21, 2007.
- [Sai11] Teruo Saito. A single-key attack on 6-round KASUMI. *IACR Cryptology ePrint Archive*, 2011(584):1013, 2011.
- [Sch98a] Bruce Schneier. Cryptographic design vulnerabilities. *Computer*, 31(9):29–33, 1998.
- [Sch98b] Bruce Schneier. Security pitfalls in cryptography. In *EDI FORUM-OAK PARK-*, volume 11, pages 65–69. The EDI Group, Ltd., 1998.
- [SDK⁺13] Daehyun Strobel, Benedikt Driessen, Timo Kasper, Gregor Leander, David Oswald, Falk Schellenberg, and Christof Paar. Fuming acid and cryptanalysis: Handy tools for overcoming a digital locking and access control system. In *33rd International Cryptology Conference, Advances in Cryptology (CRYPTO 2013)*, volume 8042 of *Lecture Notes in Computer Science*, pages 147–164. Springer-Verlag, 2013.
- [Sha49] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

- [Sha04] Adi Shamir. Invited talk: Stream ciphers: Dead or alive? In *10th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2004)*, volume 3329 of *Lecture Notes in Computer Science*, page 78. Springer-Verlag, 2004.
- [She94a] SJ Shepherd. An approach to the cryptanalysis of mobile stream ciphers. In *IEE Colloquium on Security and Cryptography Applications to Radio Systems*, volume 1994/141, 1994. (COMMERCIAL-IN-CONFIDENCE).
- [She94b] SJ Shepherd. Cryptanalysis of the GSM A5 cipher algorithm. In *IEE Colloquium on Security and Cryptography Applications to Radio Systems*, volume 1994/141, 1994. (COMMERCIAL-IN-CONFIDENCE).
- [SHXZ11] Siwei Sun, Lei Hu, Yonghong Xie, and Xiangyong 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.
- [Sie84] Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, 30(5):776–780, 1984.
- [Sie85] Thomas Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, 100(1):81–85, 1985.
- [Sim64] William Simon. *Mathematical magic*. Courier Dover Publications, 1964.
- [Sin66] Abraham Sinkov. *Elementary cryptanalysis: a mathematical approach*. Mathematical Association of America, 1966.
- [SIR02] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *9th Network and Distributed System Security Symposium (NDSS 2002)*. The Internet Society, 2002.
- [SIR04] Adam Stubblefield, John Ioannidis, and Aviel D. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP). *ACM Transactions on Information and System Security*, 7(2):319–332, 2004.
- [SKK⁺97] Christoph L Schuba, Ivan V Krsul, Markus G Kuhn, Eugene H Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a denial of service attack on tcp. In *18th IEEE Symposium on Security and Privacy (S&P 1997)*, pages 208–223. IEEE Computer Society, 1997.

- [Sko05] Sergei P Skorobogatov. Semi-invasive attacks - a new approach to hardware security analysis. Technical report, University of Cambridge, Computer Laboratory, 2005.
- [Sko09] Sergei Skorobogatov. Local heating attacks on flash memory devices. In *2nd IEEE International Workshop on Hardware - Oriented Security and Trust (HOST 2009)*, pages 1–6. IEEE Computer Society, 2009.
- [Smi71] John Lynn Smith. The design of lucifer, a cryptographic device for data communications. Technical report, IBM Thomas J. Watson Research Laboratory, 1971.
- [SNC09] Mate Soos, Karsten Nohl, and Claude 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.
- [SS02] Pamela Samuelson and Suzanne Scotchmer. The law and economics of reverse engineering. *Yale Law Journal*, pages 1575–1663, 2002.
- [SSAQ02] David Samyde, Sergei Skorobogatov, Ross Anderson, and J-J Quisquater. On a new way to read data from memory. In *1st International IEEE Security in Storage Workshop (SISW 2002)*, pages 65–69. IEEE Computer Society, 2002.
- [SSB11] Asia Slowinska, Traian Stancescu, and Herbert 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.
- [SSVV13] Pouyan Sepehrdad, Petr Susil, Serge Vaudenay, and Martin Vuagnoux. Smashing WEP in a passive attack. In *20th International Workshop on Fast Software Encryption (FSE 2013)*, volume 8424 of *Lecture Notes in Computer Science*, pages 155–178. Springer-Verlag, 2013.
- [ST02] 13.56 mhz short range contactless memory chip with 4096 bit EEPROM, anti-collision functions and anti-clone functions, SRIX4K. Preliminary Product Datasheet, May 2002. ST Microelectronics.
- [Ste99] Frank A. Stevenson. Cryptanalysis of contents scrambling system (CCS), November 1999.
- [Ste13] Marc Stevens. New collision attacks on SHA-1 based on optimal joint local-collision analysis. In *32nd International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2013)*, volume 7881 of *Lecture Notes in Computer Science*, pages 245–261. Springer-Verlag, 2013.

