Cryptanalysis of Lightweight Ciphers

Borghoff, Julia

Publication date:
2011

Document Version
Early version, also known as pre-print

Technical University of Denmark
DTU Mathematics
Department of Mathematics

Cryptanalysis of Lightweight Ciphers

Julia Borghoff

Ph.D. Thesis

Kongens Lyngby
December 2010

Date Julia Borghoff

Technical University of Denmark
Department of Mathematics
Matematiktorvet Building 303S
DK-2800 Kongens Lyngby
Denmark
Phone +45 45253031
Fax +45 45881399
www.mat.dtu.dk

Title of Thesis:
Cryptanalysis of Lightweight Ciphers

Ph.D. student:
Julia Borghoff
Department of Mathematics
Technical University of Denmark
Address: Matematiktorvet, DTU Building 303 S, DK-2800 Lyngby, Denmark
E-mail: J.Borghoff@mat.dtu.dk

Supervisors:
Lars Ramkilde Knudsen
Department of Mathematics
Technical University of Denmark
Address: Matematiktorvet, DTU Building 303 S, DK-2800 Lyngby, Denmark
E-mail: Lars.R.Knudsen@mat.dtu.dk

Censors:
Peter Beelen, DTU mathematics
Thomas Johansson, Lund University
Vincent Rijmen, KU Leuven

Summary

Lightweight encryption denotes a class of cryptographic algorithms that are suitable for extremely resource constrained environments and offer a moderate security level. The demand for such lightweight encryption algorithms increases because small computing devices such as RFID tags become more and more popular and establish a part of the pervasive communication infrastructure. But this extensive employment of computing devices is not only convenient, it also carries security risks.

Lightweight encryption algorithms can be divided into two classes: lightweight block ciphers and stream ciphers. Before a new algorithm can be deployed an extensive assessment and analysis of its security is necessary. In this thesis we focus on the cryptanalysis of lightweight encryption schemes and we consider the two block cipher C2 and Maya, as well as the stream cipher Trivium. We start with an introduction to block ciphers and stream ciphers and give an overview of the most general techniques in cryptanalysis. Furthermore, we investigate block ciphers with secret components. The idea of using secret components is to increase the security of the cipher. We present a cryptanalysis of the cipher C2 where we apply a newly developed technique for recovering the secret S-box, and a cryptanalysis of the PRESENT-like cipher Maya, which involves a differential-style attack. In the analysis of the stream cipher Trivium we combine optimization techniques with cryptanalysis. In this new direction of research we examine the use of mixed-integer optimization and neighborhood search algorithms such as simulated annealing for solving non-linear Boolean equation systems.

Resumé (in Danish)

Letvægtskryptering betegner en klasse af kryptografiske algoritmer, som er specielt egnede til miljøer med ekstremt begrænsede ressourcer, og som giver et moderat sikkerhedsniveau. Efterspørgslen efter sådanne letvægtskrypteringsalgoritmer vokser, fordi mikro-computere, såsom RFID-tags, bliver mere og mere populære som en del af vores voksende kommunikationsinfrastruktur. Denne omfattende brug af mikro-computere er ikke kun praktisk, men medfører også potentielle sikkerhedsrisici.

Letvægtskrypteringsalgoritmer kan inddeles i to klasser: letvægtsblokchifre og strømchifre. Før ibrugtagning skal en ny algoritme underkastes en omfattende sikkerhedsvurdering og -analyse. I denne afhandling fokuserer vi på kryptoanalyse af letvægtskrypteringalgoritmer, og vi undersøger de to blokchifre C2 og Maya samt strømchifferet Trivium. Vi begynder med en introduktion til blokchifre og strømchifre og giver et overblik over de mest generelle metoder til kryptoanalyse. Desuden undersøger vi blokchifre med hemmelige komponenter, hvis formål er at øge chifferets sikkerhed. Vi præsenterer en kryptoanalyse af chifferet C2, hvor vi anvender en nyudviklet teknik til at finde den hemmelige S-box, og en kryptoanalyse, som inkluderer et differentielt angreb, af det Present-lignende chiffer Maya. I analysen af strømchifferet Trivium kombinerer vi optimeringsmetoder med kryptoanalyse. I denne nye forskningsretning undersøger vi brugen af blandet-heltalsoptimering og omegns-søgealgoritmer såsom "simulated annealing" til løsning af ikke-lineære boolske ligningssystemer.

Preface

Most people use cryptography several times a day, often without being aware of it. Home banking, the Internet, mobile phones, NemID or access cards are just some examples. The digital world becomes more and more important in our lives, therefore it also becomes more and more important that our transactions in that world are secure such that nobody can manipulate our data, eavesdrop on our communications etc. Mathematical algorithms called cryptographic primitives found the basis of the security. Any trapdoor in these primitives threatens the security of the application. Therefore it is important to analyze them carefully before they are employed in practice. This analysis is called cryptanalysis.

Cryptanalysis advances by discovering unforeseen and unexpected structure of cryptographic problems. Such structural properties can often be exploited to decrease the complexity of breaking the cryptosystem below the designated security level.

In this thesis we investigate in particular the security of lightweight encryption schemes. These are cryptographic algorithms which can be employed in resource constrained environments. We try to identify certain properties in the different algorithms which can be exploited to break the ciphers.

In **Part I** we give a short introduction into symmetric cryptographic primitives and their cryptanalysis.
Chapter 1 provides a motivation and a general introduction to the field of cryptography. We introduce the cryptographic concepts such as symmetric and public-key encryption and define terminology such as security definitions, attack goals and scenarios.

Chapter 2 introduces the cryptographic primitives called block ciphers. We provide a short description of both Feistel ciphers and substitution-permutation networks, and introduce the ciphers C2 and PRESENT as examples of these design concepts.

Chapter 3 gives a short overview of stream cipher design. We compare the two most common stream cipher designs, synchronous and self-synchronizing stream ciphers and afterwards we focus on the design of synchronous stream ciphers. We discuss the properties of building blocks such as LFSRs and sketch designs based on them. After a short discussion of security considerations the chapter concludes with a description of the stream cipher Trivium.

Chapter 4 provides an introduction to cryptanalysis applied to block and stream ciphers. We give a short description of generic attacks such as the exhaustive key search, the table look-up attack, the dictionary attack, and the time-memory trade-offs. These attacks set the bounds on the security of block and stream ciphers. Subsequently, we present the two most important techniques in the cryptanalysis of symmetric encryption schemes: differential and linear cryptanalysis. Additionally, we describe the

boomerang and the cube attacks as variants of the classical differential attack. The chapter concludes with algebraic attacks. Algebraic representations of symmetric encryption scheme are the basis for the novel cryptanalysis presented in Part III.

In **Part II** we examine the security of block ciphers with secret components where we in particular consider block ciphers with secret S-boxes.
Chapter 5 is concerned with the cryptanalysis of the cipher C2. C2 is 64-bit block Feistel cipher with a 56-bit key. The 8-bit S-box is application-dependent and kept secret. We show a trial-and error attack which recovers the S-box in only 2^{24} queries to the device and present a boomerang attack to determine the secret key. The differential used in the attack is independent of the S-box. These two attacks can be combined to an attack that enables us to recover the key and the S-box at the same time. This chapter is based on [22].

Chapter 6 addresses the cryptanalysis of PRESENT-like ciphers with secret S-boxes. We present a new differential-style attack which enables us to recover the S-boxes of the first round of encryption. Furthermore, we show that an S-box can be uniquely determined if we know all sets of input pairs, which lead to an output difference of Hamming weight one, for the S-box and its inverse. The attack is successfully carried out on the cipher Maya and we are able to break the full version consisting of 16 rounds with a practical complexity. Based on a mathematical model we infer that our attack can be successfully applied to ciphers with a randomly chosen S-box layer for up to 28 rounds, which is better than the best attack known on PRESENT. This chapter is based on [24].

In **Part III** we investigate the use of optimization methods in cryptanalysis. Our main target is the stream cipher Trivium, because its internal state can be described by a very sparse quadratic Boolean equation system.
Chapter 7 provides a short introduction to optimization with emphasis on mixed-integer optimization and generalized hill climbing methods such as simulated annealing. As optimization is a large research field we restrict ourselves to the concepts of optimization which required in later chapters.

Chapter 8 describes several representations of Boolean functions as real-valued polynomials and the corresponding conversion methods. Next to the four commonly known representation (the standard, the dual, the sign, and the Fourier representation) which can be applied to any Boolean function, we also introduce conversion methods which only apply to Boolean functions in algebraic normal form. While the adapted standard conversion converts a Boolean equation into a set of equations over the reals, the integer adapted standard conversion converts a Boolean equation into an equation over the integers. The integer adapted standard conversion was introduced in [28].

Chapter 9 presents the state recovery problem of Trivium as a mixed-integer linear programming problem. More precisely, we explain how to convert the Boolean equa-

tion system describing the small scale variant of Trivium called Bivium A into a set of constraints that describes the feasible set of a mixed-integer linear programming problem. Experiments on variants with smaller state size are carried out to find the optimal settings for the mixed-integer programming problem, such as the objective function, binary constraints on the variables and the number of observed keystream bits. The approach breaks Bivium A in about 4.5 hours and Bivium B in $2^{63.7}$ seconds. The results have been published in [28].

Chapter 10 explains how the state recovery problem of Trivium can be presented as a discrete optimization problem. As the equation system given by Trivium exhibits properties such as sparsity and variable locality we believe that the problem is suitable for simulated annealing. We present an improved simulated annealing variant which depends on a new parameter called nochangebound. Applied to Trivium this variant shows a significant improvement in the complexity compared to the classical simulated annealing. We are able to recover the initial state of Trivium in 2^{210} bit flips. Parts of this chapter will be published in [26].

Chapter 11 summarizes the results of this thesis.

The work presented in this thesis was performed at the Department of Mathematics, Technical University of Denmark, in partial fulfillment of the requirement for acquiring the PhD degree. The author was funded by a grant from the Danish research council for Technology and Production Sciences grant number 274-07-0246 and supervised by Professor Lars Ramkilde Knudsen.

The thesis describes the work done by the author during her PhD studies from January 2008 to December 2010. The contribution is presented in Chapters 5 (where the author in particular worked on the search for differentials and the S-box recovery attack), 6 (where the author focused in particular on the mathematical model), 9 and 10. During the three years of PhD studies the following papers were published:

Borghoff, J., Knudsen, L. R., Leander, G., and Matusiewicz, K. Cryptanalysis of C2. In *Advances in Cryptology – CRYPTO 2009,* (2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, pp. 250–266

Borghoff, J., Knudsen, L. R., and Stolpe, M. Bivium as a mixed-integer linear programming problem. In *Cryptography and Coding, 12th IMA International Conference* (2009), M. G. Parker, Ed., vol. 5921 of *Lecture Notes of Computer Science*, Springer, pp. 133–152

Borghoff, J., Knudsen, L. R., and Matusiewicz, K. Hill climbing algorithms and Trivium. to appear in Proceedings of Selected Areas in Cryptology 2010, Lecture notes in Computer Science, 2010

The following paper has been submitted and is at the time of writing awaiting notification:

Borghoff, J., Knudsen, L. R., Leander, G., and Thomsen, S. S. Cryptanalysis of PRESENT-like ciphers with secret s-boxes. submitted to FSE 2011, 2010

Further publication by the author:

Borghoff, J., Knudsen, L. R., Leander, G., and Matusiewicz, K. Cryptanalysis of C2 – extended abstract. Pre-proceedings of WEWoRC 2009, Graz, 2009

Borghoff, J., Knudsen, L. R., and Stolpe, M. Bivium as a mixed-0-1 programming problem – extended abstract. Pre-proceedings of WEWoRC 2009, Graz, 2009

Borghoff, J., Knudsen, L. R., and Matusiewicz, K. Analysis of Trivium by a simulated annealing variant. Proceedings of Ecrypt II Workshop on Tools for Cryptanalysis 2010, Egham, 2010

Acknowledgments

I would like to take the opportunity to thank the people who supported and accompanied me during my PhD studies.

First and foremost I would like to thank my supervisor Lars Ramkilde Knudsen. Thank you for posing a very interesting research problem to me, introducing me to the international community of crytologists and sharing your knowledge. I believe that I became a better researcher and a stronger personality under your supervision and I am grateful for that. I would like to say thank you for having an open door for all my question, for our discussions, and for your comments on my work. I appreciate that you gave me the freedom to do the research I liked and supported me when necessary and finally I would like to thank you for staying calm whenever I lost my head.

I am especially grateful to Krystian Matusiewicz and Mathias Stolpe. I would like to thank you, Krystian, for your endless patience when answering my questions, for sharing your expertise, for your positive attitude and motivation when our research seemed to be stuck and for your always friendly smile. Krystian, I learned a lot from you. I also give my deep-rooted thanks to Mathias. Thank your for your open mind towards our collaboration, our discussions, for introducing me to different methods in the field of optimization and for giving me the push in the right direction. My work would not have been as successful without you.

Next I would like to say thank you to the two of my co-authors I have not mentioned yet, Gregor Leander and Søren Steffen Thomsen. I truly enjoyed working with you. Furthermore I would like to thank all members of our discrete mat group, who make the department a really nice place to work: Krystian, Charlotte, Kristian and Diego (who are not longer with the department), Lars, Gregor, Erik, Søren, Praveen, Valerie, Mohamed, Hoda, Tom, Peter, Johan, Carsten and also Martin and Nasour (who were visiting the crypto group). Thank you all for broadening my knowledge, for the serious and the not so serious lunch discussions and for an always open ear. Sincerely thanks to Tom Høholdt for taking good care of me, being always concerned about my well-being and the attempts to teach me Danish. I am truly grateful. Furthermore I would like to say thank you to Charlotte Vikkelsø Miolane, not only for all the helpful comments on my thesis but also for support and friendship especially in the first year of my PhD.

I would also like to thank all my PhD colleagues. Thanks for the fun at our trips, for the meaningful and in particular for the senseless discussions during our coffee breaks and lunches, thanks for the international encounters which really broadened my mind and thanks for the friendship.

It has been a pleasure to be part of DTU mathematics the last three years. I enjoyed the friendly atmosphere, in particular I would like to highlight the PhD trips, the social clubs especially the beer club, and the Christmas lunches.

I would also like to thank the members of COSIC at the KU Leuven, in particular my host Vincent Rijmen, for welcoming me in their group, inspiring seminars,

delightful coffee breaks and giving me a proper desk. My stay in Leuven was a great experience I will never forget.

During my PhD studies I was part of an international project in cooperation with the TU Graz and KU Leuven. I would like to say thank you to Tomislav Nad, Elmar Tischhauser, Mario Lamberger, Vincent, Krystian and Lars for all the nice meetings and the inspiring discussions and presentations.

Furthermore my heartfelt thanks to Anders Astrup Larsen for patiently proofreading my thesis, giving me helpful comments and cheering me up in the stressful times towards the end.

And last but not least I should say thanks to Volker Krummel for pointing me to the job and encouraging me to apply which was the starting point of my Danish adventure.

Kgs. Lyngby, December 2010
Julia Borghoff

Contents

I Introduction to Symmetric Cryptanalysis **5**

1 Introduction **7**
1.1 Symmetric Encryption 8
1.1.1 Public-key Encryption versus Symmetric Encryption 9
1.1.2 Block and Stream Ciphers 10
1.2 Security Considerations 10
1.2.1 Security Measurements 11
1.2.2 The Adversary's Goals 13
1.2.3 Classification of Attack Scenarios 14

2 Block Ciphers **17**
2.1 Feistel Ciphers 17
2.1.1 The Cipher C2 19
2.2 Substitution-Permutation Networks 20
2.2.1 The Block Cipher PRESENT 22

3 Stream Ciphers **25**
3.1 The General Structure of Stream Ciphers 26
3.1.1 Synchronous Stream Ciphers 27
3.1.2 Self-Synchronizing Stream Ciphers 28
3.2 Design and Building Blocks 29
3.2.1 Linear Feedback Shift Registers 30
3.2.2 LFSR-based Stream Ciphers 32
3.2.3 Non-Linear Feedback Shift Register 34
3.3 Security Considerations 34
3.4 Trivium 35
3.4.1 Keystream Generation 35
3.4.2 Key Setup 36
3.4.3 Cryptanalytic Results on Trivium 36
3.4.4 Bivium 37
3.4.5 Trivium as an Equation System 39

4 Classical Cryptanalysis **41**
4.1 Exhaustive Key Search 41
4.2 Table Look-up Attack 42
4.3 Dictionary Attack 42
4.4 Cryptanalytic Time-Memory Trade-off 42
4.4.1 Time-Memory Trade-off Attack for Block Ciphers 43
4.4.2 Time-Memory-Data Trade-off for Stream Ciphers 45
4.5 Differential Cryptanalysis 46

4.5.1 The Probability of a Differential Characteristic 50
4.5.2 Iterative Characteristics 52
4.5.3 Key Recovery 53
4.5.4 The Boomerang Attack 55
4.5.5 Cube Attacks 57
4.6 Linear Cryptanalysis 60
4.6.1 Key Recovery 62
4.6.2 The Probability of a Linear Characteristic 63
4.7 Algebraic Attacks 65

II Cryptanalysis of Block Ciphers with Secret Components 67

5 Cryptanalysis of C2 69
5.1 Recovering the Secret S-box with a Chosen Key Attack 69
5.1.1 Generating Plaintexts that Fit for Seven Rounds 71
5.1.2 A Three-Round Test 73
5.2 Search for S-box Independent Characteristics 75
5.3 Key-Recovery Attack for a Known S-box 76
5.3.1 Recovering Bits of the First Round Key 77
5.4 Key and S-box Recovery with Chosen Ciphertext Attack 80
5.4.1 Recovering Remaining Unknown Round Key Bits 81
5.4.2 Attacking the Second Round 82
5.5 Conclusion 83

6 Cryptanalysis of PRESENT-like Ciphers with Secret S-boxes 85
6.1 PRESENT-like Ciphers 86
6.2 The Principle of the Attack 87
6.2.1 S-box Recovery given Slender Sets 89
6.2.2 Generalizing to All S-boxes and their Inverses 91
6.2.3 Partitioning Pairs into Sets 91
6.2.4 Filtering Methods 92
6.2.5 Relaxed Truncated Differentials 93
6.3 The Attack in Practice 94
6.3.1 Data Collection Phase 94
6.3.2 S-box Recovery Phase 94
6.4 Case study: the Block Cipher Maya 96
6.5 Model for the Complexity of Recovering Sets D_e 99
6.6 Fully random PRESENT-like ciphers 102
6.7 Linear-style Attack 103
6.8 Conclusion 104

III Cryptanalysis by Optimization 107

7 Optimization 109
7.1 Mixed-Integer Optimization 110
7.1.1 Modeling 113
7.1.2 Algorithms 114
7.2 Generalized Hill Climbing Algorithms 123
7.2.1 Simulated Annealing 124

8 Conversion Methods 127
8.1 Representations of Boolean Functions as Polynomials over the Reals . 128
8.2 Conversion Methods for Boolean Functions in Algebraic Normal Form 133
8.2.1 The Integer Adapted Standard Conversion Method 135

9 Bivium as a Mixed Integer Programming Problem 137
9.1 Bivium A as a Mixed-Integer Linear Programming Problem 138
9.1.1 Linearization Using the Standard Conversion Method 138
9.1.2 Linearization Using the Integer Adapted Standard Conversion Method 140
9.1.3 Bivium A as a Feasibility Problem 141
9.2 Results 142
9.2.1 Parameters Using the Standard Conversion Method 142
9.2.2 Results on Bivium A Using Standard Conversion 147
9.2.3 Results on Bivium B Using the Standard Conversion 148
9.2.4 Results Using the Integer Adapted Standard Conversion 150
9.3 Possible Improvements through Extra Constraints 151
9.3.1 AND-gate Constraints 151
9.3.2 CNF Constraints 152
9.4 Conclusion 154

10 Hill Climbing Algorithms and Trivium 155
10.1 Trivium as a Discrete Optimization Problem 156
10.2 Properties of Trivium 157
10.3 Solving the Trivium Systems with Modified Simulated Annealing . . . 158
10.4 Experimental Results 159
10.5 Some Variations 163
10.5.1 Guessing Strategies 163
10.5.2 Overdetermined System of Equations 168
10.5.3 Variable Persistence 170
10.6 Conclusion and Future Directions 171

11 Conclusion 173

A Five-round Characteristics for C2 177

B A Short Analysis of the Trivium Equation System **179**

C CNF for the Bivium B Keystream Equation **187**

Part I

Introduction to Symmetric Cryptanalysis

Chapter 1

Introduction

The name cryptography comes from the Greek words "kryptos" which means "hidden, secret" and "graph" which means "writing" and is the science of hiding information. Some experts claim that cryptography was developed spontaneously after writing was invented. However, the first documented form of cryptography dates back to 1900 B.C. when an Egyptian used non-standard hieroglyphs in an inscription. One of the most famous historic ciphers is the Caesar cipher, named after Julius Caesar. In order to protect messages with military content against his enemies he wrote his messages with an alphabet shifted by three positions, e.g., he wrote a "D" instead of an "A". This form of simple encryption prevented Caesar's enemies from reading his messages. Nowadays, where information technologies become more and more important in our society, cryptography has more objectives than merely providing confidentiality.

The development of telecommunication systems and digital information opened a wide range of new possibilities. Billions of people are connected through the Internet or digital mobile networks and exchange large amounts of data and sensitive information over those networks. Digital information has the advantage that it can be transmitted in different ways and easily copied, meaning that one can store the exact same information on a USB-stick or on a hard drive, send it over a wireless network or transmit it over an optical fiber. The transmission of data is fast and easy. The advantages of digital information and networks such as the Internet and mobile networks are obvious, but what is the drawback compared to an old-fashion letter? A receiver of a letter can check if the letter and therefore the information has been compromised. A handwritten signature does not only verify the origin of the letter, it also prevents the sender from later denying that he sent the message. An adversary, who reads the letter, has to open a sealed envelope. If he wants to change the information he has to rewrite the letter. Both opening the envelope and changing information will leave traces. This is not the case for digital data. Therefore cryptography has to provide the characteristics a letter automatically provides. Providing confidentiality is still the first objective of cryptography that comes to mind. But also authentication, data integrity and non-repudiation are important objectives of cryptography and characteristics which are also provided by a letter. This list is by no means complete. The reader is referred to [82] for more information.

Confidentiality When transmitting a message over an insecure channel only the intended recipient should be able to read the message. An unauthorized eavesdropper should not get any information about the content of the message. Also stored data should be protected against unauthorized access.

Authentication The receiver of a message wants to verify its origin, i.e., the receiver wants to make sure that the message comes from a certain sender and not from somebody else.

Data integrity The receiver can verify that the data was not modified (neither accidentally nor on purpose) during the transmission.

Non-Repudiation The sender should not be able to later deny that he sent a message.

In this thesis we focus on symmetric encryption schemes which are supposed to provide confidentiality.

1.1 Symmetric Encryption

The general setting is that two parties - a sender, called Alice, and a receiver, called Bob, - want to communicate over an insecure channel without allowing an eavesdropper, called Eve, to obtain any information about their conversation. We assume that beforehand Alice and Bob have the possibility to exchange a small amount of information, the secret key, over a secure channel. The purpose of the encryption algorithms is to protect the secrecy of the message transmitted over an insecure channel.

In general a cryptosystem, also called an encryption scheme, consists of two functions. An encryption function E, which takes the message or plaintext p as input and transforms it into the corresponding ciphertext $c = E(p)$, and a decryption function $D = E^{-1}$, which inverts the encryption function. Thus, it holds that $D(c) = p$.

In order to provide confidentiality the encryption function should be designed such that Eve cannot deduce any information about the plaintext from an intercepted ciphertext and the decryption function should be kept secret. As the encryption and decryption functions are often similar and can be deduced from each other, keeping D secret also implies keeping E secret. However, keeping a whole algorithm secret is from the practical point of view never a good idea because it means that several algorithms are needed for different communication partners [41]. The idea to overcome this problem is to construct a parametrized encryption algorithm. In such an encryption scheme the encryption and decryption function have an additional secret parameter k as input such that $D_{k'}(E_k(p))$ for $k' \neq k$ does not reveal any information about the plaintext p. The parameter k is called the secret key and is usually a bitstring of length ranging from 56 to a couple of hundred bits which is shared by the parties beforehand over a secure channel. The secret key is also referred to as their common secret. Alice encrypts the message p under Bob's and her shared, secret key k and sends the resulting ciphertext $c = E_k(p)$ over the insecure channel. After receiving the ciphertext c Bob applies the decryption algorithm with the key k and ciphertext c as inputs and obtains the plaintext $p = D_k(c)$ as depicted in Figure 1.1. Formally, we define a symmetric encryption scheme as follows.

Definition 1.1 (Symmetric cryptosystem [97])**.**
A symmetric cryptosystem, also called a symmetric encryption scheme, is a five-tuple

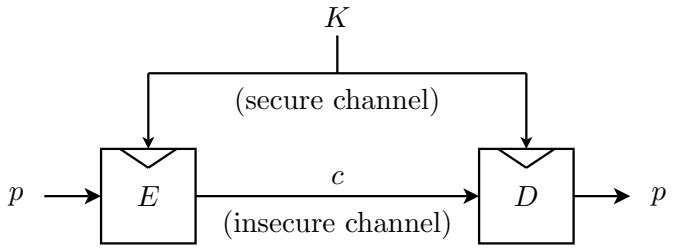


Figure 1.1: Symmetric encryption

$(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ where $\mathcal{P}$ is the finite set of plaintexts, $\mathcal{C}$ is the finite set of ciphertexts and $\mathcal{K}$ is the key space. For each key $k \in \mathcal{K}$ there is an encryption function $E_k \in \mathcal{E}$ with respect to k,

$$E_k \quad : \quad \mathcal{P} \to \mathcal{C}$$

and a corresponding decryption function $D_k \in \mathcal{D}$,

$$D_k : \mathcal{C} \to \mathcal{P}$$

such that $D_k(E_k(p)) = p$ for all plaintexts $p \in \mathcal{P}$.

The security of an encryption scheme should depend solely on the secret key. This assumption, which builds the basis of modern cryptography, was first made by the Dutch cryptographer Auguste Kerckhoffs in 1883 [64] and is generally referred to as Kerckhoffs' principle.

Theorem 1.2 (Kerckhoffs' principle)**.**
A cryptosystem should be secure even if everything about the cipher, except the secret key, is public.

In practice Kerckhoffs' principle is not always obeyed. One example is the block cipher GOST [93] which was developed in the 1970s by the soviet government and kept secret. After the dissolution of the USSR the specification were released. GOST's contains secret S-boxes which vary for different implementations. The design principle behind this idea is not clear, one speculation is that the government would supply weak S-boxes for those it would like to spy on.

Kerckhoffs' principle does not only apply to symmetric cryptosystems but to cryptosystems in general, also to the so-call public key encryption schemes.

1.1.1 Public-key Encryption versus Symmetric Encryption

The drawback of symmetric encryption schemes is that the two parties, who want to communicate over an insecure channel, have to exchange a secret key over a secure channel beforehand. The key for encryption and decryption is the same and must therefore be kept secret. This implies that not only the decryption but also the encryption function is secret. However, in the 1970s Diffie and Hellman found

out that it is not a necessary condition to keep the encryption function secret in order to construct a secure encryption scheme. This insight was the hour of birth of public-key cryptography. Public-key cryptosystems are based on functions which are easy to evaluate but hard to invert unless some extra information is known. Such functions are called trapdoor one-way functions, where the extra information which enables one to easily invert the function is referred to as the trapdoor. Alice holds a key pair consisting of a public and a private key, where the public key defines the encryption function and the private key is the trapdoor information needed to invert the encryption function. When Bob wants to send a message to Alice he obtains her public key from a trusted source, often referred to as certificate authority or trusted authority, and encrypts the message under Alice's public key. Alice can then decrypt the message using her private key. Public-key cryptography has the huge advantage that Alice and Bob do not have to exchange information over a secure channel before they can start exchanging messages. However, symmetric cryptography still plays an important role in practical systems because symmetric encryption is an order of magnitude more efficient than public key encryption. Often hybrid systems are employed, where public key encryption is used to establish a secret key for the symmetric encryption, which is performed afterwards. In the remainder of this thesis we only consider symmetric encryption schemes.

1.1.2 Block and Stream Ciphers

Symmetric encryption algorithms can be divided into the two categories: block ciphers and stream ciphers. Roughly spoken, a block cipher divides the input message into blocks of a fixed size (usually 64 or 128 bits) and then encrypts each block independently under the same key. The input to a stream cipher is a continuous stream of plaintext bits, which is encrypted according to an internal state. The internal state is initialized by the secret key and an initial value and is updated during the encryption process independently of the message for most designs [41].

In practice these definitions are not that strict. A block cipher is usually used in a mode of operation where the encryption of a block often depends on the encryption of the previous block or the block cipher is used in a stream cipher mode. However, modes of operation are out of the scope of this thesis. A more detailed description of block and stream ciphers can be found in the Chapters 2 and 3.

1.2 Security Considerations

In the previous section we pointed out that the purpose of encryption is to enable two parties to securely communicate over an insecure channel. But what does it actually mean that a cipher is secure. The definition of security is quite difficult, also because the attacker may have various goals and abilities. In this section we will list various definitions of security, name different goals an attacker may have and give an overview of typical attack scenarios.

1.2.1 Security Measurements

We can find several approaches to evaluate the security of a cryptosystem in the literature. We consider two different concepts of security in this section: unconditional and computational security.

Unconditional Security

A cryptosystem is unconditionally secure if it cannot be broken even with infinite computational resources. A cryptosystem might be unconditional secure in one attack scenario but easy breakable in another. A more formal definition is given by Shannon [95] under the name "perfect secrecy" in his paper "Communication Theory of Secrecy Systems". Informally, perfect secrecy means that an adversary cannot obtain any information about the plaintext by observing the ciphertext.

Definition 1.3 (Perfect secrecy)**.**
A cryptosystem has perfect secrecy if

$$\Pr[P = p|C = c] = \Pr[P = p]$$

for all $p \in \mathcal{P}$ and $c \in \mathcal{C}$.

Note that unconditional security and perfect secrecy are not equivalent definitions, a cipher which provides perfect secrecy is unconditional secure against a ciphertext only attack (cf. Section 1.2.3) but not necessarily unconditional secure in any other attack scenario. Perfect secrecy is a strong condition, which can only be achieved if the key length equals or exceeds the message length [95]. Therefore, it is rather impractical to require perfect secrecy. Nevertheless perfect secrecy is achievable. Shannon proved that the cipher One-time pad provides perfect secrecy. The idea of One-time pad is very simple:

Definition 1.4 (One-time pad)**.**
Let $n > 1$ be the length of the message, then $\mathcal{P} = \mathcal{K} = \mathcal{C} = \mathbb{Z}_2^n$. For $p = (p_1, \ldots, p_n) \in \mathcal{P}$, $c = (c_1 \ldots, c_n) \in \mathcal{C}$ and $k = (k_1, \ldots, k_n) \in \mathcal{K}$ the encryption and decryption is defined as bitwise exclusive-or.

$$E_k(p) = (p_1 \oplus k_1, \cdots, p_n \oplus k_n)$$

and

$$D_k(c) = (c_1 \oplus k_1, \cdots, c_n \oplus k_n).$$

For the security of One-time pad it is crucial that the key $k \in \mathcal{K}$ is chosen uniformly at random and that no two messages are encrypted using the same key.

We have seen that unconditional security or even perfect secrecy implies rather impractical requirements and in practice we can assume that the adversary does not have infinite computational power to his disposal. This leads us to the definition of computational security.

Computational Security

If we have the possibility to somehow verify that the solution we found is correct, we can always apply an exhaustive search.

Definition 1.5 (Exhaustive search)**.**
The method of trying all possible elements in the search space until the correct one is found is called exhaustive search. On average half of the elements have to be tested.

In practice it is reasonable to require that a cryptosystem is secure against an adversary with bounded computational resources.

Definition 1.6 (Computational security)**.**
A cryptosystem provides n bits of security if the most efficient attack requires a computational effort which is equivalent to an exhaustive search over 2^n values.

From this definition we can infer that a cryptosystem is computationally secure if it provides n bits of security where 2^n operations are computationally infeasible with the resource that are currently available or that will be available in the near future. The parameter n in general is determined by the key size, because it is always possible to exhaustively search over all 2^n keys until the correct one is found if the plaintext is known (see Section 4.1 for further details). Thus the key size yields an upper bound for the parameter n and in the optimal case, i.e., if there is no attack faster than exhaustive key search, also a lower bound.

Computational security is in general hard to prove. Modern designs usually follow design principles that allow the designer to derive lower bounds for a successful attack. However, these bounds are derived using certain assumption or models and are often limited to a certain attack.

In his paper "Communication Theory of Secrecy Systems" [95] Shannon identified two properties a secure cipher should have for obscuring redundancies.

Confusion means according to Shannon "to make the relation between the simple statistics of ciphertext and the simple description of key a very complex and involved one". This was reinterpreted by Massey [70] to mean that the ciphertext statistics should depend on the plaintext statistics in a manner too complicated to be exploited by a cryptanalyst. Confusion is usually realized by substitutions such as S-boxes.

Diffusion "In the method of diffusion the statistical structure of the plaintext which leads to its redundancy is dissipated into long range statistics, i.e., into statistical structure involving long combinations of letters in the cryptogram." [95]. That means that each digit of the plaintext and an secret key should influence many digits of the ciphertext [70]. A cipher has a good diffusion if each bit of the ciphertext changes with probability $\frac{1}{2}$ if one bit of the plaintext is flipped. Permutations provide diffusion in block ciphers.

An important question when evaluating computational security is the question of which components determine the complexity of an attack. There are three different categories that are used to measure the complexity.

- *Processing complexity:* The time needed to actually perform the attack.
- *Data complexity:* The amount of data needed for the attack to be successful.
- *Storage complexity:* The amount of memory that is required during the attack.

A design goal for an encryption scheme is to maximize the complexity of known attacks while minimizing the complexity of encryption and decryption. Here, it is important that the designer also keeps possible trade-offs between the different complexities in mind such as the time-memory trade-off described in Chapter 4. As opposed to the designer's goals an adversary tries to find an attack with a complexity that is lower than the bound estimated by the designer. On the one hand, not every successful attack implies that the cipher becomes useless in practice because even though the attack is faster than exhaustive search it might still be computational infeasible. On the other hand, every attack reveals a previously unknown weakness of a cipher and carries the risk of becoming a practical threat in the future.

In this section we have given two basic definitions of security against a successful attack. But what does it mean that an attack is successful? In the next section we give a short overview of the different attack goals, an adversary may have.

1.2.2 The Adversary's Goals

Clearly, the highest goal of an adversary is to recover the secret key which is used to encrypt the ciphertexts he intercepts. But the key is not the only information an adversary can obtain. In the following we list different attack goals in decreasing order of severity using the classification of [67].

1. **Total break:** An attacker finds the secret key k which has been used under the encryption.
2. **Global deduction:** An attacker finds an algorithm A, that is equivalent to E_k or D_k without knowing the actual key k.
3. **Local deduction:** An attacker can recover the plaintext of an intercepted ciphertext.
4. **Information deduction:** An attacker obtains some information about the key or the plaintext. This can be a few bits of the key or plaintext, as well as some information about the structure of the plaintext etc.

Not included in the classification given in [67] are distinguishing attacks. These are the most basic attacks. In a distinguishing attack the adversary wants to distinguish the output of a cipher from the output of a random permutation.

After we have clarified the different attack goals the next step is to classify an adversary's abilities. An attacker, who can require decryptions of ciphertexts he chooses himself, might be more powerful than an attacker, who can just eavesdrop a conversation.

1.2.3 Classification of Attack Scenarios

The overall assumption is that Kerckhoffs' principle applies, meaning that the adversary knows all details about the encryption algorithms except for the secret key. Furthermore, we assume that the attacker can access the full communication between the two parties. Under these assumptions we can classify five possible cryptanalytic attacks.

Ciphertext only attack: The attacker possesses the ciphertexts of several messages, all of which have been encrypted using the same key.

Known plaintext attack: The attacker has access to several plaintexts and the corresponding ciphertexts. However, he has no control over the plaintexts available to him.

Chosen plaintext attack: The attacker can choose a number of plaintexts a priori and obtains the corresponding ciphertexts.

Adaptively chosen plaintext attack: An adaptively chosen plaintext attack is a chosen plaintext attack where the attacker does not choose the plaintexts a priori but depending on the ciphertexts he received from previous requests.

Chosen ciphertext attack: The attacker chooses a number of ciphertexts and asks for the decryption of those. Equivalent to chosen plaintext attacks there are chosen ciphertext and adaptively chosen ciphertext attacks.

Clearly ciphers which are vulnerable to ciphertext only attacks are considered very weak. A known plaintext scenario is still a very realistic assumption while there are easy strategies to impede the more powerful chosen text attacks in a real-world application. For example, we can require that the plaintexts have certain structure. And if a plaintext does not have this structure the application denies encryption or decryption. In general a symmetric cryptosystem that is secure against adaptively chosen plaintext attacks might still be vulnerable to chosen ciphertext attacks but is secure against the other three attack types.

In the last years another attack scenario, which can be combined with the attacks above, has become more popular, the so-called related-key attacks. The attacker obtains the encryptions of a plaintext under two or more related keys, where he either knows or chooses the relation between the keys while the keys themselves are unknown. Related-key attacks constitute a not very likely scenario in practice. (However, there are applications where a related-key attack might be practical.) Nonetheless every successful attack even if it is just possible in theory compromises the security of the cipher. Hence, being able to prove that a cipher is secure against a very powerful attacks such as adaptively chosen ciphertexts under related keys is a convincing argument for the security of the cipher, especially if the implementation of the algorithm would not allow such an attack.

The different classes of attack and the adversary's goals we have seen in the last two sections consider mainly block ciphers. Due to the different designs, the assumptions,

we make, and the goals, an adversary has, are usually slightly different when it comes to stream ciphers. Of course, the highest goal, the recovery of the secret key, which leads to a total break of the cipher, stays the same, but in general deducing information about the plaintext for a given ciphertext plays a subordinate role. Instead we want to predict bits of the keystream which is continuously generated. As we lack the design details which are necessary to understand the adversary's goals and attack classes we postpone the description of those until Chapter 3.

Chapter 2

Block Ciphers

As the name suggests block ciphers operate on plaintext blocks of a fixed length. A block cipher maps an n-bit block of plaintext to a ciphertext block of the same size. The mapping depends on the key. Therefore a block cipher can be seen as a family of bijections indexed by a key. Here the block size determines the space of all bijections that can be generated by the cipher and the key size determines the subspace of bijections that will be generated by the cipher. The two important parameters of a block cipher are the block size which is usually 64 or 128 bits and the key size which normally ranges between 56 and 256 bits. For a secure block cipher we expect that the ciphertext does not leak exploitable information about the plaintext or the key used in the encryption.

Most block cipher are so-called iterated ciphers. That means that a simple function, called the round function g, is repeatedly applied to the text. Typically the round function takes the output of the previous round and a round key as the input and outputs the input to the next round. One application of the round function is called a round. Let c_i denote the intermediate value after i rounds of encryption and k_i is the ith round key. Then

$$c_i = g(c_{i-1}, k_{i-1}).$$

The input to the first round is the n-bit plaintext block also denoted by c_0 and the last round outputs the ciphertext c_r. The round keys or subkeys are usually derived from the secret key by a key schedule algorithm and are called key schedule.

For a fixed key the round function is by definition bijective because it must be possible to invert the encryption process with the knowledge of the key. Thus, the encryption is a permutation if the key is fixed.

Most block ciphers are either realized as Feistel ciphers or substitution-permutation networks which we will look into in the remainder of this chapter.

2.1 Feistel Ciphers

Feistel ciphers are named after the German cryptographer Horst Feistel. He worked as an IBM researcher on the project "Lucifer" [49] which was the predecessor of the Data Encryption Standard (DES) [88].

The round function of ciphers in Feistel structure operates only on one half of the block while the other half remains unchanged.

Definition 2.1 (Feistel cipher [82])**.**
A Feistel cipher is an iterated cipher which maps a $2t$-bit plaintext block (L_0, R_0),

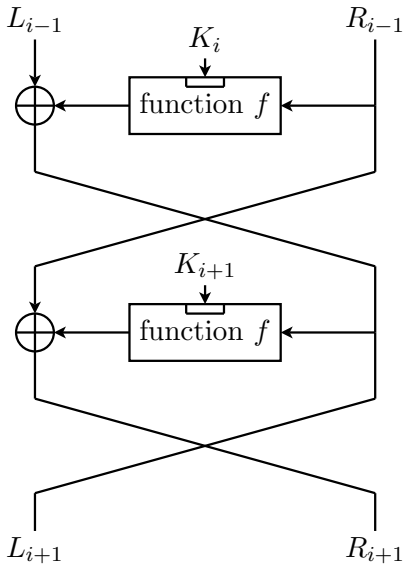


Figure 2.1: Feistel cipher.

where L_0 and R_0 are t-bit blocks each, to a $2t$-bit ciphertext block (L_r, R_r) through r rounds of encryption. One round of encryption is performed as follows. For $1 \leq i < r-1$

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, k_{i-1}) \end{aligned}$$

where k_i is the ith subkey derived from the secret key k and f is a function that takes a subkey and an t-bit block as input and transforms it into an t-bit output block. In the last round the swap of the two halves is omitted:

$$\begin{aligned} L_r &= L_{r-1} \oplus f(R_{r-1}, k_{r-1}) \\ R_r &= R_{r-1}. \end{aligned}$$

In a Feistel network (cf Figure 2.1) the plaintext is split into two halves L_0 and R_0. The right half is together with the round key input to the function f. The output of the function f is exclusive-ored to the left half of the text and the two halves are swapped afterwards. These operations are repeated $r-1$ times, each time with a new subkey. In the last round the swap of the halves is omitted. The reason for this is that decryption is achieved by using the encryption algorithm with the subkeys in reverse order. The function f itself does not need to be invertible to invert the Feistel encryption (or one round of the Feistel cipher). The Data Encryption Standard is probably the best known example of a Feistel cipher. Here we present the cipher C2.

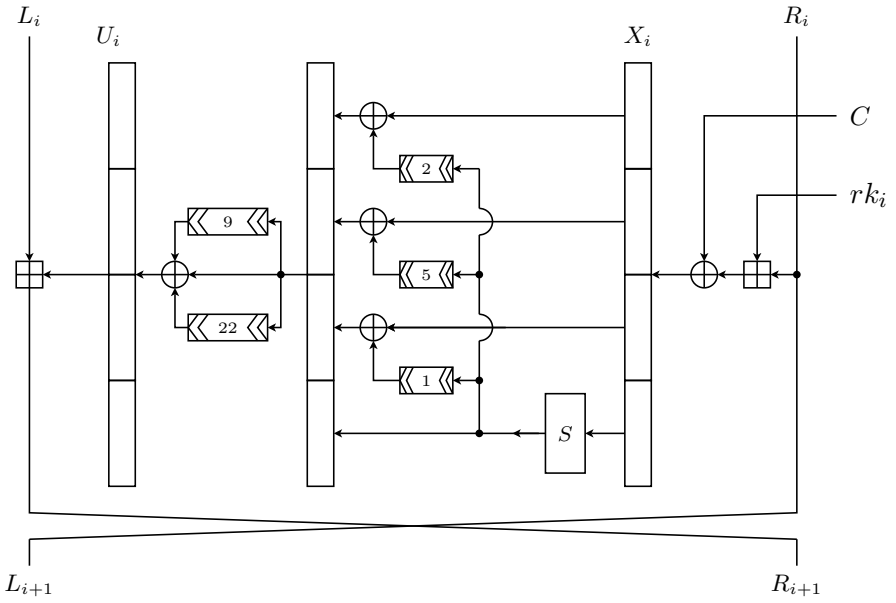


Figure 2.2: The round transformation of C2.