- [Sti05] Douglas R Stinson. *Cryptography: theory and practice*. CRC press, 2005.
- [Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.
- [Sun94] Tzu Sun. *Art of war*. Basic Books, 1994.
- [SVV11a] Pouyan Sepehrdad, Serge Vaudenay, and Martin Vuagnoux. Discovery and exploitation of new biases in RC4. In *17th International Workshop on Selected Areas in Cryptography (SAC 2010)*, volume 6544 of *Lecture Notes in Computer Science*, pages 74–91. Springer-Verlag, 2011.
- [SVV11b] Pouyan Sepehrdad, Serge Vaudenay, and Martin Vuagnoux. Statistical attack on RC4. In *30th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2009)*, volume 6632 of *Lecture Notes in Computer Science*, pages 343–363. Springer-Verlag, 2011.
- [SW06] Yaniv Shaked and Avishai Wool. Cryptanalysis of the Bluetooth E0 cipher using OBDD’s. In *9th International Conference on Information Security (ISC 2006)*, volume 4176 of *Lecture Notes in Computer Science*, pages 187–202. Springer-Verlag, 2006.
- [SWT01] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *10th USENIX Security Symposium (USENIX Security 2001)*. USENIX Association, 2001.
- [TB09] Erik Tews and Martin Beck. Practical attacks against WEP and WPA. In *2nd ACM Conference on Wireless Network Security (WISEC 2009)*, pages 79–86. ACM, 2009.
- [Tec05] Constructivecard Technologies. A review of the cryptomemory algorithm performance, jan 2005.
- [Tew12] Erik Tews. *DECT Security Analysis*. PhD thesis, Doctoral Dissertation, May 2012, TU Darmstadt, 2012.
- [TI04] Radio frequency identification systems – digital signature transponder plus, DST+. Product Specification, July 2004. Texas Instruments Incorporated.
- [Tre08] Jan 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.
- [TT80] Moiez A. Tapia and Jerry H. Tucker. Complete solution of boolean equations. *IEEE Transactions on Computers*, 100(7):662–665, 1980.

- [TWP07] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit WEP in less than 60 seconds. In *8th International Workshop on Information Security Applications (WISA 2007)*, volume 4867 of *Lecture Notes in Computer Science*, pages 188–202. Springer-Verlag, 2007.
- [vdBP13] Fabian van den Broek and Erik Poll. A comparison of time-memory trade-off attacks on stream ciphers. In *6th International Conference on Cryptology in Africa, Progress in Cryptology (AFRICACRYPT 2013)*, volume 7918 of *Lecture Notes in Computer Science*, pages 406–423. Springer-Verlag, 2013.
- [VdKGG12] Roel Verdult, Gerhard de Koning Gans, and Flavio D. Garcia. A toolbox for RFID protocol analysis. In *4th International EURASIP Workshop on RFID Technology (EURASIP RFID 2012)*, pages 27–34. IEEE Computer Society, 2012.
- [VDWKP09] Gauthier Van Damme, Karel M. Wouters, Hakan Karahan, and Bart Preneel. Offline NFC payments with electronic vouchers. In *1st ACM Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld 2009)*, pages 25–30. ACM, 2009.
- [Ver08a] Roel Verdult. Proof of concept, cloning the OV-chip card. Technical report, Radboud University Nijmegen, 2008.
- [Ver08b] Roel Verdult. Security analysis of RFID tags. Master’s thesis, Radboud University Nijmegen, 2008.
- [VGB12] Roel Verdult, Flavio D. Garcia, and Josep Balasch. Gone in 360 seconds: Hijacking with Hitag2. In *21st USENIX Security Symposium (USENIX Security 2012)*, pages 237–252. USENIX Association, 2012.
- [VGE13] Roel Verdult, Flavio D. Garcia, and Barış Ege. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *Presentation at 22nd USENIX Security Symposium (USENIX Security 2013)*. USENIX Association, 2013.
- [VK11] Roel Verdult and François 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] Petr Štembera and Martin Novotný. Breaking Hitag2 with reconfigurable hardware. In *14th Euromicro Conference on Digital System Design (DSD 2011)*, pages 558–563. IEEE Computer Society, 2011.
- [VOW96] Paul C Van Oorschot and Michael J Wiener. Improving implementable meet-in-the-middle attacks by orders of magnitude. In *16th International Cryptology Conference, Advances in Cryptology (CRYPTO*

- 1996), volume 1109 of *Lecture Notes in Computer Science*, pages 229–236. Springer-Verlag, 1996.
- [VOW99] Paul C Van Oorschot and Michael J Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.
- [VV07] Serge Vaudenay and Martin Vuagnoux. Passive-only key recovery attacks on RC4. In *14th International Workshop on Selected Areas in Cryptography (SAC 2007)*, volume 4876 of *Lecture Notes in Computer Science*, pages 344–359. Springer-Verlag, 2007.
- [Wag99] David Wagner. The boomerang attack. In *6th International Workshop on Fast Software Encryption (FSE 1999)*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer-Verlag, 1999.
- [WBD74] Edward L Wilson, Klaus-Jürgen Bathe, and William P Doherty. Direct solution of large systems of linear equations. *Computers & Structures*, 4(2):363–372, 1974.
- [WCAA00] DJ Weaver, JRA Cleaver, L Avery, and H Ahmed. Multilayer integrated-circuit imaging with contrast enhancement in a large-area, high-resolution electron-beam system. *Microelectronic engineering*, 53(1):641–644, 2000.
- [Wei00] Steve H Weingart. Physical security devices for computer subsystems: A survey of attacks and defenses. In *2nd International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000)*, volume 1965 of *Lecture Notes in Computer Science*, pages 302–317. Springer-Verlag, 2000.
- [WFY⁺02] Dai Watanabe, Soichi Furuya, Hirotaka Yoshida, Kazuo Takaragi, and Bart Preneel. A new keystream generator mugl. In *9th International Workshop on Fast Software Encryption (FSE 2002)*, volume 2365 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 2002.
- [WGB98] David Wagner, Ian Goldberg, and Marc Briceno. Gsm cloning. *Web page about COMP-128 version*, 1, 1998.
- [WHF02] Doug Whiting, Russ Housley, and Niels Ferguson. Aes encryption & authentication using CTR mode & CBC-MAC. *IEEE P802*, 11, 2002.
- [WHWC07] Pang-Chieh Wang, Ting-Wei Hou, Jung-Hsuan Wu, and Bo-Chiuan Chen. A security module for car appliances. *International Journal of World Academy Of Science, Engineering and Technology*, 26:155–160, 2007.