C2 is a cipher in Feistel structure with a small variation. The XORs are replaced by a modular addition.

2.1.1 The Cipher C2

In this section we consider the block cipher C2. We will present a cryptanalysis of this cipher in Chapter 5. C2 [5] is the short name for the proprietary block cipher Cryptomeria, which is defined and licensed by the 4C Entity [6]. The 4C Entity is a consortium consisting of IBM, Intel, Matsushita and Toshiba. C2 was designed for the CPRM/CPPM Digital Right Management scheme which is used on DVD-Audio discs and Secure Digital cards.

C2's specification has been published by the 4C Entity. C2 is a Feistel cipher with a 56-bit secret key, supporting message blocks of size 64 bits. In each of the 10 Feistel rounds a 32-bit round key rk_i, derived from the 56-bit secret key, is used. The round function is illustrated in Figure 2.2 and can be described as

$$
\begin{aligned}
L_{i+1} &= R_i \\
X &= (R_i \boxplus rk_i) \oplus \texttt{0x2765ca00} \\
Z_{0..7} &= S[X_{0..7}] \\
Z_{8..15} &= X_{8..15} \oplus \mathsf{rotl}_8(Z_{0..7}, 1) \\
Z_{16..23} &= X_{16..23} \oplus \mathsf{rotl}_8(Z_{0..7}, 5) \\
Z_{24..31} &= X_{24..31} \oplus \mathsf{rotl}_8(Z_{0..7}, 2) \\
R_{i+1} &= L_i \boxplus (Z \oplus \mathsf{rotl}_{32}(Z, 9) \oplus \mathsf{rotl}_{32}(Z, 22)), \quad i = 0, \ldots, 9,
\end{aligned}
$$

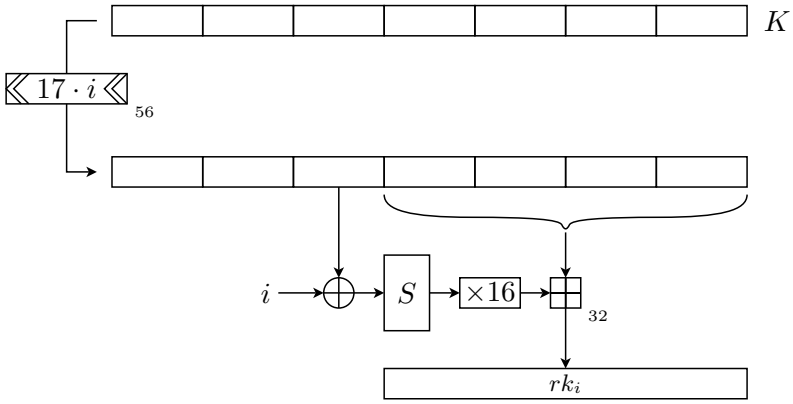


Figure 2.3: One step of the key scheduling algorithm generates 32-bit round key rk_i.

where L_i is the left and R_i is the right half of the word after the ith round of encryption, $rotl_m(b, n)$ denotes a cyclic left rotation of an m bit sequence b by n positions and $X_{i,p..q}$ the sequence of consecutive bits $X_{i,p}, X_{i,p+1}, \ldots, X_{i,q}$ of the word X_i. The addition modulo 2^{32} of the words X and Y is denoted by $X \boxplus Y$ and the bitwise XOR respectively by $X \oplus Y$.

Note that the description differs from the original reference code [5] where three byte constants are used. We present a simpler but equivalent description using only one 4-byte constant $C =$ `0x2765ca00`.

The 10 round keys $rk_0, \ldots, rk_9$ are derived from the 56-bit secret key K in the following way.

$$\begin{aligned} K'_i &= \mathsf{rotl}_{56}(K, 17 \cdot i) \ , \\ rk_i &= K'_{i,0..31} \boxplus (S[K'_{i,32..39} \oplus i] \ll 4), \quad i = 0, \ldots, 9. \end{aligned}$$

The key schedule algorithm is depicted in Figure 2.3.

The 8-bit S-box S, used in both the round transformation and the key scheduling, is secret and available under license from the 4C Entity. An example S-box provided by 4C for the purpose of validating the implementations is available online [7]. The S-box might be considered as part of the secret key.

Depending on the application different S-boxes are used but the same S-box is used for the same application. A CPRM compliant device is given a set of secret device keys when manufactured. The device keys can be revoked.

We can think of different attack scenarios on C2 like an S-box recovery, the recovery of the secret key when the S-box is known and an attack which recovers the S-box and the key at the same time. These three attack scenarios will be discussed in Chapter 5 in detail.

2.2 Substitution-Permutation Networks

In a substitution-permutation network (see Figure 2.4) or short SP network the round function combines layers of invertible functions such as substitutions and permuta-

tions. The substitution layer realizes the non-linear part of the cipher. Often it consists of the parallel application of functions $S : \{0,1\}^m \to \{0,1\}^m$ which operate on a small chunk of data. These functions are called S-boxes (Substitution boxes) and can be implemented as a look-up table with 2^m entries. (Note that the input and output size of an S-box may also be different but in those cases the S-box is not bijective.) The S-boxes are non-linear and introduce local confusion. The permutation layer is an affine transformation that operates on the complete block and introduces diffusion.

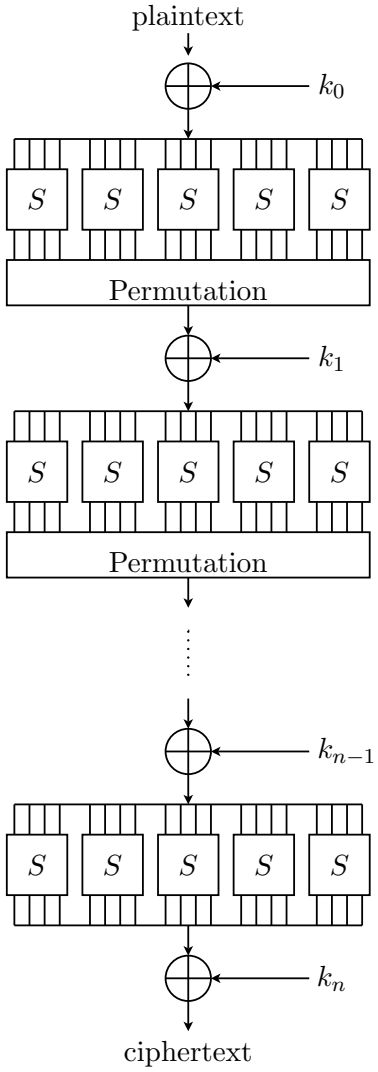


Figure 2.4: Substitution Permutation network.

Definition 2.2 (SP-network)**.**
A substitution-permutation network is an iterated cipher where the round function is built from a substitution layer, a permutation layer and a subkey application.

The Advanced Encryption Standard (AES) is the most famous block cipher based

on an SP network. Also the lightweight cipher PRESENT is based on an SP-network. Later on we will consider PRESENT-like block ciphers with secret components. Therefore we give PRESENT as an example for an SP network in the following section.

2.2.1 The Block Cipher PRESENT

PRESENT is an ultra-lightweight block cipher proposed by Bogdanov et al. [21]. It has been designed for extremely resource-constrained environments such as RFID tags and sensor networks where just a moderate security level is required. PRESENT has implementation requirements similar to stream ciphers (see Chapter 3).

PRESENT is an SP-network which consists of 31 rounds and operates on blocks of size 64 bits. Two key lengths of 80 or 128 bits are supported, where an 80-bit key is recommended. Each of the 31 rounds consists of a round-key application, followed

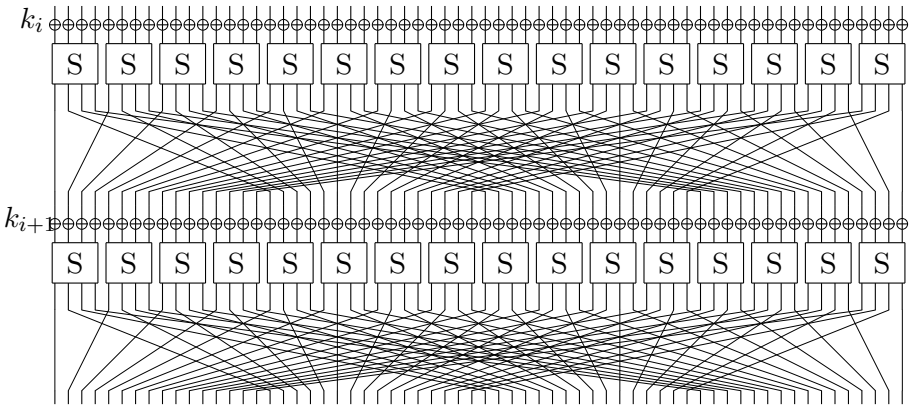


Figure 2.5: The S/P network for PRESENT.

by a substitution and a permutation layer. Two rounds of PRESENT are shown in Figure 2.5.

In each round a 64-bit round key K_i for $1 \leq i \leq 32$ is xored to the current state, where the last round key K_{32} is used for post-whitening. Let $K_i = \kappa^i_{63} \dots \kappa^i_0$ for $1 \leq i \leq 32$ be the ith round key and $b_{63} \dots b_0$ the current state. Then the round-key addition is a bitwise XOR for $0 \leq j \leq 63$ as shown in (2.1).

$$b_j \rightarrow b_j \oplus \kappa^i_j. \tag{2.1}$$

The non-linear substitution layer, also called sBoxLayer by the authors, is built from a single 4 to 4-bit S-box $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ which is applied 16 times in parallel. This S-box in hexadecimal notation is given in Figure 2.6. The current state $b_{63} \dots b_0$ is split into sixteen 4-bit words $w_0, \dots, w_{15}$ where $w_i = b_{4i+3}||b_{4i+2}||b_{4i+1}||b_{4i}$ for $0 \leq i \leq 15$ and then each of the words is processed by the S-box. The concatenated outputs $S[w_i]$ for $i = 0, \dots, 15$ update the state value.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Figure 2.6: The S-box used in PRESENT in hexadecimal notation.

Table 2.1: The PRESENT permutation.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

The permutation layer, also called pLayer by the authors, uses a regular bit permutation which can be efficiently implemented in hardware. The permutation is given in Table 2.1.

The S-box S (cf. Figure 2.6) fulfills some additional conditions to improve the avalanche of change. It has been carefully chosen to resist against differential and linear cryptanalysis. (The concepts of differential and linear cryptanalysis are explained in Chapter 4.) This is not generally the case for randomly chosen S-boxes. In Chapter 6 it will be shown how weak differential properties can be exploited.

The key schedule of PRESENT can take an 80 or 128-bit secret key. In this thesis we focus on the 80-bit version which is the recommended key size in [21] . The 80-bit secret key is stored in the key register K and presented as $k_{79} \dots k_0$. The i-th round key K_i consists of the 64 leftmost bits of the current state of the key register K. Thus $K_i = \kappa_{63} \dots \kappa_0 = k_{79} \dots k_{16}$. After the round key K_i has been extracted the key register is updated using the three following steps.

1. $[\mathtt{k}_{79}\mathtt{k}_{78} \dots \mathtt{k}_1\mathtt{k}_0] = [\mathtt{k}_{18}\mathtt{k}_{17} \dots \mathtt{k}_{20}\mathtt{k}_{19}]$,
2. $[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}] = S[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}]$,
3. $[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] = [\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}]. \oplus \mathtt{round\_counter}$

This means that the key register is first rotated by 61 positions to the left, then the leftmost four bits are processed by the S-box. This S-box is the same as the one used in the encryption algorithm. At last the bits $k_{19}k_{18}k_{17}k_{16}k_{15}$ of K are xored with the least significant bits of `round_counter`. The key schedule for 128 bit keys involves two applications of the S-box for each update of the key register. For further details the reader is referred to [21].

Chapter 3

Stream Ciphers

Stream ciphers form the second class of symmetric encryption algorithms. While block ciphers transform a large block of plaintext data into a ciphertext block, stream ciphers take a continuous stream of plaintext bits as an input and transform one bit (or a small chunk of bits) at a time. Stream ciphers are in general faster than block cipher and smaller in terms of hardware implementations [82]. These properties and the fact that the bits can be processed one by one as they arrive, make stream ciphers appropriate for applications where buffering is limited like in telecommunications. Another advantage is that stream ciphers have a very limited or no error propagation.

Despite all the advantages of stream ciphers over block ciphers for some applications stream ciphers have been systematically replaced by block ciphers. The stream cipher A5/1 used in the GSM standard has been replaced by the block cipher Kasumi (A5/3) after weaknesses in A5/1 were pointed out [14, 20]. A similar example are the wireless network standards. While WEP ('wired equivalent privacy') uses the stream cipher RC4 [93], the newer standards WPA and WPA2 (Wi-Fi protected access) [1] are based on AES.

One reason for this trend might be that the security of block ciphers is better understood. It is quite clear what the design principles are and with what building blocks and internal structure they can be achieved. Stream ciphers, on the contrary, come in a wide variety of designs of which many have turned out to be flawed.

However, stream ciphers may still play an important role in environments where resources are very restricted and where a high throughput is critical. Therefore in 2004 the eSTREAM stream cipher project [3] was launched with a call for stream cipher designs in two different categories. The first profile has focused on stream ciphers for software applications with high throughput requirements, while the second profile has aimed for stream cipher for hardware applications with restricted resources such as limited storage, gate count, or power consumption [3].

One aim of the project was to restore the confidence in stream ciphers. A thorough cryptanalysis of new stream cipher proposals might help to gain a better understanding of stream cipher design. Furthermore, during the process simple and reliable design criteria might be developed. A second challenge of the project was to improve the efficiency advantage stream ciphers have over block ciphers. The eSTREAM project ended in 2008 with a portfolio of 7 recommended stream ciphers, four in the software and three in the hardware category.

In this chapter we will first categorize two different types of stream ciphers. Secondly we examine building blocks and various designs, followed by a discussion of security issues. The chapter will conclude with the specification of the stream cipher Trivium and a short review of cryptanalytic results on Trivium. Trivium will

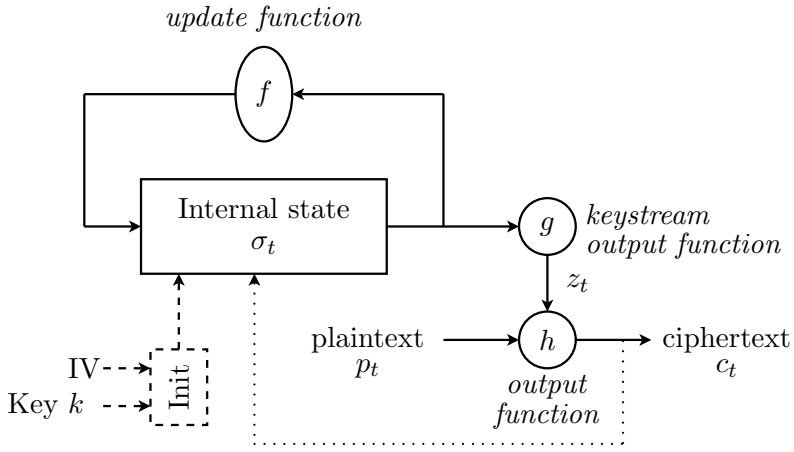


Figure 3.1: General structure of a stream cipher.

be the main target in Part III of this thesis where we apply optimization methods in cryptanalysis.

3.1 The General Structure of Stream Ciphers

The design of stream ciphers is inspired by the cipher One-time Pad (cf. Definition 1.4). As mentioned in Chapter 1 Shannon proved that One-time pad provides perfect secrecy if the key is of the same length as the message and chosen uniformly at random. The drawback of this design is that the key must have the same length as the message and can only be used once. The idea behind the stream cipher design is to use a short secret key to generate a long sequence of bits, which appears to be random. This random looking sequence is called a keystream. As encryption the keystream is combined with the plaintext to produce the ciphertext.

Informally, a *cryptographically secure pseudorandom* sequence is a sequence that has the following three attributes. The sequence is produced by a deterministic algorithm, which takes a short truly random sequence as an input, meaning that the sequence can be reproduced easily. Despite this the sequence appears to be random, which means that it is difficult to distinguish this sequence from a truly random sequence. Furthermore, it should be difficult to predict the next bit of the sequence. More precisely, given l bits of the sequence there is no polynomial-time algorithm that predicts the $(l+1)$st bit with a probability better than $\frac{1}{2}$. In order for a stream cipher to be secure the keystream must be cryptographically secure pseudorandom.

Alternatively, a stream cipher can be described as a key-dependent algorithm with an internal memory that processes a message m bit by bit and produces the corresponding ciphertext bit in parallel or with a short delay using a transformation that varies as the plaintext is processed. In other words a stream cipher can be seen as a finite state machine that in each clocking produces one bit (or a couple of bits) of the pseudorandom sequence in dependency of the internal state, updates the internal state, and combines the keystream bit and the plaintext bit in order to obtain the

ciphertext bit. The general structure of a stream cipher is depicted in Figure 3.1.

Most stream ciphers can either be classified as synchronous or self-synchronizing stream ciphers.

3.1.1 Synchronous Stream Ciphers

A stream cipher is called synchronous if the next state is defined independently of the plaintext or ciphertext and thus the keystream is generated independently of the plaintext and ciphertext.

Definition 3.1 (Synchronous Stream Cipher [82])**.**
A synchronous stream cipher is a finite state machine whose update function does not depend on the plaintext and ciphertext but only on the current state (and in some cases on the key k). A synchronous stream cipher consists of

- *a internal state which is denoted by σ_t at time t. The internal state is initialized using initialization function or key setup function $Init$ which takes an initial value IV and the key k as inputs*

$$\sigma_0 = Init(k, IV),$$

- *an update function f which updates the internal state of the finite state machine depending on the current state at time t*

$$\sigma_{t+1} = f(\sigma_t),$$

- *a keystream output function g which maps the internal state to a keystream bit z_t*

$$z_t = g(\sigma_i),$$

- *and an output function h that combines the keystream and the plaintext in order to generate the corresponding ciphertext*

$$c_t = h(p_t, z_t).$$

The part of the cipher that generates the keystream is also called keystream generator (see Figure 3.2).

The update function f and the keystream function g may also take the secret key k as an input, but as for most cases the stream cipher proceeds independently of the key after the key set-up we omit it here. In order to decrypt, the keystream is generated in the same way, thus the only requirement is that the output function h is invertible. Usually h is the exclusive-or and hence self-inverse.

As each bit of the ciphertext is decrypted independently any error that occurs in one bit of the ciphertexts during the transmission will not affect the decryption of other ciphertext bits. This means there is no error propagation for corrupted bits. However, in order to be able to decrypt correctly the decryption has to stay in step, which means that deleting or inserting a ciphertext bit will cause an error in the

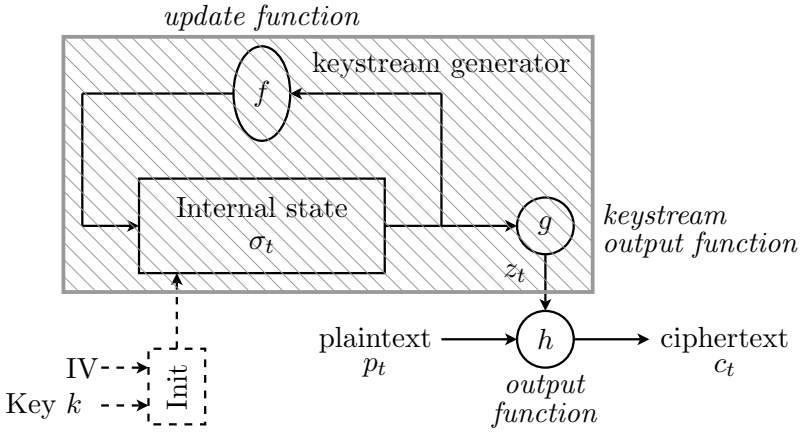


Figure 3.2: Synchronous stream cipher.

decryption of all consecutive ciphertext bits. This synchronization problem can be solved by using marker positions, which enable a re-synchronization of the decryption and encryption process. Then a lost or inserted ciphertext bits just causes incorrect decryption until the next marker position [91]. These marker positions are usually realized by frames [78]. A frame is of a fixed size and contains a frame number and a ciphertext block. Before the plaintext of a frame is encrypted the encryption is re-initialized using the secret key and an initial value which can be derived from the frame counter. The receiver of a frame initializes the keystream generator with the same secret key and initial value and is able to decrypt the ciphertext. Thus, a lost or inserted ciphertext bit can at most affect all bits in a frame.

An example of a synchronous stream cipher is the cipher Trivium which we will present in Section 3.4.

3.1.2 Self-Synchronizing Stream Ciphers

The synchronization problem can also be solved using self-synchronizing stream ciphers.

Definition 3.2 (Self-synchronizing stream cipher [82])**.**
A self-synchronizing or asynchronous stream cipher is a stream cipher in which the keystream is generated as a function of the key and a fixed number of the previous ciphertext digits.

The internal state σ_t at time t is then $\sigma_t = (c_{t-l}, c_{t-l+1}, \ldots, c_{t-1})$ where $\sigma_0 = (c_{-l}, \ldots, c_{-1})$ is the initial value. A keystream z_t is generate using the keystream output function g with input σ_t and the secret key k:

$$z_t = g(\sigma_t, k).$$

The ciphertext is produced as a combination of the keystream and the plaintext under

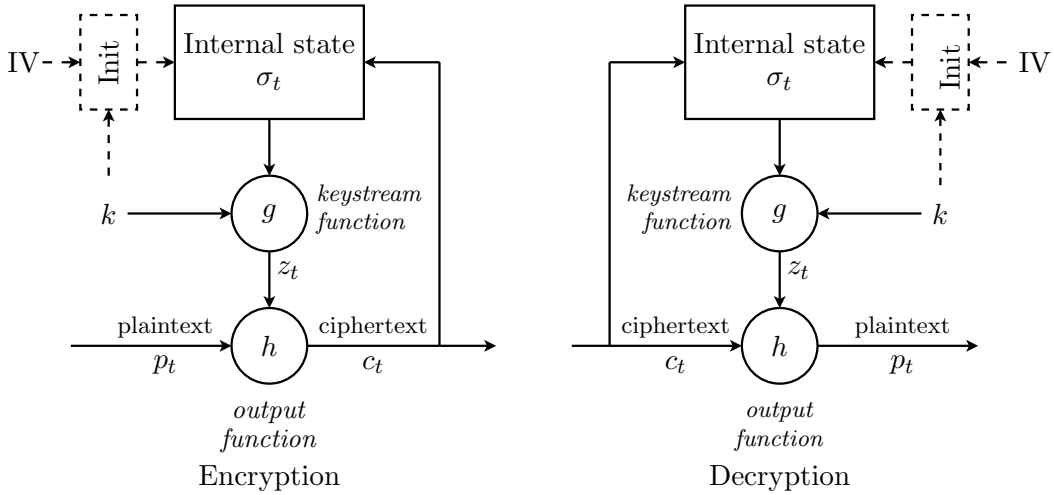


Figure 3.3: Self-synchronizing stream cipher.

a function h as in the case of a synchronous stream cipher. The encryption and decryption process of a self-synchronizing stream cipher is depicted in Figure 3.3.

The main advantage of a self-synchronizing stream cipher is that it can resume correct decryption if the keystream falls out of synchronization after a ciphertext bit was inserted or deleted. This is because the keystream depends only on a fixed number of the preceding ciphertext bits. Also the error propagation is limited. We assume that the state of the cipher depends on l previous ciphertext bits. If one ciphertext bit is corrupted, inserted or lost then the decryption of the following l ciphertext bits may be incorrect. After that the system is able to resynchronize itself after receiving l correct ciphertext bits.

The drawback of self-synchronizing stream ciphers is that the attacker knows the ciphertext bits and with them already parts of the inputs to the keystream generator. Also the text dependency of the keystream makes it hard to estimate the security.

In general there are more synchronous than self-synchronizing stream cipher designs, maybe due to the fact that for the latter security assessments are difficult. Out of the 34 submission to the eSTREAM project only two were self-synchronizing stream ciphers while the other 32 were synchronous stream ciphers. Therefore, we will focus entirely on keystream generators and synchronous stream cipher for the remainder of this thesis.

3.2 Design and Building Blocks

In this section we focus on keystream generators. The keystream shall be a cryptographically secure pseudorandom sequence. We know that we can model the keystream generator as a finite state machine. The machine is regular clocked and at each clocking the internal state is deterministically updated where the update is determined by the current state. Furthermore, at each clocking a bit (or a couple of bits) of the

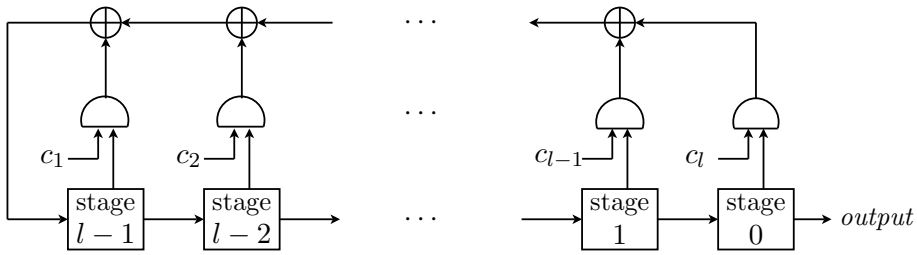


Figure 3.4: A linear feedback shift register (LFSR) of length l.

keystream are generated. Since we consider a finite state machine, and thus only a finite number of states is available, the machine will eventually reach a state where it has already been before (not necessarily the initial state). As the update procedure is deterministic the finite state machine will generate the same states in the same order as it did before from this point on. Hence, also the keystream will be identical to that produced when this state previously occurred. The number of bits before the keystream recurs is called the period of the keystream.

Firstly, it is desirable that the period of the keystream is as long as possible. If the period is too short, parts of the plaintext will be encrypted using an identical part of the keystream and make the cipher vulnerable to a known plaintext attack. If an attacker knows for example the beginning of the message he can easily calculate the corresponding part of the keystream and use it to decrypt the remaining message [82].

Secondly, the sequence should satisfy some statistical properties. The keystream symbols/bits should be distributed uniformly at random, meaning that a keystream bit takes the value 0 or 1 with equal probability. Also the joint distribution of two or more bits in a window should be uniform and any linear combination of these bits should be uniformly distributed as well.

In order to achieve good statistical properties and a long period keystream generators are often based on linear feedback shift register (LFSR).

3.2.1 Linear Feedback Shift Registers

Many stream ciphers are based on linear feedback shift registers (LFSRs). The reason is that they can be easily analyzed using algebraic techniques. They are well-suited for hardware implementation and furthermore they have good statistical properties and can be designed such that they produce a long period.

Definition 3.3 (Linear feedback shift register)**.**
A linear feedback shift register (LFSR) of length l consists of l stages or memory cells numbered $0, 1, \ldots, l-1$. Each memory cell can store one bit and a clock controls the movement of the data. The content of the stages at time t is called the state of the LFSR at time t and is denoted by $S_t = (s_{t+l-1}, s_{t+l-2}, \ldots, s_t)$, where s_{t+i} is the content of stage i at time t. The state S_0 at time zero is called the initial state of the LFSR.

At each clocking of the LFSR the following operations are performed:

(i) the content of the stage 0 *is output,*

(ii) the content of stage i *is moved to stage* $i-1$ *for* $i = 1, \ldots, l-1$,

(iii) the new content of stage $l-1$ *is the feedback bit* s_{l+t} *with*

$$s_{l+t} = \sum_{i=0}^{l-1} c_{l-i} \cdot s_{t+i}.$$

The relation in (iii) is called a linear recurrence relation. The constants $c_1, c_2, \ldots, c_l \in \mathbb{F}_2$ *are called feedback positions.*

The general structure of an LFSR is given in Figure 3.4. The positions where $c_i = 1$ are also called tap positions. Algebraically an LFSR can be described through the feedback polynomial of the LFSR

$$g(x) = 1 + c_1 x + c_2 x^2 + \cdots + c_l x^l \text{ in } \mathbb{F}_2[x].$$

The feedback polynomial can be used to determine the period of the LFSR. We know that an LFSR is a finite state machine with an internal state of size l, therefore it has at most 2^l possible state. Thus the maximum period of the LFSR for an initial state is $2^l - 1$, because the state (0,...,0) obviously has period 1 and can not occur in any periodic sequence with a period greater than 1. Furthermore the following theorem holds.

Theorem 3.4.
If the feedback polynomial $g(x)$ *over* $\mathbb{F}_2$ *of a linear feedback shift register is a primitive polynomial of degree* l*, then for every non-zero initial state the period of the LFSR is* $2^l - 1$ *[78].*

An LFSR with a primitive feedback polynomial of degree l is called a maximum-length LFSR. The tap positions for a maximum length LFSR must be relatively prime.

As mentioned before LFSRs are easily implementable in hardware and software and the generated sequence has good statistical properties. The drawback of a linear feedback shift register is, however, its linear properties. If the length l of the LFSR is known, then the knowledge of l symbols at known positions from the output are sufficient to recover the initial state of the linear feedback shift register. But even if the length of the LFSR is not known there are fast algorithms to recover the LFSR. We denote by s^n the sequence $s_0, s_1, \ldots, s_{n-1}$ of length n and by s the infinite sequence $s_0, s_1, \ldots$. We say that an LFSR generates a sequence s if there is an initial state S_0 for which the output sequence of the LFSR is s. Similar an LFSR is said to generate the finite sequence s^n if there is an initial state such that the output sequence of the LFSR equals s^n for the first n terms. Given a sequence s^n we now consider the problem of finding the shortest linear recurrence generating this sequence or equivalent the shortest LFSR generating this sequence.

Definition 3.5 (Linear complexity [82])**.**
The linear complexity of an infinite binary sequence s, denoted $L(s)$, is defined as follows:

(i) If s is the zero sequence, then $L(s) = 0$.

(ii) If s is a truly random sequence, then $L(s) = +\infty$,

(iii) otherwise, $L(s)$ is the length of the shortest LFSR that generates the sequence s.

The Berlekamp-Massey algorithm [62] is an efficient algorithm for finding the shortest LFSR and can be used to calculate the shortest LFSR that generates a sequence s with linear complexity k after observing $2k$ consecutive output bits. While the details of the algorithm are out of the scope of this thesis it is clear that high linear complexity of a keystream is a necessary condition. However, there are several constructions of stream ciphers based on LFSR which try to exploit the desirable properties of an LFSR and add non-linearity in order to increase the linear complexity at the same time. In the following we briefly review some of the constructions of LFSR-based keystream generators.

3.2.2 LFSR-based Stream Ciphers

In this section we give a short overview of different keystream generators based on LFSRs [62]. An analysis of the properties of those such as linear complexity etc. is out of the scope of the thesis.

The filter generator

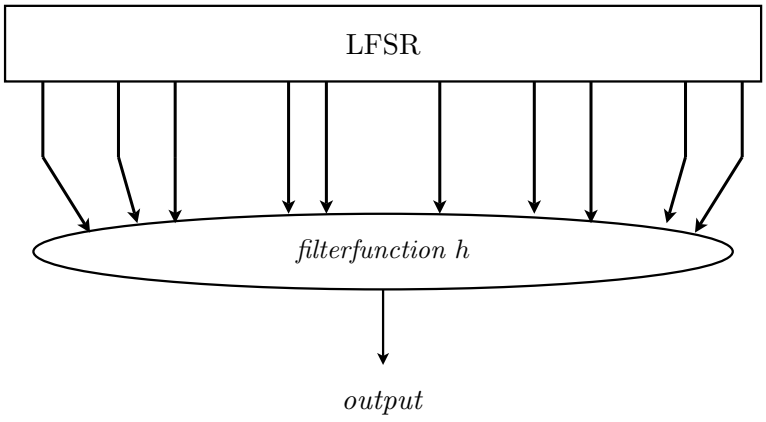


Figure 3.5: Non-linear filter keystream generator.

The filter generator introduces non-linearity through a non-linear output function h, called the filter function, which takes a few bits of the LFSR as input. At each

clocking the filter function h takes t bits from fixed positions of the LFSR as input and generates the output in a non-linear manner. Usually these positions are neither consecutive nor evenly spaced in the register. The function h should be carefully chosen, a too simple function yields a weak generator. However the function should still be efficiently evaluable in order to obtain a fast stream cipher. Figure 3.5 illustrates the construction of a filter generator.

Non-linear output from multiple generators

This can be seen as an extension of the filter generator. The idea is to run several LFSRs in parallel and combine their output bits using a non-linear combining function h. This is depicted in Figure 3.6.

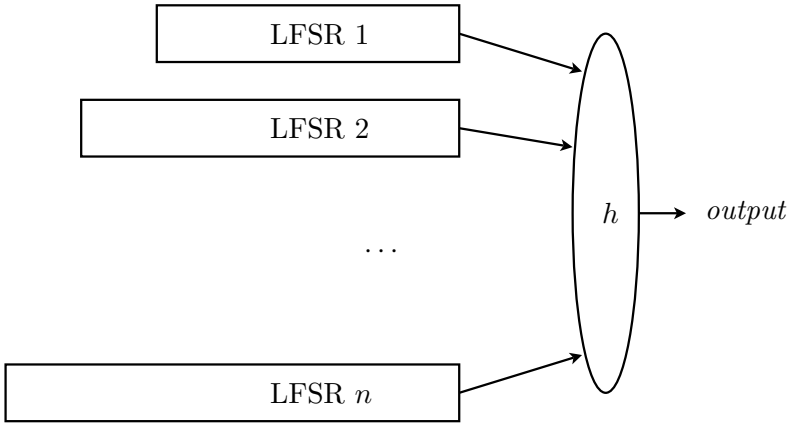


Figure 3.6: The keystream is generated as a non-linear combination h of the outputs of n LFSRs.

A simple example for such a generator is the Geffe generator [82], which consists of three LFSRs that run in parallel. The first LFSR controls the output while the other two generate the output bits. In each clocking of the algorithm all three LFSR are clocked. If the output of the first LFSR, denoted by s_{10}, is 0 then the output z of the keystream comes from the second LFSR, if $s_{10} = 1$ then z is the output bit of the third LFSR. Let s_{20} and s_{30} denote the output bit of the second and third LFSR respectively. Then the non-linear combining function h can be represented as $h(s_{10}, s_{20}, s_{30}) = \bar{s}_{10} \cdot s_{20} + s_{10} \cdot s_{30}$, where $\bar{s}_{10}$ is the negation of s_{10}.

Clock controlled generators

By definition a linear shift register is clocked regularly and the content of the stages is updated in each clocking. One way to introduce non-linearity is to vary the rate at which the register is clocked. This can be done by a so-called “control LFSR” that

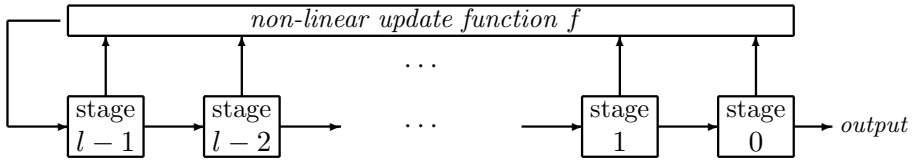


Figure 3.7: A non-linear feedback shift register (NFSR) of length l.

controls when the LFSR which produces the output is clocked. An example is the alternating step generator [82] which consists of one control and two output LFSRs. At each step the XOR of the output bits of the two output LFSRs is output as a keystream bit. Furthermore, at each step the control LFSR and exactly one of the output LFSRs are clocked. The output bit of the control LFSR determines which of the output LFSRs advances. When the control bit is 0 the first output LFSR is clocked and otherwise the second.

3.2.3 Non-Linear Feedback Shift Register

In more recent stream cipher designs such as Trivium [42] and Grain [57] a new building block has been introduced in order to overcome the problem of too low linear complexity. This building block is called the non-linear feedback shift register (NFSR) and is very similar to a linear feedback shift register. The only difference is that the feedback function f which updates the internal state of the feedback shift register is non-linear for an NFSR (cf. Figure 3.7). Like LFSRs non-linear feedback shift registers are easy to implement in hardware and software and the sequence generated by an NFSR is hard to predict. Furthermore, the linear complexity is high and the updated state variables depend on the initial state in a non-linear manner such that their algebraic expressions quickly grow in complexity and degree through recursion.

However, NFSR are not easy to analyze. Therefore there is no guaranty that the maximum period on average is as expected. Moreover, we cannot guarantee good statistical properties as we could for LFSR.

In current designs the problems are overcome by using e.g., a NFSR with a large internal state, so that we with a high probability can infer that the period is large. Another possibility is to use an LFSR to somehow control the NFSR. We can conclude that so far not enough is known about the analysis of NFSRs in order to guarantee certain properties and give design guidelines for NFSRs with cryptographic use. However, designs based on NFSRs such as Trivium and Grain seem to be very promising.

3.3 Security Considerations

In this thesis we focus on synchronous stream ciphers. When the security of a synchronous stream cipher is considered, usually only the keystream generator is exam-

ined because in most designs a simple XOR is used to combine the keystream and the plaintext. Thus, we assume that an adversary has access to a certain number of keystream bits. This can also be seen as a known plaintext attack. A more advanced attack scenario are resynchronization attacks. Most stream ciphers have an initial value (IV) as an additional input which enables resynchronization during the encryption process. In a resynchronization attack the adversary is allowed to manipulate the IV and he obtains the corresponding keystreams initialized with the different IVs, from which he tries to deduce information about the secret key.

The adversary's goal is to recover the internal state of a stream cipher or the secret key. If the key setup is reversible the recovery of the internal state leads to the recovery of the secret key, because one can just run the key setup backwards. However, already the knowledge of the internal state breaks the cipher because an adversary will be able to run the keystream generator forwards and generate the full keystream which he can use to decrypt.

A weaker goal is to predict keystream bits without knowing the key or the internal state. Such an attack reveals a severe weakness of the cipher, because such a knowledge can be used to decrypt the corresponding ciphertext bits.

A distinguishing attack is also interesting. In a distinguishing attack the adversary tries to determine whether a given sequence is a truly random sequence or generated by the keystream generator. In the next section we will consider a specific stream cipher design based on non-linear feedback shift registers. It is called Trivium.

3.4 Trivium

As mentioned before, the eSTREAM project [3] within ECRYPT in 2004 called for secure and fast stream ciphers in two categories: hardware and software. In the hardware category the focus was on stream ciphers for hardware applications with restricted resources such as limited storage, gate count or power consumption. In 2008 the eSTREAM project ended and amongst the recommended hardware oriented stream ciphers was Trivium [42].

Trivium is a synchronous stream cipher and designed to generate up to 2^{64} bits of keystream from an 80 bit key and 80 bit IV. The cipher consists of two phases. In the first phase, the key setup, the internal state of the keystream generator is initialized using the secret key and the IV. In the second phase, the keystream generation, the generator is repeatedly clocked, where in each clocking one bit of keystream is produced and output. As the key setup is very similar to the keystream generation we start with a description of the second phase.

3.4.1 Keystream Generation

Trivium has an 80 bit secret key and an 80 bit IV. The internal state of size 288 bits is divided into three interconnected NFSRs. Five tap positions from each NFSR are chosen in order to generate one bit z_i of keystream and update one bit of each NFSR in each clocking, meaning a total of 15 tap positions are used to generate one keystream

bit and update three state bits. The keystream generation proceeds as follows, where s_j denotes the jth bit of the state and z_i is the keystream bit at time i:

for $i = 1, 2, \ldots$ **do**
  $z_i \leftarrow s_{66} + s_{93} + s_{162} + s_{177} + s_{243} + s_{288}$ ▷ Generate output bit z_i
  $t_{i,1} \leftarrow s_{66} + s_{93} + s_{91} \cdot s_{92} + s_{171}$
  $t_{i,2} \leftarrow s_{162} + s_{177} + s_{175} \cdot s_{176} + s_{264}$
  $t_{i,3} \leftarrow s_{243} + s_{288} + s_{286} \cdot s_{287} + s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_{i,3}, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_{i,1}, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_{i,2}, s_{178}, \ldots, s_{287})$
end for

Here the '+' and '$\cdot$' operations stand for addition and multiplication over $\mathbb{F}_2$ respectively.

3.4.2 Key Setup

During the key setup phase, the key is loaded into the first 80 bits of the state, followed by 13 zero bits, then the IV is loaded into the next 80 bits of the state and the remaining bits are filled with constant values. In this constant all bits are zero except the last three which are set to one. Then $4 \cdot 288$ clockings are computed without producing any keystream bits. The pseudo-code of the key setup is given below.

$(s_1, s_2, \ldots, s_{93}) \leftarrow (K_1, \ldots, K_{80}, 0, \ldots 0)$
$(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots 0)$
$(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$
for $i = 1$ to $4 \cdot 288$ **do**
  $t_{i,1} \leftarrow s_{66} + s_{93} + s_{91} \cdot s_{92} + s_{171}$
  $t_{i,2} \leftarrow s_{162} + s_{177} + s_{175} \cdot s_{176} + s_{264}$
  $t_{i,3} \leftarrow s_{243} + s_{288} + s_{286} \cdot s_{287} + s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_{i,3}, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_{i,1}, s_{94}, \ldots, s_{176})$
  $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_{i,2}, s_{178}, \ldots, s_{287})$
end for

The cryptanalytic results which are presented in Chapter 9 and 10 of this thesis do not depend on this procedure.

3.4.3 Cryptanalytic Results on Trivium

Due to its elegant design and simple structure Trivium has been a target for many cryptanalysts. The authors say "It was designed as an exercise in exploring how far a stream cipher can be simplified without sacrificing its security, speed or flexibility." [43]. In this section we want to give a brief state of the art overview of the properties of and cryptanalytic results on Trivium, whereby this list does not claim to be exhaustive.

In [43] it is claimed that the probability for a given key and IV pair to cause a period smaller than 2^{80} is 2^{-208} and it is hoped for a period of at least 2^{128} bits.

Furthermore, it is stated that each state bit depends on each key and IV bit in a non-linear way after $2 \cdot 288$ iterations of the cipher. As the key setup consists of four full cycles ($4 \cdot 288$ iterations) it is believed that resynchronization attacks are not possible. To our knowledge, there is no attack on Trivium faster than the exhaustive key search so far. However, several attacks have been proposed which are faster than the naive guess-and-determine attack with complexity 2^{195} which was considered by the designers [43]. A more intelligent guess-and-determine attack with complexity 2^{135} using a reformulation of Trivium has been sketched in [65]. Furthermore, Maximov and Biryukov [79] described an attack with complexity $2^{85.5}$ and Raddum proposed a new algorithm for solving non-linear Boolean equations and applied it to Trivium in [90]. The attack complexity was 2^{164}. McDonald et al. [80] considered Trivium as a Boolean satisfiability problem and used the SAT-solver MiniSAT to solve it. This approach was faster than exhaustive key search for the small scale variants of Trivium called Bivium [90] (see Subsection 3.4.4). However, for the full Trivium the estimated complexity is about $2^{159.9}$ seconds and is therefore worse than exhaustive search.

A common approach is to consider a small scale variant of the cipher, if we are not able to successfully attack its full version. In the case of Trivium the primarily considered small-scale variants were introduced in [90] and are called Bivium A and B.

3.4.4 Bivium

Raddum [90] introduced two small-scale variants of Trivium called Bivium A and Bivium B (the latter is often referred to as Bivium). Both variants are obtained by removing the last NFSR which yields an internal state of size 177 bits. The pseudo code given as Algorithm 1 specifies how to generate one bit z of the keystream for Bivium B where s_j denotes the jth state bit. The keystream generator of Bivium B is depicted in Figure 3.8.

Algorithm 1 Pseudo-code of Bivium B.

for $i = 1, 2, \ldots$ **do**
  $t_{i,1} \leftarrow s_{66} + s_{93}$
  $t_{i,2} \leftarrow s_{162} + s_{177}$
  $z_i \leftarrow t_{i,1} + t_{i,2}$ ▷ Generate output bit z_i
  $t_{i,1} \leftarrow t_{i,1} + s_{91} \cdot s_{92} + s_{171}$
  $t_{i,2} \leftarrow t_{i,2} + s_{175} \cdot s_{176} + s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_{i,2}, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_{i,1}, s_{94}, \ldots, s_{176})$
end for

Bivium A only differs in the keystream bit equation from Bivium B. In Bivium A the keystream bit is $z = s_{162} + s_{177}$ which means that the keystream bit only depends directly on the second register (see Algorithm 2 and Figure 3.9).

It is important to note that not only the state size is smaller for Bivium compared to Trivium but also the linear equation or the keystream equation is simpler because

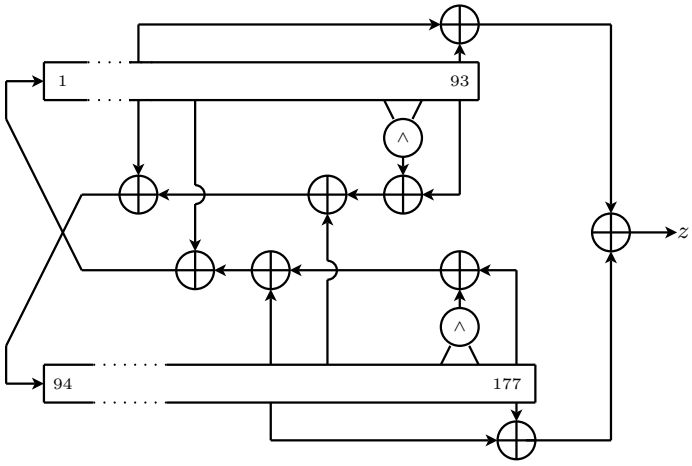


Figure 3.8: Bivium B.

Algorithm 2 Pseudo-code of Bivium A.

for $i = 1, 2, \ldots$ **do**
  $t_{i,1} \leftarrow s_{66} + s_{93}$
  $t_{i,2} \leftarrow s_{162} + s_{177}$
  $z_i \leftarrow t_{i,2}$ ▷ Generate output bit z_i
  $t_{i,1} \leftarrow t_{i,1} + s_{91} \cdot s_{92} + s_{171}$
  $t_{i,2} \leftarrow t_{i,2} + s_{175} \cdot s_{176} + s_{69}$
  $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_{i,2}, s_1, \ldots, s_{92})$
  $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_{i,1}, s_{94}, \ldots, s_{176})$
end for

it only depends on four and two bits for Bivium B and Bivium A respectively.

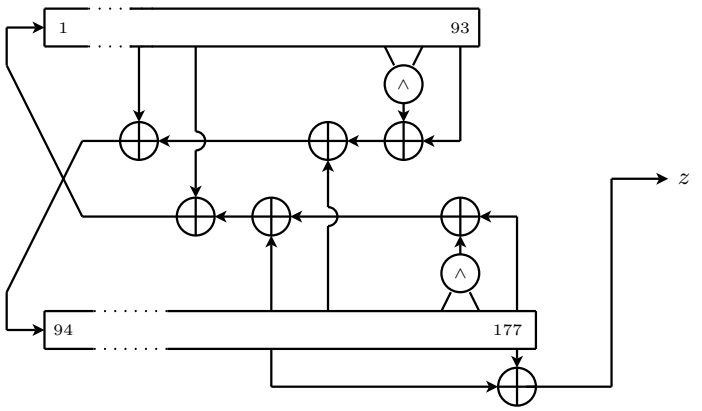


Figure 3.9: Bivium A.

Bivium has also been a target of cryptologists. In [80] an attack using MiniSAT is presented which can recover the initial state of Bivium A in 21 seconds and the initial

state of Bivium B in $2^{42.7}$ seconds which corresponds to a search through 2^{56} keys. Also Raddum's new algorithms for solving Boolean equations could successfully be applied to Bivium A and B with a complexity faster than exhaustive search. Raddum also made the valuable observation that the internal state of Trivium can be represented as a system of sparse non-linear Boolean equations.

3.4.5 Trivium as an Equation System

In [90] is stated that the initial state, which is the state of the registers at the time when the key generation starts, (or any other internal state) can be expressed as a system of sparse linear and quadratic Boolean equations. We consider the initial state bits as variables and label them with $s_1 \ldots, s_{288}$. In each clocking of the Trivium algorithm three state bits are updated. The update function is a quadratic Boolean function of the state bits. In order to keep the degree low and the equations sparse we introduce new variables for each updated state bit $t_{i,1}$, $t_{i,2}$, $t_{i,3}$. We get the following equations from the first clocking

$$\begin{aligned} s_{66} \oplus s_{93} \oplus s_{91} \cdot s_{92} \oplus s_{171} &= s_{289}, \\ s_{162} \oplus s_{177} \oplus s_{175} \cdot s_{176} \oplus s_{264} &= s_{290}, \\ s_{243} \oplus s_{288} \oplus s_{286} \cdot s_{287} \oplus s_{69} &= s_{291}, \\ s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288} &= z. \end{aligned} \tag{3.1}$$

where the last equation is the keystream equation with z being the known keystream bit.

After observing 288 keystream bits we can set up a fully determined system of 954 Boolean equations in 954 unknowns [90]. We only need to consider 954 equations and unknowns instead of 1152 since we do not care about the last 66 state updates for each register. These variables will not be used in the keystream equation because the new bits are not used for the keystream generation before 66 further clockings of the cipher. By clocking the algorithm more than 288 times we can easily obtain an overdetermined equation system. We know that the initial state together with the corresponding updated state bits satisfies all the generated equations (3.1).

In the same way we can obtain a sparse, nonlinear Boolean equation system in 399 equations and unknowns for Bivium A and B after observing 177 bits of the keystream. Each clocking of the algorithm yields one linear and two quadratic equations and two new variables are introduced for the updated state bits $t_{i,1}, t_{i,2}$. We obtain the following equations for the first clocking of Bivium A

$$\begin{aligned} s_{162} \oplus s_{177} &= z \\ s_{66} \oplus s_{93} \oplus (s_{91} \wedge s_{92}) \oplus s_{171} \oplus s_{178} &= 0 \\ s_{162} \oplus s_{177} \oplus (s_{175} \wedge s_{176}) \oplus s_{69} \oplus s_{179} &= 0, \end{aligned} \tag{3.2}$$

and Bivium B

$$\begin{aligned} s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} &= z \\ s_{66} \oplus s_{93} \oplus (s_{91} \wedge s_{92}) \oplus s_{171} \oplus s_{178} &= 0 \\ s_{162} \oplus s_{177} \oplus (s_{175} \wedge s_{176}) \oplus s_{69} \oplus s_{179} &= 0. \end{aligned} \tag{3.3}$$

Solving a random system of non-linear Boolean equations is an NP-hard problem. However, the equation system for Trivium is not random but exhibits a lot of structure. Therefore we hope that efficient methods for solving this kind of equation systems exist. For that reason these equation systems are the starting point of the cryptanalysis of Bivium and Trivium presented in Chapter 9 and Chapter 10.

Chapter 4

Classical Cryptanalysis

The security of block and stream ciphers is continuously evaluated by cryptanalysts all over the world in order to examine the resistance of the designs towards different kinds of attacks. In general the security of block ciphers seems to be better understood than the security of stream ciphers. Many of the standard techniques that have been developed for block ciphers are nevertheless also often applicable to stream ciphers.

We differentiate between two classes of attacks, the generic attacks and short cut attacks. In this chapter we describe the exhaustive key search, the table look-up, the dictionary and the time-memory trade-off attacks as they are important examples of generic attacks. Generic attacks do not depend on the internal structure of the design and can only be avoided by choosing the parameters such as block size, key size or internal state size such that these attacks become computational infeasible. Generic attacks yield important bounds on the security of block and stream ciphers. In short cut attacks the attacker makes use of the internal structure of the cipher. Here, cryptanalysis focuses on efficient ways of exploiting, perhaps unexpected, structure. This could be a difference which propagates with a high probability through the cipher as used in differential cryptanalysis [34, 17] or a linear approximation of the non-linear parts of a cipher that holds for many of the possible inputs as is the case in linear cryptanalysis [75]. With differential and linear cryptanalysis [17, 75] we have mentioned the two most powerful cryptanalytic techniques. In this chapter we describe both techniques for the block cipher case, however both techniques are also suitable for the analysis of stream ciphers [54, 102]. We conclude the chapter with a short introduction to algebraic attacks. Representing a cipher as a system of Boolean equations has become more and more popular in the recent years. This algebraic description also yields the starting point of our novel cryptanalysis presented in Part III.

4.1 Exhaustive Key Search

The exhaustive key search is the most general attack that can be applied to any cipher. We assume that a block cipher encrypts a plaintext p to a ciphertext c using the encryption function $E_k(\cdot)$ under a secret $k \in \mathcal{K}$. Given a plaintext/ciphertext pair the attacker encrypts p under all possible keys $k_i \in \mathcal{K}$ until he finds a key k_j such that

$$E_{k_j}(p) = c.$$

That means the attacker tries exhaustively all keys until he finds the correct one. If the key length equals the block size the attack requires on average one plaintext/ciphertext

pair and the attacker has to try all keys in the worst case but on average he is expected to find the key after he trying about half of the keys.

An exhaustive key search is also possible in a ciphertext only setting under the assumption that the attacker has some information about the plaintext he can check, for example that the plaintext is an English text. In this case the attacker decrypts the ciphertext under all key guess until he obtains a meaningful plaintext.

For a stream cipher the exhaustive key search works similar. If the attacker knows sufficiently many keystream bits he runs the key set up and the keystream generation for all possible keys and compares the newly generated keystream with the given. When these are equal the attack is successful. The number of keystream bits required depends on the design of the stream cipher.

Exhaustive key search is in theory always possible but for all recent cipher designs it is computationally infeasible.

4.2 Table Look-up Attack

In the table look-up attack the idea is to reduce the on-line computation time by doing the bulk computation in the precomputation phase. The attacker builds up a table by encrypting a likely message under all possible keys. The table is sorted according to the ciphertext values. When the attacker intercepts the ciphertext of the likely message, which he has chosen to build the table upon, he can just look up the key in the table. The table look-up attack is therefore a chosen plaintext attack.

The advantage of the attack is that the table can be reused several times and for different keys as long as the attacker can intercept an encryption of the chosen plaintext. The drawback is that the attacker has to intercept a previously specified ciphertext instead of just any plaintext/ciphertext pair. Moreover, the attack requires a non-negligible amount of memory. The attacker needs to store 2^{n_k} ciphertext blocks where n_k is the size of the key and he also has to perform 2^{n_k} operations during the precomputation phase.

4.3 Dictionary Attack

In a dictionary attack the attacker does not recover the secret key of a block cipher but collects a table of plaintexts and corresponding encryptions. If the attacker has intercepted all 2^n possible plaintexts and corresponding ciphertexts where n is the block size, he can build up a dictionary and decrypt all future messages by just looking up the plaintext that corresponds to the ciphertext he intercepted. This is possible as long as the same encryption key is used.

4.4 Cryptanalytic Time-Memory Trade-off

In a time-memory trade-off attack there are five parameters

- N is the size of the search space which is the key space for block cipher and the set of initial states for stream ciphers,
- P represents the time which is required in the precomputation phase,
- M denotes the amount of memory required,
- T represents the time required in the online phase of the attack,
- D is the amount of data which is available to the attacker.

4.4.1 Time-Memory Trade-off Attack for Block Ciphers

For block ciphers we have seen that the exhaustive-key search requires no memory or precomputation ($M = P = 1$) but the attacker has to perform $T = N$ online operations after he intercepted a plaintext/ciphertext pair. The opposite of this is the table look-up. The attacker generates a table that requires $M = N$ blocks of memory in the precomputation phase, such that after intercepting the ciphertext associated to his chosen plaintext he can look up the key in one operation ($T = 1$). Hellman [58] proposed a time-memory trade-off attack which is not only applicable to block ciphers but to random functions in general. As the table look-up the time-memory trade-off attack is a chosen plaintext attack. The attack was originally designed for DES, for which $|\mathcal{K}| = 2^{56}$ and $|\mathcal{C}| = 2^{64}$, and therefore it is assumed $|\mathcal{C}| \geq |\mathcal{K}|$.

We consider a block cipher $E_k(p)$ with block size n using an n_k-bit key k. Let p_0 be a fixed, chosen plaintext. We define

$$f(k) = R(E_k(p_0)),$$

where R is a simple reduction function from n to n_k bits. If $n_k = n$ this reduction function can simply be the identity. We note that inverting the function f is equivalent to cryptanalysis of the cipher E_k.

There are two steps in the attack: a precomputation phase and an online phase. In the precomputation phase the attacker chooses m starting points $SP_1, SP_2, \ldots, SP_m$ independently and uniformly at random from the key space. For $i = 1, \ldots, m$ he sets $X_{i0} = SP_i$ and from each starting point generates the chain

$$X_{ij} = f(X_{ij-1}) \text{ for } j = 1, \ldots, t.$$

This yields a $m \times t$ matrix containing m chains of t elements each, depicted in Figure 4.1.

The endpoint of the ith chain is called EP_i and it holds $EP_i = f^t(SP_i)$. The pairs of start and endpoints $(SP_i, EP_i)_{i=1}^m$ are sorted by the endpoints and stored in a table. All intermediate points are discarded to save memory.

In the online phase the attacker intercepts a ciphertext $c_0 = E_k(p_0)$ and he can easily calculate

$$Y_1 = R(c_0) = f(k).$$

$$\begin{array}{ccccccccc}
SP_1 = X_{10} & \xrightarrow{f} & X_{11} & \xrightarrow{f} & X_{12} & \xrightarrow{f} & \cdots & \xrightarrow{f} & X_{1t} = EP_1 \\
SP_2 = X_{20} & \xrightarrow{f} & X_{21} & \xrightarrow{f} & X_{22} & \xrightarrow{f} & \cdots & \xrightarrow{f} & X_{2t} = EP_2 \\
\vdots & & & & & & & & \vdots \\
SP_m = X_{m0} & \xrightarrow{f} & X_{m1} & \xrightarrow{f} & X_{m2} & \xrightarrow{f} & \cdots & \xrightarrow{f} & X_{mt} = EP_m
\end{array}$$

Figure 4.1: Matrix of images under f.

Then he checks if Y_1 equals one of the endpoints. If $Y_1 = EP_i$ for some $i \in \{1, \ldots, m\}$, then either $k = X_{i,t-1}$ or EP_i has more than one preimage under f. We refer to this event as a false alarm. If $Y_1 = EP_i$ the attacker obtains the corresponding starting point SP_i from the table and calculates $X_{i,t-1}$ by repeatedly applying the function f. It holds $X_{i,t-1} = f^{t-1}(SP_i)$. To verify that $X_{i,t-1}$ is the desired key and no false alarm occurred the attacker can test if $E_{X_{i,t-1}}(p_0) = c_0$ holds.

If Y_1 is not one of the endpoints the attacker calculates $Y_2 = f(Y_1)$ and checks if this is an endpoint. If yes it is either a false alarm or the key is in the $t-2$ column of the matrix in Figure 4.1. Otherwise the attacker continues to iteratively compute $Y_3 = f(Y_2), \ldots, Y_t = f(Y_{t-1})$ and to check whether this value is amongst the endpoints.

If we assume that all mt elements in the matrix are distinct the success probability of the attack is

$$\Pr(success) = \frac{mt}{N},$$

where $N = 2^{n_k}$ is the size of the key space.

However, since f is a random function we expect some overlap in the covered points; because chains collide or repeat themselves. To find the critical values we make use of the birthday paradox.

Lemma 4.1 (The Birthday Paradox)**.**
Suppose $f : \mathcal{C} \to \mathcal{K}$ is a random function with $|\mathcal{K}| = N$. Then we expect a collision after $N^{1/2}$ evaluations of the function.

Now we assume that a matrix with mt distinct elements and an additional chain containing t elements is given. These two sets are likely to be disjunct if $mt^2 \leq N$. Thus, we choose m and t such that $mt^2 = N$ holds. That means a single matrix covers only $\frac{1}{t}$ of the key space. Hellman's idea to cover the whole key space is to generate t unrelated matrices by using different variants f_i of the original function f defined as $f_i(K) = h_i(f(K))$ where h_i is an output modification. The advantage is that even if there is a collision in two points between two different tables this does not imply that also the successive points collide because the generating functions are different.

When we use t matrices of size mt to cover a fixed fraction of the key space the precomputation requires around $P = N$ operations. The total required memory is $M = mt^2$ and the online computation time is $T = t^2$ because we have to perform t applications of some f_i for $i = 1, \ldots, t$. The best trade-off is achieved by choosing the

parameters $m = t = N^{1/3}$. Then the attack requires $M = N^{2/3}$ words of memory and $T = N^{2/3}$ operations in the online phase (cf. Table 4.1).

Table 4.1: Comparison between exhaustive key search, table look-up and time-memory trade-off attack.

	Exhaustive Search		Table Look-up		Time-Memory Trade-off	
	Pre.	Online	Pre.	Online	Pre.	Online
time	0	N	N	small	N	$N^{2/3}$
memory	0		N		$N^{2/3}$	

4.4.2 Time-Memory-Data Trade-off for Stream Ciphers

Hellman's time-memory trade-off for block ciphers is a chosen plaintext attack. The precomputation depends on this fixed plaintext, meaning that additional data would not improve the complexity of the attack. In the case of stream ciphers the setting is different. When we consider synchronous stream ciphers the search space is the set of internal states of the keystream generator and the real time data consists of the first D output bits of the keystream generator which can be obtained by xoring a known plaintext to the ciphertext. The goal is to recover the internal state of the keystream generator. If we know the internal state we can run the keystream generator forward and use the keystream to decrypt further ciphertexts. For many ciphers it is also possible to run the key setup backwards and recover the secret key from a given internal state but this is, however, not the goal of the attack.

The first time-memory trade-off for stream ciphers was independently introduced in [9, 53]. The idea is to use a mapping $f(S) = Y$ that maps all possible states to the first $\log(N)$ bits produced by the keystream generation. This function can be seen as a random function over a set of N elements. In the preprocessing phase the attacker chooses M points S_i for $i = 1, \ldots, M$ at random from the set of all possible internal states, computes the corresponding images Y_i under the function f and stores the pairs sorted by Y_i. In the online phase the attacker obtains $D + \log(N) - 1$ bits of the keystream and derives from it all D possible windows of $\log(N)$ consecutive bits. If one of these windows is contained in the table the attacker can look up the corresponding S_i and recover the internal state. From the birthday paradox we know that two random sets drawn from a set of N elements are likely to intersect if the product of their sizes exceeds N. This leads to the condition $DM = N$, the precomputation time is $P = M$, and the online time is $T = D$.

A more sophisticated time-memory-data trade-off, based on Hellman's time-memory trade-off for block ciphers, was proposed in [18]. The attacker generates t tables containing mt elements by using variants f_i of the function f. However, in the case of stream cipher we can make use of additional data and thus reduce the precomputation. Assuming that we obtain D data points in the online phase we only have to cover N/D

points in the precomputation. This is done by producing only t/D instead of t tables. Thus the required memory is reduced to $M = tm/D$ and the total preprocessing time to $P = N/D$. During the online phase we have to evaluate t/D different functions f_i at the D data points, which yields a complexity of $T = t^2$. Possible parameters are $P = T = N^{2/3}$ and $M = D = N^{1/3}$. In this case the attack seems to be computationally feasible almost up to $N = 2^{100}$. Thus, in order to design a secure stream cipher the internal state size should be reasonable large even if the key size is only say, 80 bits.

4.5 Differential Cryptanalysis

Differential cryptanalysis is together with linear cryptanalysis (Section 4.6) one of the most powerful attacks on symmetric cryptographic primitives. Invented by Biham and Shamir differential cryptanalysis was original used to demonstrate weaknesses of the block cipher FEAL but did not receive much attention until the technique was generalized and applied to DES [16, 17]. DES shows a certain resistance to differential cryptanalysis which suggests that the designer of DES knew about the technique 10 years earlier. Differential cryptanalysis is a very versatile attack, and even though it was designed for iterated block ciphers it has many applications to stream ciphers and hash functions. Furthermore, there exist many variants of the classical differential attack such as Boomerangs (cf. Subsection 4.5.4, [99]), truncated [68] or impossible differentials [13].

As suggested by its name, differential cryptanalysis analyzes the effect of a difference in the plaintext pair on the difference of the corresponding ciphertexts. The purpose is to examine how a difference propagates through a cipher and to use this information to recover parts of the secret key or a round key.

Definition 4.2 (Difference)**.**
The difference Δ between to elements $m_1, m_2 \in G$ is defined as

$$\Delta m = \Delta(m_1, m_2) = m_1 \otimes m_2^{-1}$$

where m_2^{-1} denotes the inverse of m_2 with respect to the group operator $\otimes$.

For most block ciphers the group G is the set of all plaintext blocks, in the case of stream ciphers it might be set of keystreams of a certain length, i.e., in both cases we consider a set of fixed-length bit strings. The group operator $\otimes$ is often the exclusive-or operator. An element x with respect to the exclusive-or is self-inverse. In the sequel we consider a group of fixed-length bit strings together with the exclusive-or as group operator and therefore omit the inverse in the notation.

The reason that the exclusive-or is a popular choice as operator for differences is that in many block ciphers the key application in the round function is realized as a simple XOR. It holds

$$(m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2 \oplus k \oplus k = m_1 \oplus m_2 = \Delta m,$$

meaning that the XOR application of the key preserves the difference. The permutation layer usually does not preserve the difference but changes it in a predictable deterministic way which does not depend on the concrete input words but only on the difference itself. That means the permutation layer is linear with respect to an XOR difference. However, every cipher contains components which show non-linear behavior with respect to the exclusive-or operator; these are often realized as S-boxes (c.f Chapter 2). For these components we cannot predict the output difference to a certain input difference with probability one as we can for the linear parts. But we are still able to gather some information about the propagation of a difference through the S-box. We can use this information to predict the output difference with a certain probability.

The main task in differential cryptanalysis is to find series of input and output differences over several rounds (c.f. Definition 4.3) with high probability. This can be done step by step. We first focus on finding a pair of an input/ an output difference (α, β) that holds with a high probability through one round of encryption. Such a pair of differences is also call one-round differential and denoted by $\alpha \to \beta$. In order to find such a one-round differential the attacker considers the single components of the round functions, tries to find good differentials for those, and combines these differentials to a differential for the whole round.

We consider an iterated block cipher with the round function f

$$c^{(i+1)} = f(c^i, k_{i+1}) = P(S(c^i \oplus k_{i+1}))$$

where c^i denotes the cipher text after i rounds, P is the permutation layer and S the S-box layer.

For the components of the cipher which are linear with respect to the chosen operator it is straight forward to find a differential with probability one. As mentioned before the tricky parts are the non-linear components, the S-boxes. (Another possibility to realize the non-linear part of a cipher is for example modular addition but here we will focus on small S-boxes.) As S-boxes operate on small chunks of data, usually 4 to 8 bits, we can exhaustively search over all input pairs and compute their input difference and the associated output difference. This information is stored in a table, the so-called difference distribution table, where the rows are labeled by input differences and the columns by output differences. Each entry (i, j) denotes the number of input pairs with difference i that lead to an output difference j. The probability of a differential $\alpha \to \beta$ is defined as the ratio of all pairs (m_1, m_2) with input difference α and output difference β to all pairs with input difference α. That means we obtain the probability of an S-box differential by dividing the corresponding entry in the difference distribution table by the number of pairs with the associated input difference. Table 4.2 shows the differences distribution table for the S-box S_D depicted in Figure 4.2 with respect to XOR. For example, the input difference $\alpha = \texttt{8}$ leads with probability $\frac{1}{2}$ to the output difference $\beta = \texttt{5}$. We can observe that all entries in the table are even, this is because the pairs $(m, m \oplus \alpha)$ and $(m \oplus \alpha, m)$ lead to the same difference. Furthermore, if there is no difference in the input to the S-box the output difference is zero with probability one.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	8	c	1	a	7	6	d	4	e	9	5	f	2	3	b	0

Figure 4.2: The S-box S_D in hexadecimal notation.

Table 4.2: The difference distribution table table of $S_D(\cdot)$. The first column contains the input differences while the first row contains the output differences. The entry (i, j) contains the number of pair with the input difference i and output difference j. Here '.' denotes a zero value.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	4	.	.	2	.	.	2	.	2	2	4	.	.	.	.
2	.	.	2	2	.	.	4	.	.	4	2	2	.	.	.	.
3	.	2	4	2	.	.	.	.	2	.	.	2	2	2	.	.
4	.	.	.	.	.	.	.	.	.	.	4	.	4	.	4	4
5	.	.	.	.	2	4	.	2	.	.	.	4	.	2	2	.
6	.	.	.	.	.	4	2	2	2	2	.	.	4	.	.	.
7	.	2	2	.	.	.	2	2	.	.	.	.	2	4	2	.
8	.	.	.	.	4	8	4	.	.	.	.	.	.	.	.	.
9	.	2	2	.	4	.	.	.	.	.	.	.	.	2	2	4
a	.	.	.	4	.	.	2	2	.	.	.	.	2	2	.	4
b	.	.	.	.	2	.	2	4	2	2	.	.	.	2	2	.
c	.	.	.	.	.	.	.	.	2	2	6	2	.	.	.	4
d	.	6	2	.	.	.	.	.	2	.	.	2	.	.	4	.
e	.	.	2	6	.	.	.	.	.	4	.	.	2	2	.	.
f	.	.	2	2	2	.	.	2	6	.	2	.	.	.	.	.

Based on the difference distribution table of the S-boxes we can establish a differential for one round by combining the S-box differentials with the differential of the linear components. These one-round differentials can then be concatenated to a chain of differences over several rounds of the cipher by assuring that the output difference of each round equals the input differences of the following round. Such a chain of differences is called a characteristic.

Definition 4.3 (Characteristic [70])**.**
An s-round characteristic is an (s+1)-tuple of differences $(\alpha_0, \ldots, \alpha_s)$*, where* α_i *is the anticipated difference* Δc^i *after i rounds of encryption, i.e., the difference between the values of the partially encrypted plaintexts. The value of the chosen plaintext difference* $\Delta m = \Delta c^0$ *is denoted by* α_0*.*

In order to clarify the concept of a characteristic and determine its probability we consider a toy block cipher (see Figure 4.3) which operates on blocks of 16 bits. In one round of encryption first the round key is applied by an XOR operation, then the block is divided in chunks of 4 bits which are processed by the S-box S_D given in

Figure 4.2. Afterwards a symmetric bit permutation is applied. The cipher consists of 4 rounds where in the last round the permutation is omitted and an additional post-whiting key is applied. We start with a plaintext pair (m_1, m_2) which has an

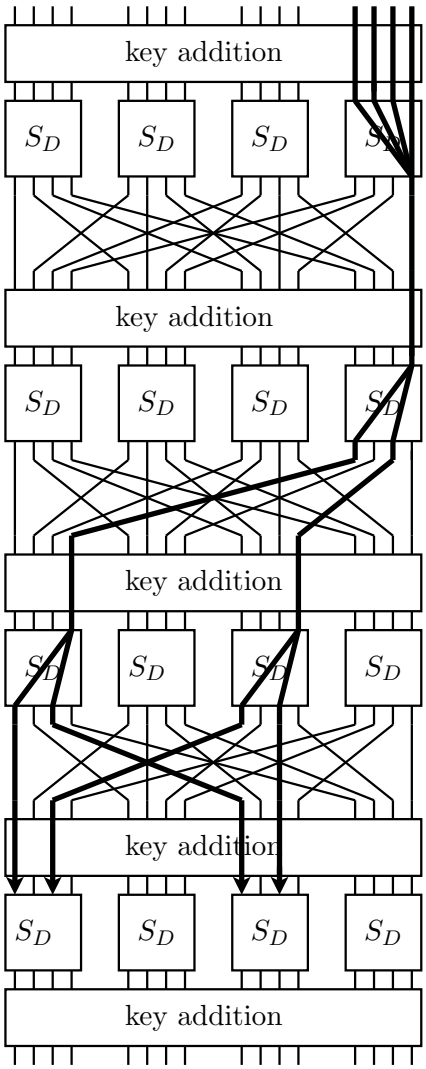


Figure 4.3: A toy block cipher. The bold lines depict a differential characteristic.

input difference $\Delta m =$ `000f`. This means the input differences of the first three S-boxes are zero and the input difference for the last S-box is `f`. From the difference distribution table (Table 4.2) we can infer that the difference `f` will be transformed into the difference `8` with probability $\frac{3}{8}$ by the S-box S_D and that the whole difference will be translated into $\Delta c^1 =$ `0008`. Again, only the fourth S-box has a non-zero input

difference, which will be transformed into difference `5` with probability $\frac{1}{2}$. Continuing this analysis yields the following 3-round characteristic for our toy example:

$$\texttt{000f} \rightarrow \texttt{0008} \rightarrow \texttt{8080} \rightarrow \texttt{5050}.$$

This characteristic is depicted in Figure 4.3.

4.5.1 The Probability of a Differential Characteristic

Before we consider the probability of an s-round characteristic we want to determine the probability of a single round. For the linear components we can predict the propagation of the difference with probability one. The difference distribution table yields the probabilities for the S-box differentials. We call an S-box active if its input difference is non-zero. The probability for a passive S-box is 1 because a zero input difference will trivially lead to a zero output difference. As the S-boxes are applied in parallel to different chunks of the text they are independent. Therefore, the probability of a one-round differential can be calculated as the product of the probabilities of the active S-boxes. As an example we remember the one-round differential $\texttt{8080} \rightarrow \texttt{5050}$. There are two active S-boxes in the example, the S-box differential $\texttt{8} \rightarrow \texttt{5}$ holds with probability $\frac{1}{2}$ for both active S-boxes. This yields

$$\Pr_{\mathcal{P},\mathcal{K}}(\Delta c = \texttt{8080} | \Delta m = \texttt{5050}) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$

for the one-round differential.

The probability of an s-round characteristic is the conditional probability that $\Delta c^i = \alpha_i$ after i rounds of encryption under the condition that $\Delta c^{i-1} = \alpha_{i-1}$ is the difference after $i-1$ rounds of encryption for $i = 1, \ldots, s$, taken over all keys and text pairs:

$$\Pr_{\mathcal{P},\mathcal{K}}(\Delta c^i = \alpha_i, \Delta c^{i-1} = \alpha_{i-1}, \ldots, \Delta c^1 = \alpha_1 | \Delta c^0 = \alpha_0).$$

This probability might be difficult to determine but for a certain class of ciphers, Markov ciphers, we can calculate this probability from the probability of the one-round differentials.

Definition 4.4 (Markov Cipher [73])**.**
An iterated cipher with round function $c^{i+1} = f(c^i, k)$ is a Markov cipher with respect to differential cryptanalysis, if there is a group operator $\otimes$ defining the difference Δ such that for all choices of $\alpha \neq e$ and $\beta \neq e$ (where e is the neutral element of the group)

$$\Pr(\Delta c^{i+1} = \beta | \Delta c^i, c^i = \gamma)$$

is independent of γ when the key k is chosen uniformly at random.

This means that for a Markov cipher the probability of a one-round characteristic taken over all keys and plaintexts equals the probability of a one-round characteristic taken only over all keys. Furthermore, if we have given an iterated Markov cipher

over r rounds together with r independent round keys which are chosen uniformly at random, then the sequence of differences $\Delta c^0, \ldots, \Delta c^r$ is a homogeneous Markov chain.

Definition 4.5 (Markov chain)**.**
A Markov chain is a sequence of random variables $X_0, X_1, \ldots, X_r$ with the property that given the current state the past and the future states are independent, i.e., for $0 \leq i \leq r$

$$\Pr(X_{i+1} = \beta_{i+1} | X_i = \beta_i, \ldots, X_0 = \beta_0) = \Pr(X_{i+1} = \beta_{i+1} | X_i = \beta_i).$$

A Markov chain is called homogeneous if

$$\Pr(X_{i+1} = \beta | X_i = \alpha) = \Pr(X_i = \beta | X_{i-1} = \alpha)$$

i.e., the probability is independent of i for all α and β.

Thus, for a Markov cipher with independent round keys the probability of an s-round characteristic $(\alpha_0, \ldots, \alpha_s)$ can be calculated from the probability of the one-round differential as

$$\Pr(\Delta c^s = \alpha_s, \ldots, \Delta c^1 = \alpha_1 | \Delta c^0 = \alpha_0) = \prod_{i=1}^{s} \Pr(\Delta c^1 = \alpha_i | \Delta c^0 = \alpha_{i-1}). \tag{4.1}$$

In real-life ciphers the round keys are usually not independent but derived from a master key via the key schedule algorithm. However, for many ciphers experiments have shown that this theoretical probability is a very good approximation for the actual probabilities.

Using formula (4.1) we can calculate the probability of the three-round characteristic in Figure 4.3 from the probabilities of the one-round differentials used in the characteristic

$$\begin{aligned} \Pr_{\mathcal{K}}(\Delta c = \texttt{000f} | \Delta m = \texttt{0008}) &= \tfrac{3}{8}, \\ \Pr_{\mathcal{K}}(\Delta c = \texttt{0008} | \Delta m = \texttt{8080}) &= \tfrac{1}{2}, \\ \Pr_{\mathcal{K}}(\Delta c = \texttt{8080} | \Delta m = \texttt{5050}) &= \tfrac{1}{4}. \end{aligned}$$

This yields

$$\Pr_{\mathcal{K}}(\Delta c^3 = \texttt{5050}, \Delta c^2 = \texttt{8080}, \Delta c^1 = \texttt{0008} | \Delta c^0 = \texttt{000f}) = \frac{3}{8} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{3}{64}.$$

In an actual attack the attacker cannot check if the input difference follows the s-round characteristic in each step. He can choose the input difference and has the possibility to check the output difference after s rounds of encryption. That means for carrying out the attack only the input and output difference of the characteristic are important and often there is more than a single characteristic with the same input and output difference. The collection of all s-round characteristics with input difference α_0 and output difference α_s is called a differential.

Definition 4.6 (Differential [73])**.**
An s-round differential is a pair of differences (α_0, α_s), also denoted by $\alpha_0 \to \alpha_s$, where α_0 is the chosen input difference and α_s the expected output difference Δc^s.

The probability of an s-round differential (α_0, α_s) is the conditional probability that given an input difference Δc^0 at the first round, the output difference after the sth round will be Δc^s. More precisely, the probability of a differential (α_0, α_s) is the sum of the probabilities of all characteristics with input difference α_0 and output difference α_s

$$\Pr_{\mathcal{P},\mathcal{K}}(\Delta c^s = \alpha_s | \Delta c^0 = \alpha_0) = \sum_{\alpha_1} \cdots \sum_{\alpha_{s-1}} \Pr_{\mathcal{P},\mathcal{K}}(\Delta c^s = \alpha_s, \ldots, \Delta c^1 = \alpha_1 | \Delta c^0 = \alpha_0). \quad (4.2)$$

It is usually impossible to enumerate all characteristics contained in a differential. Therefore we search for a high probability characteristic which yields a lower bound for the differential.

The probability of a differential (4.2) is taken over all keys and all plaintext pairs. When we actually attack the cipher the key is fixed and the plaintext is variable, thus the probability is taken over all plaintext for a fixed key

$$\Pr_{\mathcal{P}}(\Delta c^i = \alpha_i | \Delta c^{i-1} = \alpha_{i-1}, K = k).$$

Since the key is not known during the attack, we cannot calculate this probability. To overcome this problem the following common assumption is made.

Assumption 4.7 (Hypothesis of stochastic equivalence [72])**.**
We consider an r-round iterated cipher. For virtually all high probability s-round differentials, $s \leq r$, (α, β)

$$\Pr_{\mathcal{P}}(\Delta c^s = \beta | \Delta c^0 = \alpha, K = k) = \Pr_{\mathcal{P},\mathcal{K}}(\Delta c^s = \beta | \Delta c^0 = \alpha)$$

holds for a substantial fraction of the key values k.

Under this assumption we can calculate the probability of a differential and later also the data requirements of an attack using the average probability over all keys. However, for certain keys the differential might have a higher or lower probability than estimated. If the actual probability for a fixed key is much lower than the estimated probability taken as an average over all keys the attack will fail.

4.5.2 Iterative Characteristics

As mentioned before the main task in differential cryptanalysis is finding a high probability characteristic/differential over s rounds. We have seen before that we can construct an s-round characteristic as a concatenation of s one-round differential. However, for a real-life block cipher it is usually very time consuming to find a good characteristic over a sufficient number of rounds and starting with a high-probability

one-round characteristic in the first round will not necessary lead to a high-probability differential over s rounds. Therefore it is desirable to find a characteristic over one or a few rounds with a high probability, where the input difference equals the output difference. This iterative characteristic can then be concatenated with itself to obtain a characteristic of the desired length s.

Definition 4.8 (Iterative characteristic)**.**
An t-round iterative characteristic is a characteristic

$$(\alpha_0, \ldots, \alpha_t),$$

where $\alpha_0 = \alpha_t$.

In our toy cipher example we find the iterative one-round characteristic (`1000` $\rightarrow$ `1000`) with probability $\frac{1}{4}$ which can be concatenated to a characteristic over 3 rounds with probability 2^{-6}.

4.5.3 Key Recovery

A high probability differential over $r-1$ rounds can be used to find parts of the last round key of an r-round iterated block cipher by partially decrypting the last round. We assume that we have given an $(r-1)$-round differential (α, β) with probability p for an iterated r-round block cipher. As before $c^r = f(c^{r-1}, k_r)$ where f denotes the invertible round function and c^r is the intermediated value after r rounds of encryption. The main idea is to encrypt N plaintext pairs (m, m') with the chosen difference $\Delta m = \alpha$ and to obtain the corresponding ciphertext pair (c, c'). Then we make a guess k_G for (parts of) the last round key, (partly) decrypt the last round, and check if the ciphertext pair decrypted with the key guess yields the desired difference after $r-1$ rounds. If the difference is as expected we say that the pair (m, m') suggests the key candidate k_G. When we encrypt N plaintext pairs we expect that around pN pairs follow the differential (α, β). Such pairs are called right pairs.

Definition 4.9 (right pair)**.**
A pair (m, m') with $\Delta m = \alpha$ and associated ciphertexts (c, c') is called a right pair with respect to the $(r-1)$-round differential (α, β) if $\Delta c^{r-1} = \beta$. Otherwise, it is called a wrong pair.

In order for the attack to work we need at least one right pair. However, the basic aim of differential cryptanalysis is to identify a statistically unusual distribution of differences, e.g., we want to identify a right pair, a plaintext pair which follows our differential. But this signal is disguised by noise, wrong pairs that do not follow the characteristic. Therefore it is desirable to eliminate such wrong pairs from consideration in an early stage of the attack in order to make the search for the signal easier.

Often wrong pairs can be eliminated by considering the associated ciphertexts. This process is called filtering. The differential in our toy cipher example (Fig. 4.3) demonstrate an easy filtering criterion. We see that the input difference to the second

and last S-box of the last round is zero, thus the output difference also has to be zero. Therefore we can discard all pairs which ciphertexts have a non-zero difference in the second and last nibble of four bits. A slightly more advanced criterion is to enumerate all possible output differences for the input difference `5` and discard all pairs whose ciphertext differences do not show one of these differences for the first and third S-box. However, this filtering method reduces the number of pairs that have to be considered but not the number of wrongly suggested keys, because such a ciphertext pair will for no key guess yield the desired input difference to the last round and hence not suggest a key. Depending on the cipher and the differential other filtering methods are possible.

Now we are able to outline the attack on an r-round iterated block cipher using an $r-1$ round differential where we recover parts of the last round key. During the attack we demand the encryption of N plaintext pairs. Later we will specify how large N has to be for the attack to be successful.

1. Establish difference distribution tables for the non-linear parts of the cipher.
2. Find a good characteristic $(\alpha_0, \ldots, \alpha_{r-1})$ with probability p.
3. Initialize a counter T_j for all possible round key guesses k_j in the last round.
4. For $i = 1, \ldots, N$ do
   (a) Choose m_i at random, compute $m_i' = m_i \oplus \alpha_0$ and obtain the corresponding ciphertexts (c_i, c_i').
   (b) Use filtering.
   If (m_i, m_i') is a wrong pair, discard it and continue with the next iteration of the loop.
   Otherwise continue with Step 4c.
   (c) For each key guess k_j:
      i. (Partly) decrypt the last round
      $(c_i^{(r-1)}, c_i'^{(r-1)}) = (f^{-1}(c_i, k_j), f^{-1}(c_i', k_j))$.
      ii. If $c_i^{(r-1)} \oplus c_i'^{(r-1)} = \alpha_{r-1}$ then increment the counter T_j.
5. Find the counter $T_{\max}$ with the maximum value amongst all counters.
6. Output $k_{\max}$ as the guess for the correct key.

One pair might suggest several key candidates. However a right pair will always suggest the right key guess, while a wrong pair is expected to suggest a set of key candidates which are basically chosen at random but do not include the correct key [70].

In order for the attack to work the attacker must be able to identify the differential at least once, otherwise he will not be able to distinguish the correct key guess from the wrong key guesses. Furthermore, the correct key value must be suggested significantly more often than other candidates. We expect that these conditions are satisfied for a differential of probability p if we select approximately $\frac{c}{p}$ plaintexts uniformly at

random where c is a small constant. For a more thorough analysis of the attack complexity the reader is referred to [70, 16].

The attack described above recovers parts of the last round key using a differential covering all but the last round of encryption. It is also possible to use shorter differentials and/or to recover parts of the round keys for the first and last rounds.

Moreover several extensions and variations of the attack are known, such as truncated differentials [68] where the difference is not completely defined, or impossible differentials [13] where key candidates can be discarded because they lead to a differential which is not possible. Another variant of a differential attack is the boomerang attack by Wagner [99] which is a chosen plaintext and chosen ciphertext attack. We will describe this attack in the next section and show an application in Chapter 5.

4.5.4 The Boomerang Attack

The boomerang attack [99] is a differential-style attack. It is especially efficient for ciphers where there does not exist a high-probability differential covering the whole cipher but instead there are two good differentials which together cover the whole cipher. These differentials do not have to be related to each other.

Let $E(\cdot)$ be the encryption function of an r-round iterated block cipher and $E = E_1 \circ E_0$ its decomposition where E_0 represents the first s rounds and E_1 the remaining $r-s$ rounds. Furthermore, we suppose that there is a differential $\Delta \to \Delta^*$ through E_0 with probability p_1 and that also its reverse differential $\Delta^* \to \Delta$ holds with probability p_2 through the decryption E_0^{-1}. Moreover we suppose that a differential $\nabla \to \nabla^*$ with probability q through the decryption of the second part of the cipher exist, i.e., through E_1^{-1}.