- [Wie90] JM Wiesenfeld. Electro-optic sampling of high-speed devices and integrated circuits. *IBM Journal of Research and Development*, 34(2.3):141–161, 1990.
- [Wie07] I.C. Wiener. Philips/NXP Hitag2 PCF7936/46/47/52 stream cipher reference implementation. <http://cryptolib.com/ciphers/hitag2/>, 2007.
- [WIF03] Wi-Fi protected access: Strong, standards-based, interoperable security for today’s Wi-Fi networks, 2003.
- [WKR⁺08] Jung-Hsuan Wu, Chien-Chuan Kung, Jhan-Hao Rao, Pang-Chieh Wang, Cheng-Liang Lin, and Ting-Wei Hou. Design of an in-vehicle anti-theft component. In *8th International Conference on Intelligent Systems Design and Applications (ISDA 2008)*, volume 1, pages 566–569. IEEE Computer Society, 2008.
- [WMT⁺13] Michael Weiner, Maurice Massar, Erik Tews, Dennis Giese, and Wolfgang Wieser. Security analysis of a widely deployed locking system. In *20th ACM Conference on Computer and Communications Security (CCS 2013)*, pages 929–940. ACM, 2013.
- [WP07] Hongjun Wu and Bart Preneel. Differential-linear attacks against the stream cipher Phelix. In *14th International Workshop on Fast Software Encryption (FSE 2007)*, volume 4593 of *Lecture Notes in Computer Science*, pages 87–100. Springer-Verlag, 2007.
- [WS02] Anthony D Wood and John A Stankovic. Denial of service in sensor networks. *Computer*, 35(10):54–62, 2002.
- [WSD⁺99] David Wagner, Leone Simpson, Ed Dawson, John Kelsey, William Millan, and Bruce Schneier. Cryptanalysis of ORYX. In *5th International Workshop on Selected Areas in Cryptography (SAC 1998)*, volume 1556 of *Lecture Notes in Computer Science*, pages 631–631. Springer-Verlag, 1999.
- [WSK97] David Wagner, Bruce Schneier, and John Kelsey. Cryptanalysis of the cellular message encryption algorithm. In *17th International Cryptology Conference, Advances in Cryptology (CRYPTO 1997)*, volume 1294 of *Lecture Notes in Computer Science*, pages 526–537. Springer-Verlag, 1997.
- [WSvRG⁺08] Ronny Wichers Schreur, Peter van Rossum, Flavio D. Garcia, Wouter Teepe, Jaap-Henk Hoepman, Bart Jacobs, Gerhard de Koning Gans, Roel Verdult, Ruben Muijers, Ravindra Kali, and Vinesh Kali. Security flaw in MIFARE Classic. *Press release, Digital Security group, Radboud University Nijmegen, The Netherlands*, March 2008.

- [WTHH11] Michael Weiner, Erik Tews, Benedikt Heinz, and Johann Heyszl. Fpga implementation of an improved attack against the DECT standard cipher. In *13th International Conference on Information Security and Cryptology (ICISC 2010)*, volume 6829 of *Lecture Notes in Computer Science*, pages 177–188. Springer-Verlag, 2011.
- [WWW07] Marko Wolf, Andre Weimerskirch, and Thomas Wollinger. State of the art: Embedding security in vehicles. *EURASIP Journal on Embedded Systems*, 2007:074706, 2007.
- [WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *24th International Conference on the Theory and Application of Cryptographic Techniques, Advances in Cryptology (EUROCRYPT 2005)*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, 2005.
- [XHW94] SB Xu, DK He, and XM Wang. An implementation of the GSM general data encryption algorithm A5. In *Advanced in Cryptology (CHINACRYPT 1994)*, volume 94, pages 287–291, 1994.
- [ZF06] Bin Zhang and Dengguo Feng. New guess-and-determine attack on the self-shrinking generator. In *12th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology (ASIACRYPT 2006)*, volume 4284 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 2006.
- [ZJS⁺09] Yang Zhenye, Yang Jiexue, Wang Songming, Hong Jiexin, and Chen Kuncheng. New method of hardware encryption against piracy. In *Information Technology and Applications (IFITA 2009)*, volume 2, pages 737–739. IEEE Computer Society, 2009.
- [ZXF13] Bin Zhang, Chao Xu, and Dengguo Feng. Real time cryptanalysis of Bluetooth encryption with condition masking. In *33rd International Cryptology Conference, Advances in Cryptology (CRYPTO 2013)*, volume 8042 of *Lecture Notes in Computer Science*, pages 165–182. Springer-Verlag, 2013.
- [ZYSQ13] Yuanyuan Zhou, Yu Yu, François-Xavier Standaert, and Jean-Jacques Quisquater. On the need of physical security for small embedded devices: a case study with COMP128-1 implementations in SIM cards. In *17th International Conference on Financial Cryptography and Data Security (FC 2013)*, volume 7859 of *Lecture Notes in Computer Science*, pages 230–238. Springer-Verlag, 2013.