As mentioned before the boomerang attack is a chosen plaintext and chosen ciphertext attack and proceeds as follows (cf. Figure 4.4). First we request the ciphertexts c and c' corresponding to the plaintexts m and $m' = m \oplus \Delta$. We know that $a \oplus a' = E_0(m) \oplus E_0(m') = \Delta^*$ holds with probability p_1 but we cannot predict the difference $c \oplus c'$. From c and c' we calculate two new ciphertexts $d = c \oplus \nabla$ and $d' = c' \oplus \nabla$ and request the corresponding encryptions n and n'. We know that $a \oplus b = E_1^{-1}(c) \oplus E_1^{-1}(d) = \nabla^*$ and $a' \oplus b' = E_1^{-1}(c') \oplus E_1^{-1}(d') = \nabla^*$ holds with probability q^2. Then we have

$$\begin{aligned} E_0(n) \oplus E_0(n') &= E_0(m) \oplus E_0(m') \oplus E_0(m) \oplus E_0(n) \oplus E_0(m') \oplus E_0(n') \\ &= E_0(m) \oplus E_0(m') \oplus E_1^{-1}(c) \oplus E_1^{-1}(d) \oplus E_1^{-1}(c') \oplus E_1^{-1}(d') \\ &= \Delta^* \oplus \nabla^* \oplus \nabla^* = \Delta^* \end{aligned}$$

with probability $p_1 q^2$. This means under the condition that $b \oplus b' = E_1^{-1}(d) \oplus E_1^{-1}(d') = \Delta^*$ it holds $n \oplus n' = \Delta$ with probability p_2. Starting with a plaintext pair (m, m') with difference Δ this yields a plaintext pair (n, n') with the same difference with probability $p_1 p_2 q^2$ after encrypting the plaintext pair and decrypting an associate ciphertext pair. Often, we assume that the differential and its reverse have the same probability p. This leads to a probability $p^2 q^2$.

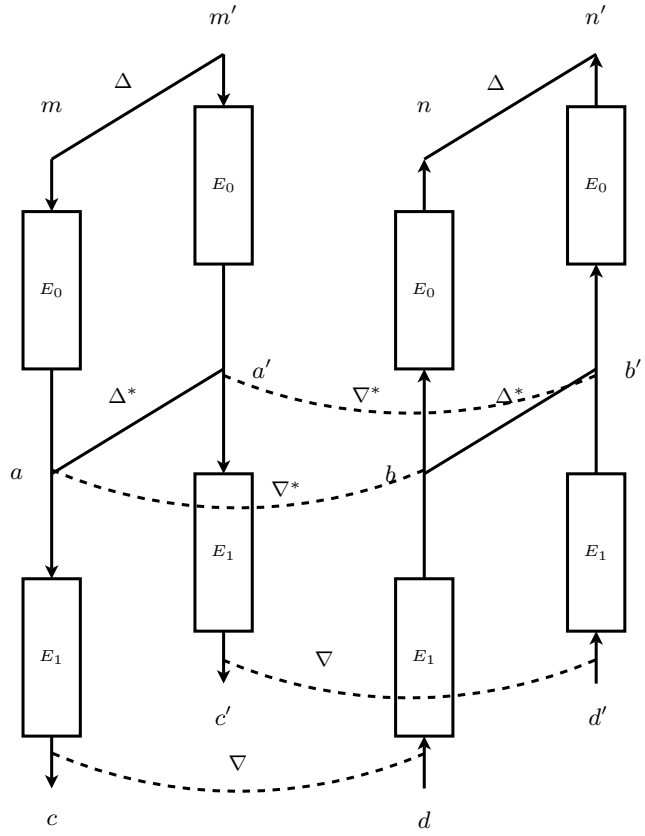


Figure 4.4: Illustration of the Boomerang attack.

There are improvements/extensions of the boomerang attack. The amplified boomerang [63] turns the boomerang attack into a chosen plaintext attack at the expense of an increased amount of required text pairs. The idea is to consider quartets (m', m, n, n') with differences $m \oplus m' = \Delta$ and $n \oplus n' = \Delta$ and try to find a quartet such that $E_0(m) \oplus E_0(m') = \Delta^*$, $E_0(n) \oplus E_0(n') = \Delta^*$, $E_0(m) \oplus E_0(n) = \nabla^*$ and thus $E_0(m') \oplus E_0(n') = \nabla^*$. The attacker can check for the property $c \oplus d = \nabla$ and $c' \oplus d' = \nabla$ in order to identify right quartets. When we encrypt N pairs (m, m') and (n, n') with difference Δ the expected number of right quartets is $N^2 2^{-n} p^2 q^2$ [15, 96].

Another improvement introduced in [63] is that the difference ∇^* in the differential $\nabla^* \to \nabla$ for the second part E_1 of the cipher does not have to be specified. Instead we can allow any differential $\nabla_i^* \to \nabla$. That gives us an improvement on the probability of the differential through E_1. The probability $\Pr(\nabla^* \to \nabla)^2 = q^2$ of the pairs (a, b) and (a', b') (cf. Figure 4.4) to have output difference ∇ can be replaced by the probability $\sum_i \Pr(\nabla_i^* \to \nabla)^2$.

A similar improvement is the rectangle attack [15]. Instead of fixing the output difference for the differential through E_0 to Δ^* the attacker sorts the pairs into piles according to their output difference and performs the original attack for each of those piles.

Coming back to the original boomerang attack we note that its name is derived from the property that we get a plaintext pair with the same difference as we started with: "when you send it properly, it always comes back to you" [99]. However, recently Murphy [85] discovered that some boomerangs never return. In [99] it is stated that the probability of a boomerang is $p_B \geq p^2q^2$, nevertheless, Murphy constructed examples of boomerangs for round-reduced versions of AES and DES with $p > 0$ and $q > 0$ but the probability of the boomerangs itself is 0, i.e., the boomerang never comes back. These counter examples point to a flaw in the analysis of boomerang attacks. Therefore attacks based on boomerangs or related concepts should be handled carefully and the existence of a boomerang as well as its estimated probability should be experimentally verified.

Despite Murphy's concerns successful attacks based on boomerangs do exists. The existence of the boomerang has been experimentally verified, for example in Wagner's attack on COCONUT98 [99] and in the attack on C2 [22], which will be presented in Chapter 5.

4.5.5 Cube Attacks

The cube attack, also published under the name AIDA [98] as an attack on the stream cipher Trivium, is a differential-style attack and can be seen as a special case of higher order differentials [71]. We follow the notation and naming of [45].

Cube attacks are applicable to cryptographic schemes that can be described by a tweakable polynomial over $\mathbb{F}_2$ which contains both secret variables and public variables. Secret variables may be key bits while public variables are for example plaintext or IV bits. The complexity of a cube attack depends mainly on the degree of the polynomial. Given a random polynomial of degree d in l secret variables the estimated attack complexity is $2^{d-1}l + l^2$ bit operations. However, the degree of a polynomial that describes a cryptographic scheme is usually quite high and will therefore dominate the expression.

The attack is divided into two phases. In the preprocessing phase the attacker should be able to set the values of all variables, the secret and the public ones. This corresponds to that he has a full description of the cipher. The tweakable polynomial itself does not have to be known, it is sufficient if it is available as black box. In the online phase the attacker should be able to tweak the public variables only. For the attack to be successful it is necessary that the number m of public variables exceeds $d + \log_d l$.

Before we explain the procedure of the two different phases we start with some notations and definitions. The tweakable polynomial which describes one output bit

of the cipher is called the master polynomial p

$$\begin{aligned} p : \mathbb{F}_2^{l+m} &\to \mathbb{F}_2 \\ (k_1, \ldots, k_l, v_1, \ldots, v_m) &\to p(k, v) \end{aligned} \tag{4.3}$$

where $k_1, \ldots, k_l$ denote the secret and $v_1, \ldots, v_m$ denote the public variables. This polynomial is not known but can be evaluated as a black box.

In the following we ignore the distinction between secret and public variables and label all variables with $x_1, \ldots, x_n$ where $n = l + m$. For an index set $I \subset \{1, \ldots, n\}$, t_I denotes the monomial which contains all variables $x_{i_1}, \ldots, x_{i_r}$ with $i_j \in I$ for $j = 1, \ldots, r$. It is possible to denote all monomials over $\mathbb{F}_2$ in this way since $x_i^2 = x_i$ modulo 2. Every polynomial over $\mathbb{F}_2$ can then be represented as a sum of terms t_I. Furthermore given an index set I and a polynomial p, we can represent the polynomial as a collection of monomials which are indexed by supersets of I and a collection of monomials whose index sets lack at least one index of I. This leads to the definition of the superpoly.

Definition 4.10 (Superpoly)**.**
Given $I \subset \{1, \ldots, n\}$ and a polynomial p, p can be represented as

$$p(x_1, \ldots, x_n) = t_I \cdot p_{S(I)} + q(x_1, \ldots, x_n),$$

where $p_{S(I)}$ does not contain any common variable with t_I and q misses at least one variable of t_I. The polynomial $p_S(I)$ is called superpoly of I in p.

Of special interest are terms t_I such that the superpoly is linear.

Definition 4.11 (Maxterm)**.**
A term t_I is called maxterm of p if $\deg(p_{S(I)}) = 1$, i.e., if the superpoly of I in p is a linear polynomial and not a constant.

If we consider an index set I of size r, it defines a r-dimensional Boolean cube C_I. (Hence the name cube attacks). This Boolean cube contains all 2^r vectors v in which the variables in I take all possible assignments in $\{0, 1\}^r$, while the remaining $n - r$ variables are undetermined. If we plug a vector $v \in C_I$ into the polynomial p we obtain a new polynomial $p_{|v}$ in $n - r$ variables with degree less than or equal to d. This leads us to the main observation that cube attacks are based on.

Theorem 4.12.
Any vector $v \in C_I$ defines a polynomial $p_{|v}$ in $n - r$ variables. Summing over all 2^r possible polynomials generated by vectors in C_I yields the superpoly of I in p

$$p_{S(I)} = \sum_{v \in C_i} p_{|v}.$$

The reader is referred to [45] for the proof of this theorem. This theorem also says that if we sum over a cube of size r we will obtain a superpoly in $n - r$ variables,

meaning that the degree of the superpoly is at most $n - r$. If we sum over a cube corresponding to a maxterm t_I we will obtain a linear polynomial.

The idea of the attack is now that given a master polynomial p of degree d the adversary has to find sufficiently many maxterms which only contain public variables and derive the corresponding superpolys in the preprocessing phase. These superpolys should be degree one polynomials in some of secret variables. In the online phase the attacker sums over at most 2^{d-1} values for each maxterm and obtains a linear equations system in the secret variables.

The goal of the preprocessing phase is to find a description of the secret bits or a part of the secret bits as a linear equation system where the right-hand side of the equation system is for the time being undetermined. This equation system is obtained by finding sufficiently many maxterm in the master polynomial p and is independent of th secret key.

Identifying the maxterm would be easy if the master polynomial were given in an explicit way. However, the master polynomial is only given as a black box, but the attacker is allowed to tweak the secret and public variables. Under the assumption that t_I is a maxterm he derives the corresponding linear polynomial in the following way.

1. Summing the values of p over the cube C_I modulo 2 where all variables which are not cube variables are set to zero yields the constant term of the linear superpoly.

2. By summing the values of p over the cube C_I modulo where $x_i = 1$ and all other variables which are neither cube variables nor x_i are set to zero one obtains the coefficient of x_i. If flipping a the value of x_i in the summation changes the value of the sum, the coefficient of x_i is 1.

Given a random polynomial of degree d, each term t_I consisting of $d - 1$ public variables corresponds to a superpoly of degree at most 1, the probability that t_I actually is a maxterm is $1 - 2^{-n}$. In order to obtain l linear equations l maxterms are needed. Thus, we need $d + \log_d l$ public variables. The complexity of the preprocessing phase for a random polynomial of degree d is $O(l^2 \cdot 2^{d-1})$.

However, master polynomials corresponding to an encryption scheme are not random. This causes several problems for the attack. First of all, in the theoretical attack we look for terms of the form $v_{i_1} \cdots v_{i_d} \cdot k_j$ which consist of several public and only one secret variable. For a non-random polynomial there is no guarantee that such terms exist at all. Secondly, the degree d of the master polynomial is unknown. That means that the attacker has to guess the dimension of the Boolean cube and test if the corresponding superpoly is constant, non-linear or the desired polynomial of degree 1.

In the online phase the attacker is able to tweak the public variables while the secret variables are set to their fixed secret values. The online phase consists of two parts. In the first part, for each maxterm t_I the attacker sums the values of p over C_I in order to obtain the right hand side of the corresponding equation, meaning that the attacker obtains the evaluation of the polynomial p for all vectors where the variables

in t_I take all possible values. The remaining public variables are fixed to arbitrary values and the secret variables are fixed to their secret values. The complexity is $2^d \cdot l$ where d is the dimension of the cubes.

When the adversary has obtained the right hand side for all l linear equations, he has to solve the system of l linear equations in l unknowns.

Cube attacks were applied to a reduced version of Trivium [45]. The authors claim that Trivium can be broken with an online complexity of 2^{30} if the key setup is reduced from 1152 to 736 initialization rounds.

We conclude this section on differential cryptanalysis with pointing out once more that differential style attacks are extremely powerful tools for a cryptanalyst. We show applications of differential style attacks in Part II of this thesis.

4.6 Linear Cryptanalysis

Linear cryptanalysis is another important technique in the cryptanalysis of block ciphers. It was developed and successfully applied to DES by Matsui in 1993 [75]. Linear cryptanalysis is a known plaintext attack where the idea is to find linear relations between plaintext, ciphertext and key bits with a good probability such that information about the secret key can be deduced from the plaintext and the ciphertext. For an iterated block cipher we establish this linear relation by searching for good linear approximations of each round of the cipher and concatenating these approximations to a linear expression over several rounds (cf. differential cryptanalysis). It is trivial to describe the linear functions of a cipher by a linear expression that holds with probability 1. Therefore our aim is to find good linear approximations for the non-linear functions of the cipher, for example the S-boxes.

For a non-linear function $S : \{0,1\}^n \to \{0,1\}^m$ we consider the input variables $X = (x_{n-1}, \ldots, x_0)$ and output variables $Y = (y_{m-1}, \ldots, y_0) = S(X)$ of $S(\cdot)$ and try to find an n-bit binary vector α and an m-bit binary vector β such that

$$\alpha \cdot X = \beta \cdot Y$$

holds for a number of inputs bounded away from $\frac{2^n}{2}$. The binary vectors α and β are called linear masks. Usually, so-called linear approximation tables are computed for the non-linear functions of a block cipher. A linear approximation table contains all possible input and output masks together with the corresponding probability that the resulting linear approximation is preserved through the non-linear function. Instead of considering the probability $p = \Pr(\alpha \cdot Y = \beta \cdot Y)$ we consider the bias $\epsilon = p - \frac{1}{2}$. Here a high absolute value of the bias ϵ is interesting, while the sign of the bias does not matter. For the S-box $S_L(\cdot)$ shown in Figure 4.5 the linear masks $\alpha = (0,0,1,1)$ and $\beta = (0,0,0,1)$ yield the linear equation $x_1 \oplus x_0 = y_0$ which holds with probability $p = \frac{1}{8}$ (cf. Table 4.3). That means that the bias is negative. But the linear expression $x_1 \oplus x_0 \oplus 1 = y_0$ holds with probability $1 - p = \frac{7}{8}$. Thus this linear approximation has a positive bias.

An approximation for the non-linear functions can be combined with a description of the linear functions to obtain a linear approximation for one round of the cipher.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S_L(x)$	c	b	a	1	f	4	e	5	9	0	8	3	7	6	2	d

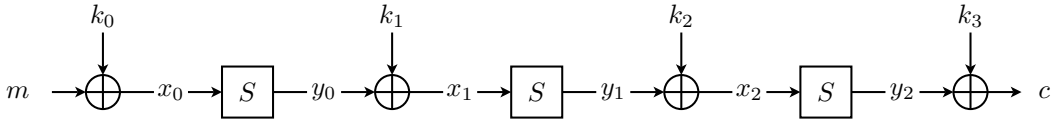


Figure 4.5: A toy cipher. The table above shows the S-box S_L in hexadecimal notation.

Afterwards the one-round linear approximations can be joined to a linear characteristic (Definition 4.13), which is a linear approximation for several rounds of the cipher.

Definition 4.13 (Linear Characteristic)**.**
An s-round linear characteristic is an $(s+1)$-tuple $(\alpha_0, \ldots, \alpha_s)$ with probability p where α_i is the input mask to the ith round.

An s-round linear characteristic for an s-round block cipher yields one bit of information about the key. In order to clarify the attack idea and explain how one-round approximations are joined to longer linear characteristics we consider the toy cipher defined in Figure 4.5. This cipher can be seen as an iterated block cipher consisting of three rounds together with a post-whitening key after the last round. There is no permutation layer, therefore the only linear function in this cipher is the key addition. The non-linear part, the substitution layer, is realized as a 4-bit S-box S_L. We now assume that we have given the following linear approximation for the single parts of the cipher

$$\begin{aligned}
\alpha \cdot (m \oplus k_0) &= \alpha \cdot x_0 && \text{with probability 1,} \\
\alpha \cdot x_0 &= \beta \cdot y_0 && \text{with probability } p_0, \\
\beta \cdot (y_0 \oplus k_1) &= \beta \cdot x_1 && \text{with probability 1,} \\
\beta \cdot x_1 &= \gamma \cdot y_2 && \text{with probability } p_1, \\
\gamma \cdot (y_2 \oplus k_2) &= \gamma \cdot x_3 && \text{with probability 1,} \\
\gamma \cdot x_3 &= \delta \cdot y_3 && \text{with probability } p_2, \\
\delta \cdot (y_3 \oplus k_3) &= \delta \cdot c && \text{with probability 1.}
\end{aligned}$$

The linear description of the key addition and the linear approximation for the S-box of each round can be added to a linear approximation of the whole round. The linear approximation $\alpha \cdot m \oplus \alpha \cdot k_0 = \beta \cdot y_0$ holds for example with probability p_0. Together the linear approximations for the single rounds yield the linear characteristic $(\alpha, \beta, \gamma, \delta)$ over several rounds and the corresponding linear equation

$$\alpha \cdot m \oplus \delta \cdot c = \alpha \cdot k_0 \oplus \beta \cdot k_1 \oplus \gamma \cdot k_2 \oplus \delta \cdot k_3 \tag{4.4}$$

for the toy cipher in Figure 4.5. As in the case of differential cryptanalysis the search for good linear characteristics over several rounds is difficult. Therefore, it is convenient to build an r-round linear characteristic from a shorter s-round linear characteristic with a high bias where the input and output masks are the same.

Definition 4.14 (Iterative linear characteristic)**.**
An s-round iterative linear characteristic is an s-round linear characteristic $(\alpha_0, \ldots, \alpha_s)$ *where* $\alpha_0 = \alpha_s$.

In our example we can build a 3-round characteristic `(b,b,b,b)` for the iterative one-round characteristic `(b,b)` which has a bias of $-\frac{6}{16}$ (Table 4.3). After we have established a linear characteristic over a sufficient number of rounds, we can use this knowledge to deduce information about the secret key.

4.6.1 Key Recovery

At first we use a linear characteristic to deduce one bit of information about the key. The attack can then be extended to allow us to recover more than one bit of information at once. The idea is to incorporate partial decryption similar to the differential attack, meaning that we guess (parts of) the last round key and decrypt (parts of) the last round.

Let us assume the linear approximation (4.4) holds with probability $p = \frac{1}{2} + \epsilon$ and $\epsilon > 0$. (Later in the chapter we will discuss how to determine the probability of a linear characteristic.) The right hand side of Equation (4.4) only depends on key variables and is therefore fixed to a value $b \in \{0, 1\}$. If N plaintexts are encrypted we expect that the left hand side of the equation takes value b for a fraction of pN plaintext/ciphertext pairs and $b \oplus 1$ for $(1-p)N$ pairs. The attack proceeds as follows. We encrypt N plaintexts and evaluate the left hand side of Equation (4.4) for each of these plaintext/ciphertext pairs. We initialize two counters T_0 and T_1. If the left hand side of the equation is 0 for a plaintext we increment the counter T_0, otherwise we increment T_1. If T_0 is the higher counter in the end, we deduce that the right hand side of Equation (4.4) equals 0, otherwise it equals 1. This gives us one bit of key information.

However, we would like to recover more than one bit of key information at the time. Using partial decryption of the last round the attack can be extended as follows. We consider an r-round iterated block cipher.

1. Find an $(r-1)$-round linear characteristic

   $$(\alpha_0, \ldots, \alpha_{r-1})$$

   with a high absolute value of the bias ϵ for the r-round block cipher.
   This characteristic corresponds to the linear approximation

   $$\alpha_0 \cdot m \oplus \alpha_{r-1} \cdot x_{r-1} = \alpha_0 \cdot k_0 \oplus \cdots \oplus \alpha_{r-1} \cdot k_{r-1} \tag{4.5}$$

   where x_{r-1} is the input to the last round.

2. Create a list of key guesses k_G for the last round key (or the partial last round key), create counters T_0^G and T_1^G.
3. For each of the N known plaintext/ciphertext pairs and for each key guess k_G do
   (a) Decrypt (or partial decrypt) the last round to get the value of x_{s-1} under key k_G.
   (b) Evaluate the left hand side of Equation (4.5).
   (c) Increment the counter T_0^G if the result is zero, otherwise increment T_1^G .
4. Identify the counter T_j^G with the maximum value and take k_G as the correct round key. (For a wrong key guess we expected the counters to be around $\frac{N}{2}$ each.)

We note that often it is not necessary to guess the whole last round key. The linear approximation usually only contains a few bits of the input x_{r-1} to the last round. This means that we only have to guess the key bits of the last round key which are necessary to decrypt the last round partially in such a way that we get the bits of x_{r-1} used in the linear approximation.

If the linear approximation used in the linear attack holds with bias ϵ then approximately $c \cdot \epsilon^{-2}$ known plaintext/ciphertext pairs are needed for the attack to be successful where c is a small constant.

4.6.2 The Probability of a Linear Characteristic

In order to determine the attack complexity it is necessary to estimate the probability of a linear characteristic. The linear approximation of one round can be seen as a random variable of the form $\alpha_0 X_0 \oplus \cdots \oplus \alpha_n X_n \oplus \beta_0 Y_0 \oplus \cdots \oplus \beta_m X_m$ which either takes the value zero or one (depending on the key bits). A linear characteristic is then the XOR of these random variables and the probability of the linear characteristic can be computed using the Piling-up Lemma (Lemma 4.15). Before we state the lemma we give an example to motivate it.

We consider two independent random variables X_1 and X_2. It holds $\Pr(X_i = 0) = p_i$ and $\Pr(X_i = 1) = 1 - p_i$ for $i \in \{1, 2\}$. Then it follows from the independency of X_1 and X_2 that $\Pr(X_1 = 0, X_2 = 0) = p_1 p_2$ and $\Pr(X_1 = 1, X_2 = 1) = (1 - p_1)(1 - p_2)$. Thus

$$\Pr(X_1 \oplus X_2 = 0) = p_1 p_2 + (1 - p_1)(1 - p_2).$$

This results holds in general and is known as the Piling-up Lemma.

Lemma 4.15 (Piling-up Lemma)**.**
Let X_i for $i = 1, \ldots, n$ be n independent binary random variables and assume that it holds $\Pr(X_i = 0) = p_i$ for all $1 \leq i \leq n$. Then

$$\Pr(X_1 \oplus \cdots \oplus X_n = 0) = \frac{1}{2} + 2^{m-1} \prod_{i=1}^{m} (p_i - \frac{1}{2}).$$

Table 4.3: The linear approximation table of $S_L(\cdot)$. The first column contains the input masks while the first row contains the output masks. The entry (i, j) divided by 16 contains the bias of the probability that the linear approximation with input masked by i and output masked by j holds. Here '.' denotes a zero value.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	2	-2	.	.	2	2	-4	-4	-2	-2	.	.	2	-2	.
2	.	.	.	4	-2	-2	2	-2	.	4	.	.	-2	2	2	2
3	.	-6	2	.	-2	.	.	-2	.	-2	-2	.	2	.	.	2
4	.	.	2	2	6	-2	.	.	-2	-2	.	.	.	.	2	2
5	.	2	4	-2	-2	.	-2	.	-2	.	-2	.	-4	-2	.	2
6	.	.	2	-2	.	.	-2	2	-2	2	.	.	2	6	.	.
7	.	2	.	-2	.	-6	.	-2	2	.	-2	.	2	.	-2	.
8	.	.	.	.	-2	-2	-2	-2	-2	-2	6	-2	.	.	.	.
9	.	2	-2	.	-2	.	.	2	-2	.	.	2	4	-2	2	4
a	.	.	.	4	.	.	-4	.	-2	2	-2	-2	2	-2	-2	-2
b	.	2	2	.	.	2	2	.	2	.	.	-6	2	.	.	2
c	.	.	-2	-2	.	.	-2	-2	.	.	-2	-2	.	.	6	-2
d	.	2	.	2	.	2	-4	-2	4	-2	.	2	.	2	.	2
e	.	.	-2	2	-2	-2	.	4	.	-4	-2	-2	-2	2	.	.
f	.	2	4	2	-2	.	2	.	.	-2	.	2	2	.	2	-4

A requirement of the Piling-up Lemma is that the random variables or the linear approximation respectively are independent. This cannot always be guaranteed. In order to be able to use the Piling-up Lemma we make some independence assumptions. We define a Markov cipher with respect to linear cryptanalysis analogous to a Markov cipher with respect to differential cryptanalysis.

Definition 4.16 (Markov cipher [70])**.**
An iterated cipher is called a Markov cipher with respect to linear cryptanalysis, if

$$|\Pr(\alpha \cdot X = \beta \cdot Y | Y = \gamma) - 1/2|$$

is independent of γ for all α and β, when the round key k is chosen uniformly at random.

In practice it might not be possible to determine the key even though we found a good linear characteristic. The reason being that there might be other linear characteristics with the same first and last linear mask but their biases contradict the bias of the linear characteristic used in the attack. This effect can also lead to a higher probability than expected. For the attack only the first and last mask of the linear characteristic matter. We do not need to specify the intermediate masks. This leads to the definition of a linear hull.

Definition 4.17 (Linear hull)**.**
An s-round linear hull is tuple (α_0, α_s) of linear masks with probability p. It predicts

that the input masked with α_0 equals the output after s round masked with α_s with probability p.

The linear hull contains all s-round characteristics $(\alpha_0, \alpha_{1_i}, \ldots, \alpha_{s-1_i}, \alpha_s)$ for all possible values α_{j_i}, $j = 1, \ldots, s-1$. It is unclear how to determine the bias of a linear hull correctly [84]. The actual complexity of the attack is $c \cdot \epsilon_L^{-2}$ where ϵ_L is the bias of the linear hull. Even with the difficulty to estimate or lower bound the attack complexity linear cryptanalysis is a powerful technique. It has been successfully applied to DES by Matsui [75]. The full 16-round DES was broken using 2^{43} known plaintexts.

Nowadays it is a general design principle that a block cipher or stream cipher is strong towards linear and differential cryptanalysis.

4.7 Algebraic Attacks

More recently, the so-called algebraic attacks have received much attention. They exploit the fact that many cryptographic primitives can be described by a sparse multivariate non-linear equation system over the binary field in such a way that solving this equation system recovers the secret key (or the initial state in the case of stream ciphers). The equation system is usually generated by finding a description of the single components of the cipher and then joining these together to an equation system that represents the whole cipher. Considering an SP-network we first obtain an algebraic description of the linear components of the round function such as the key addition and the permutation layer and then we set up an equation system describing the S-boxes which normally contains non-linear equations. Merging these equations yields an algebraic description of the round function which can be extended iteratively to a multivariate equation system of the whole cipher. This equation system can be set up generally; given a plaintext/ciphertext pair or a part of the keystream we can plug this information into our general equation system and use it for recovering the secret key.

In general, solving random systems of multivariate non-linear Boolean equations is an NP-hard problem [51]. However, when the system has a specific structure, we can hope that methods exist which are faster than enumerating all possible solutions.

A first approach is to try to solve the equation system with Gröbner bases techniques. However, most of the time this approach is not successful because the memory requirements and the complexity of the Gröbner bases algorithms are hard to predict.

Another technique to tackle such equation systems is linearization. Here each non-linear term is replaced by an independent linear variable. The resulting linear systems can be solved using Gaussian elimination. Linearization works only if there are enough linearly independent equations in the resulting system which requires the initial system to be very sparse and highly overdetermined. An approach is to generate an overdetermined system by adding new equations to it. The XL algorithm [36] increases the number of equations by multiplying them with all monomials of a certain degree. It has been refined to the XSL algorithm [38], which, when applied to the

AES, exploits the special structure of the equation system. Neither the XL nor the XSL algorithm have been able to break AES but algebraic attacks were successful in breaking a number of stream cipher designs [8, 37].

The representation of a cipher as a sparse multivariate non-linear Boolean equation system is also the starting point for the novel cryptanalysis introduced in Part III of this thesis. We believe that the special structure which is exhibited by the equations makes them suitable for optimization methods. The difference between algebraic attacks and this novel cryptanalysis is that in the latter we are not operating over the finite fields anymore but consider either integer and real-valued equations or mappings from a finite set into the reals.

Part II

Cryptanalysis of Block Ciphers with Secret Components

Chapter 5

Cryptanalysis of C2

In this chapter we present several attacks on the block cipher C2 [5], which is used for encrypting DVD Audio discs and Secure Digital cards. We presented C2 in Chapter 2 as an example of a Feistel network operating on 64-bit blocks with a 56-bit key. The tweak in this cipher is that the 8-to-8 bit S-box is kept secret and is available under license from the 4C entity. The S-box is application dependent and can be considered as part of the secret key.

The specification of the system gives rise to several attack scenarios for C2.

1. The 56-bit key can be chosen by the attacker, who will attempt to determine the values in the secret S-box.

2. The S-box is known to the attacker, who will attempt to determine the value of a secret 56-bit key.

3. Both the 56-bit key and the S-box are unknown to the attacker, who will attempt to determine the values of both.

We attack C2 in all three scenarios. The first attack requires 2^{24} chosen plaintext queries with a negligible amount of other computations required in the on-line phase. The complexity of the second attack is around 2^{48} adaptive chosen ciphertext queries and a similar amount of computations. The third attack requires $2^{53.5}$ adaptively chosen ciphertext queries.

The only other cryptanalytical result on C2 we are aware of is the S-box recovery attack (scenario 1) on 8 rounds of the cipher by Weinmann [100].

5.1 Recovering the Secret S-box with a Chosen Key Attack

The first attack depends on the details of the key schedule. We show that by carefully selecting the value of the 56-bit key, we can ensure that only three S-box entries are used in the first seven rounds of encryption of a chosen plaintext. By a trail-and-error approach these three entries can be determined. Subsequently the other entries of the S-box can be determined in a similar approach.

We assume in this attack scenario that we are allowed to set the encryption key. The ith round key is generated from the master key by rotating it $17 \cdot i$ positions to the left, xoring the round counter to the 5th byte, processing the output by the S-box, shifting it and adding it to the lower 4 bytes of the master key (cf. Chapter 2

Figure 2.3). The resulting four bytes are output as the round key. It is important to note that during the round key generation only one S-box application is used and that the master key itself is not updated. Thus, given the master key it is easy to predict the inputs to the S-box during the key schedule. Furthermore, we can observe that some keys generate only very few different inputs to the secret S-box in the key scheduling. It is easy to verify using a computer search that the smallest number of inputs generated in the key scheduling is three. An example of such a master key is

$$\texttt{0x40}, \texttt{0x84}, \texttt{0x88}, \texttt{0x40}, \texttt{0x02}, \texttt{0x80}, \texttt{0x09}$$

and the inputs to the S-box in rounds 1 to 10, generated by this master key, are the following

$$\texttt{0x88},\ \texttt{0x4},\ \texttt{0x27},\ \texttt{0x27},\ \texttt{0x4},\ \texttt{0x4},\ \texttt{0x27},\ \texttt{0x27},\ \texttt{0x88},\ \texttt{0x88}.$$

For the attack we first fix the above key and guess the possible outputs of the S-box for the inputs `0x04`, `0x27` and `0x88`. Under the assumption that our guess is correct we know the round keys used during the encryption. For each possible guess we generate one plaintext that only uses these three S-box inputs during 7 rounds of encryption. For such a plaintext, again under the assumption that our guess is correct, we know the output of the encryption process after 7 rounds, i.e., (L_7, R_7).

These plaintexts are completely independent of the actual S-box used in the attack (but depend on the different guesses for the S-box entries). Therefore this precomputation has only to be done once, is trivially parallelizable and the data can be reused for any actual S-box in the implementation of C2 we want to attack. Generating a plaintext for one fixed key guess requires approximately $2^{19.25}$ C2-encryptions and as there are 2^{24} possible values for the three entries in the secret S-box the complexity for this step is approximately $2^{43.25}$ C2-encryptions. We will give the details in Section 5.1.1. We computed a table containing one plaintext for each guess. The actual running time was 96 hours and the size of the table is less than 400 MBytes.

In the online phase, when we attack an actual device or implementation using a secret S-box, we proceed as follows. We set the device key to the master key given above which only triggers the three different S-box inputs. Then we encrypt each of the 2^{24} plaintexts in the table. This is one plaintext corresponding to one possible guess for the three S-box entries. If our guess is correct we know the output after round 7. It is possible to check whether the observed ciphertext fits to our guess of the 7th round output. We explain this test in Section 5.1.2. However, this test will never fail for the right guess and has a (heuristic) probability of accepting a wrong guess of 2^{-29}. Thus, on average, only the right guess will survive. Using the outlined approach we can recover three S-box entries with 2^{24} encryptions using the actual device and marginal overhead for the test.

After the first three entries have been recovered we continue in a very similar way. First, it is now easy to recover (up to) three additional entries corresponding to the inputs triggered in the last three rounds without querying the device. For all other entries we now generate plaintexts that do not trigger any unknown inputs in the

first six rounds. Using the three round test explained in Section 5.1.2 on any possible output of the S-box in round 7 we can recover the output of the S-box in the 7th round and later recover the output of the S-box in the last three rounds again.

Assuming that the inputs to the S-box in rounds $7, 8, 9$ and 10 behave randomly, an estimate for the complexity (in terms of C2 encryptions) of successfully recovering the whole S-box is derived from the well-known coupon collector's problem [50, Section II.7]. The coupon collector's problem deals with the following question. Given n different coupons which are drawn with replacement, what is the probability that more than T samples are needed to collect all n coupons. It holds that the expected time T to collect all n coupons is

$$E(T) = n \cdot H_n,$$

where H_n is the nth harmonic number.

For each plaintext we can recover a maximum of 4 S-box entries. Applying the coupon collector problem leads to a complexity given by

$$C\frac{(256 \cdot H_{256})}{4} \approx 2^{19.4},$$

where H_{256} is the 256th harmonic number and C is the complexity to generate a plaintext that fit for 6 rounds. As explained in Section 5.1.1, C can be upper bounded by

$$C \leq \left(\frac{256}{6}\right)^2.$$

However, it turns out that those inputs do not behave purely random and experimentally we measured a slightly higher complexity of $2^{20.2}$ as an average of 10000 trials (100 tests for 100 randomly generated S-boxes).

To sum up, when we are allowed to choose an encryption key, the S-box can be recovered with less than 2^{24} queries to the device on average. Of course, the actual running time highly depends on the encryption speed of the device, but for an implementation on a standard PC the whole S-box can be recovered in less than 30 seconds.

5.1.1 Generating Plaintexts that Fit for Seven Rounds

Next, we describe a procedure to generate a plaintext such that for known (or guessed) round keys and a set of known (or guessed) input/output pairs D the inputs to the S-box in the first seven rounds are within the set D.

A naive method would be to randomly generate plaintexts and verify that the plaintext satisfies the conditions in all seven rounds. Under the assumption that those inputs behave randomly, the effort to generate such a plaintext is $\left(\frac{256}{|D|}\right)^7$. For the first part of the attack, where $|D| = 3$, this is

$$\left(\frac{256}{3}\right)^7 \approx 2^{44.9}.$$

As we have to generate one plaintext for each possible guess of the three S-box entries, meaning that we have to generate a total of 2^{24} such plaintexts, the complexity of this naive approach is $2^{68.9}$ and thus too high. However, it is easy to generate plaintexts that fulfill the conditions for four out of the seven rounds by construction. Then, again assuming things behave randomly, the effort is reduced to $\left(\frac{256}{|D|}\right)^3$ which for $|D| = 3$ and 2^{24} plaintexts to be generated gives an overall complexity of approximately $2^{43.25}$.

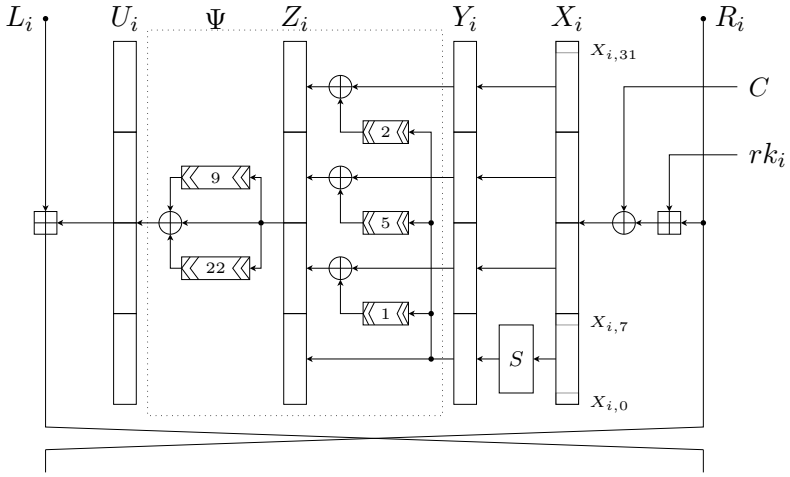


Figure 5.1: Equivalent description of the round transformation of C2

In the following the names of the variables refer to Figure 5.1. In order to generate a plaintext that satisfies the conditions in the first four rounds we proceed as follows. First, we choose the inputs to the S-box in the first four rounds in order to get these inputs correct. That means we fix $X_{0,0..7}, X_{1,0..7}, X_{2,0..7}$ and $X_{3,0..7}$ to arbitrary input values in the set D. Furthermore we choose $X_{1,8..31}$ and $X_{2,8..31}$ randomly. Given these values we can compute backwards

$$\begin{aligned} R_{0,0..7} &= (X_{0,0..7} \oplus C_{0..7}) - rk_{0,0..7} \pmod{2^8}, \\ R_{1,0..7} &= (X_{1,0..7} \oplus C_{0..7}) - rk_{1,0..7} \pmod{2^8}, \\ R_{2,0..7} &= (X_{2,0..7} \oplus C_{0..7}) - rk_{2,0..7} \pmod{2^8}, \\ R_{3,0..7} &= (X_{3,0..7} \oplus C_{0..7}) - rk_{3,0..7} \pmod{2^8}, \end{aligned}$$

where $C =$ `0x2765ca00` iss a constant. Let F denote the function mapping X_i to U_i. We observe that for any 8-bit vector x it holds that

$$F(X \oplus (x << 23))_{0..7} = F(X)_{0..7} \oplus x.$$

In particular we can choose bytes z_1 and z_2 such that

$$F(X_1 \oplus (z_1 << 23))_{0..7} - R_{2,0..7} = R_{0,0..7} \pmod{2^8}$$

and

$$F(X_2 \oplus (z_2 << 23))_{0..7} + R_{1,0..7} = R_{3,0..7} \pmod{2^8}.$$

Thus, if we choose

$$L'_2 = (X_1 \oplus (z_1 << 23) \oplus C) - rk_1 \pmod{2^{32}}$$

and

$$R'_2 = (X_2 \oplus (z_2 << 23) \oplus C) - rk_2 \pmod{2^{32}}$$

and decrypt two rounds to get a plaintext (L'_0, R'_0), then we ensure that for this plaintext all inputs to the S-box in the first four rounds are previously fixed and thus in the set D. We experimentally verified that the complexity of generating plaintext that also fit in the fifth, sixth and seventh round for $|D| = 3$ is approximately $(\frac{256}{3})^3 \approx 2^{19.25}$ as predicted by the heuristic. The overall running time to generate all 2^{24} plaintexts for each guess was less than one hour when distributed to 100 CPUs.

5.1.2 A Three-Round Test

To verify that we guessed the correct S-box entries we need to check whether the intermediate value after 7 rounds of encryption, which we predict based on our guess, matches the obtained ciphertext. This means that we have to check if encrypting (L_7, R_7) with three rounds gives us the correct ciphertext (L_{10}, R_{10}). This test would be trivial if we knew the S-box. However, we still can do it efficiently and with very good probability even without knowing the S-box.

We do this by establishing a system of linear equations which holds for the correct guess of the S-box entries with probability 1. Since we know the values of R_7 and R_{10}, we can compute $U_8 = R_{10} - R_7$ and going backwards through the linear function, we can determine Y_8. Since we do not know the S-box, we know only the 24 most significant bits of X_8. We known the round key rk_8 for our guess of the S-box entries. This means that if we knew whether a carry in the modular addition occurred or not, we would know the 24 most significant bits of R_8 and L_9. Thus, we can calculate $R_8 = L_9$ for both possibilities (a carry occurs and no carry occurs).

Now, using this knowledge and the values of L_7 and L_{10} we can determine 24 most significant bits of $U_7 = R_8 - L_7$ and $U_9 = L_{10} - L_9$. Again, we do not know the carry bit so we have to test two possibilities for each of the words, either assuming a carry occurred or not. Provided that the carries are as predicted, we know the 24 most significant bits of U_7 and U_9.

In the next step we check if the values obtained by this procedure match the values we expect if our guess is correct. We can apply such a test in round 7 and 10. We focus on the 7th round first. We want to compare the values of U_7 obtained by the above procedure with $U'_7 = \Psi(Y_7)$, where Ψ is a $\mathbb{F}_2$-linear map (marked with a dotted box in Figure 5.1). The mapping

$$\begin{aligned} \Psi : \mathbb{F}_2^{32} &\longrightarrow \mathbb{F}_2^{32} \\ Y &\longmapsto U \end{aligned}$$

can be described by the set of linear equations given in Figure 5.2.

$$
\begin{aligned}
u_0 &= y_0 + y_1 + y_2 + y_{10} + y_{23} \\
u_1 &= y_1 + y_2 + y_6 + y_{11} + y_{24} \\
u_2 &= y_2 + y_3 + y_7 + y_{12} + y_{25} \\
u_3 &= y_0 + y_3 + y_4 + y_{13} + y_{26} \\
u_4 &= y_1 + y_4 + y_5 + y_{14} + y_{27} \\
u_5 &= y_2 + y_5 + y_6 + y_{15} + y_{28} \\
u_6 &= y_6 + y_{16} + y_{29} \\
u_7 &= y_7 + y_{17} + y_{30} \\
u_8 &= y_7 + y_8 + y_{18} + y_{31} \\
u_9 &= y_6 + y_9 + y_{19} \\
u_{10} &= y_7 + y_{10} + y_{20} \\
u_{11} &= y_0 + y_{11} + y_{21} \\
u_{12} &= y_1 + y_{12} + y_{22} \\
u_{13} &= y_2 + y_{13} + y_{23} \\
u_{14} &= y_6 + y_{14} + y_{24} \\
u_{15} &= y_7 + y_{15} + y_{25}
\end{aligned}
\qquad
\begin{aligned}
u_{16} &= y_0 + y_3 + y_7 + y_{16} + y_{26} \\
u_{17} &= y_1 + y_4 + y_7 + y_8 + y_{17} + y_{27} \\
u_{18} &= y_0 + y_2 + y_5 + y_9 + y_{18} + y_{28} \\
u_{19} &= y_1 + y_3 + y_6 + y_{10} + y_{19} + y_{29} \\
u_{20} &= y_2 + y_4 + y_7 + y_{11} + y_{20} + y_{30} \\
u_{21} &= y_0 + y_3 + y_5 + y_{12} + y_{21} + y_{31} \\
u_{22} &= y_0 + y_1 + y_4 + y_{13} + y_{22} \\
u_{23} &= y_1 + y_2 + y_5 + y_{14} + y_{23} \\
u_{24} &= y_2 + y_{15} + y_{24} \\
u_{25} &= y_7 + y_{16} + y_{25} \\
u_{26} &= y_0 + y_{17} + y_{26} \\
u_{27} &= y_1 + y_{18} + y_{27} \\
u_{28} &= y_2 + y_{19} + y_{28} \\
u_{29} &= y_3 + y_{20} + y_{29} \\
u_{30} &= y_0 + y_4 + y_7 + y_8 + y_{21} + y_{30} \\
u_{31} &= y_0 + y_1 + y_5 + y_9 + y_{22} + y_{31}
\end{aligned}
$$

Figure 5.2: Equations describing $\Psi : Y \to U$.

We cannot compare U_7 with U'_7 directly because we do not know bits $U_{7,0..7}$ and the unknown output of the S-box masks bits of U'_7, meaning we do not known the bits $Y_{7,0..7}$. However, we can compare linear combinations of bits of U_7 and $\Psi(Y_7)$ that do not depend on any of the unknown bits $U_{7,0..7}$ and $Y_{7,0..7}$. As an example we consider the equations for u_8 and u_{10}. The linear combination of these bits

$$u_8 + u_{10} = y_8 + y_{18} + y_{31} + y_{10} + y_{20}$$

does not depend on any unknown bits. Thus we can check whether it holds that $U_{7,8} + U_{7,10} = U'_{7,8} + U'_{7,10}$. There are 16 linear equations $\xi_j(U_7) = \xi_j(\Psi(Y_7))$ involving bits of U_7 and Y_7 that do not use any unknown bits. If the pair $(L_7, R_7), (L_{10}, R_{10})$ matches and we guessed all the carries correctly, all these equations will be satisfied. For an unrelated pair of inputs and outputs, this happens with probability 2^{-16}. The same happens for the test in round 10. We combine those two tests with a simple guessing of all the carries we need to know to obtain our testing procedure. For each of the two possible values of the carry in round 9, we test independently two possible carries in round 7 and round 10. If for any combination of these all the 32 pairs of check equations are satisfied, we conclude the pair matches. Otherwise, we reject the pair.

This procedure always accepts right pairs $(L_7, R_7), (L_{10}, R_{10})$ as they will always produce a match in one of the tested carry combinations. To accept a wrong pair which is not coming from the encryption, all the 32 pairs of check equations would need to be satisfied for one of the 2^3 combinations of carries. This happens with

probability 2^{-29} if the values are uniformly distributed. We experimentally verified that the probability is indeed around 2^{-29}. This is sufficient for us since we need to test only 2^{24} possibilities.

5.2 Search for S-box Independent Characteristics

The only components of C2 that are not linear over $\mathbb{F}_2$ are the S-box and the two modular additions. As the S-box is secret and therefore its differential behavior is unknown, we focus on characteristics not involving the S-box. Note that if the input to the round function has zero difference in the least significant byte $R_{i,0..7}$, this zero difference cannot be destroyed by carries in the modular key addition. Thus, we can search for differential characteristics independent of the S-box by focusing on characteristics with $R_{i,0..7} \oplus R'_{i,0..7} = 0$.

To search for these characteristics we consider a linear model of the round function, that is, we replace the modular addition by XORs and assume that the S-box is the identity (or any linear mapping as the characteristic will be independent of this choice anyway). This linear model of the round function

$$(L_i,\ R_i) = (R_{i-1},\ L_{i-1} \oplus F(R_{i-1}, K_i))$$

can be written as

$$(L_i, R_i) = (L_{i-1}, R_{i-1}) \cdot M$$

where M is a 64×64 matrix over $\mathbb{F}_2$. Furthermore, the condition that the input difference to the S-box, i.e., the least significant byte of the output difference, must be zero can be described as $((L, R)M)Q = 0$ where Q corresponds to the projection on the least significant 8 bits. Thus, for the linearized version of the cipher, the problem of finding a characteristic which has a zero input differences to the S-box is reduced to the problem of calculating the kernel of the linear mapping $x \to x \cdot M \cdot Q$. The kernel of the matrix $K = [Q|M \cdot Q| \cdots |M^i \cdot Q]$ contains all differences which have a zero input difference to the S-box over $i+1$ rounds. This kernel is non trivial for $i \leq 8$ implying that for a version of C2 where the modular additions are replaced by XORs a characteristic over 9 rounds with probability 1 exists independently of the S-box.

As modular additions are not linear over $\mathbb{F}_2$ we need to estimate the probability that the modular addition behaves like an XOR. Here we are interested in the following two cases.

1. The probability that the key addition behaves like an XOR

   $$\Pr[(C \boxplus K) \oplus ((C \oplus \alpha) \boxplus K) = \alpha]$$

   where C and K are random bit strings and α is the known difference.

2. The probability that the addition of the left half and the output of function F behaves like an XOR

   $$\Pr[(L \boxplus F) \oplus ((L \oplus \alpha) \boxplus (F \oplus \beta)) = \alpha \oplus \beta]$$

   where L and F random bit strings and α and β fixed known differences.

These probabilities have been studied for example in [74] where it was shown that

$$\Pr[(C \boxplus K) \oplus ((C \oplus \alpha) \boxplus K) = \alpha] = 2^{-(\mathrm{hw}(\alpha) - \mathrm{msb}(\alpha))}$$

and

$$\Pr[(L \boxplus F) \oplus ((L \oplus \alpha) \boxplus (F \oplus \beta)) = \alpha \oplus \beta] = 2^{-(\mathrm{hw}(\alpha \vee \beta) - \mathrm{msb}(\alpha \vee \beta))}$$

where $\mathrm{hw}(\alpha)$ denotes the Hamming weight of α and $\mathrm{msb}(\alpha)$ the most significant bit.

Since the probability of an XOR characteristic mainly depends on the Hamming weight of all the intermediate input differences, we searched for characteristics minimizing it. This problem is equivalent to searching for low weight code words in the linear code generated by the matrix $B \cdot [I|M|\cdots|M^i]$ where B is the basis matrix of the kernel of K. Such an approach has been used before for finding differential characteristics in dedicated hash functions, cf. [81].

The best five round characteristic we found is

$$\begin{aligned}\Delta =&(\texttt{0x00020800}\,\texttt{0x80200100})\\ \rightarrow&(\texttt{0x80200100}\,\texttt{0x80000000})\\ \rightarrow&(\texttt{0x80000000}\,\texttt{0x00000000})\\ \rightarrow&(\texttt{0x00000000}\,\texttt{0x80000000})\\ \rightarrow&(\texttt{0x80000000}\,\texttt{0x80200100})\\ \rightarrow&(\texttt{0x80200100}\,\texttt{0x00020800})\end{aligned}$$

which does not require non-zero differences to the S-box in any round, and it has Hamming weight 15 over all intermediate input differences. Using the above formulas from [74] one gets a probability of 2^{-12} for independent round inputs and keys. Experimentally, the probability for randomly chosen master keys and S-boxes was even better, namely approximately $2^{-11.17}$, which is due to a differential effect which takes place inside the 5-round characteristic.

The differential characteristic can also be specified for the last five rounds of C2 and the average probability was estimated to be similar to the one for the first five rounds.

During the search for differentials we found a total of 11 code words with Hamming weight 15. They are listed in Appendix A. The characteristic given above has the highest probability amongst these 11 characteristics and therefore we believe it is the best five-round characteristic which is independent of the S-box.

5.3 Key-Recovery Attack for a Known S-box

We have seen in Section 5.2 that we can establish a characteristic with a high probability which is independent of the S-box and covers the first as well as the second half of the cipher. This is the condition for mounting a boomerang attack on the whole cipher C2 [99, 63, 15]. As explained in Section 4.5.4 a boomerang attack is a chosen plaintext and a chosen ciphertext attack that involves two encryptions and

Table 5.1: Examples of boomerang plaintext pairs for different keys and S-boxes.

S-box used	key (hex)	plaintext
AES	00 00 00 00 00 00 00 00	5707aec0 48a9c942
	00 30 20 08 00 20 28	0f42cd03 b7b5f077
	'c' 'r' 'y' 'p' 't' '0' '9'	b4b32db5 589913dc
C2 facsimile [7]	00 00 00 00 00 00 00 00	3af32bac 960693e1
	ee 9b 7f 2b 7c 26 cd	69676fdc 339879d4
	'c' 'r' 'y' 'p' 't' '0' '9'	d6b44956 36771c9d

two decryptions. As the five-round characteristic has an experimentally determined probability of $2^{-11.17}$ for the encryption and decryption direction the theoretical probability of the corresponding boomerang is $2^{-44.68}$. In the light of [85] it is important to experimentally verify the existence of such a boomerang and to confirm the estimated probability. By testing 1000 random keys and multiplying probabilities of passing the first five and the last five rounds, we confirmed that boomerangs exist with an average probability of $2^{-44.5}$. We observed that for all such boomerangs the pairs of texts followed the characteristic in the first round every time, but not always in later rounds. The reason why one can obtain a boomerang anyway is the differential effect which is utilized also in the so-called rectangles [15]. We further observed a large variability in the probabilities over the keys and some keys were found for which the probability of the boomerang is as high as 2^{-32} but there are also keys for which no boomerangs were found. We present some of actual boomerangs we found in Table 5.1.

We use this boomerang to recover bits of the first round key. The possibility of finding boomerangs enables us to test if the differences in the first round propagate according to the characteristics. If not, we do not expect to get any boomerangs. We will use this observation to recover bits of the first round key rk_0 by a careful analysis of the carries appearing in the addition $R_0 \boxplus rk_0$. This method resembles the approach used by Contini and Yin to partially recover HMAC keys using a pseudo-collision differential for MD5 [33].

5.3.1 Recovering Bits of the First Round Key

In this section we describe how the boomerang attack which is outlined above is applied to recover up to 22 bits of the first round key.

Given a plaintext pair (L_0, R_0) and $(L_0 \oplus \alpha, R_0 \oplus \beta)$, for which the boomerang returns, we know that the pair follows the five-round characteristic at least for the first round. That means that we know with an overwhelming probability the difference after the first round. We also know that the difference after the modular key addition is still β because we assume that the modular additions behave like XORs when a plaintext pairs follows the characteristic. Thus, the first round key satisfies the

equation

$$(R_0 \boxplus rk_0) \oplus ((R_0 \oplus \beta) \boxplus rk_0) \quad = \quad \beta. \tag{5.1}$$

We can represent the modular addition of the right half R_0 and the round key rk_0 as exclusive-or of the right half, the round key and a vector of carry bits

$$R_0 \boxplus rk_0 = R_0 \oplus rk_0 \oplus c(R_0, rk_0)$$

where $c(R_0, rk_0)$ denotes the vector of carry bits of the modular addition of R_0 and rk_0 with

$$c_{-1} = 0 \text{ and } c_i = R_{0,i} rk_{0,i} \oplus c_{i-1} R_{0,i} \oplus c_{i-1} rk_{0,i}.$$

Hence, (5.1) can be rewritten as

$$R_0 \oplus rk_0 \oplus c(R_0, rk_0) \oplus R_0 \oplus \beta \oplus rk_0 \oplus c(R_0 \oplus \beta, rk_0) = \beta$$

which is equivalent to

$$c(R_0, rk_0) \quad = \quad c(R_0 \oplus \beta, rk_0) \tag{5.2}$$

and furthermore implies

$$\beta_i \; rk_{0,i} = \beta_i \; c_{i-1}.$$

This means that whenever $\beta_i = 1$ the previous carry bit equals the key bit. This carry bit potentially depends on all previous key bits. Hence, we can use this relationship to recover key bits by carefully choosing plaintexts of a special structure and checking whether there exists a boomerang in the class of these plaintexts.

On the downside, Equation 5.2 implies that we cannot extract any key bits beyond first most significant non-zero bit of β which is not the most significant bit of β. (As the addition of the most significant bit does not introduce any carry we cannot determine this bit even though it is one.) Using the five-round characteristic from Section 5.2 we can therefore at most recover 22 bits of the first round key using (5.2).

In the following we describe how bits of the round key can be found one at a time. We start with recovering the first 9 key bits starting from the key bit $rk_{0,8}$ and then recursively recover the remaining bits $rk_{0,7}, \cdots, rk_{0,0}$. In order to determine $rk_{0,8}$ we start by fixing the 8 least significant bits of R_0 to zero, the remaining bits of the plaintext are chosen uniformly at random. Fixing the 8 least significant bits to zero ensure that $c_7 = 0$. Equation (5.2) implies that boomerangs with this additional constraint exist if and only if $rk_{0,8} = 0$.

Thus, if we find a boomerang we can conclude $rk_{0,8} = 0$ and if after sufficiently many trials, we do not find any boomerang, we can conclude that $rk_{0,8} = 1$. Let us estimate the probability of making a mistake and wrongly assuming that $rk_{0,8} = 1$ while in reality it holds that $rk_{0,8} = 0$.

If 2^{-b} is the probability of a boomerang and we make our decision after $t2^b$ tries, then the error probability can be approximated by

$$(1 - 2^{-b})^{t2^b} = \left((1 - 2^{-b})^{2^b}\right)^t \approx \left(\frac{1}{e}\right)^t.$$

After recovering $r_{0,8}$ we modify our choice of plaintexts adaptively in order to recover the previous round key bits. Our considerations are based on the actual value of the round key bits we have already determined and the relation between the carries

$$c_i = R_{0,i}rk_{0,i} \oplus c_{i-1}R_{0,i} \oplus c_{i-1}rk_{0,i}.$$

In order to recover $r_{0,7}$ we choose plaintexts such that the 8 least significant bits of R_0 equal 10000000. This means that we ensure that $c_6 = 0$. First, we consider the case $rk_{0,8} = 0$. This means that a boomerang only exists if $c_7 = 0$. However, $c_7 = 0$ if and only if $r_{0,7} = 0$. Thus, after sufficiently many tries, we can with a good probability recover $rk_{0,7}$.

Next, we consider the case where $rk_{0,8} = 1$. Again $c_7 = 0$ if and only if $rk_{0,7} = 0$. However, in this case boomerangs exist only when $c_7 = 1$, thus when $rk_{0,7} = 1$.

This procedure can now be applied recursively to finally recover all the key bits $rk_{0,0\ldots7}$. After those bits have been successfully recovered a very similar argument allows the recovery of the key bits $rk_{0,21\ldots8}$.

Now, we want to estimate the complexity of our attack. We assume that the average complexity for finding the boomerang is 2^{44}. Furthermore, we assume that in half of the cases a boomerang is found after 2^{44} trails and that in half of the cases we do not find a boomerang, meaning that we have to try $t2^{44}$ plaintexts for $t > 1$. Then the overall complexity of this procedure to recover B bits for a random key can be estimated to

$$B \cdot \left(\frac{t2^{44} + 2^{44}}{2} \right) \tag{5.3}$$

and the error probability is approximately

$$1 - \left(1 - \left(\frac{1}{e} \right)^t \right)^B \quad . \tag{5.4}$$

If we want to recover 8 bits with a success probability of more than 0.5 we have to choose $t = 2.48$ and the effort will be $2^{47.8}$. The remaining 48 bits of the master key can then be recovered with a brute force search.

If we want to recover all 22 bits with a success probability of more than 0.99 we have to choose $t = 7.7$ and the effort will be $2^{50.59}$.

We have mentioned before that the complexity of the boomerang varies a lot for different keys, meaning that for a given key the actual probability of a boomerang is unclear. However, there are several ways to deal with this problem. One possibility is to first get an estimate of the probability by running the boomerang search for randomly selected plaintexts. Another possibility is to double the time until we decide on a key bit when no boomerang has been found step by step until the right key has been found.

It should be noted that, even though our attack has a better overall complexity, it might still be slower in practice than a simple exhaustive key search which can be distributed. Actually such a brute force attack on C2 has been carried out [2] but was unsuccessful as the S-box guess turned out to be wrong.

Table 5.2: A list of master key bits used to generate the round keys in rounds 1 up to 10.

round	master key bits used for the addition	bits input to the S-box
1	$\{0,\ldots,31\}$	32 33 34 35 36 37 38 39
2	$\{39,\ldots,55\}\cup\{0,\ldots,14\}$	15 16 17 18 19 20 21 22
3	$\{22,\ldots,53\}$	54 55 0 1 2 3 4 5
4	$\{5,\ldots,36\}$	37 38 39 40 41 42 43 44
5	$\{44,\ldots,55\}\cup\{0,\ldots,19\}$	20 21 22 23 24 25 26 27
6	$\{27,\ldots,55\}\cup\{0,1,2\}$	3 4 5 6 7 8 9 10
7	$\{10,\ldots,41\}$	42 43 44 45 46 47 48 49
8	$\{49,\ldots,55\}\cup\{0,\ldots,24\}$	25 26 27 28 29 30 31 32
9	$\{32,\ldots,55\}\cup\{0,\ldots,7\}$	8 9 10 11 12 13 14 15
10	$\{15,\ldots,46\}$	47 48 49 50 51 52 53 54

5.4 Key and S-box Recovery with Chosen Ciphertext Attack

In this attack scenario we want to recover the secret key and the secret S-box at the same time. The first observation we make is that we can determine the master key from the first two and the last round keys (see Table 5.2).

As explained in Section 5.3.1 we can recover the least significant 22 bits of the first round key with an average complexity of $2^{50.59}$ and an error probability less then 0.01 using the boomerang attack. But turning the boomerang upside down, we can similarly recover 22 bits of the last round key with the same complexity. We explain in Section 5.4.1 how we can recover the remaining bits of these round keys and one entry of the secret S-box with an average complexity of 2^{52}. The knowledge of the first round key and one entry of the S-box allows us to recover the second round key (see Section 5.4.2) with an average effort of $2^{45.32}$. Thus we can determine the entire master key uniquely.

After we have recovered the master key we focus on determining the remaining S-box entries. We can recover additional entries of the secret S-box with an effort of 2^{44} by again applying the approach of Section 5.4.2. After recovering four more entries (with an effort of 2^{44+2}) of the S-box corresponding to what is triggered in the key scheduling in rounds $3, 4, 5$ and 6 we can use an attack very similar to the attack described in Section 5.1 to recover the remaining entries of the S-box. Namely, we guess the remaining three S-box entries triggered in the key scheduling in rounds $7, 8$ and 9. For each possible guess we generate a plaintext that does not trigger any unknown (or not guessed) S-box entries in the first seven rounds. As we know or guessed 10 entries already the effort of generating such a plaintext is $(256/10)^3 \approx 2^{14}$. We encrypt each of those plaintexts and use the three-round test of Section 5.1.2 to verify our guess. This way we recover all 10 S-box entries used in the key scheduling and afterwards the remaining entries are recovered just as in Section 5.1 with a complexity of less than 2^{20}. The complexity of recovering the S-box is therefore

$2^{24+14} = 2^{38}$ and the overall complexity of the attack is

$$2 \cdot 2^{50.59} + 2 \cdot 2^{52} + 2^{45.3} + 2^{44+2} + 2^{38} + 2^{20} \approx 2^{53.5}$$

on average.

For this attack scenario, a brute force search is not an option, as not only the key but the entire S-box would have to be guessed, all together 2104 bits or 1740 if we assume that the S-box is a permutation.

5.4.1 Recovering Remaining Unknown Round Key Bits

Once we know bits $rk_{0,0..21}$ of the first round key we can recover the remaining most significant bits of the round key and the output of the S-box using the carry behavior of the left addition $L_0 \boxplus U_0$.

Given a plaintext for which the boomerang exists, we know that the following equation is true

$$(L_0 \boxplus U_0) \oplus [(L_0 \oplus \alpha) \boxplus (U_0 \oplus \Psi(\beta))] = \texttt{0x80000000},$$

where $\alpha = \texttt{0x00020800}$ and Ψ is a linear function mapping bits of Y_0 to U_0, hence $\Psi(\beta) = \texttt{0x80020800}$. Since the difference in the most significant bit always propagates linearly as it does not induce any carries, we can focus on a simplified version of the above equation

$$(L_0 \boxplus U_0) \oplus [(L_0 \oplus \alpha) \boxplus (U_0 \oplus \alpha)] = 0.$$

As in Section 5.3.1 we can represent the modular addition as an XOR of the summands and a vector of carry bits $c(\cdot, \cdot)$ introduced by modular addition

$$L_0 \oplus U_0 \oplus c(L_0, U_0) \oplus L_0 \oplus \alpha \oplus U_0 \oplus \alpha \oplus c(L_0 \oplus \alpha, U_0 \oplus \alpha) = 0.$$

This yields

$$c(L_0, U_0) = c(L_0 \oplus \alpha, U_0 \oplus \alpha)$$

and it simplifies to the condition

$$\alpha_i(L_{0,i} \oplus U_{0,i} \oplus 1) = 0. \tag{5.5}$$

In α the 11th and 17th bit are set to 1. Thus (5.5) allows us to determine bits $U_{0,11}$, $U_{0,17}$ by trying to find boomerang plaintexts for all of the four possible combinations of $L_{0,11}$, $L_{0,17}$ in parallel. One of the choices will yield a boomerang and it contains the correct combination of values of $L_{0,11}$, $L_{0,17}$ that determine the values of bits of U_0.

We have $U_{0,11} = Y_{0,0} \oplus Y_{0,11} \oplus Y_{0,21}$ and we can compute the values of $Y_{0,11}, Y_{0,21}$ because we know R_0 and the round key bits $rk_{0,0..21}$. Thus, we learn one bit $Y_{0,0}$ of the output of the S-box for a known input. Furthermore, we get another equation $U_{0,17} = Y_{0,1} \oplus Y_{0,4} \oplus Y_{0,7} \oplus Y_{0,8} \oplus Y_{0,17} \oplus Y_{0,27}$.

The same principle can be used to recover more bits. In order to do this, we need differences to appear at other bit positions in the addition $L_0 \boxplus U_0$. We can achieve

this by inducing carry chains in the first addition $R_0 \boxplus rk_0$ by appropriately setting some bits of the plaintext so that the difference $\beta =$ `0x80200100` will trigger more bit flips in $R_0 \boxplus rk_0$. More precisely, we find plaintexts R_0 such that

$$(R_0 \boxplus rk_0) \oplus [(R_0 \oplus \beta) \boxplus rk_0] = \beta \oplus \gamma.$$

for some carry-induced difference γ. Remember that Ψ, mapping Y_0 to U_0, is linear and induces an extra difference $\Psi(\gamma)$, thus $U_0 \oplus U_0' = \Psi(\beta) \oplus \Psi(\gamma)$.

Later, we try to compensate for this extra difference in U_0 by the additional difference $\Psi(\gamma)$ in L_0. This situation can be described as

$$(L_0 \boxplus U_0) \oplus [(L_0 \oplus \alpha \oplus \Psi(\gamma)) \boxplus (U_0 \oplus \Psi(\beta) \oplus \Psi(\gamma))] = \texttt{0x80000000}.$$

If this equation holds (and we know this when we find a boomerang) we have the following conditions

$$(\alpha_i \oplus \Psi_i(\gamma))(L_{0,i} \oplus U_{0,i} \oplus 1) = 0$$

which allow us to determine bits of U_0 at positions i where $\alpha_i \oplus \Psi_i(\gamma) = 1$.

Because of the effect of Ψ, each bit in γ usually requires 3 additional compensating bits of the difference in L_0 and this means we need to search for 8 boomerangs in parallel to determine the correct values of L_0. After we find one, we obtain three more equations as explained before.

The complexity of this procedure depends on the configuration of the carry chains we are able to induce and this in turn depends on the round key. Assuming we can extend the difference in $\beta =$ `0x80200100` at position 8 to chains at positions 8-9, 8-10, 8-11, 8-12, 8-13, 8-14, 8-15 (so γ is `00000200`, `00000600`, `00000c00`, etc.) we get enough equations to uniquely determine the unknown bits of Y_0. We need to test 2^3 combinations of values of bits in L_0 and the total complexity is $2^3 \cdot 2^3 \cdot 2^{44} = 2^{52}$ where 2^{44} is the cost of finding the boomerang plaintext. For other configurations of secret key bits we may not be able to extend γ by one bit at a time and we will need to test more bits in L_0 each time. In that situation we usually need to test less cases though because we learn more bits of U_0 at the same time. The exact increase in complexity depends very much on the particular case.

Note that we can always perform a search for the 13 missing bits (10 key bits and 3 bits output of the S-box) by randomly choosing plaintext pairs (L_1, R_1) and (L_1', R_1') with a difference corresponding to the second round difference of the five-round characteristic, decrypting them using all possible guesses for the missing 13 bits and searching (in parallel) for a boomerang for all 2^{13} pairs. This upper bounds the complexity of recovering the remaining bits in the first round by $2^{13} \cdot 2^{44} = 2^{57}$.

5.4.2 Attacking the Second Round

We know the entire first round key and one entry of the secret S-box. This means that we can start the boomerang from the second round. For this we choose pairs (L_1, R_1) and (L_1', R_1') with the input difference of the five-round characteristic where $L_{1,0..7} = R_{0,0..7}$ is fixed to the known S-box entry and compute backwards the corresponding

values for (L_0, R_0) and (L'_0, R'_0). For the lower part of the boomerang we can now use the five-round characteristic truncated to the first 4 rounds:

$$\begin{aligned}\Delta =&(\texttt{0x00020800}\,\texttt{0x80200100})\\ \rightarrow&(\texttt{0x80200100}\,\texttt{0x80000000})\\ \rightarrow&(\texttt{0x80000000}\,\texttt{0x00000000})\\ \rightarrow&(\texttt{0x00000000}\,\texttt{0x80000000})\\ \rightarrow&(\texttt{0x80000000}\,\texttt{0x80200100}).\end{aligned}$$

This shortened boomerang will give a right pair (L''_0, R''_0) and (L'''_0, R'''_0) with an average probability of $2^{-(2\cdot 11+2\cdot 8)} = 2^{-38}$. However, we cannot directly verify that the pair (L''_1, R''_1) and (L'''_1, R'''_1) has the required difference because with a high probability we do not know the S-box entry in the first round of encryption and thus cannot decrypt the first round. But for a right pair it holds that

$$L''_1 \oplus L'''_1 = R''_0 \oplus R'''_0 = \beta = \texttt{0x80200100},$$

and we can check this. Furthermore, by exhaustively trying all possible output values for the S-box for pairs with the correct right half difference, we get an additional $32-8$ bit check for the left half difference. Thus, with high probability we detect correctly pairs following the boomerang characteristic.

Now, repeating the procedures outlined in Section 5.3.1 and 5.4.1 we first recover the 22 least significant bits of the second round key ($rk_{1,0..21}$) and afterwards the remaining 7 bits of the round key ($rk_{1,22..29}$) as well as one additional entry of the S-box. Note that the bits $rk_{1,29..31}$ are known from the last round key. The complexity of this is now

$$22 \cdot \left(\frac{7.7 \cdot 2^{38} + 2^{38}}{2}\right) \approx 2^{44.58}$$

for the first step where we choose $t = 7.7$ in order to have success probability of 0.99 and

$$2^3 2^3 2^{38} = 2^{44}$$

for the second step.

Using this shortened boomerang described in the last section, we can moreover recover arbitrary S-box entries by fixing $R_{1,0..7}$ appropriately. The complexity for this is again 2^{44} on average.

5.5 Conclusion

We have shown three kinds of attack on the block cipher C2.

When we are allowed to set the encryption key once and then encrypt chosen plaintexts, we can recover the secret S-box with only 2^{24} queries to the device and a reasonable precomputation phase that we have already done. The attack implemented on a PC recovers the whole S-box in less than 30 sec. Due to the low query complexity,

we believe that this attack could be applied in practice to recover the S-box from an actual device.

We presented a boomerang attack that, when the S-box is known, recovers the key with complexity equivalent to 2^{48} C2 encryptions and works for all possible S-boxes.

For the most difficult case, when both the key and the S-box are unknown and we are faced with an equivalent of at least 1740-bit long key, we presented an attack that recovers both of them with complexity of around $2^{53.5}$ queries to the encryption device.

Furthermore, we showed that the main strength of the cipher lies in the modular additions rather than the S-box. With modular additions replaced by XORs, one can find 9-round differentials with probability 1 and boomerangs for all 10 rounds with probability 1, both regardless of the S-box that is used.

None of the attacks assume anything about the S-box, not even its bijectivity. It is surprising that the addition of the secret S-box does not substantially improve the overall security of the design. It shows that to achieve the desired effect, the algorithm using a secret S-box must be designed very carefully. Probably a better option would be to use a longer secret key instead.

Chapter 6

Cryptanalysis of PRESENT-like Ciphers with Secret S-boxes

In this chapter we investigate PRESENT-like block ciphers with secret key-dependent S-boxes. The block cipher PRESENT [21] is an important example of a lightweight cipher. It consists of alternate layers of substitutions and permutations.

Important design principles of lightweight ciphers are efficient hardware implementation, good performance, and a moderate security level. Usually there is a trade-off between the performance and the security level. In order to speed up the algorithm we want as few rounds of encryption as possible but at the same time a minimum number of rounds is required to assure the security level.

PRESENT was described in Section 2.2.1 and is a 64-bit iterated block cipher with an 80-bit key. It consists of 31 rounds, each round has three layers, a substitution layer consisting of 16 parallel applications of the same 4-bit S-box, a permutation layer consisting of a bit-wise permutation of 64 bits, and a key addition layer, where a subkey is xored to the text. The PRESENT S-box is chosen carefully to resists differential and linear cryptanalysis. There are neither any 1 to 1-bit differential characteristics nor linear approximations with a bias larger than 2^{-3} such that the Hamming weight is 1 for the input and output mask. Also the best differential characteristic for the S-box has probability 2^{-2} and the best linear approximation has a bias of 2^{-2}. These properties are not generally fulfilled for randomly chosen S-boxes.

PRESENT was designed to allow fast and compact implementation in hardware. The best known cryptanalytic attack on PRESENT is a linear attack on 26 of the 31 rounds [31]. The attack requires all possible 2^{64} texts and has a running time of 2^{72}. Although this attack is hardly practical, it illustrates that the number of rounds used should not be dramatically reduced.

An idea of how to strengthen the cipher in a way that enables one to reduce the number of rounds has been presented by two researchers from Princeton University. The cipher Maya [55] is a 16 round SP-network similar to PRESENT. The main difference is that the substitution layer of Maya consists of 16 different S-boxes which are key dependent and therefore kept secret. The bit permutation between the S-box layers is fixed and public. In each round a round key is xored to the text. It is argued that this cipher can be implemented efficiently in practice and also that "differential cryptanalysis is infeasible". In this chapter we will investigate if such a cipher is stronger than the original, and if so how much stronger.

The Maya design is only one possibility to design a PRESENT-like cipher with secret components. In the extreme case one could choose 16 S-boxes uniformly at

random and independently for every round. Furthermore, one could also make the bit permutation part of the key, where the bit permutation could be chosen uniformly at random from the set of all such permutations and used repeatedly. As another extreme, a bit permutation could be chosen uniformly at random and independently for every round.

The idea of having ciphers where the substitutions are not publicly known and part of the secret key is not new, most notable examples are Khufu [83] and the Khufu variation Blowfish [92] as well as other proposals [12, 94].

We focus on the cipher Maya and present a novel differential-style attack that enables us to find the S-boxes in the first round one by one with a practical complexity. In the next section we give a description of a PRESENT-like cipher. We consider both cases, the Maya case where the layer of 16 S-boxes is kept secret and used throughout the cipher and the extreme case where the S-box layer and the bit permutation are chosen uniformly at random for each round.

6.1 PRESENT-like Ciphers

We focus on a PRESENT-like cipher where the secret consists of one round key for each round and 16 secret S-boxes. We assume that the round keys and the S-boxes are randomly chosen. In practice these secret components might be derived from a master key using a key schedule which generates key-dependent round keys and S-boxes. These 16 randomly chosen S-boxes form the substitution layer which is used repeatedly throughout all the rounds. The permutation layer consists of a bit permutation which is fixed and publicly known.

One round of encryption works as follows (cf. Algorithm 3). The current text is divided into nibbles of 4 bits which are processed by the 16 S-boxes in parallel. Then the bit permutation is applied to the concatenation of the output of the S-boxes and the output is xored with the round-key.

The cipher Maya, proposed by Gomathisankaran and Lee [55], is an instance of the cipher described in Algorithm 3 with $N = 16$. The authors claim that it is efficient in a hardware implementation.

We attack this cipher by recovering all 16 S-boxes. However, in the general case, we do not know the last-round key, and therefore what we recover is in fact the 16 S-boxes xored with the last round key. Once this is done, we can peel off the first and last layers of encryption, and attack the cipher with two rounds less; this time, the S-boxes are known and a standard differential or linear attack can be mounted to extract the round keys. What we obtain in the end is an equivalent description of the cipher, but not necessarily the key. Still, the equivalent description of the cipher will allow us to encrypt or decrypt any text of our choice.

Furthermore, we will outline how our attack can be applied to a generalization of the cipher. Here, the S-boxes are chosen uniformly at random for each round. Additionally, the bit permutation can be chosen randomly for each round and kept secret as part of the key. In this case, the addition of the round keys is not necessary because it can be seen as part of the S-boxes. Furthermore the permutation is omitted

Algorithm 3 Pseudo-code of a PRESENT-like cipher with secret S-boxes. The number of rounds is N.

Require: X is a 64-bit plaintext
Ensure: $C = E_k(X)$ where E_k means the encryption function with key k
  Derive 16 S-boxes S_i and N round keys k_i from k
  $\mathsf{STATE} \leftarrow X$
  for $i = 1$ to N **do**
    Parse STATE as $\mathsf{STATE}_0 \| \cdots \| \mathsf{STATE}_{15}$, where each STATE_j is a four-bit nibble
    for [**do**Substitution layer] $j = 0$ to 15
      $\mathsf{STATE}_j \leftarrow S_j(\mathsf{STATE}_j)$
    end for
    Reassemble STATE
    Apply bit permutation to STATE
    Add round key k_i to STATE
  end for
  $C \leftarrow \mathsf{STATE}$

in the last round. In this variant nothing but the block size and the number of rounds is known. The pseudo-code of this variant is described as Algorithm 4.

6.2 The Principle of the Attack

In this section, we explain the idea of our approach to recover the S-boxes in the basic variant of a PRESENT-like cipher with secret S-boxes.

In the basic variant of the cipher (cf. Algorithm 3), the substitution layer consists of 16 secret S-boxes which are applied in all rounds. We denote these 16 S-boxes S_i, $0 \leq i < 16$, and we note that all S_i are bijective mappings $S_i : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$.

For convenience, we introduce the following notation.

Definition 6.1 (Slender set)**.**
Given the S-box S and $e \in \mathbb{F}_2^4$, we denote the set of all pairs $\{x, y\}$ such that $S(x) \oplus S(y) = e$ by D_e. Here, we consider the pairs $\{x, y\}$ and $\{y, x\}$ to be identical. A pair $\{x, y\}$ belonging to a set D_e where e has Hamming weight 1 is called a slender *pair. A set consisting of slender pairs is called a slender set.*

If we consider the S-box S_D (see Figure 4.2), then the four slender sets are

$$\begin{aligned} D_1 =&\{\{2, f\}, \{c, d\}, \{7, a\}, \{4, 5\}, \{0, 9\}, \{3, e\}, \{1, 6\}, \{8, b\}\}, \\ D_2 =&\{\{c, f\}, \{2, d\}, \{5, 7\}, \{4, a\}, \{0, 3\}, \{a, e\}, \{1, 8\}, \{6, b\}\}, \\ D_4 =&\{\{7, f\}, \{2, a\}, \{5, c\}, \{4, d\}, \{0, 1\}, \{6, 9\}, \{3, 8\}, \{b, e\}\}, \\ D_8 =&\{\{0, f\}, \{4, b\}, \{2, 9\}, \{3, c\}, \{d, e\}, \{1, 7\}, \{6, a\}, \{5, 8\}\}. \end{aligned}$$

In the following we explain how to recover the leftmost S-box S_0. The recovery of the other 15 S-boxes works in the same way. We want to obtain information about

Algorithm 4 Pseudo-code of a PRESENT-like cipher with secret S-boxes and secret bit permutations, all unique for each of the N rounds.

Require: X is a 64-bit plaintext
Ensure: $C = E_k(X)$ where E_k means the encryption function with key k
  Derive $16 \cdot N$ S-boxes $S_{i,j}$, $1 \leq i \leq N$, $0 \leq j \leq 15$ and $N - 1$ bit permutations P_i from K
  $\mathsf{STATE} \leftarrow X$
  for $i = 1$ to N **do**
    Parse STATE as $\mathsf{STATE}_0 \| \cdots \| \mathsf{STATE}_{15}$, where each STATE_j is a four-bit nibble
    for [**do**Substitution layer] $j = 0$ to 15
      $\mathsf{STATE}_j \leftarrow S_{i,j}(\mathsf{STATE}_j)$
    end for
    Reassemble STATE
    if $i < N$ **then**
      Apply bit permutation P_i to STATE
    end if
  end for
  $C \leftarrow \mathsf{STATE}$

the S-box in a differential-style attack and proceed as follows. We encrypt a certain number t of structures P_{r_i} of plaintexts of the form

$$P_{r_i} = \{(x \| r_i) \mid x \in \mathbb{F}_2^4\}$$

where each $r_i \in \mathbb{F}_2^{60}$ for $0 \leq i < t$ is chosen uniformly at random. Two different plaintexts $(x\|r_i), (y\|r_i)$ in P_{r_i} have an input difference of the form

$$(x\|r_i) \oplus (y\|r_i) = (?\|0^{60}),$$

where 0^n denotes the bit string consisting of n zeros.

We obtain the corresponding ciphertexts and check if there is an input pair for which only one S-box is active in the ciphertext difference. For now, let $p(\{x, y\})$ denote the probability that only one S-box is active in the ciphertext pair when the plaintext pair is $\{x\|r, y\|r\}$, taken over all the different choices of $r \in \mathbb{F}_2^{60}$. The attack is based on some assumptions. The first assumption is a standard one in differential cryptanalysis:

Assumption 6.2.
The probability $p(\{x, y\})$ depends only on the value of $S(x) \oplus S(y)$, not specifically on the pair $\{x, y\}$. Hence, given $e = S(x) \oplus S(y)$, we can denote this probability p_e.

We are particularly interested in identifying slender pairs. In order to do this, we need the following assumption, which has been experimentally verified to hold in most cases.

Assumption 6.3.
The probability p_e is higher when e has Hamming weight 1, than when e has Hamming weight greater than 1.

In order to learn all the probabilities p_e we would have to encrypt all 2^{64} possible plaintexts, but we can estimate the probabilities by introducing counters

$$C(\{x,y\}) = \left|\{r_i \mid \exists j:\ E(x\|r_i) \oplus E(y\|r_i) = (0^{4j}\|?\|0^{60-4j})\}\right|$$

for all pairs $\{x,y\}$, $x,y \in \mathbb{F}_2^4$. Hence, the counter $C(\{x,y\})$ counts how often only one S-box is active in the ciphertext pair when the input pair to S-box S_0 is $\{x,y\}$.

Assumption 6.2 states that pairs in the same set have the same probability. That means, that the counter values for pairs in the same set should be similar when sufficiently many plaintexts have been encrypted. Assumption 6.3 says that the highest counter values will (usually) correspond to slender pairs. In the attack we will to try to identify the slender sets, and this will be relatively easy if the probabilities p_e and $p_{e'}$, $e \neq e'$, are sufficiently different. Experiments show that this condition is often satisfied.

The counter C consists of 120 values since there are $\binom{16}{2} = 120$ different pairs $\{x,y\}$. After encrypting sufficiently many structures we may sort C in descending order, and thereby hopefully obtain a partitioning of the 120 pairs into a number of sets corresponding to D_e for different values of e. We will return to this partitioning method later. Our final goal is to learn all four slender sets D_e. It is enough to learn all slender sets because all other sets will not give any additional information about the S-box. The reason is that if D_e and $D_{e'}$ are known for some S-box S, then $D_{e\oplus e'}$ can be derived from D_e and $D_{e'}$ and therefore $D_{e\oplus e'}$ does not give any new information. Clearly, if $\{x,y\} \in D_e$ and $\{x,z\} \in D_{e'}$, then $\{y,z\} \in D_{e\oplus e'}$. This observation generalizes to more than two sets. In general, given sets D_{e_i} one can construct all sets D_e where e can be written as a linear combination of the vectors e_i. This statement is proven in Lemma 6.4 in the next subsection.

In a similar manner we can obtain information about the inverse of the S-boxes as explained in Subsection 6.2.2.

6.2.1 S-box Recovery given Slender Sets

We assume that we know one or more sets D_e from the partitioning of the counters. We focus on sets for the S-box itself and not for its inverse. However, we remark that it is only possible to recover the S-box up to equivalence when no set for the inverse S-box is given. We call two S-boxes S and S' equivalent if they differ only by a permutation of the output bits and by adding a constant. In other words there exists a bit permutation P and a constant c such that

$$S'(x) = P(S(x)) \oplus c.$$

These two S-boxes can never be distinguished.

Lemma 6.4.
Given r sets $D_{e_1}, \ldots, D_{e_r}$ for $1 \leq r \leq 4$, and $e_i \in \mathbb{F}_2^4$ we can construct all sets D_y where $y \in \operatorname{span}(e_1, \ldots, e_r)$.

Proof. If $y \in \text{span}(e_1, \ldots, e_r)$ then there exists a (not unique) chain of values

$$y_0 = e_{j_0}, y_1, \ldots, y_s = y$$

such that $y_i \oplus y_{i+1} = e_{j_i}$ for $j_i \in \{1, \ldots, r\}$. We can inductively construct the sets D_{y_i}. We already know the set $D_{y_0} = D_{e_{j_0}}$ and we can construct $D_{y_{i+1}}$ using the set D_{y_i} and $D_{e_{j_i}}$ given that

$$\{a, b\} \in D_{y_i \oplus e_{j_i}} \Leftrightarrow \exists c \in \mathbb{F}_2^4 \text{ such that } \{a, c\} \in D_{y_i} \text{ and } \{c, b\} \in D_{e_{j_i}}$$

□

Based on this lemma we can prove the following theorem.

Theorem 6.5.
Let $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ be a bijective S-box and for $e \in \mathbb{F}_2^4$ with hw$(e) = 1$,

$$D_e = \{\{x, y\} \mid S(x) \oplus S(y) = e\}.$$

where hw *denotes the hamming weight. Given r sets $D_{e_1}, \ldots, D_{e_r}$ for $1 \leq r \leq 4$, up to equivalence, there are*

$$\prod_{i=1}^{2^{4-r}-1} 2^4 - i2^r$$

possibilities for S. More concretely,

1. *given* 4 *sets the S-box is determined uniquely,*
2. *given* 3 *sets there are* 8 *possible S-boxes,*
3. *given* 2 *sets there are* 384 *possible S-boxes, and*
4. *given* 1 *set there are* 645120 *possible S-boxes.*

Proof. We assume that r sets $D_{e_1}, \ldots, D_{e_r}$ are given. Without loss of generality we can assume that $S(0) = 0$ and $e_1 = (1, 0, 0, 0)$, $e_2 = (0, 1, 0, 0)$ and so on. We claim that given this information, S is fixed on the set

$$\{x \mid S(x) \in \text{span}(e_1, \ldots, e_r)\}.$$

Let $y \in \text{span}(e_1, \ldots, e_r)$. The proof of Lemma 6.4 describes how we can construct the set D_y. As a set D_y has to cover the whole $\mathbb{F}_2^4$, there exists a pair $\{0, x\} \in D_y$ for some $x \in \mathbb{F}_2^4$. It follows that we found an $x \in \mathbb{F}_2^4$ such that

$$S(0) \oplus S(x) = S(x) = y.$$

More generally, the same argument shows that, given $D_{e_1}, \ldots, D_{e_r}$, fixing $S(x') = y'$ the values of S are fixed for all x such that $S(x)$ is in the coset $y' \oplus \text{span}(e_1, \ldots, e_r)$. Noting there are 2^{4-r} cosets of $\text{span}(e_1, \ldots, e_r)$ and taking into account the bijectivity of the S-box, the theorem follows. □

However, when we attack a PRESENT-like cipher where we use the same layer of S-boxes in every round, we can also get information about the inverse of the S-boxes. This means that we can actually determine an S-box uniquely if we accumulate sufficient information, as explained in the following subsection.

6.2.2 Generalizing to All S-boxes and their Inverses

In order to break the cipher we have to eventually recover all S-boxes and not only the S-box S_0. The above observations can clearly be generalized to all S-boxes by introducing additional types of structures and additional counters.

Moreover, as mentioned above the symmetry between encryption and decryption in the cipher we are considering allows us to obtain the same type of information about the inverse S-boxes as we obtain about the S-boxes themselves. This can even be done in a chosen-plaintext setting, although it may require more texts than in a chosen-ciphertext setting.

We assume now that we have identified u slender sets for some S-box S, and v slender sets for its inverse S^{-1}. The following table shows the average number of S-boxes that would give rise to the same $u+v$ sets; these averages are based on 100000 randomly generated S-boxes.

$u\backslash v$	1	2	3	4
1	207	3.52	1.44	1.19
2	3.52	1.16	1.03	1.01
3	1.44	1.03	1.01	1.01
4	1.19	1.01	1.01	1.01

Evidently, if $u+v \geq 6$, the S-box is usually uniquely determined from the $u+v$ sets, and in many cases, fewer sets are sufficient. However, there exist S-boxes S which are not uniquely determined even if all four slender sets are known for both S and S^{-1}. Note that there is no contradiction to Theorem 6.5, because there we argue that we can determine the S-box uniquely up to equivalence, meaning that we obtain a class of equivalent S-boxes, while the information about the inverse usually enables us to uniquely determine the S-box, meaning we find exactly one S-box in the end.

In the next subsections we describe a number of ways to partition the pairs into sets and to check that this partitioning is correct.

6.2.3 Partitioning Pairs into Sets

We focus again on recovering the S-box S_0. Our starting point for partitioning pairs (in particular the slender pairs) into sets is the counter C.

The straightforward partitioning method simply sorts C in descending order, and takes the first eight pairs as the first set, the next eight pairs as a second set, etc. However, when for example pairs from two different sets D_e and $D_{e'}$ have a similar probability, we expect that the counters have similar values. This indicates that using this method we will often make the wrong partitioning into sets.

But it can be checked that the partitioning is correct using the very strong *filtering methods* described in the following subsection.

6.2.4 Filtering Methods

In this section we present a couple of filtering methods to check whether the sets we obtain by partitioning the counter values are correct. If the filter methods fail, meaning one or all filters indicate that the sets are not correct we can infer that this is true. However, even if the sets survive the filtering this does not guarantee that we found the correct sets.

The most simple filtering method is what we call the *existence filter*. Given u sets for an S-box S and v sets for its inverse S^{-1}, we can count to how many S-boxes would give rise to the same sets. If no S-box gives rise to these sets, then clearly the sets must be wrong. However, counting the number of S-boxes that give rise to these sets is somewhat inefficient, and as we have seen, if we only know a few sets, there are usually several S-boxes that give rise to the same sets, and so the probability of a false positive is high in this case.

The *cover filter* is a much more efficient method. It is based on the trivial observation that for any valid set D_e, the pairs cover all values in $\mathbb{F}_2^4$, meaning that it holds that

$$\{x, y \ : \ \{x, y\} \in D_e\} = \mathbb{F}_2^4 .$$

Hence, if we have identified a candidate set D_e containing two pairs $\{x, y\}$ and $\{x, z\}$, then D_e cannot be a valid set. Although this method is very simple, it is in fact a very strong filter; the probability that eight randomly chosen pairs among the 120 pairs cover all values in $\mathbb{F}_2^4$ is only

$$\frac{\prod_{i=1}^{8} \binom{2i}{2}}{\prod_{i=0}^{7} \binom{16}{2} - i} = \prod_{i=1}^{7} \frac{\binom{2i}{2}}{\binom{16}{2} - i} \approx 2^{-18.7},$$

and therefore in practice, many wrong candidate sets are discovered by this method.

From the proof of Theorem 6.5 we can deduce the following about the cover filter.

Corollary 6.6.
The cover filter is necessary and sufficient. This means that given a number of sets D_e where e runs through a subspace of $\mathbb{F}_2^4$, there exists an S-box corresponding to these sets if and only if each of the sets D_e passes the cover filter.

The final filtering method that we describe here is called the *bowtie filter*. We can observe that if $\{x_1, y_1\}$ and $\{x_2, y_2\}$ belong to the set D_e, then $\{x_1, y_2\}$ and $\{x_2, y_1\}$ will also belong to the same set $D_{e'}$ for some $e' \neq e$, and likewise, $\{x_1, x_2\}$ and $\{y_1, y_2\}$ will belong to the same set $D_{e''}$ for some $e'' \notin \{e, e'\}$. The reason for this is that if $\{x_1, y_1\}$ and $\{x_2, y_2\}$ belong to the same set D_e, then by definition

$$S(x_1) \oplus S(y_1) = S(x_2) \oplus S(y_2) = e,$$

and therefore

$$S(x_1) \oplus S(y_2) = S(x_2) \oplus S(y_1) = e \oplus S(y_1) \oplus S(y_2) = e' \neq e$$

and

$$S(x_1) \oplus S(x_2) = S(y_1) \oplus S(y_2) = e \oplus S(y_1) \oplus S(x_2) = e'' \neq e' \neq e.$$

Hence, we assume that we know two sets D_e and $D_{e'}$, which both passed the cover filter, and that $\{a, b\} \in D_e$ and $\{a, c\} \in D_{e'}$. Now, if $\{c, d\} \in D_e$, then for both these two sets to be valid, it must hold that $\{b, d\} \in D_{e'}$. If we follow the partner b of a in set D_e to set $D_{e'}$, we find that $b's$ partner in set $D_{e'}$ is d. Following d back into set D_e, its partner is c. In set $D_{e'}$ c belongs to the pair $\{a, c\}$ and following a back to set D_e finally leads us to the pair $\{a, b\}$ which we started with. These jumps back and forth between the two sets form a bowtie-shape cycle, which gave the filter method its name.

$$\begin{aligned} D_e &= \{\{a,b\},\ \{c,d\}\ \ldots \\ D_{e'} &= \{\{a,c\},\ \{b,d\}\ \ldots \end{aligned}$$

6.2.5 Relaxed Truncated Differentials

So far we have focused on one counter for each S-box in the first round which is incremented if exactly one S-box in the ciphertext difference is active. But even for slender pairs the probability that a single active S-box in the input difference leads to a single active S-box in the output difference is relatively low. That means that many plaintext pairs are needed before it is possible to partition pairs into sets.

It is much more likely that the weight one difference spreads moderately through the cipher resulting in a few active S-boxes in the ciphertext. Hence, we might find slender pair candidates more efficiently by looking at ciphertext pairs with more than one active S-box. The more active S-boxes we allow, the more noise we will get, and so there is a trade-off between the signal-to-noise ratio, and the strength of the signal.

It turns out that allowing even a relatively large number of active S-boxes does not introduce too much noise. This can be used to make the attack more efficient. Instead of keeping only one counter for each input S-box and each pair $\{x, y\}$ which we increment when a single S-box is active, we have one counter for each possible number of active S-boxes in the ciphertext (except 16 because if all S-boxes are active we do not obtain any information). For each input S-box S_i and for each pair $\{x, y\}$ we introduce counters $C_{i,j}(\{x, y\})$. We increment the counter $C_{i,j}(\{x, y\})$ every time the input pair $\{x, y\}$ to S-box S_i (with a random but fixed input to the other S-boxes) leads to exactly j S-boxes being active in the last round, where j ranges from 1 to 15. When we have done a number of encryptions we sort the counters $C_{i,j}$ for some pair i, j. If the cover filter identifies sets based on this sorting, we assume that these are correct slender sets. When we have several sets, we use the bowtie filter to check the validity of the sets. We do this for increasing j from 1 to 15. Since the cover filter is a very strong filter, the risk of errors is low, both in the cases where the signal is weak (small values of j), and also in the cases where there is a lot of noise (large values of j).

6.3 The Attack in Practice

We now describe how the attack is carried out in practice. The attack consists of a data collection phase followed by an S-box recovery phase, and those two phases are repeated until all or almost all S-boxes have been recovered.

6.3.1 Data Collection Phase

In the data collecting phase we simply encrypt structures and increment counters when applicable. Each structure consists of 16 plaintexts differing in only a single input S-box:

$$P_{r_i} = \{(r_{i_1}||x||r_{i_2}) \mid x \in \mathbb{F}_2^4, r_i = r_{i_1}||r_{i_2}, r_{i_1} \in \mathbb{F}_2^{4j}, r_{i_2} \in \mathbb{F}_2^{4(15-j)}\}$$

The active S-box is chosen randomly among the S-boxes that have not already been recovered. From each structure we can form 120 pairs. After encryption, we check all 120 pairs of ciphertexts to see how many S-boxes are active in the ciphertext difference and we increment the corresponding counter for the input pair to the S-box that was active in the plaintext.

We also carry out decryptions in order to obtain information about the inverse S-boxes.

6.3.2 S-box Recovery Phase

Every once in a while, we stop collecting data and try identifying sets for each S-box. Our aim is to recover up to all four slender sets for each S-box in each direction, meaning for an S-box and its inverse. We have seen before that it is usually enough to find 6 slender sets in total to determine an S-box uniquely. In order to identify sets for an S-box S we first sort the counters for each number of active output S-boxes, meaning we have 15 different sortings of the counters for the 120 input pairs, one for each possible number of active S-boxes between 1 and 15. We start with the lowest number of active S-boxes because here we expect fewest noise, however the signal might not be strong enough. We check if the top eight counter values in the sorted list passes the cover filter. If so, we consider these eight pairs a slender set and add it to a collection of identified sets, unless the set is already present in the collection. If we have already identified sets earlier, meaning that there are multiple set in the collection, we can check if they pass the bowtie filter. We then look at the next eight pairs and so forth. We stop adding sets when we have identified four sets, or we run into an inconsistency such as failing the bowtie test or non-disjoint sets. In case of an inconsistency, we give up identifying sets for this S-box and return to it after collecting more data.

As mentioned before we are only interested in slender sets because we can derive all other sets from the four slender sets. We can use the bowtie filter to filter out candidate sets that can be derived from existing sets. Consider as an example a

situation where the following two candidate sets D_e and $D_{e'}$ have been identified

$$\begin{aligned} D_e &= \{\{0,1\},\{2,3\},\{4,5\},\{6,7\},\{8,9\},\{\mathrm{a},\mathrm{b}\},\{\mathrm{c},\mathrm{d}\},\{\mathrm{e},\mathrm{f}\}\}, \\ D_{e'} &= \{\{0,2\},\{1,3\},\{4,6\},\{5,7\},\{8,\mathrm{a}\},\{9,\mathrm{b}\},\{\mathrm{c},\mathrm{e}\},\{\mathrm{d},\mathrm{f}\}\}. \end{aligned}$$

Both set pass the cover and the bowtie filter. From these two sets we can derive the set $D_{e\oplus e'}$ directly as

$$D_{e\oplus e'} = \{\{0,3\},\{1,2\},\{4,7\},\{5,6\},\{8,\mathrm{b}\},\{9,\mathrm{a}\},\{\mathrm{c},\mathrm{f}\},\{\mathrm{d},\mathrm{e}\}\}.$$

As an example,

$$S(0) \oplus S(3) = (S(0) \oplus S(1)) \oplus (S(1) \oplus S(3)) = e \oplus e'.$$

Thus, the pair $\{0,3\}$ is contained in the set $D_{e\oplus e'}$. Hence, if we identify a set that can be derived from two sets which have already been identified, then we should not add the third set to our collection. Because if we assume that the first two sets are slender sets, then we know that the third is not.

We repeat the above method of identifying sets for the inverse S-boxes as well, maintaining separate counters for these.

Once we have identified a sufficient number of set (as many as possible) for an S-box and its inverse using the above method we can simply check whether an S-box exists that gives rise to these sets in order to see if the sets are valid. If there is no S-box generating these sets, then the sets are obviously not valid. As mentioned in Section 6.2, applying the existence filter is not terribly efficient; on the other hand, it is not terribly slow either. A way to implement the recovery of the S-box from a number of given sets is the following. We make guesses for value of $S(0)$ and the exact values e_i for the identified sets D_{e_i} until one runs into an inconsistency with the candidate sets. Note that once these guesses have been made, we may find the "partner" of 0 in all candidate sets. For instance, if the two sets D_e and $D_{e'}$ in the example above are our candidate sets, and we guess that $S(0) = 0$, then we would know that $S(1) = 2^i$ and $S(2) = 2^j$ for some (guessed) i, j, $i \neq j$ and $0 \leq i, j < 4$. We obtain similar information about the inverse S-box from the candidate sets for the inverse S-box. This method is able to find all candidate S-boxes in a fraction of a second given at least one set for the S-box and one set for its inverse.

We stop considering an S-box both in the data collection and the S-box recovery phase when exactly one S-box candidate has been found. If not all S-boxes have been recovered, we continue the data collection phase. In some cases during the attack, we have to give up recovering one or more S-boxes because we are unable to identify sufficiently many sets, or because we consistently get no candidates for the S-box based on the identified sets. In the latter case, there is obviously an error in the partitioning into sets. If we consistently obtain multiple candidates for an S-box, we may also accept this and consider the S-box recovered, keeping a record of all candidates.

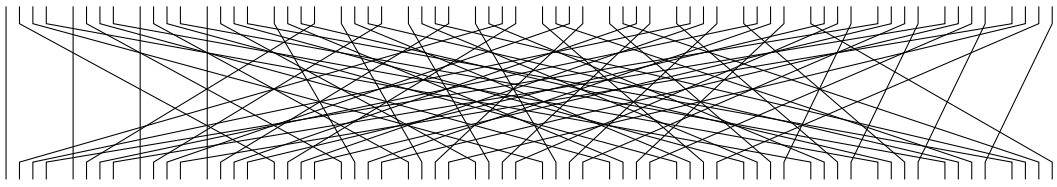
A reason that sets pass the cover and the bowtie filter but then fail the existence filter, meaning that the pairs have been wrongly partitioned is the following. We consider the sets D_e and $D_{e'}$ and swap the pairs $\{0,1\}$ and $\{2,3\}$ in D_e with the pair

$\{0, 2\}$ and $\{1, 3\}$ in $D_{e'}$, then the resulting set will still pass the cover and the bowtie filter. That means that swapping two "bowtie pairs" in two valid sets will result in two sets that will still pass both the cover and the bowtie test. This is a potential cause for errors; if two sets have roughly the same probability of causing a single active S-box (or a limited number of active S-boxes) in the ciphertext, then we are likely to generate wrong sets that pass both the cover and the bowtie test. This error may be caught by the existence filter and lead to the case that we cannot find an S-box that gives rise to the identified sets. However, in rare cases this error stays undetected and we will recover the wrong S-box.

In the next section we report experimental results for the attack when applied to the cipher Maya.

6.4 Case study: the Block Cipher Maya

In the following case study we show how to break the cipher Maya with practical complexity using the attack described in the previous section. Maya is a block cipher proposed at WCC 2009 [55]. It is a PRESENT-like cipher of the kind described in Algorithm 3, meaning that the S-box layer consists of secret key-dependent S-boxes which are repeated in every round. The bit permutation (see Figure 6.1) is fixed and publicly known. Each round also contains an addition of a round key. The round keys and the S-boxes are derived from the 1024-bit master key.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	16	32	48	4	20	36	52	8	24	40	56	12	28	44	60	21	37	53	5	25	41	57	9	29	45	61	13	33	49	1	17
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
42	58	10	26	46	62	14	30	50	2	18	34	54	6	22	38	63	15	31	47	3	19	35	51	7	23	39	55	11	27	43	59

Figure 6.1: The Maya bit permutation.

Since the S-boxes are the same in every round we can use the differential-style attack explained in Sections 6.2 and 6.3 to get information on the S-boxes and their inverses. We get information on both directions for every encrypted pair. However, if necessary we can also choose to do decryptions to obtain information about the inverse of a specific S-box. In this way we often recover at least two sets in each direction, which usually means all the S-boxes can be determined uniquely. But we do not recover exactly the S-box which is used during the encryption of the first round because of the round key additions. This means that we only obtain the correct S-box up to an XOR by the last round key, which is unknown. This additive constant will be compensated when recovering the round keys, meaning that we obtain an equivalent description of the cipher in the end. Recovering the S-box layer xored with the last

round key enables us to peel of the last round and the S-box layer in the first round of encryption, after which the attack can be repeated on this reduced cipher. Moreover, we expect that once the S-boxes are known, a dedicated differential or linear attack is more efficient than our general attack.

The cipher Maya is proposed with 16 rounds and in Table 6.1 the logarithm to basis 2 of the median complexity to recover the secret S-boxes as function of the number of rounds is shown. The median is taken over 16 cases. Moreover, Table 6.1 shows the logarithm to basis 2 of the complexity (number of texts) as a function of the number of rounds for the same example keys. The complexities in italics are extrapolated values from running the attack on fewer rounds. The complexities refer to obtaining all 16 S-boxes (whenever possible, see discussion below), so that the first and the last round can be peeled off, and the cipher with two rounds less can be attacked. The complexities are graphically presented in Figure 6.2.

Table 6.1: The log of the complexity (number of texts encrypted or decrypted) of 16 test runs of the attack on Maya as a function of the number of rounds. The complexities in italics are extrapolations based on the assumption of a linear relationship between the number of rounds and the log complexity. The median was computed on the assumption that non-existent complexities are infinite.

	Rounds										
Case	6	7	8	9	10	11	12	13	14	15	16
1	14.4	16.2	18.6	21.0	24.3	28.5	31.6	35.5	40.5		*46.8*
2	14.1	15.6	17.3	19.7	22.0	23.7	26.9	29.1	32.0	33.8	36.0
3	14.3	16.3	17.4	19.5	22.2	24.7	27.4	29.7	31.3	33.6	35.9
4	14.8	16.1	17.6	19.8	22.3	25.3	27.9	30.1	32.1	34.8	36.9
5	14.6	15.7	17.4	19.4	21.4	23.5	26.0	27.6	30.0	31.4	35.7
6	15.0	16.1	18.3	20.2	22.7	25.6	28.7	31.8	34.2	36.3	*39.3*
7	14.2	15.6	17.7	19.7	22.4	25.4	27.4	29.9	32.6	35.4	37.4
8	14.5	15.7	17.5	19.4	21.5	24.4	26.9	29.6	31.9	35.5	37.1
9	15.2	16.8	19.1	21.1	23.6	26.5	28.7	31.5	36.3	39.0	*41.2*
10	14.9	16.5	18.1	20.2	23.0	24.5	27.6	29.8	34.7	38.6	38.5
11	14.4	15.6	17.5	19.8	22.1	25.1	27.5	30.5	33.4	37.7	*39.4*
12	15.0	15.7	17.5	19.9	22.4	25.3	29.1	31.5	34.2	36.1	*39.5*
13	14.9	15.9	17.1	19.6	21.7	24.4	27.9	29.3	31.8	35.8	36.0
14	14.4	15.6	17.5	19.3	21.9	24.3	27.7	30.3	32.1	35.4	36.7
15	14.4	15.6	17.2	19.5	22.3	24.0	26.6	29.9	33.0	36.5	40.5
16	14.2	15.7	17.4	19.7	22.4	24.9	27.6	30.4	32.9	34.9	37.4
Median	14.4	15.7	17.5	19.7	22.3	24.8	27.6	30.2	32.5	35.6	37.4

In our implementation of the attack, an S-box was considered correctly recovered if only one S-box gave rise to the given partitioning into sets. However, if a substantial amount of time had been spent on an S-box, the conditions were relaxed such that even if there were more than one candidate S-box, work on this S-box was still discontinued and all candidates were printed. In extreme cases, where there were no candidate

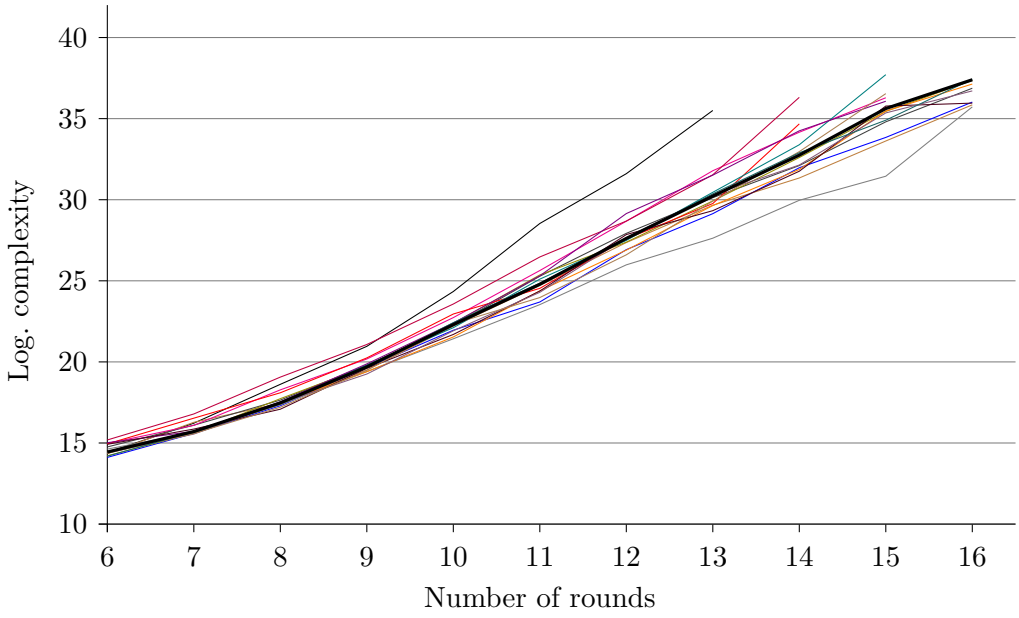


Figure 6.2: A graphical representation of the data in Table 6.1. The thick line represents the median computed for each number of rounds.

S-boxes after a long time had been spent trying to recover the S-box, that S-box was given up. The choice of when to accept multiple candidates, or when to give up an S-box, obviously affects the complexity of the attack. A more sophisticated implementation might adapt better to these situations. As an example, if the program consistently gives rise to the same partitioning into sets, and there are no candidates for this partitioning, one might try swapping elements between sets in such a way that the bowtie condition still holds.

The error rate of the attack is very low. If we consider the highest number of rounds broken in each of the 16 test cases, then the total number of S-boxes that had to be recovered was $16 \cdot 16 = 256$. Of these, 245 were correctly recovered with only a single S-box candidate. For seven S-boxes, there were multiple candidates, and the correct S-box was always one of these. The number of candidates ranged from two to four. Three out of 256 S-boxes were incorrectly recovered with only a single S-box candidate. One S-box was given up due to too much time spent trying to recover it. Thus our attack successfully recovered the S-boxes in more than 95% of the cases and only failed in less than 2% of the attempts.

In a real attack, the fact that some S-boxes were incorrectly recovered would be discovered after attempting to break the cipher reduced by the first and the last rounds. By recording as much information about the identified sets and the counter values as possible, it is likely that one would be able to locate the S-box causing the problem. For instance, there may be 16 counter values that are all similar, meaning that it is likely that two sets have been mixed up. Thus, a closer look at the data

given might enable us to still correctly recover the S-boxes in such a case.

In this section we applied our attack to the cipher Maya and showed that we can break the full cipher with a practical complexity. However, one question is how many rounds of encryption are necessary to prevent our attack. As gathering experimental results for a higher number of rounds is very time consuming, we invented a mathematical model in order to estimate the complexity for more rounds of encryption which we present in the next section.

6.5 Model for the Complexity of Recovering Sets D_e

For a small number of rounds the attack to recover one or more sets D_e has small complexity and it is possible to get sufficient experimental data. However, to be able to extrapolate the attack complexity we describe a theoretical model below.

In the attack we are faced with the problem to group 120 counters $C(\{x, y\})$, each belonging to an input pair to an S-box of the first round, into 15 distinct groups. All pairs within a group should yield the same output difference, i.e., belong to a set D_e for some e.

Interpreting the counters $C(\{x, y\})$ as random variables, a counter $C(\{x, y\})$, with $S(x) \oplus S(y) = e$ is binomially distributed with parameters n and p_e . Here p_e is the probability that the difference $(e||0^{60})$ after the first layer of S-boxes yields only one active S-box in the output and n is the number of text pairs.

Assumption 6.3 states that counters $C(\{x, y\})$ such that $S(x) \oplus S(y)$ has a weight greater than one are significantly smaller than others and we therefore focus only on the 32 counters corresponding to slender pairs. Thus, we consider 8 counters distributed with parameters (n, p_1), 8 distributed with parameters (n, p_2), 8 distributed with parameters (n, p_4) and finally 8 counters distributed with parameters (n, p_8) (here we identified $e = (0, 0, 0, 1)$ with 1, $e = (0, 0, 1, 0)$ with 2 etc.). Without loss of generality we assume $p_1 \geq p_2 \geq p_4 \geq p_8$ and that holds $p_1 \neq p_2$. The attack works by looking at the 8 highest counters and is successful if those counters correspond to the same output difference, e.g., $e = 1$, of the S-box. The attack fails whenever there exists a pair $\{x_1, y_1\}$ with output difference '1' and a pair $\{x_2, y_2\}$ with $S(x_2) \oplus S(y_2) \neq 1$ such that $C(\{x_1, y_1\}) \leq C(\{x_2, y_2\})$. In the following we estimate this error probability depending on the number of samples n.

To simplify the problem for now, we consider only two pairs $\{x_1, y_1\}$ and $\{x_2, y_2\}$ and their corresponding counters where $C(\{x_1, y_1\})$ is distributed with parameters (n, q) and $C(\{x_2, y_2\})$ is distributed with parameters (n, p) for $q > p$. The attack fails if $C(\{x_1, y_1\}) \leq C(\{x_2, y_2\})$ and thus we denote $Z = C(\{x_2, y_2\}) - C(\{x_1, y_1\})$ and the error with

$$\text{err} = \Pr(C(\{x_1, y_1\}) \leq C(\{x_2, y_2\})) = \Pr(Z \geq 0).$$

To investigate this error further consider the usual approximation of the binomial distribution by the normal distribution, $C(\{x_1, y_1\}) \sim N(nq, nq(1-q))$ and $C(\{x_2, y_2\}) \sim N(np, np(1-p))$. With this approximation, the distribution of Z can be approximated by $Z \sim N(\mu, \sigma^2)$, where $\mu = n(p - q)$ and $\sigma = n(p(1 - p) + q(1 - q))$.

The density function for the normal distribution with mean μ and variance σ^2 is given by the following formula:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

The integral of the normal density function is the normal distribution function

$$N(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-\frac{1}{2}x^2} dx.$$

The error we make is thus described by

$$\text{err} \approx 1 - \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{0} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{-\mu}{\sigma}} e^{-\frac{x^2}{2}} = 1 - N\left(\frac{-\mu}{\sigma}\right).$$

The following lemma gives an estimate of the 'tail' $1 - N(x)$ which is useful to approximate the error.

Lemma 6.7 ([50])**.**
As $x \to \infty$

$$1 - N(x) \approx x^{-1}\phi(x)$$

where $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

Using the approximation of Lemma 6.7 yields

$$\text{err} \approx 1 - N(-\frac{\mu}{\sigma}) \approx -\frac{\sigma}{\mu}\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\mu}{\sigma})^2}. \tag{6.1}$$

From (6.1) it follows that for a given error probability err the sample must be of size

$$n > \frac{-c(p^2 - p + q^2 - q)}{(p-q)^2}, \tag{6.2}$$

where $c = \text{LambertW}\left(\frac{1}{2\,\text{err}^2\,\pi}\right)$ [35] is a small constant depending on the error.

After having estimated the error probability for 2 counters, assuming independence, the total error probability err_t, that is, the probability of the event that one of the 8 counters with parameter (n, p_1) being smaller than one of the 24 counters with parameters $(n, p_2), (n, p_4), (n, p_8)$, can be bounded as

$$\text{err}_t \leq 1 - (1 - \text{err})^{8 \cdot 24}.$$

If we allow an error probability of $\text{err}_t \leq 0.5$, which in light of the strong cover filter is clearly sufficient, we need $\text{err} \leq 1 - 0.5^{1/(8 \cdot 24)} \approx 0.0036$. For this $c = 8$ is sufficient.

The next step is to find a way to estimate the probabilities p_e. Assuming the cipher is a Markov cipher we can model the propagation of differences through the cipher as a matrix multiplication of the difference distribution matrices and the permutation matrices. Considering the difference distribution table for the whole layer of S-boxes

would yield a $2^{64} \times 2^{64}$ matrix. Therefore we determine the difference distribution matrix which contains only the probabilities for 1 to 1 bit differences, which as it turns out when comparing to experimental data, is a good approximation. This matrix is only of size 64×64. This enables us to simulate the propagation of 1 to 1 bit differences through a number of rounds using matrix multiplications. For the resulting matrix an entry (i, j) contains the probability that given the single, active input bit i after the first layer of S-boxes, a single output bit j in the second last round will be active. This matrix can therefore be used to get an estimate for the parameters of the counters. We determine the probability that given a fixed 1 bit difference after the first round exactly one S-box is active in the last round (analogously for the inverse). This can be done by summing over the corresponding matrix entries. Then we use formula (6.2) to calculate the number of plaintexts needed to recover at least two sets D_e in each direction. Note that in the original attack we do not restrict ourselves to having a single active S-box in the last round but a limited number of active S-boxes. Furthermore, we can expect that a single active S-box will on average not lead to 16 active S-box after two rounds of encryption. Thus we believe that in practice we can break at least two more rounds of encryption with the sample size determined by the model, meaning the model yields an upper bound for the complexity.

The comparison between the experimental data and the modeled data supports this assumption.

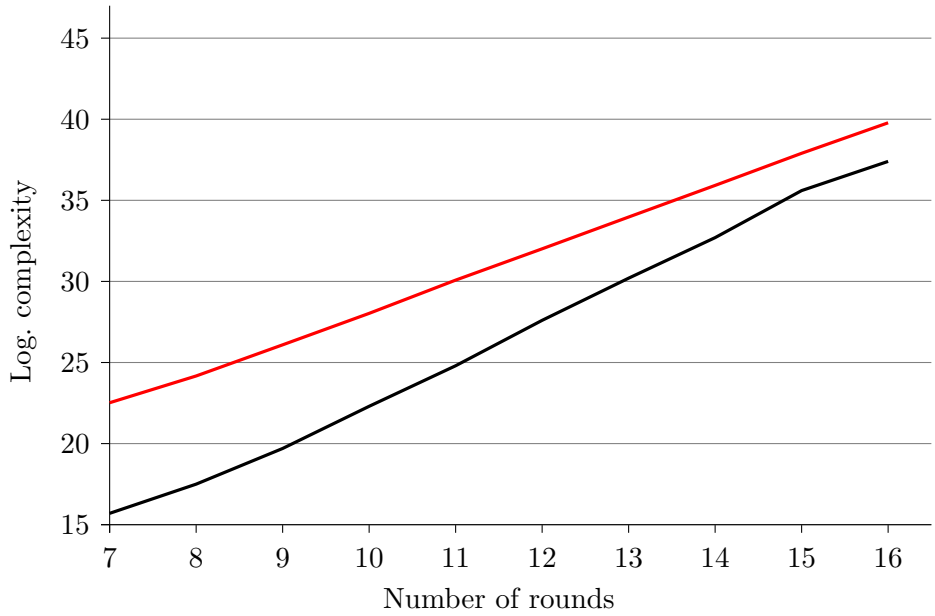


Figure 6.3: Comparison between the medians of the experimental data and the model for recovering two sets D_e in each direction. The black line shows the experimental data while the red line shows the data from the model. The complexity unit is one plaintext.

To justify the introduced model we implemented the attack for a small number of rounds (see Section 6.4). For each number of rounds we sampled 1000 ciphers in our model to determine the sample size needed to distinguish between the two distributions. Figure 6.3 gives a comparison of the experimental data with that of the model for the case that we want to recover at least two sets D_e in each direction for all 16 S-boxes. The black line shows the experimental data and the red line shows the model for an error of around 0.3% which corresponds to $c = 8$. The complexity is given as the logarithm of the number of plaintexts used. As seen, the model seems to give an upper bound on the complexity of the attack. In some rare cases the difference between p and q is close to zero, which leads to a very high attack complexity. These rare cases have a strong influence on the average complexity, hence we considered the median instead of the mean to estimate the complexity of the attack.

The modeled data suggest that we are able to break up to 28 rounds before we reach the bound of 2^{64} available plaintexts.

6.6 Fully random PRESENT-like ciphers

In this section we will shortly sketch how to break the fully random version of a PRESENT-like cipher, which was introduced in Algorithm 4. In this variant the S-boxes and the bit permutations of all rounds are chosen independently and uniformly at random.

Fully random ciphers of this kind have been investigated before. Biryukov and Shamir investigated the security of iterated ciphers where the substitutions and permutations are all key-dependent [19]. In particular they analyzed an AES-like cipher with 128-bit blocks using eight-bit S-boxes. An attack was presented on five layers (SASAS, where S stands for substitution and A stands for affine mapping) of this construction which finds all secret components (up to an equivalence) using 2^{16} chosen plaintexts and with a time complexity of 2^{28}. Using the terminology of "rounds" as in the AES, this version consists of two and a half rounds.

The second variant of our cipher, which we consider in this section, is a special instance of the SASAS cipher [19]. In fact the attack of Biryukov and Shamir applies to three rounds of this variant and has a running time of 2^{16} using 2^8 chosen texts. However, the complexity of the attack for more than three rounds is unclear, but seems to grow very quickly [19]. The SASAS attack is a multiset attack whereas we use a differential-style attack to recover the S-boxes. Also, the technique to recover the bit permutation is different.

Now, we outline the attack in the fully random case. We first focus on recovering the S-boxes. As the S-boxes layer is different for each round of encryption we will not get information about the inverse of the S-boxes like in the case of Maya. This means we cannot uniquely determine the S-boxes but just up to equivalence. Furthermore, we need to identify all four slender sets D_e in order to determine the S-boxes uniquely up to equivalence. We implemented a series of attacks on such ciphers and the results show that recovering four sets is indeed possible, but not for all S-boxes. The following table shows the results of our tests to fully recover one S-box in the first round. The

complexity is the number of chosen plaintexts needed and is given as the median of 500 tests.

Rounds	Complexity	Probability
4	$2^{12.5}$	73%
5	$2^{15.5}$	82%
8	$2^{24.5}$	81%

In each test the computation was stopped if not all 4 slender sets where obtained with 2^{30} structures. The tests are very time-consuming which is why results for 6 and 7 rounds were not implemented.

Summing up, the attack does not seem to be able to fully recover all S-boxes of the first (or last) round, merely about 80%. However, in the remaining cases, the attack identifies one, two or three sets S_e, which means that only a limited number of choices for these S-boxes remain. Depending on exactly how many choices of the S-boxes are left, one possible way to proceed is to simply make a guess, and repeat the attack on a reduced number of rounds. If S-boxes in other rounds cannot be successfully recovered, the guess might have been wrong. This is a topic for further research.

After successfully recovering the S-box layer of the first round we can peel of the S-box layer and try to recover the bit permutation in the first round.

The idea is similar to recovering the S-boxes. We encrypt plaintext pairs that differ in only a few bit positions, for example two, meaning that we encrypt pairs $(x, x \oplus 2^i \oplus 2^j)$ for two bit positions $i \neq j$ and $x \in \mathbb{F}_2^{64}$. (As we can peel of the S-box layer these plaintext pairs are inputs to the permutation in the first round.)

As in the S-box recovery attack we expect that if exactly one S-box is active in the second round only a limited number of S-boxes will be active in the last round. Hence, for each pair of bit positions we maintain a counter which we increment if a plaintext pair corresponding to these bit positions leads to a ciphertext difference with only one or a few active S-boxes. After we encrypted a sufficient number of plaintext pairs for all possible pairs of bit positions we assume that the highest counter values corresponds to pairs of bit positions that are mapped to the same S-box.

This leads to information about which bit positions are mapped to the same S-box input in the next round. We can also use plaintext pairs that differ in three or four bit positions in order to obtain more information. The complexity of this method has not been thoroughly investigated, but preliminary results indicate that it is similar to (if not lower than) the complexity of recovering S-boxes.

The attack on the fully random PRESENT-like ciphers we presented here has not been thoroughly investigated and we cannot confirm our assumptions with a sufficient amount of experimental data. This is a direction for further research.

6.7 Linear-style Attack

In this section we discuss the possibility to use linear cryptanalysis to obtain information about the S-boxes. Our differential-style attack is based on the hypothesis that the probability of a characteristic with a single-bit difference at the output of the

S-box layer in the first round is correlated to the number of active S-boxes in the last round. Under this hypothesis we try to identify sets of 8 pairs that correspond to a output difference of weight one. Under a similar hypothesis for linear characteristics we can mount a linear-style attack to extract information about the S-box. The idea is to identify a set of 8 values such that these values are mapped to outputs of the S-box with the same value in some single bit. That means that the output of the S-box masked with a linear mask of weight one yields a constant value (either 0 or 1) for all values in the set.

We proceed as follows. In order to gather information about the S-box S_0 in the first round we encrypt structures of the form

$$P_{r_i} = \{(x \| r_i) \mid x \in \mathbb{F}_2^4\}$$

where each $r_i \in \mathbb{F}_2^{60}$ for $0 \leq i < t$ is chosen uniformly as we did in the differential-style attack. Now we consider the S-boxes in the last round one by one and for each S-box generate a table in the following way. The rows of the table are labeled with all possible four-bit linear masks and the columns with the plaintext nibbles corresponding to the input to S_0 in the first round. For each entry in the table we initialize a counter with zero. For each plaintext we calculate the scalar product of the four bits of ciphertext which are output of the S-box we consider and all 15 possible linear masks. If the product is one we increment the counter in the table corresponding to the 4 bits of plaintext and the linear mask, otherwise we decrement the counter. After encrypting sufficiently many plaintexts we search for a column with 8 positive and 8 negative counters with high absolute values. This yields us a partitioning of the 16 inputs to the S-box S_0 into to groups and we expect that S-box outputs coincide in one bit for all 8 values within these sets.

We ran a small number of experiments on ciphers with a small number of rounds. It is unclear how many structures must be encrypted in order to ensure that the sets we obtain are correct with a high probability. However, there is a lot of information we can use. Each table consists of 15 columns and each of these columns may suggest a partitioning into two sets. Furthermore, we generate one such table for each S-box. In our limited number of tests on a 7-round cipher we could observe that this approach yields a correct partitioning if a set is suggested by at least two columns of an S-box and at least two S-boxes. However, more experiments are necessary and it is a natural future direction of research to combine differential-style and linear-style attacks.

6.8 Conclusion

In this section we presented a novel differential-style attack and applied it to several 64-bit PRESENT-like ciphers with secret components. A variant with 16 secret S-boxes can be attacked for up to 28 rounds with a data complexity of less than 2^{64}. It is interesting to note that the best known attack on PRESENT, a linear attack, can be used to cryptanalyze up to 26 rounds of PRESENT (which has publicly known S-boxes and bit permutation).

Also, the variant where the S-boxes and bit permutations are chosen at random for every round can also be attacked with a data complexity of less than 2^{64} for up to 16 rounds.

It is clear that our attacks exploit that there are weak differential properties for some randomly chosen four-bit S-boxes, and they do not apply to ciphers where the S-boxes are chosen as in PRESENT. However, the number of such strong (w.r.t. differential attacks), non-equivalent S-boxes is very small, so this restriction would not allow for a huge key.

Part III

Cryptanalysis by Optimization

Chapter 7

Optimization

Optimization means the action of finding the best solution for a problem. Historically optimization goes back to Gauss, the steepest descent method [44] is the first known optimization method. Also the introduction of linear programs in 1940 by Dantzig [40, 39] can be seen as the origin of optimization. Dantzig used mathematical techniques for generating training schedules for military applications which were called programs. Not only linear programming techniques but also other optimization techniques nowadays have many applications such as production scheduling, supply-chain optimization, topology optimization, structural, and process optimization. The application areas span from agriculture over microchips to airplanes, the oil industry and banking.

Generally speaking an optimization problem can be represented as follows.
Given: a function $f : S \to \mathbb{R}$ from a set S into the reals.
Sought: an element $x_{opt} \in S$ such that $f(x_{opt})$ is a minimum (or maximum) for all $x \in S$.

Usually, the set S is a subset of the Euclidean space $\mathbb{R}^n$, which is often specified by a set of constraints and integrality restrictions. Constraints are equalities and inequalities that the elements of S have to satisfy. Integrality restrictions mean that some or all of the variables are restricted to be integers. The function f is called the objective function and assigns to each element in S an objective value. This is the function that shall be minimized or maximized, where we will only consider minimization in the remainder of this thesis. If the set S is discrete its elements are also called configurations. In general we distinguish between global and local extrema.

Definition 7.1 (Extrema)**.**
Given a function $f : S \to \mathbb{R}$. An element $x_{min} \in S$ is

- *a local minimum of f if $f(x_{min}) \leq f(x)$ for all $x \in S$ with $|x - x_{min}| \leq \epsilon$, i.e., for all $x \in S$ which lie in a certain distance to x_{min},*
- *a global minimum of f if $f(x_{min}) \leq f(x)$ for all $x \in S$.*

An element $x_{max} \in S$ is

- *a local maximum of f if $f(x_{max}) \geq f(x)$ for all $x \in S$ with $|x - x_{max}| \leq \epsilon$, i.e., for all $x \in S$ which lie in a certain distance to x_{max},*
- *a global maximum of f if $f(x_{max}) \geq f(x)$ for all $x \in S$.*

We are mainly interested in global optima. However, often it is difficult to find a global optimum, especially when the function f contains many local optima.

There are many different subclasses of optimization problems. For some there exist efficient solvers, while for others the solvability depends on the structure of the instance we consider. Three typical characteristics of an optimization problem are: Linear/Non-linear, Constrained/Unconstrained and Real-valued/Integer-valued.

In a linear optimization problem or a linear programming problem the objective function and all constraints are linear. There are efficient algorithms such as the simplex method [101] for solving linear programming problems. In non-linear optimization problems the objective function and/or the constraints are non-linear. Whether such optimization problems are easily solvable depends on the degree of non-linearity and other features such as convexity.

A constrained optimization problem can be presented as

$$\min_x z = f(x)$$

$$\begin{aligned} \text{subject to} \quad & g_1(x) = b_1 \\ & g_2(x) \leq b_2, \end{aligned}$$

where f is the objective function and the constraint functions g_1 and g_2 describe a set of equalities and inequalities which have to be satisfied. In an unconstrained optimization problem we search for a minimum or maximum of a function $f : X \to \mathbb{R}$. Depending on the set of configurations X, neighborhood search or hill climbing algorithms such as simulated annealing can be applied to the problem. Roughly speaking generalized hill climbing [61] or a neighborhood search tries to optimize the objective function by moving from one configuration of the problem to a neighbor and comparing the objective values. Moves which improve the objective value are always accepted, while worsening moves are only accepted with a certain probability

Most of the time we assume that the variables in the optimization problem are real-valued, but we can also restrict some or all of the variables to integers. This is beneficial if we consider discrete problems such as scheduling problems. This leads us to mixed-integer and integer programming.

In the following sections we first explain mixed-integer linear programming which considers linear, constrained optimization problems where some of the variables are integers and some are defined over the reals. Afterwards we illustrate simulated annealing, a hill climbing or neighborhood search algorithm which is used to optimize combinatorial optimization problem. Simulated annealing works on a discrete configuration space and is therefore suitable for our kind of problems which are defined over the Boolean domain.

7.1 Mixed-Integer Optimization

Combinatorial and integer optimization deals with the problem of minimizing (or maximizing) a function of several variables subject to equality and inequality constraints and integrality restrictions on some or all of the variables.

Pure integer optimization and mixed-integer optimization have many applications including operational problems such as production scheduling and distribution of goods as well as planning problems such as capital budgeting and facility location and design problems such as network design and the design of automated production systems. Integer optimization has been successfully applied to tackle difficult problems. One example is the traveling salesman problem. The traveling salesman problem is stated as follows. Given a list of cities and their pairwise distances the task is to find the shortest tour that visits each city exactly once. It is known that the decision version of the traveling salesman problem belongs to the complexity class of NP-complete problems. However, when the traveling salesman problem is carefully formulated as an integer linear programming problem it has been successfully solved for up to 2000 cities [101]. (The most direct solution is to try all permutations of the n cities and calculate the length of the tour. This has a complexity of $\emptyset$(n!) and becomes already for around 20 cities computational infeasible.) The success of integer optimization for difficult discrete problems raises the question if these methods can also be exploited for cryptanalysis. We will discuss some possibilities to apply integer or more precisely mixed-integer optimization in the cryptanalysis of the stream cipher Trivium in Chapter 9. But to begin with we shall define a mixed-integer programming problem and explain some of the algorithms which are used to solve this kind of problems.

As mentioned before optimization deals with the problem of minimizing or maximizing a function subject to certain constraints and restrictions on the variables. Throughout this thesis we assume that the function shall be minimized. All definitions and algorithms are also valid for the problem of maximizing a function because that is equivalent to minimizing the negation of the same function. In the case of linear programming problems all constraints are linear, while in mixed-integer linear programming problems some of the variables are additionally restricted to be integer variables.

Definition 7.2 (Mixed-integer linear programming problem)**.**
A mixed-integer linear programming problem (MIP) is a problem of the form

$$\min_x c^T x$$

subject to

$$Ax \leq b,$$
$$x \in \mathbb{Z}^k \times \mathbb{R}^l$$

where c is an n-vector, A is an $m \times n$-matrix ($n = k + l$) and b is an m-vector ($c \in \mathbb{R}^n,\ A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m$) .

This means that we minimize a linear function subject to linear equality and inequality constraints. Additionally some of the variables are restricted to integer values while the remaining variables are real-valued. In fact it does not matter if we

allow both equality and inequality constraints or if we only allow inequality constraints since each equality can be expressed as two inequalities.

The linear function $c^T x$, objective to minimization, is called the objective function. A point x which fulfills all integrality restrictions and all constraints is an element of the feasible set S.

Definition 7.3 (Feasible set)**.**
The set S of all $x \in \mathbb{Z}^k \times \mathbb{R}^l$ which satisfies the linear constraints $Ax \leq b$

$$S = \{x \in \mathbb{Z}^k \times \mathbb{R}^l, Ax \leq b\}$$

is called a feasible set. An element $x \in S$ is called a feasible point.

An instance of a mixed-integer programming problem is said to be *feasible* if at least one feasible point exists, i.e., if $S \neq \emptyset$. In the case $S = \emptyset$ the problem is infeasible. If the problem instance is feasible a solution for the optimization problem is a feasible point which minimizes the objective function. That means x_{opt} is a solution if $x_{opt} \in S$ and $c^T x_{opt} \leq c^T x$ for all $x \in S$. Then $c^T x_{opt}$ is called the *optimal value*. However, even though the instance is feasible it might not have an optimal solution. In this case the instance is called *unbounded*, that means that for any $\omega \in \mathbb{R}$ there exists an $x \in S$ such that $c^T x < \omega$. Every mixed-integer programming problem either has an optimal solution, is unbounded or infeasible. Hence, solving a mixed-integer problem means either to find a solution or to show that the problem is unbounded or infeasible. However, when we apply mixed-integer optimization in cryptanalysis we consider in almost all cases problems where we know that a solution exists.

In general one may consider two different types of problems in mixed-integer programming: the optimization and the feasibility problem. While in the *optimization problem* we look for a feasible point which minimizes the objective function, meaning we are looking for an optimal solution, in the *feasibility problem* we try to find a point x that fulfills all linear constraints and restrictions on the variables regardless of the objective value. Intuitively a feasibility problem is easier to solve than an optimization problem because we are looking for just one feasible point instead of a feasible point that yields an optimal value. However, in practice finding any feasible point may be the most difficult part of the problem.

Special cases of linear mixed-integer programming are linear programming and integer programming as well as mixed-0-1 or 0-1 programming. In linear programming (LP)

$$\min_x \{c^T x | Ax \leq b, x \in \mathbb{R}^n\}$$

there are no restriction on the variables. That means all variables are continuous. There are efficient algorithms for solving linear programming problems such as the simplex algorithm [101] and the interior point method [89]. Furthermore, most algorithms for integer or mixed-integer programming use LP relaxations.

Definition 7.4 (LP relaxation)**.**
The LP relaxation or linear programming relaxation of an integer, mixed-integer or

0-1 programming problem is the linear programming problem that arises by removing the integer or binary constraints and replacing them by weaker constraints that each variable is continuous where applicable in an interval e.g., $[0, 1]$.

Thus, linear programming algorithms are applied in order to solve the linear program which is obtained from the integer programming problem by removing the integrality constraints. More details and an example for an LP relaxation are given in Subsection 7.1.2.

In an integer programming problem (IP)

$$\min_x \{c^T x | Ax \leq b, x \in \mathbb{Z}^n\}$$

all variables are integer-valued. We are mainly interested in mixed-0-1 programming problems where the integer variables are replaced by binary variables.

7.1.1 Modeling

Usually there is more than one way to model an MIP or an IP. In integer programming, formulating a 'good' model is of crucial importance for solving the problem [101]. The first step in formulating a model is usually defining variables. The variables are often already given by the definition of the desired solution. When the problem is stated the solution is usually defined by certain unknowns we want to determine. We define a variable for each of these unknowns. One might also use some additional auxiliary variables. The second task is to find a good objective function. We will consider a feasibility problem rather than an optimization problem. Hence, we can choose an arbitrary objective function. The main problem is to find a good formulation for the feasible set $S = \{x \in \mathbb{Z}^k \times \mathbb{R}^l, Ax \leq b\}$. Here it is often easy to find A and b which yield a valid formulation for S but this description of the feasible set might not be the best one for actually solving the problem. The reason for this is that integer optimization algorithms such as branch-and-bound need a lower bound on the objective function and they often determine this bound by using relaxations. One possible relaxation of the problem is to solve the corresponding linear program

$$z_{LP} = \min_x \{c^T x : Ax \leq b, x \in \mathbb{R}^n\}.$$

The feasible set S of the mixed-integer program is a subset of the feasible set P of the linear programming relaxation

$$S \subset P = \{x \in \mathbb{R}^n : Ax \leq b\}.$$

The smaller the set P is, the more precise are the bounds for the mixed-integer program and the efficiency of most algorithms depends on these bounds. Let us consider the following example (taken from [101]).

The three following sets describe the same feasible set S

$$
\begin{aligned}
S = \{x \in \{0,1\}^4 \;&:\; 93x_1 + 49x_2 + 37x_3 + 29x_4 \leq 111\}, && (7.1)\\
S = \{x \in \{0,1\}^4 \;&:\; 2x_1 + x_2 + x_3 + x_4 \leq 2\}, && (7.2)\\
S = \{x \in \{0,1\}^4 \;&:\; 2x_1 + x_2 + x_3 + x_4 \leq 2 && \\
&\quad\; x_1 + x_2 \leq 1 && (7.3)\\
&\quad\; x_1 + x_3 \leq 1 && \\
&\quad\; x_1 + x_4 \leq 1\}. &&
\end{aligned}
$$

We consider the corresponding LP relaxation of the mixed-integer programming problem using the different description of the set S

$$
\begin{aligned}
P_1 = \{x \in \mathbb{R}^4 \;&:\; 93x_1 + 49x_2 + 37x_3 + 29x_4 \leq 111\},\\
P_2 = \{x \in \mathbb{R}^4 \;&:\; 2x_1 + x_2 + x_3 + x_4 \leq 2\},\\
P_3 = \{x \in \mathbb{R}^4 \;&:\; 2x_1 + x_2 + x_3 + x_4 \leq 2\\
&\quad\; x_1 + x_2 \leq 1\\
&\quad\; x_1 + x_3 \leq 1\\
&\quad\; x_1 + x_4 \leq 1\}.
\end{aligned}
$$

It holds $P_3 \subset P_2 \subset P_1$. Hence $z_{LP_3} \geq z_{LP_2} \geq z_{LP_1}$ for a minimization problem. This means that the formulation of the problem using the set described in (7.3) yields a sharper bound and can therefore be considered a better formulation of the problem.

7.1.2 Algorithms

As integer and mixed-integer linear programming problems almost look like linear programming problems it is tempting to solve the LP relaxation of the problem and just round the solution to the integer values. The following small example shows that this will in general not lead to an optimal solution and often not even to a feasible point [30]. We consider the integer program

$$
\begin{aligned}
\text{minimize } z \;&=\; -x_1 - 5x_2\\
\text{Subject to:}\quad & x_1 + 10x_2 \leq 20,\\
& x_1 \leq 2,\\
& x_1, x_2 \geq 0,\\
& x_1, x_2 \in \mathbb{Z}.
\end{aligned}
$$

Solving the LP relaxation of this problem yields the point $(2, 1.8)$ as solution with the optimal value $z = -11$. The integrality restriction is violated for the variable x_2. Rounding to the nearest integer value yields the point $(2, 2)$. But this point is infeasible because the first constraint is violated. Rounding down yields a feasible point. However, $(2, 1)$ yields the objective value $z = -7$ while the point $(0, 2)$ yields the objective value $z = -10$ and is feasible. Therefore $(2, 1)$ is not the optimal solution

for the integer programming problem. Even though solving the corresponding linear program and rounding solution values that violate the integrality restrictions does not solve the integer programming problem, LP relaxations play an important role when solving integer and mixed-integer programming problems.

In this section we will sketch two important algorithms used in mixed-integer and integer linear programming: the branch-and-bound algorithm [30, 59] and the cutting-plane algorithm. Both algorithms solve the problem by progressively adding constraints to the problem. In practice there are many variations of these algorithms. Furthermore, the algorithms are usually combined with each other or other solvers, which will not be mentioned here, in order to achieve good running times and optimal solutions. First, we explain the basic idea of the branch-and-bound method [30, 59].

The Branch-and-Bound Technique

A natural idea for solving integer-valued problems is to simply enumerate all possible solutions and then choose the best one. But even for medium-size problems this exhaustive search becomes computationally infeasible. However, the branch-and-bound technique, one of the workhorses for solving integer programming problems, is based on the observation that the enumeration of all possible integer solutions forms a tree. The main idea of the branch-and-bound method is to avoid growing the whole solution tree by growing the tree in stages. This is done by a divide-and-conquer approach. The original problem is divided into smaller subproblems. This is called branching and done by partitioning the feasible set into subsets. Then the problems are conquered by bounding and pruning. The three basic steps of every branch-and-bound algorithms are branching, bounding and fathoming and will be explained in the following.

Branching

We grow a tree containing a subset of the feasible set in each node starting from the set of all solutions by branching. This means we consider the current stage of the tree and pick a node at which we want to grow the tree further, i.e., divide the set of solutions given at that node and the corresponding problem into smaller subsets and corresponding subproblems. In general the division is done by adding a constraint of the form $dx \leq d_0$ in order to generate one branch of the node and $dx \geq d_0 + 1$ to generate the other where $d \in \mathbb{Z}^n$ and $d_0 \in \mathbb{Z}$. If x_{LP} is a solution for relaxation of the subproblem represented by the current node then we choose (d, d_0) such that $d_0 < dx_{LP} < d_0 + 1$ holds. In practice we only branch at one variable, i.e. $d = e_j$ for $j \in \mathbb{N}$, where e_j is the jth unit vector. This is called variable dichotomy[1]. As an example we assume that the solution of the LP-relaxation of an integer optimization problem is $(1, 3.5, 2.1)$. Using variable dichotomy we generate two subproblem by adding the constraints $x_2 \leq 3$ and $x_2 \geq 4$ respectively.

[1] A dichotomy is any splitting of a whole into exactly two non-overlapping parts, meaning it is a procedure in which a whole is divided into two parts. Variable dichotomy means that we choose one variable to perform this splitting.

In the case of binary problems the straight forward way to partition the set of feasible solutions is to fix one the variables x_i to its values. This will yield two subsets, one where $x_i = 0$ and one where $x_i = 1$. The variable which is used to branch the tree at any iteration of the algorithm is called the branching variable. When we consider integer-valued problems we can still assign the branching variable all values, at least in the cases where the variables are restricted to a finite number of possibilities. A good alternative to this approach is variable dichotomy where we specify a range of values for the branching variable by introducing new constraints e.g., $x_i \leq d_0$ and $x_i \geq d_0 + 1$. This way we only get two branches/two children for each node.

If the division is done by variable dichotomy for all nodes of the tree we can prove that the tree is finite.

Proposition 7.5.
If $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ is bounded and a solution tree is developed in such a way that at each node i that requires branching a variable dichotomy of the form $(x_j \leq [x^i_j], x_j \geq [x^i_j] + 1)$ is chosen where x^i is the solution of the corresponding relaxation and x^i_j is not integral. Then the solution tree developed on variable dichotomies is finite.

The reader is referred to [101] for the proof. This proposition guaranties the termination of a branch-and-bound algorithms, at least for the cases where the feasible set of the corresponding LP relaxation is bounded.

The choice of the node at which we want to grow the tree further as well as the choice of the actual branching variable might be important. We will discuss the node selection and the variable selection policies later in this section.

Bounding

The bounding function assigns to each node in the solution tree an estimate of how good the best feasible solution for this subproblem can be. More precisely, the bounding function estimates the best value of the objective function that can be obtained by growing the tree further in this node. The estimate should be as precise as possible but it must be an optimistic estimate. In other words if we look for a minimum, the bounding function should underestimate the actual best achievable objective function value, so the bounding function must make errors in the right, the optimistic direction.

Often relaxations of the original problem which are easy to solve are used as bounding functions. In most cases a relaxation is obtained by removing the set of constraints which makes the problem difficult. In the case of integer or mixed-integer programming problems these constraints are the integer restrictions on all or some of the variables. The relaxation obtained by deleting the set of integer constraints, LP relaxation, is the most widely used relaxation.

The bounding function is used for selecting the next node for branching in each iteration of the algorithm as well as for pruning certain branches of the solution tree.

Fathoming

Sometimes the solution of the LP relaxation is already a feasible solution for the

integer or mixed-integer programming problem. Such a solution is called *incumbent* and denotes the best feasible solution so far. The incumbent is stored together with its objective value until a better solution is found and the current incumbent is replaced by this better solution. If an incumbent is found as a solution for one of the subproblems in the tree, we do not have to expand the tree any further in this node. The reason is that we will not find a better solution by expanding the node since the LP relaxation yields an optimistic estimate of the best achieve objective value in this node. It is, however, possible to find another feasible solution, which is equivalently good or worse than the one we already found, by growing the tree to the next stages at that node.

The incumbent together with the bounding function is used to prune nodes of the solution tree. Pruning is crucial in the branch-and-bound method because it prevents that the solution tree grows too big. *Pruning* means that we cut off and permanently discard nodes in the tree because neither the node nor any of its descendants will yield an optimal or feasible solution. There are two reasons to prune a node. Firstly, a node can be pruned if expanding this node will not yield a better solution than the current best feasible solution, namely the incumbent. That means a node can be pruned when the bounding function value of a node is worse than the objective value of the incumbent. Secondly, a node can be pruned if we can show that there is no feasible solution. For example if the LP relaxation of the subproblem presented by the node is infeasible also the integer or mixed-integer problem will be infeasible and the node can be discarded.

The algorithm terminates and has found a globally optimal solution for the IP or MIP or shown that the problem is infeasible if the solution tree cannot be grown any further because all nodes are either pruned or already contain a feasible solution.

In each iteration of the algorithm one has to decide which node to be expand. There are three popular node selection policies.
Best-first: We always select the most promising node for branching, that is the node with the best bounding function value. This node is the most promising one to lead to the optimal solution of the original problem.
Depth-first: We choose the node for branching among the nodes which have been created in the last step. With this selection policy we reach a leaf of the tree quickly and it is therefore a way to achieve an early incumbent, at least if there are sufficiently many elements in the feasible set. Another advantage is that the LP relaxation which has to be solved for branching can use reoptimization instead of solving the problem from scratch. Reoptimization uses the final simplex tableau of the LP relaxation solved in the previous iteration and tends to be much faster than starting from scratch.
Breadth-first: We expand the nodes in the same order in which they were created.

Once we have chosen a node for branching we have to select a branching variable. In the case of mixed-integer programming one only considers the integer-valued variables as the branching variables. One possibility is to choose the branching variables in their natural order. Another strategy is to consider the solution of the LP relaxation and choose one of the variables that violates the integrality restrictions as branching

variable. We assume that the variable $x_i = q$ where $q \in \mathbb{R} \setminus \mathbb{Z}$ in the solution of LP relaxation. Then we choose x_i as the branching variable and generate two new subproblems; one containing the additional constraint $x_i \leq \lfloor q \rfloor$ and the other containing $x_i \geq \lfloor q \rfloor + 1$ or in the case that x_i is a binary variable $x_i = 0$ and $x_i = 1$.

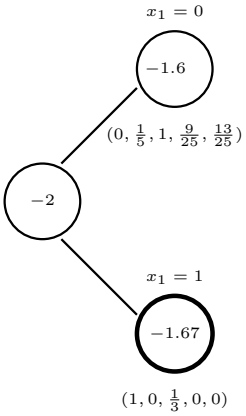
We will now clarify the steps of the algorithm with the help of a simple example. We consider a binary linear program to simplify matters. However, as explained above the algorithm works similarly for integer or mixed-integer programs. We select variables which violate the binary restrictions as branching variables and select the nodes for branching following the best-first policy. Moreover, we use the LP relaxation as the bounding function. The following binary linear program is given:

$$\begin{aligned} \text{minimize } f(x) &= -x_1 + x_2 - 2x_3 + 2x_4 - x_5 \\ \text{subject to} \quad & x_1 + x_2 \leq 1, && (7.4)\\ & x_1 - 5x_2 + 3x_3 \leq 2, && (7.5)\\ & 2x_3 + 3x_4 - 4x_5 \leq 1, && (7.6)\\ & x_2 - 2x_4 + x_5 \leq 0, && (7.7)\\ & x \in \{0,1\}^5. && (7.8) \end{aligned}$$

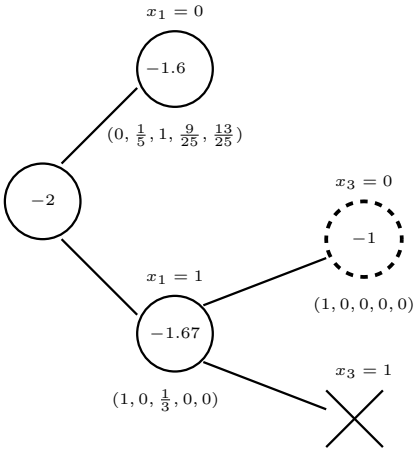
First of all we solve the LP relaxation of this problem which we obtain by replacing the binary restriction $x \in \{0,1\}^5$ by the constraints $0 \leq x_i \leq 1$ for $1 \leq i \leq 5$, meaning the variables are continuous in the interval $[0,1]$. The optimal solution of the relaxation is $(\frac{2}{3}, \frac{1}{3}, 1, \frac{7}{15}, \frac{3}{5})$ and yields -2 as a bounding function value for the node in the solution tree containing all solutions. The variables x_1, x_2, x_4 and x_5 violate the binary restrictions. We choose x_1 as branching variable and generate two subproblems of our original problem by adding the constraint $x_1 = 0$ or $x_1 = 1$ respectively. The LP relaxation of the subproblem obtained by adding the constraint $x_1 = 0$ yields $(0, \frac{1}{5}, 1, \frac{9}{25}, \frac{13}{25})$ as optimal solution and -1.6 as bounding function value. The optimal solution of the other subproblem is $(1, 0, \frac{1}{3}, 0, 0)$ and the corresponding bounding function value is -1.667. Now we pick the most promising node in order to expand it in the next iteration. This is the node corresponding to the subproblem containing the constraint $x_1 = 1$ because $-1.667 < -1.6$ and it is marked by a bold circle in Figure 7.1, Iteration 1. We select x_3 as our next branching variable because it is the only variable which violates the binary restrictions in the solution of relaxation.

The node corresponding to the subproblem generated by adding the constraints $x_1 = 1$ and $x_3 = 1$ can be pruned because the LP relaxation of the problem is infeasible. The reason is that the constraint (7.4) forces $x_2 = 0$. But in order to satisfy constraint (7.5) x_2 has to be at least $\frac{2}{5}$. Because of this contradiction the LP relaxation and therefore also the binary problem are infeasible.

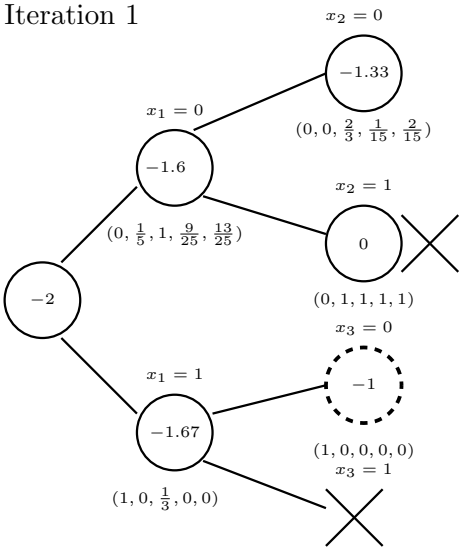
The relaxation subproblem containing the constraints $x_1 = 1$ and $x_3 = 0$ yields a feasible solution. The objective function value associate to the incumbent $(1, 0, 0, 0, 0)$ is -1 and the incumbent is marked by a dashed circle in Figure 7.1. We can use the incumbent to prune nodes in the tree which cannot lead to a better solution than the incumbent and so prevent the tree from growing too big.



Iteration 1



Iteration 2



Iteration 3

Figure 7.1: Iterations 1-3 of the branch-and-bound algorithm. The bold circle marks the most promising node, the dash circle the incumbent and a cross means that the branch of the tree is pruned, either because the LP-relaxation is infeasible (only cross) or because we already found a feasible solution which is worse than the incumbent (circle + cross).

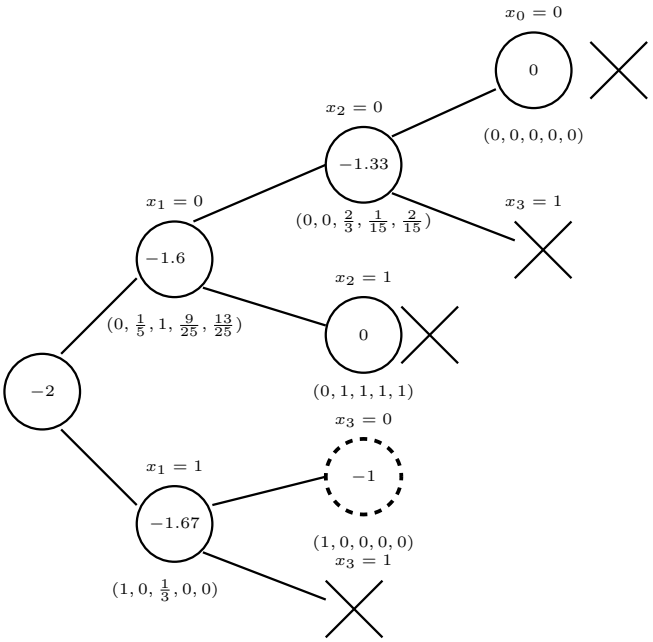


Figure 7.2: The complete solution tree for the branch-and-bound algorithm.

However, the incumbent might not be the optimal solution. Therefore we have to check if expanding any of the other nodes in the tree leads to a better solution. We consider the node labeled with $x_1 = 0$ again and choose x_2 as branching variable. Adding the constraint $x_2 = 1$ yields $(0, 1, 1, 1, 1)$ as solution for the relaxation. As this solution is already feasible but yields a worse objective function value than the incumbent, this node can be pruned.

The subproblem obtained by adding the constraint $x_2 = 0$ instead yields a fractional solution for the relaxation and the corresponding bounding function value is better than the objective function value of the incumbent. Therefore we branch again using x_3 as branching variable. The combination $x_1 = 0, x_2 = 0$ and $x_3 = 1$ violates constraint (7.5) so the corresponding subproblem is infeasible and the node can be pruned. Adding the constraint $x_3 = 0$ to the subproblem yields a feasible solution $(0, 0, 0, 0, 0)$. But the objective function value is worse than the value of the incumbent, so we prune the node. Now the algorithm terminates because we cannot grow the tree any further (cf. Figure 7.2).

A closer look at the objective function would have told us already in the second iteration that the feasible solution we found was an optimal solution. The objective function consists of integer-valued coefficients. Therefore the value of the objective function is an integer for any binary solution. That means we can tighten the bounds by rounding up. In our example all nodes generated in the first iteration would be bounded by -1 and therefore we could terminate the algorithm after we found a feasible

solution with -1 as corresponding objective function value. This observation can be used to tighten the bounds and thus further restrict the size of the search tree in branch-and-bound computations.

This example demonstrates a pure branch-and-bound algorithm for binary linear programming problem. However, in practice the branch-and-bound algorithm is usually combined with cutting-plane method to the so-called branch-and-cut algorithm. We will roughly sketch the idea of the cutting-plane method in the following subsection.

The Cutting-Plane Method

The cutting-plane method in general denotes a family of optimization methods which iteratively refines the feasible set by adding linear constraints to it. These linear constraints are called cuts. Gomory [56] introduced cutting-plane methods to solve mixed-integer linear programming problems.

The core idea of all cutting-plane methods is the following. The algorithm solves the LP relaxation of the given mixed-integer or integer linear programming problem. If this solution already satisfies the integrality restrictions, it is an optimal solution for the original problem as is the case in the branch-and-bound technique. Otherwise the optimal solution for the LP relaxation is in a corner of the polytope that describes the feasible region of the LP relaxation. The idea is now to add a new linear constraint, a cut, to the feasible set which fulfills the following properties:

1. The optimal solution of the LP relaxation is not feasible for this cut.

2. None of the feasible solutions which fulfill the integrality constraints are eliminated from the feasible set by this cut.

An example for a cut fulfilling these properties is illustrated in Figure 7.3. It is guaranteed that such a linear inequality exists. There are different ways to calculate the cuts. The best known example are Gomory's cuts [56]. Here the cut is deduced from the Simplex tableau of the LP relaxation. However, the technical details of how these cuts are calculated are beyond the scope of this thesis and therefore omitted here. Instead we explain a very simple procedure of how to generate cuts for pure binary linear programming problems [59]. It is based on minimal covers of constraints.

Definition 7.6 (Minimal cover)**.**
A minimal cover is a group of N variables contained in the constraint such that the constraint is violated if all variables in this group are set to 1 and the remaining variables are set to 0 but the constraint is satisfied if the value of any of these variables is changed from 1 to 0.

In order to generate a cut we consider a constraint in the feasible set that contains only nonnegative coefficients and we look for a minimal cover of this constraint. Then

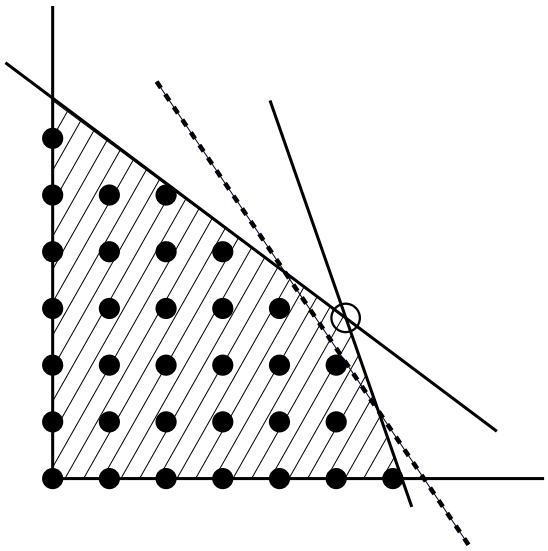


Figure 7.3: An example for a cutting-plane. The hatched area illustrates the feasible set of the LP relaxation, while the black dots are the integer feasible solutions. The optimal solution of the LP relaxation in the corner of the polytope is marked by a circle. The cut (dashed line) separates this solution from the feasible set without eliminating any integer solutions.

the new cut is that the sum over all the variables in the minimal cover has to be less or equal to $N - 1$. As an example consider the constraint

$$3x_1 + 2x_2 + 4x_3 + x_4 \leq 6$$

with $x_i \in \{0, 1\}$ for $1 \leq i \leq 4$. Then the set $\{x_1, x_3\}$ forms a minimal cover for the constraint because $(1, 0, 1, 0)$ violates the constraint whereas $(1, 0, 0, 0)$ and $(0, 0, 1, 0)$ satisfy the constraint. This yields the cut

$$x_1 + x_3 \leq 1.$$

The procedure does not guarantee that an optimal solution for the LP relaxation is not feasible for the cut as opposed to Gomory's cut. Therefore we have to choose the constraint and the minimal cover carefully so that this property is satisfied.

In practice an integer or mixed-integer problem is preprocessed before the branch-and-bound and cutting-plane methods are applied. In the preprocessing phase variables are fixed to a value, redundant constraints are eliminated and constraints are tightened. We shall not go into detail here how this preprocessing works. In all the experiments we use a commercial solver to solve our mixed-integer programming

problems. Therefore the technical details of the algorithms are not of interest. The purpose of this subsection is to give the reader a feeling of how integer optimization algorithms work.

7.2 Generalized Hill Climbing Algorithms

Generalized hill climbing algorithms [61] form a general class of heuristic optimization algorithms which includes simulated annealing [29, 66] and local search (simple hill climbing) as special cases. Generalized hill climbing is a direct search technique. This means that it does not require any derivates as opposed to optimization methods such as Newton methods. Normally, generalized hill climbing algorithms deal with discrete optimization problems and under the assumption that the objective function value of a globally optimal solution is known, the task is to identify an associated optimal solution. If the optimal value of the objective function is not known the algorithm tries to find a good approximation to the global optimum.

A discrete optimization problem is defined as follows.

Definition 7.7 (Discrete Optimization Problem)**.**
Given a discrete, finite set X of configurations and a non-negative objective function $f : X \to \mathbb{R}^+$ a discrete optimization problem is the problem of optimizing (minimizing or maximizing) the objective function over the set X of possible configurations. The objective function f is called the cost function.

In other words this means that the finite set X is composed of all possible solutions of the optimization problem. Each configuration $x \in X$ is assigned a value $f(x)$ called cost. And we want to find a configuration which minimizes or maximizes the cost. In the following we will focus only on minimization.
In order to apply generalized hill climbing methods to an optimization problem a neighborhood function $\eta : X \to 2^X$ is defined such that for each configuration $x \in X$ there is a set of neighbors $\eta(x) \subset X$. The aim of the search is to find $x_{min} \in X$ minimizing the cost function $f(x)$,

$$f(x_{min}) = \min\{f(x) : x \in X\},$$

by moving from neighbor to neighbor depending on the cost difference between the neighboring configurations. Deteriorating moves from one neighbor to another are probabilistically accepted in the hope that the algorithm does not get stuck in a local optimum.

In [61] Johnson and Jacobsen present a unified view of many hill climbing algorithms by describing conditions on accepting a move from one configuration to another. The *transition probability* $p_k(x, y)$ of accepting a move from a configuration $x \in X$ to its neighbor $y \in \eta(x)$ is defined as the product of a *configuration generation probability* $g_k(x, y)$ and a *configuration acceptance probability* $\Pr[R_k(x, y) \geq f(y) - f(x)]$, where $R_k(x, y)$ is a random variable and k is an iteration index. A general form of a hill climbing algorithm is depicted in Algorithm 5.

Algorithm 5 General formulation of hill climbing algorithms

```
x_best ← x
while stopping criterion not met do
    k ← 0                                          ▷ set the outer loop counter
    while k < K do
        for m = 0, ..., M − 1 do
            generate a neighbor y ∈ η(x) with probability g_k(x, y)
            compute the cost function f(y) of the candidate
            if R_k(x, y) ≥ f(y) − f(x) then
                x ← y                                          ▷ accept the move
                if f(x) < f(x_best) then
                    x_best ← x                    ▷ store the best configuration
                end if
            end if
        end for
        k ← k + 1
    end while
end while
```

By changing the definition of $R_k(x, y)$ we obtain different variants of hill climbing algorithms. When $R_k(x, y) = 0$ no move that will increase the value of the cost function is accepted. Therefore we obtain a local search or simple hill climbing algorithm which does not accept deteriorating moves. In this thesis we focus on the simulated annealing algorithm.

7.2.1 Simulated Annealing

The simulated annealing algorithm uses a key parameter called the temperature t_k. The configuration generation probability of a neighbor y of the current configuration x is defined as $g_k(x, y) = \frac{1}{|\eta(x)|}$, so it is uniform, i.e., each neighbor is equally likely to be picked at each state. The acceptance probability depends on the difference $f(y) - f(x)$ in the cost function between the current state x and the selected neighbor y and the current temperature t_k. The move is always accepted when it decreases the cost and accepted with probability $e^{-(f(y)-f(x))/t_k}$ when the cost increases. Hence the configuration acceptance probability is

$$\Pr[R_k(x, y) \geq f(y) - f(x)] = \min\{1, e^{-\frac{f(y)-f(x)}{t_k}}\}.$$

In terms of the general formulation presented above, we get this behavior when we define

$$R_k(x, y) = -t_k \ln(U),$$

where U is a uniform random variable on $[0, 1]$.

The way the "temperature" t_k of the system decreases over time k is called the cooling schedule. The condition necessary for the global convergence of the method

is that $t_k \geq 0$ and $\lim_{k\to\infty} t_k = 0$. In practice, the two most commonly used cooling schedules are the exponential cooling schedule

$$t_k = \alpha \cdot \beta^k$$

for some parameter $0 < \beta < 1$ and the logarithmic cooling schedule

$$t_k = \alpha / \log_2(k+1)$$

proposed in [52], where α is a constant corresponding to the starting temperature.

Simulated annealing was proposed by Kirkpatrick et al. [66] in 1983 and independently by Cerny [29] in 1983 for finding the global minimum of a cost function with several local minima and has received much attention in the following years because the algorithm often obtains good results for difficult problems. It was motivated by the analogy between the physical annealing of solids and combinatorial optimization problems [46]. Physical annealing is used to find a low energy state of a solid. This is analog to finding a good approximation for the global minimum of a combinatorial optimization problem. Physical annealing involves heating and controlled cooling of a material and works as follows. First the material is melted and then the temperature is slowly lowered so that the temperature is close to the freezing point of material for a long time. The heating frees the atoms and allows them to randomly move through states with higher energy. During the slow cooling they will probably find a configuration with a lower internal energy than the initial one. When the temperature is close to the freezing point the atoms can only take constitutions with lower energy than the previous one. In the analogy, the internal energy of the material corresponds to the value of the cost function and the constitution of the atoms in the material to the configuration of the combinatorial optimization problem. Furthermore the outer loop in Algorithm 5 corresponds to the current temperature, therefore the parameter t_k is called the temperature. Hence the inner loop corresponds to the time which is spent at the same temperature. We start with a high temperature which then decreases following a cooling schedule. Here we can again see the analogy to physical annealing. In the simulated annealing algorithms the new configuration $y \in \eta(x)$ is accepted if $U \leq e^{-(f(y)-f(x))/t_k}$ where U is uniformly distributed on $[0, 1]$. This means that moves which decrease the costs are always accepted, while moves that increase the costs are more likely to be accepted at the beginning of the process when the temperature is still high.

We mainly use a logarithmic cooling schedule

$$t_k = \alpha / \log(k \cdot M)$$

where α is the initial temperature and M the number of iterations for the inner loop. It is possible to use different cooling schedules or to define the acceptance probability differently. Several suggestions for different acceptance probabilities are made in [61] but as we use the classical acceptance probability we do not want to go into detail here. Further improvements are suggested in [46]. One of these is to save the best solution found so far. We incorporate this suggestion in our implementation of the simulated annealing algorithm. We furthermore introduce a new idea called nochangebound which works well for our special problem. More details are given in Chapter 10.

Chapter 8

Conversion Methods

The representation of Boolean functions as polynomials over the reals is of scientific interest and used in research fields such as Circuit Complexity [10]. This chapter gives an overview over the different representations of Boolean functions over the reals and integers. Firstly, the four most widely use representations and corresponding conversion methods are presented, these are the standard representation, the dual representation, the sign and the Fourier representation. We use the naming of [10]. Secondly, we introduce two related conversion methods which are specially tailored for the kind of Boolean equations which we consider in this thesis - namely Boolean equations in algebraic normal form. These methods are the adapted standard conversion and the integer adapted standard conversion.

We begin with the definition of a Boolean function.

Definition 8.1 (Boolean Function)**.**
A Boolean function is a mapping $f : B^k \to B$ *, where* $B = \{FALSE, TRUE\}$ *is a Boolean domain and* $k \in \mathbb{N}$*.*

In order to explicitly describe a Boolean function the operators from Boolean algebra are used. These are the complement $\neg$, the conjunction or AND-operator $\wedge$ and the disjunction or OR-operator $\vee$. We consider additionally the exclusive-or or the XOR-operator $\oplus$ because it is frequently used in the description of symmetric cryptographic schemes. The XOR-operator can be represented by the three previous operators in the following way $x \oplus y = (x \wedge \neg y) \vee (\neg x \wedge y)$.

For the analysis of Boolean functions or the application of different algorithms such as SAT-solvers [60] or an implementation of the conversion methods which will be presented later in this chapter it is convenient to represent Boolean functions in a normal form. The conjunctive normal form and the disjunctive normal form are two basic normal forms.

Definition 8.2 (Conjunctive Normal Form)**.**
A Boolean function f *is said to be in conjunctive normal form if it is a conjunction of clauses, where each clause is a disjunction of literals.*

$$\bigwedge_{I \subseteq M} \left(\bigvee_{i \in I} x_i \right)$$

where $M = \{1 \dots n\}$*.*

Definition 8.3 (Disjunctive Normal Form)**.**
A Boolean function f is said to be in disjunctive normal form if it is a disjunction of clauses, where each clause is a conjunction of literals.

$$\bigvee_{I \subseteq M} \left(\bigwedge_{i \in I} x_i \right)$$

where $M = \{1 \dots n\}$.

A third way to present a Boolean function is the algebraic normal form which uses XOR- and AND-operators.

Definition 8.4 (Algebraic Normal Form)**.**
A Boolean function f is said to be in algebraic normal form (ANF) if it is an XOR of clauses, where each clause is a conjunction of literals.

$$\bigoplus_{I \subseteq M} a_I \wedge \left(\bigwedge_{i \in I} x_i \right),$$

where $M = \{1 \dots n\}$ and $a_I \in \{FALSE, TRUE\}$.

A Boolean function in ANF is also called a Boolean polynomial due to the equivalence between Boolean functions in ANF and polynomials over $\mathbb{F}_2$. In cryptography the representation of a Boolean function as a polynomial over $\mathbb{F}_2$ is often used:

$$f(x_1, \dots, x_n) = \sum_{I \subseteq M} a_I \prod_{i \in I} x_i$$

where $M = \{1, \dots, n\}$ and $a_I, x_i \in \{0, 1\}$. In Section 8.2 we will introduce conversion methods which have been invented especially for Boolean equations in the algebraic normal form because this is the most frequently used representation of a Boolean function in cryptography. But prior to that we will define a couple of conversion methods which are applicable to all representations of Boolean functions.

8.1 Representations of Boolean Functions as Polynomials over the Reals

If we want to represent a Boolean function - no matter in which normal form it is given - as a polynomial over the reals we have to map the values FALSE and TRUE to real numbers. The representation of the Boolean operators follows from this mapping. The most natural representation is the standard representation.

Definition 8.5 (Standard Representation)**.**
Given a Boolean function $f(x_1, \dots, x_n)$ with $x_i \in \{FALSE, TRUE\}$ for $1 \leq i \leq n$ and a mapping $t : \{FALSE, TRUE\} \to \{0, 1\}$ such that

$$t(x) = \begin{cases} 0 & \text{if } x\text{=}FALSE \\ 1 & \text{if } x\text{=}TRUE, \end{cases}$$

r is the standard representation of f if

$$r(t(x_1), \ldots, t(x_n)) = t(f(x_1, \ldots, x_n))$$

holds for all possible configurations of $(x_1, \ldots, x_n)$.

This representation of FALSE and TRUE as real numbers leads to the polynomial expressions of the Boolean operators which is given in Lemma 8.6. A Boolean function can be converted recursively into a polynomial over the reals using these polynomial expressions under consideration of the distributive law in the Boolean algebra. This is true for all representations presented in this section.

Lemma 8.6. *(Standard Conversion Method)*
Let f be a Boolean function and r the corresponding standard representation. Then it holds

1. $f(x_1, x_2) = x_1 \wedge x_2 \quad \Longrightarrow \quad r(y_1, y_2) = y_1 y_2$
2. $f(x_1, x_2) = x_1 \vee x_2 \quad \Longrightarrow \quad r(y_1, y_2) = y_1 + y_2 - y_1 y_2$
3. $f(x_1, x_2) = x_1 \oplus x_2 \quad \Longrightarrow \quad r(y_1, y_2) = y_1 + y_2 - 2y_1 y_2$
4. $f(x_1) = \neg x_1 \quad \Longrightarrow \quad r(y_1) = 1 - y_1$

where $y_i = t(x_i)$ *for* $i = 1, 2$.

Lemma 8.6 can be proven by inspecting a truth table.

Swapping 0 and 1 in the mapping yields the dual representation.

Definition 8.7 (Dual Representation)**.**
Given a Boolean function $f(x_1, \ldots, x_n)$ *with* $x_i \in \{FALSE, TRUE\}$ *for* $1 \leq i \leq n$ *and a mapping* $t : \{FALSE, TRUE\} \to \{0, 1\}$ *such that*

$$t(x) = \begin{cases} 1 & \text{if } x\text{=}FALSE \\ 0 & \text{if } x\text{=}TRUE, \end{cases}$$

r is the dual representation of f if

$$r(t(x_1), \ldots, t(x_n)) = t(f(x_1, \ldots, x_n))$$

holds for all possible configurations of $(x_1, \ldots, x_n)$.

Lemma 8.8. *(Dual Conversion Method)*
Let f be a Boolean function and r the corresponding dual representation. Then it holds

1. $f(x_1, x_2) = x_1 \wedge x_2 \quad \Longrightarrow \quad r(y_1, y_2) = y_1 + y_2 - y_1 y_2$
2. $f(x_1, x_2) = x_1 \vee x_2 \quad \Longrightarrow \quad r(y_1, y_2) = y_1 y_2$

3. $f(x_1, x_2) = x_1 \oplus x_2 \quad \Longrightarrow \quad r(y_1, y_2) = 1 - y_1 - y_2 + 2y_1y_2$

4. $f(x_1) = \neg x_1 \quad \Longrightarrow \quad r(y_1) = 1 - y_1$

where $y_i = t(x_i)$ *for* $i = 1, 2$.

If we map TRUE and FALSE into the set $\{-1, 1\}$ this yields the sign and the Fourier representation and the corresponding conversion methods.

Definition 8.9 (Sign Representation)**.**
Given a Boolean function $f(x_1, \ldots, x_n)$ *with* $x_i \in \{FALSE, TRUE\}$ *for* $1 \leq i \leq n$ *and a mapping* $t : \{FALSE, TRUE\} \to \{0, 1\}$ *such that*

$$t(x) = \begin{cases} -1 & \text{if } x\text{=}FALSE \\ 1 & \text{if } x\text{=}TRUE, \end{cases}$$

r *is the sign representation of* f *if*

$$r(t(x_1), \ldots, t(x_n)) = t(f(x_1, \ldots, x_n))$$

holds for all possible configurations of $(x_1, \ldots, x_n)$.

Lemma 8.10. *(Sign Conversion Method)*
Let f *be a Boolean function and* r *the corresponding sign representation. Then it holds*

1. $f(x_1, x_2) = x_1 \wedge x_2 \quad \Longrightarrow \quad r(y_1, y_2) = \frac{1}{2}(y_1 + y_2 + y_1y_2 - 1)$

2. $f(x_1, x_2) = x_1 \vee x_2 \quad \Longrightarrow \quad r(y_1, y_2) = \frac{1}{2}(y_1 + y_2 - y_1y_2 + 1)$

3. $f(x_1, x_2) = x_1 \oplus x_2 \quad \Longrightarrow \quad r(y_1, y_2) = -y_1y_2$

4. $f(x_1) = \neg x_1 \quad \Longrightarrow \quad r(y_1) = -y_1$

where $y_i = t(x_i)$ *for* $i = 1, 2$.

We obtain the Fourier representation by exchanging -1 and 1 in the mapping.

Definition 8.11 (Fourier Representation)**.**
Given a Boolean function $f(x_1, \ldots, x_n)$ *with* $x_i \in \{FALSE, TRUE\}$ *for* $1 \leq i \leq n$ *and a mapping* $t : \{FALSE, TRUE\} \to \{0, 1\}$ *such that*

$$t(x) = \begin{cases} 1 & \text{if } x\text{=}FALSE \\ -1 & \text{if } x\text{=}TRUE, \end{cases}$$

r *is the Fourier representation of* f *if*

$$r(t(x_1), \ldots, t(x_n)) = t(f(x_1, \ldots, x_n))$$

holds for all possible configurations of $(x_1, \ldots, x_n)$.

Lemma 8.12. *(Fourier Conversion Method)*
Let f be a Boolean function and r the corresponding Fourier representation. Then it holds

1. $f(x_1, x_2) = x_1 \wedge x_2 \quad \Longrightarrow \quad r(y_1, y_2) = \frac{1}{2}(y_1 + y_2 - y_1 y_2 + 1)$
2. $f(x_1, x_2) = x_1 \vee x_2 \quad \Longrightarrow \quad r(y_1, y_2) = \frac{1}{2}(y_1 + y_2 + y_1 y_2 - 1)$
3. $f(x_1, x_2) = x_1 \oplus x_2 \quad \Longrightarrow \quad r(y_1, y_2) = y_1 y_2$
4. $f(x_1) = \neg x_1 \quad \Longrightarrow \quad r(y_1) = -y_1$

where $y_i = t(x_i)$ for $i = 1, 2$.

These four conversion methods have different properties. It depends on the application and on the Boolean function itself which of these is the best. The standard representation and the dual representation have the property that $x^2 = x$ while for the sign and the Fourier representation it holds that $x^2 = 1$. This ensures that the polynomials are automatically multilinear[1].

If we consider a Boolean function in ANF and convert it using any of the four representations, the total degree of the real-valued polynomial equals the number of variables contained in the Boolean function. The monomial degree, i.e., the number of monomials contained in the polynomial, varies for the different representations. We do not want to give a full analysis of the behavior of the different conversion methods with respect to the monomial degree. However, for a Boolean function in ANF the monomial degree seems to be largest using the dual representation. Whether the standard or the sign/Fourier representation provides a lower monomial degree depends on the structure of the special Boolean function e.g., on the number of the AND and XOR operators and the variable sharing of the monomials. If some of the clauses in the Boolean function share variables this might decrease the monomial degree because some of the monomials might be canceled out. If we consider a Boolean function in ANF without variable sharing, the monomial degree t the corresponding polynomial in standard representation is exponential in the number of clauses in the ANF. This means that if m is the number of clauses in the ANF then the monomial degree $mdeg(r)$ of the corresponding polynomial r is $mdeg(r) = 2^m - 1$. As a rule of thumb one can say that the standard conversion method is a better choice in terms of a low monomial degree if there are more ANDs than XORs and that the Fourier or sign conversion method yield a simpler result if the exclusive-or operator dominates in the ANF.

In general we are not restricted to map {FALSE,TRUE} to $\{0, 1\}$ or $\{-1, 1\}$. In the following we present a generalization of the representation of Boolean function as polynomials over the reals where TRUE and FALSE maps to $a, b \in \mathbb{R}$ [77].

[1] A function in several variables is said to be multilinear if it is linear in each variable seperately.

Definition 8.13 (Generalized Representation)**.**
Given a Boolean function $f(x_1, \ldots, x_n)$ *with* $x_i \in \{FALSE, TRUE\}$ *for* $1 \leq i \leq n$ *and a mapping* $t : \{FALSE, TRUE\} \to \{a, b\}$ *with* $a, b \in \mathbb{R}, a \neq b$ *such that*

$$t(x) = \begin{cases} a & \text{if } x\text{=}FALSE \\ b & \text{if } x\text{=}TRUE, \end{cases}$$

r *is the representation of* f *over the reals if*

$$r(t(x_1), \ldots, t(x_n)) = t(f(x_1, \ldots, x_n))$$

holds for all possible configurations of $(x_1, \ldots, x_n)$*.*

Using the generalized representation for TRUE and FALSE yields the following polynomial expressions for the Boolean operators over the reals.

Lemma 8.14. *(Generalized Conversion Method)*
Let f *be a Boolean function and* r *the corresponding representation over the reals. Then it holds*

1. $f(x_1, x_2) = x_1 \wedge x_2 \quad \Longrightarrow \quad r_\wedge(y_1, y_2) = \frac{1}{(b-a)^2}[ab^2 - a^2b + (a^2 - ab)(y_1 + y_2) + (b - a)y_1y_2]$
2. $f(x_1, x_2) = x_1 \vee x_2 \quad \Longrightarrow \quad r_\vee(y_1, y_2) = \frac{1}{(b-a)^2}[a^2b - ab^2 + (b^2 - ab)(y_1 + y_2) + (a - b)y_1y_2]$
3. $f(x_1, x_2) = x_1 \oplus x_2 \quad \Longrightarrow \quad r_\oplus(y_1, y_2) = \frac{1}{(b-a)^2}[a^3 - ab^2 + (b^2 - a^2)(y_1 + y_2) + 2(a - b)y_1y_2]$
4. $f(x_1) = \neg x_1 \quad \Longrightarrow \quad r_\neg(y_1) = a + b - y_1$

where $y_i = t(x_i)$ *for* $i = 1, 2$*.*

Lemma 8.14 captures all possible conversion methods. In particular we get the standard conversion for $a = 0$ and $b = 1$, the dual for $a = 1, b = 0$, the sign for $a = -1, b = 1$ and the Fourier representation for $a = 1$ and $b = -1$.

Proof. The generalized representation (Definition 8.13) is given. This means that the value a corresponds to FALSE and the value b to TRUE. We consider the truth tables for the Boolean operators AND, OR, XOR and define analog real-valued variants of the truth tables for the functions $r_\wedge, r_\vee, r_\oplus \in \mathbb{R}[y_1, y_2]$ as

y_1	y_2	$r_\wedge$	$r_\vee$	$r_\oplus$
a	a	a	a	a
a	b	a	b	b
b	a	a	b	b
b	b	b	b	a

This leaves us with a classical interpolation problem in two variables. We have to find polynomials that take the appropriate values defined by the above table in the points $(a,a),(a,b),(b,a)$ and (b,b). The interpolation polynomial can be written as

$$\begin{aligned} r(y_1,y_2) &= T(a,a)\frac{(b-y_1)(b-y_2)}{(b-a^2)^2} + T(a,b)\frac{(b-y_1)(-a+y_2)}{(b-a^2)^2} \\ &\quad + T(b,a)\frac{(-a+y_1)(b-y_2)}{(b-a^2)^2} + T(b,b)\frac{(-a+y_1)(-a+y_2)}{(b-a^2)^2} \end{aligned}$$

where $T(\cdot,\cdot)$ denotes the truth table entries corresponding to the Boolean function we want to represent. When we plug in the values from the table, we obtain the general polynomials $r_\wedge, r_\vee$ and $r_\oplus$ corresponding to AND, OR and XOR.

As a has to be the complement of b and vice versa $r_\neg$ is obviously the polynomial corresponding to the complement. □

All conversion methods in this section have in common that they increase the total as well as the monomial degree of the real-valued polynomial compared to the Boolean polynomial. The result is that linear Boolean polynomials are converted into highly non-linear polynomials over the reals and we loose the advantage of having linear equations when we want to solve an equation system. For our applications it is important to keep the total and the monomial degree of the polynomial over the reals as small as possible. In the next section we introduce two conversion methods which are designed especially for Boolean equation systems in ANF and meet the requirements of the application they are used in.

8.2 Conversion Methods for Boolean Functions in Algebraic Normal Form

In cryptanalysis a secret key or an internal state of a stream cipher can often be described as a system of non-linear Boolean equations in ANF. Solving this equation system then yields the secret key or the internal state and breaks the cipher. However, solving a non-linear Boolean equation system or equivalent a system of non-linear equations over $\mathbb{F}_2$ is in general an NP-hard problem. But exploiting the special structure of such a system might make it solvable in time faster than exhaustive key search. One idea, which is examined in this thesis, is to convert the Boolean equation system into an equation system over the reals and solve it with optimization tools. We can roughly say that the simpler the equations and the lower their degree the more likely it is that an optimization algorithm will successfully solve the system. In the previous section we have seen conversion methods which recursively convert Boolean operators into polynomial expressions. The main problem of these methods is the high increase in the total and monomial degree of the equations. In order to minimize this effect the so-called adapted standard conversion was introduced [87]. The adapted standard conversion method does not convert the Boolean operators one by one but considers the equation as a whole. This has the advantage that the basic

structure of the Boolean equation is kept in the equation over the reals, in particular the monomial degree stays the same and the total degree increases by at most one. The drawback of this approach is that the number of variables as well as the number of equations increases.

As in the standard representation also in the adapted standard conversion the Boolean values FALSE and TRUE are mapped to the real values 0 and 1 such that

$$t(x) = \begin{cases} 0 & \text{if } x = \text{FALSE} \\ 1 & \text{if } x = \text{TRUE}, \end{cases}$$

where $t : \{\text{FALSE,TRUE}\} \to \{0, 1\}$.

We clarify how the adapted standard conversion methods works using the following example:

$$x_1 \oplus x_2 \oplus (x_3 \wedge x_4) = FALSE. \tag{8.1}$$

Now we consider this equation as an equation over the reals where we replace the conjunction by multiplication and the exclusive-or by addition over $\mathbb{R}$

$$y_1 + y_2 + y_3 y_4 = 0. \tag{8.2}$$

where $y_i = t(x_i)$ for $i = 1 \ldots 4$. We determine the truth table for the LHS of equation (8.1) and evaluate the LHS of equation (8.2) for all possible configurations of $(y_1, y_2, y_3, y_4) \in \{0, 1\}^4$.

y_1	y_2	y_3	y_4	(8.1)	(8.2)	y_1	y_2	y_3	y_4	(8.1)	(8.2)
0	0	0	0	FALSE	0	0	0	0	1	FALSE	0
0	0	1	0	FALSE	0	0	1	0	0	TRUE	1
1	0	0	0	TRUE	1	0	0	1	1	TRUE	1
0	1	0	1	TRUE	1	1	0	0	1	TRUE	1
0	1	1	0	TRUE	1	1	0	1	0	TRUE	1
1	1	0	0	FALSE	2	0	1	1	1	FALSE	2
1	0	1	1	FALSE	2	1	1	0	1	FALSE	2
1	1	1	0	FALSE	2	1	1	1	1	TRUE	3

From this table we can see that for all possible configurations of y there are four possible outcomes of the LHS of (8.2), namely 0,1,2 and 3. Because we want to find a configuration such that the LHS of the Boolean equation (8.1) is FALSE we only have to consider the rows in the table where the outcome of the Boolean function is FALSE. The possible real values for these rows are 0 and 2 (or in a more general case all real-valued outcomes which are equal to 0 modulo 2). The converted equations over the reals should cover both of the cases. Therefore one real-valued equation for each of these values is generated by keeping the original structure of the equation and subtracting the outcome value from it. We obtain

$$\begin{aligned} y_1 + y_2 + y_3 y_4 &= 0 \\ y_1 + y_2 + y_3 y_4 - 2 &= 0, \end{aligned}$$

where $y_i = t(x_i)$ for $i = 1 \dots 4$. Obviously only one of the above equations can be true at the same time. Therefore we multiply the LHS of the equations with new variables, exclusion variables, and add an additional constraint which guarantees that exactly one of the exclusion variables is 1 and so exactly one of the above equations is satisfied. This yields the following adapted standard representation for the Boolean equation (8.1)

$$\begin{aligned} e_1(y_1 + y_2 + y_3y_4) &= 0 \\ e_2(y_1 + y_2 + y_3y_4 - 2) &= 0 \\ e_1 + e_2 &= 1, \end{aligned}$$

with $e_i \in \mathbb{R}$ for $i = 1, 2$. In the case of a linear equation with just two variables the adapted standard conversion yields a linear equation over the reals. The form of the polynomial depends on the RHS of the equation. We consider the Boolean equation

$$x_1 \oplus x_2 = z \text{ where } z \in \{FALSE, TRUE\}.$$

In the case $z =$ FALSE the resulting equation over the reals is $y_1 - y_2 = 0$ and in the case $z =$ TRUE the adapted standard conversion yields $y_1 + y_2 - 1 = 0$. This conversion can be extended to Boolean polynomial with two monomials.

As mentioned before a drawback of the adapted standard conversion method is the increase in the number of variables and equations and also the increase of the total degree by one if the equation contains more than two monomials, which means we loose linearity.

We can improve this conversion method when we require that the polynomial equations only hold over the integers instead of over the reals. In Chapter 9 we will use the conversion methods in order to apply mixed-integer programming to a Boolean equation system that describes the internal state of Bivium. Therefore the requirement that the converted equations hold over the integers is strong enough. This leads to the integer adapted standard conversion method [28].

8.2.1 The Integer Adapted Standard Conversion Method

The main idea of how to convert a Boolean equation into an equation over the integers is taken from the adapted standard conversion. We consider a Boolean equation where the RHS equals FALSE and interpret it as an equation over the integers by replacing XOR by addition and AND by multiplication. Then we evaluate the integer equation for all solutions of the Boolean equation as we did in the ASC method (Here we map TRUE to 1 and FALSE to 0). Afterwards we subtract these results from the LHS of the equation. We observe that all results are multiples of 2. This means that instead of generating several equations and introducing exclusion variables we introduce a single new integer-valued variable in a linear term and get one equation.

As an example we consider the Boolean equation

$$x_1 \oplus x_2 \oplus (x_3 \wedge x_4) \oplus x_5 \oplus x_6 = \text{FALSE} \tag{8.3}$$

If we evaluate the corresponding real-valued polynomial $y_1 + y_2 + y_3y_4 + y_5 + y_6$ for all solutions of (8.3) we get $0, 2, 4$ as results. That means that a solution of (8.3) is a solution to the following equation over the integers

$$y_1 + y_2 + y_3y_4 + y_5 + y_6 - 2y_{\text{Int}} = 0$$

where $y_{\text{Int}} \in \{0, 1, 2\}$ and $y_i \in \{0, 1\}$ for $i = 1 \dots 6$. The degree is the same and the number of variables and monomials per equation is increased only by one. We call this conversion method Integer Adapted Standard Conversion (IASC).

For an equation which only contains two monomials we can use the same trick as in the adapted standard conversion and convert it depending on the RHS of the Boolean equation such that we keep the total degree without introducing a new variable to the equation.

We will compare performance of the standard and the integer adapted standard conversion methods in the context of mixed-integer linear programming in Chapter 9.

Chapter 9

Bivium as a Mixed Integer Programming Problem

Trivium is one of the three stream ciphers that were recommended in the eSTREAM project portfolio in the hardware category. In Chapter 3, Subsection 3.4.5 we showed that the internal state of Trivium and also of its small-scale variants Bivium A and B can be described by a system of sparse non-linear Boolean equations. Solving this equation system is equivalent to recovering the internal state of the cipher. It is known that solving a random system of non-linear Boolean equations is an NP-hard problem. However, the equation systems generated by the ciphers Trivium and Bivium are not random. They are very sparse and contain a lot of structure and mixed-integer optimization methods may therefore be suitable for solving them.

In this chapter we present an attack on the Biviums using mixed-integer optimization methods. The main idea is to transform the problem of solving the sparse system of quadratic equations over $\mathbb{F}_2$ into a combinatorial optimization problem. We convert the Boolean equation system into an equation system over $\mathbb{R}$ and formulate the problem of finding a 0-1-valued solution for the system as a mixed-integer programming problem. This enables us to make use of the well-developed commercial optimization tool Cplex [4] in order to find a solution for the problem and recover the initial state of Bivium. In particular this gives us an attack on Bivium B in estimated time complexity of $2^{63.7}$ seconds.

We explain our approach of conversion and formulation of a mixed-integer programming problem on the simplest of the three variants of Trivium, named Bivium A. We recall that for each clocking of the algorithm we get three equations of the form

$$s_{162} \oplus s_{177} = z \tag{9.1}$$

$$s_{66} \oplus s_{93} \oplus (s_{91} \wedge s_{92}) \oplus s_{171} \oplus s_{178} = 0 \tag{9.2}$$

$$s_{162} \oplus s_{177} \oplus (s_{175} \wedge s_{176}) \oplus s_{69} \oplus s_{179} = 0 \tag{9.3}$$

After 177 clockings of the algorithm we have a fully determined system of 399 equations in 399 unknowns. For each further clocking we get an overdetermined system because we obtain three new equations and introduce two new variables [90] (see also Chapter 3).

9.1 Bivium A as a Mixed-Integer Linear Programming Problem

In this section we will explain step by step how to model Bivium A as a mixed-integer linear programming problem. But first we want to recap the definition of a mixed-integer programming problem given in Section 7.1.

A mixed-integer linear programming problem (MIP) is a problem of the form

$$\min_x \{c^T x | Ax \leq b, x \in \mathbb{Z}^k \times \mathbb{R}^l\}$$

where c is an n-vector, A is an $m \times n$-matrix ($n = k+l$) and b is an m-vector. However, in the sequel we will mostly be concerned with the feasible set

$$S = \{x \in \mathbb{Z}^k \times \mathbb{R}^l, Ax \leq b\}$$

which is the set of all points that satisfy the integrality restrictions and the linear constraints.

We will use the equations of the form (9.1) - (9.3) which describe the initial state to define the feasible set of the mixed-integer problem. This means that we formulate the problem of finding the internal state of Bivium A given a sufficiently long part of the keystream as a feasibility problem rather than as an optimization problem. As mentioned in Section 7.1.1 the modeling of the MIP is important. In the following subsections we will describe two different ideas of how to model Bivium A as an MIP. One model is based on the standard conversion method (Definition 8.5 and Lemma 8.6) and the other on the integer adapted standard conversion method (IASC) (Section 8.2.1). Afterwards we will compare the results of the different models.

9.1.1 Linearization Using the Standard Conversion Method

Using the standard conversion method the degree of the real-valued equation is equal to the number of variables involved in the Boolean equation. We want to keep the degree of the equation as small as possible because later we have to replace terms of higher order by new variables in order to get linear constraints. We also want to keep the number of monomials per equation small because the less complex the constraints are, the more likely it is that we can solve the MIP. Therefore the first step is to split up the equations by introducing new auxiliary variables.

Splitting

First we replace $s_{162} \oplus s_{177}$ by the keystream bit z in the quadratic equation. This is a good opportunity to reduce the number of variables in the equation and the complexity of the resulting constraint. This is different from the case of Bivium B and Trivium because of the more complex keystream equation. We introduce new variables in such a way that there are at most four variables per equation. Using a small trick of rewriting the equations we can achieve that the equations over the

reals have at most degree two after applying the standard conversion method. If we consider the two equations

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0 \tag{9.4}$$
$$x_1 \oplus x_2 = x_3 \oplus x_4 \tag{9.5}$$

then we see that both equations have the same set of solutions. However applying the standard conversion method to Equation (9.4) yields a real-valued polynomial of degree four containing 15 monomials while Equation (9.5) is converted into a real-valued polynomial equation of degree two containing six monomials.

We split and rewrite the system in the following way:

$$\begin{aligned} s_{162} \oplus s_{177} &= z \\ r_1 &= s_{66} \oplus s_{93} \\ r_2 &= s_{91} \wedge s_{92} \\ r_1 \oplus r_2 &= s_{171} \oplus s_{178} \\ r_3 &= s_{175} \wedge s_{176} \\ z \oplus r_3 &= s_{69} \oplus s_{179} \end{aligned}$$

This means we introduce three splitting variables for each clocking (except for the last 66 clockings). Starting from a fully determined system with 399 equations this splitting yields a system of 732 equations in 732 unknowns. The equations are still Boolean. Hence the next step is to convert the system of Boolean equations into a system over the reals.

Conversion Using the Standard Conversion Method

The only requirement is that a solution of the Boolean system is also a solution of the equation system over the reals. We can ignore additional non-binary solutions of the real system. Thus it does not matter whether we convert each side of the equation and subtract the resulting terms afterwards or write the equation of the form $p(x) = TRUE/FALSE$ and convert the Boolean polynomial $p(x)$ as a whole into a polynomial over the reals using the standard conversion method. The advantage of splitting the equation into two parts is that the degree of the resulting polynomial is at most two as explained above. This yields equations over the reals of the following form:

$$\begin{aligned} x_{162} + x_{177} - 2x_{162}x_{177} &= z \\ \tilde{r}_1 - x_{66} - x_{93} + 2x_{66}x_{93} &= 0 \\ \tilde{r}_2 - x_{91}x_{92} &= 0 \\ \tilde{r}_1 + \tilde{r}_2 - 2\tilde{r}_1\tilde{r}_2 - x_{171} - x_{178} + 2x_{171}x_{178} &= 0 \\ \tilde{r}_3 - x_{175}x_{176} &= 0 \\ (1-2z)\tilde{r}_3 - x_{69} - x_{179} + 2x_{69}x_{179} &= -z \end{aligned}$$

where $x_i = t(s_i)$ and $\tilde{r_i} = t(r_i)$ and t is the mapping that maps a Boolean variable to a real variable (cf. Definition 8.5).
We still have quadratic equations. All these constraints are equality constraints. The last step is to replace the quadratic terms by new variables which will be forced to take the correct values by additional constraints.

Linearization

In order to linearize the equations we introduce a new binary variable y for each quadratic term $x_i x_j$. We want this new variable to be zero if x_i or x_j is zero and to be one if and only if x_i and x_j are both one. We can achieve this by adding the following inequalities to the system of constraints:

$$y \leq x_i, \tag{9.6}$$
$$y \leq x_j, \tag{9.7}$$
$$x_i + x_j - 1 \leq y. \tag{9.8}$$

The inequality constraints (9.6) and (9.7) make sure that $y = 0$ if $x_i = 0$ or $x_j = 0$ while (9.8) ensures that $y = 1$ if $x_i = 1$ and $x_j = 1$. We introduce one new variable and three inequality constraints for each quadratic term. The equality constraints are of the form

$$\begin{aligned} x_{162} + x_{177} - 2y_1 &= z \\ \tilde{r}_1 - x_{66} - x_{93} + 2y_2 &= 0 \\ \tilde{r}_2 - y_3 &= 0 \\ \tilde{r}_1 + \tilde{r}_2 - 2y_4 - x_{171} - x_{178} + 2y_5 &= 0 \\ \tilde{r}_3 - y_6 &= 0 \\ (1-2z)\tilde{r}_3 - x_{69} - x_{179} + 2y_7 &= -z \end{aligned}$$

This linearization yields a system of 732 equality and 2529 inequality constraints in 1575 variables (again starting from a fully determined system with 399 variables in 399 unknowns).

9.1.2 Linearization Using the Integer Adapted Standard Conversion Method

In this linearization method we use the fact that we want to use the resulting system of equalities and inequalities in a mixed-integer programming problem. This means that we can restrict some (or all) variables to be integers. Consequently, when we convert a Boolean equation, we do not have to ensure that the equation holds over the reals but over the integers. Hence, we can use the integer adapted standard conversion to convert the Boolean equations into integer equations.

It is assumed that the smaller the number of monomials per equation is, the easier it is to solve the resulting mixed-integer prgramming problem. For this reason

we start by splitting the equations by introducing auxiliary variables. We introduce $t_1 = s_{66} \oplus s_{93}$ and also replace $s_{162} \oplus s_{177}$ by the keystream bit z in (9.3). We apply the standard conversion method to equations with no more than three monomials. In this case the number of monomials per equation and the number of new variables is the same as when using the IASC. But by replacing a quadratic term by a new variable we get three constraints instead of only the restriction that the variable is binary. That means we get stronger constraints by using the standard conversion in these cases. For equations with more than three monomials we use the IASC. This yields equations of the form:

$$\begin{aligned} x_{162} + x_{177} - 2x_{162}x_{177} &= z \\ t_1 - x_{66} - x_{93} + 2x_{66}x_{93} &= 0 \\ t_1 + x_{91}x_{92} + x_{171} + x_{178} - 2y_1 &= 0 \\ x_{175}x_{176} + x_{69} + x_{179} - 2y_2 &= z \end{aligned}$$

where $y_1 \in \{0, 1, 2\}$ and $y_2 \in \{0, 1\}$. Finally we replace the quadratic terms by new variables and add the corresponding inequality constraints (9.6)-(9.8) (as in Section 9.1.1.)

$$\begin{aligned} x_{162} + x_{177} - 2y_3 &= z \\ t_1 - x_{66} - x_{93} + 2y_4 &= 0 \\ t_1 + y_5 + x_{171} + x_{178} - 2y_1 &= 0 \\ y_6 + x_{69} + x_{179} - 2y_2 &= z \end{aligned}$$

This means for each clocking of the algorithm we introduce one new variable for splitting, two new integer-valued variables for the integer adapted standard conversion and four new binary variables for the linearization. As $y_2 \in \{0, 1\}$ actually six of these variables are binary and only one is an integer. We introduce three variables less per clocking this way than when using the standard conversion method. Starting from a fully determined Boolean system this yields a system of 2040 equalities and inequalities in 1020 variables.

9.1.3 Bivium A as a Feasibility Problem

From the linearization we obtain a system of linear equalities and inequalities over the reals and integers, respectively, which describes the internal state of Bivium A. Since the number of variables exceeds the number of equality constraints we cannot use Gauss elimination to solve the system. Therefore we use these constraints to describe the feasible set of a mixed-integer linear programming problem. Let

$$\min_x \{c^T x : Ax \leq b, x \in \{0, 1\}^{k_1} \times \mathbb{Z}^{k_2} \times \mathbb{R}^l\} \tag{9.9}$$

be the mixed-integer programming problem describing Bivium A where the matrix A and the vector b are given by the constraints obtained from converting and linearizing the Boolean equations of Bivium A and by the upper and lower bounds on the

variables. If we can find a feasible binary/integer-valued solution for this MIP for an arbitrary objective function, this solution can be converted into a solution for the Boolean system. Hence, it is not important to find a minimal solution but a feasible point. It is possible that more than one feasible point exists, it is also possible that the Boolean system does not have a unique solution. But it is likely that an overdetermined Boolean system has a unique solution and the corresponding feasible region contains only one element.

Furthermore, we can observe that most of the variables are dependent on the initial state variables (all except the variables introduced by IASC in Section 9.1.2). This means that we do not have to restrict all variables to be integer or binary. It is sufficient to force the initial state variables to be binary (and the IASC-variables to be integers).

In the next section we will discuss which function might be appropriate as objective function, how many additional keystream equations we should generate to obtain an overdetermined system and which variables should be restricted to be binary or integers. Furthermore, we will compare the two different linearization methods.

9.2 Results

In this section we present our observations and results from experiments with various variants of Bivium A and Bivium B as mixed-integer programming problems. We focus mainly on MIPs that use the constraints obtained using the standard conversion method and linearization. We consider some variants of Bivium A with a smaller state size, see Table 9.1, to compare the increase of the solution time to the number of variables. We also tried to find the optimal number of additional keystream equations, a good objective function and the variables which should be restricted to be binary. Later we will compare this approach to MIPs using constraints obtained by the IASC.

For all experiments we use the commercial linear optimization tool CPLEX by ILOG [4]. CPLEX has a user's choice for emphasis on feasibility or optimality. We choose emphasis on feasibility because we are not interested in optimality and stop after we find the first solution because we assume that there is only one element in the feasible set. We use CPLEX version 9.130 on a Sun Fire E25K SF 12K with shared processors. On this machine 2^{24} Bivium simulations over 5×177 steps take $2^{14.5}$ seconds. This means we can approximately search through $2^{9.5}$ keys per second.

9.2.1 Parameters Using the Standard Conversion Method

For finding good parameters (objective function, binary variables, additional equations) we ran most of the tests for the smallest variant Bivium A Step 1. This variant has internal state size of 118 bits and keeps the structure of Bivium A as much as possible. The pseudo code of Bivium A Step 1 is given in Figure 9.2.1. We confirmed these parameters for the variants with larger state size by running some spot tests.

Table 9.1: Variants of Bivium A with smaller state size.

Name	Step 1	Step 2	Step 3	Bivium A
state size	118	133	147	177
keystream bits required	158	177	196	236
variables	1530	1718	1894	2283
equalities	728	817	901	1086
constraints	3254	3652	4027	4854

$$
\begin{aligned}
t_1 &\leftarrow s_{44} + s_{62} \\
t_2 &\leftarrow s_{108} + s_{118} \\
z &\leftarrow t_1 + t_2 \\
t_1 &\leftarrow t_1 + s_{60}s_{61} + s_{114} \\
t_2 &\leftarrow t_2 + s_{116}s_{117} + s_{46} \\
(s_1, s_2, \ldots, s_{62}) &\leftarrow (t_2, s_1, \ldots, s_{61}) \\
(s_{63}, s_{64}, \ldots, s_{118}) &\leftarrow (t_1, s_{63}, \ldots, s_{117})
\end{aligned}
$$

Figure 9.1: Pseudo code Bivium A Step 1.

Binary vs. Continuous

Using the standard conversion method to convert the Boolean equations into real equations, all newly introduced variables (the newly introduced state variables, as well as the auxiliary variables and the linearization variables) are dependent on the initial state variables. This means that forcing the initial variables to take binary values will also force all other variables to take binary values. This raises the question, whether it is an advantage to have binary restrictions only for the initial state variables instead of for all. A branch-and-bound algorithm determines a variable for branching in each step. In the binary case this variable will be assigned 0 and 1 to grow the search tree further in one node. Intuition tells us that it is better to choose variables for branching which automatically force many other variables to take binary values. This can be achieved by only restricting these variables to be binary or by giving the algorithm an order in which it should consider the variables. We can confirm our intuition by experiments, see Table 9.2.

Table 9.2 shows the running time for 10 typical instances of the problem. An 80 bit key is chosen at random to generate a fully determined Boolean equation system. This system is used to formulate a mixed-integer linear programming problem as explained before. As objective function the sum over all variables is used. The last two rows of the upper table in Table 9.2 give the average and the standard deviation taken over a larger sample. The lower table contains the average and standard deviation over a sample where the Boolean system is overdetermined. We can see that the

Table 9.2: Comparison of running time in seconds for Bivium A Step 1 between the MIP with binary restriction on the initial state and on all variables. The two last rows of the upper table contain the average running time in seconds and the standard deviation for a sample using a fully determined system. The lower table contains the average running time and the standard deviation for a sample using an overdetermined system.

No.	mixed	binary
1	385	1583
2	550	890
3	170	478
4	157	277
5	629	1297
6	42	27
7	1209	620
8	213	1011
9	256	286
10	484	1979
average	**548**	783
std	393	563

	mixed	binary
average	**493**	930
std	272	289

average running time is cut by 30% to almost 50% when we compare the case where all variables are restricted to be binary to the case where only the initial state variables have binary restrictions. Even though the running times vary a lot for both cases the standard deviation is smaller for the mixed problems. Furthermore, we note a decrease of the standard deviation when we consider overdetermined problems. Considering these improvements it seems to be a good strategy to restrict only the initial state variables to be binary. All other variables are continuous variables.

Objective Function

As mentioned before, Bivium A is formulated as a feasibility problem and the objective function does not influence the solution in a way that matters for us. That means we can choose an arbitrary objective function. Even if the objective function is not important to get the correct solution of the problem it is important for the performance of many mixed-integer algorithms. In branch-and-bound algorithms the bounding function estimates the best value of the objective function obtainable by growing the tree one node further. This value is an important factor in the process of choosing the next node in the search tree. The closer the value of the bounding function to the objective function the better.

The only restriction for the objective function we have is that it must be linear.

Natural choices are

1. the zero-function,
2. the sum over the initial state variables,
3. the sum over all variables which are introduced by the original Boolean system i.e., the sum over all state variables,
4. the sum over all variables,
5. the sum over all but the initial state variables.

Already a few tests showed that the zero-function is not a good objective function in practice. This is not surprising because the constant zero-function gives the algorithm no information and does not show any improvement. We focused on the four remaining candidates for the objective function. Our experiments show, see Table 9.3, that we achieve the best running time if we use the sum over all variables as the objective function. Here again the first 10 rows contain the running times of 10 typical instances of the mixed-integer programming problem we get from a randomly chosen key and the corresponding overdetermined Boolean equation system. In our formulation of the MIP only the initial state variables are restricted to be binary. The last two rows contain the average running time and standard deviation taken over a larger sample. As mentioned before, the sum over all variables turns out to be the best objective function in this sample and is used in all further experiments. But the 5th candidate, the sum over all non-state variables, is almost as good and we expect that these two functions will yield similar results because the functions have similar properties.

For a random point we expect that approximately half of the bits are zero and half of the bits are one. This is different when we consider a solution for our equation system generated by a randomly chosen key. For a key chosen at random we expect that the state variables behave like a random sequence, meaning that half of the initial state variables as well as the updated state bits are zero. Otherwise this would exhibit a severe weakness of the cipher. The same holds for the splitting variables which represent an XOR sum of two state variables. However some of the splitting variables and all variables which are introduced in the linearization process represent a product of two binary variables. Thus we can expect a bias in the distribution of ones and zeros in these variables. A product of two binary variables is one if both binary variables are set to one. Hence, when we consider the variables in the solution which represent products of binary variables we can expect that only one quarter of them are one. This means for a key chosen at random approximately one third of the variables take the value one.

Using the sum of certain variables as objective function means looking for a solution where the Hamming weight of these variables is minimal. For the second and third candidates the expected value of the objective function for the solution is the same as for a random point. But for the fourth candidate, the sum over all variables, the expected value for the correct solution is only one third of the number of variables

Table 9.3: Comparison of running time in seconds of Bivium A Step 1 as MIP with different objective functions. The first 10 rows contain the running time in seconds corresponding to the different objective functions. The row labeled with average shows the average run time in seconds and the last row gives us the standard deviation. Both values are calculated based on a larger sample.

No.	$\sum_{i=1}^{118} x_i$	$\sum_{i=1}^{270} x_i$	$\sum_{i=1}^{\#variables} x_i$	$\sum_{i=118}^{\#variables} x_i$
1	510	365	262	202
2	477	635	198	270
3	180	1325	791	723
4	495	931	621	512
5	627	1151	411	168
6	145	506	244	180
7	544	342	939	394
8	387	1057	416	542
9	613	527	641	493
10	972	779	136	382
average	573	678	**449**	463
std	290	320	232	286

Table 9.4: Running times in seconds of Bivium A for instances with low Hamming weight (All values are averages over samples of size 5.).

HW initial state	HW solution	time in sec
10	209.4	0.2
20	346.8	17.2
40	496.8	780.3
60	617.0	9113.6
80	657.4	9342.0
100	770.9	29135.7
120	868.8	27777.7

which is significantly less than half of the number of variables which is the expected value for a random point. Here minimizing the objective function leads us in the right direction. Also the last candidate has the property that the correct solution has a lower Hamming weight than a random point.

The observation that the sum over all variables is the best objective function raises the question if the MIP is solved faster if the Hamming weight of the solution is low. We tested this on an overdetermined system of Bivium A (59 additional keystream equations and the corresponding quadratic equations) and our tests show that problems with a low weight solution are in average solved much faster, see Table 9.4. The hypothesis of our experiments is:
The sum over all variables $\sum_{i=1}^{m} x_i$ where m is the number of variables, is a good choice for the objective function.

Table 9.5: Average running times over 10 test cases for overdetermined systems.

Bivium A Step 1

add. keystream bits	+5	+10	+20	+40	+59
average time in sec	743	548	719	**438**	2063

Bivium A Step 2

add. keystream bits	+10	+40	+44	+67
average time in sec	3493	2446	**2209**	3767

Bivium A

add. keystream bits	+10	+59	+89
average time in sec	25759	**15309**	21950

We use this objective function in the remaining experiments.

Overdetermined System

The fully determined non-linear Boolean equation system (meaning n equations in n unknowns where $n = 399$ in the case of Bivium A) has possibly more than one solution. For an overdetermined system it is likely that the solution is unique. Fortunately, it is very easy to generate an overdetermined Boolean system for Bivium. After we have generated a fully determined system each additional keystream bit gives us three equations and two new variables. We believe that as little as 5 to 10 additional keystream bits are already sufficient to get a unique solution in most cases. On the one hand, the advantage of adding even more equations is that the feasible set will get smaller and hence the algorithm will become faster. On the other hand more constraints come into play which will slow down the algorithm. The question is how many additional keystream equations and corresponding quadratic Boolean equations we should add. Here the hypothesis of our experiments is, see Table 9.5:
Generate one third more keystream equations and the corresponding quadratic equations to define the system.

9.2.2 Results on Bivium A Using Standard Conversion

We ran tests for all variants of Bivium A where the key was chosen at random. We used the following settings which are good according to our earlier experiments:

- The objective function is the sum over all variables.
- We obtain an overdetermined system by generating one third additional keystream equations.
- We restrict the initial state variables to be binary, all other variables are continuous.

Table 9.6: Overview over Bivium A with different state size.

Name	Step 1	Step 2	Step 3	Bivium A
state size	118	133	147	177
keystream bits required	158	177	196	236
variables	1530	1718	1894	2283
equalities	728	817	901	1086
constraints	3254	3652	4027	4854
time in sec.	555	2110	2535	15267

We summarize the results in Table 9.6 where the last row contains the average running time taken over a samples ranging from 10 to 30 instances.

We are able to break Bivium A in less than 4.5 hours on average. This shows us that our approach is faster than Raddum's [90] (about a day) but slower than using MiniSAT [80] (reported to be 21 sec).

9.2.3 Results on Bivium B Using the Standard Conversion

In the same manner as for Bivium A we can convert Bivium B into a mixed-integer programming problem. We use the same settings as we did for Bivium A, i.e., the objective function is the sum over all variables, we generate an overdetermined system by adding 59 additional keystream and corresponding state update equations, and we restrict only the first 177 variables to be binary. This yields an MIP in 2821 variables and 5865 constraints, 1388 of these are equality constraints. We start by considering variants of Bivium B with smaller state sizes of 59 and 118 bits respectively.

Table 9.7: Running time in seconds of Bivium B and its variants with smaller state sizes.

Bivium B state size 59	Bivium B state size 118	Bivium B full size
$2^{10} \cdot 2^{11.5}$	$2^{30} \cdot 2^{12.6}$	$2^{50} \cdot 2^{13.7}$

Initial experiments showed that Cplex was not able to solve even the small instances of Bivium B as a mixed-integer programming problem in a reasonable running time. As an example we compared Bivium B with a state size of 118 bits to Bivium A Step 3 which has a state size of 147 bits. The associated MIPs approximately correspond to each other in terms of the number of variables and constraints. A comparision of the problem size is given in Table 9.8. While the MIP corresponding to Bivium A Step 1 can be solved in approximately 42 minutes on average, no solution was found for the MIP which was derived from the small-scale variant of Bivium B after a about a day. Thus, we reduced the complexity of the problem by guessing bits. When we run all experiments in parallel the expected running time of the algorithm

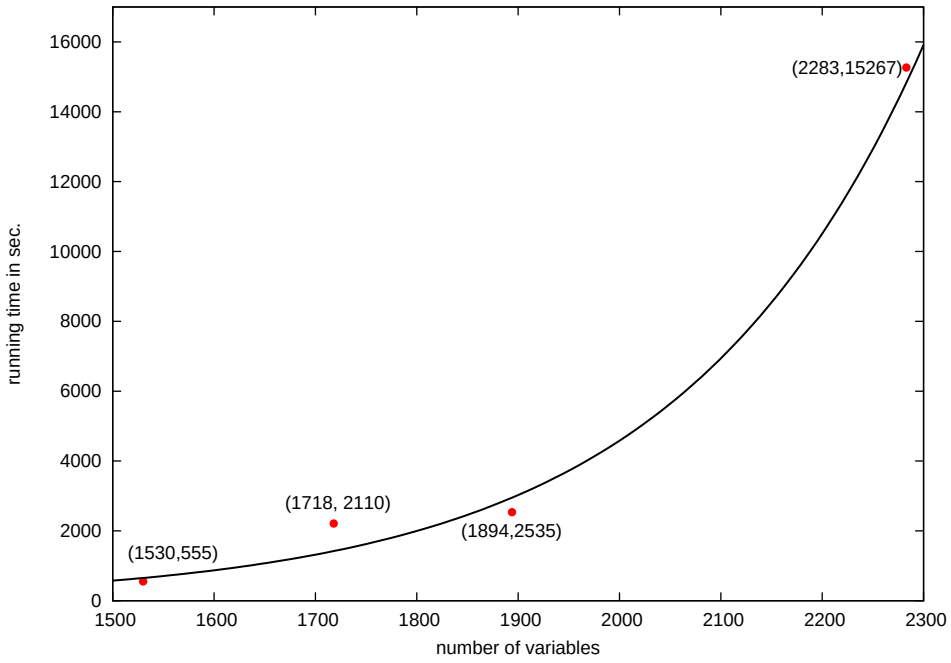


Figure 9.2: Running time for Bivium A with different state sizes (exponential compensating curve).

is the time it needs to find the solution for the correct guess times 2^n where n is the number of guessed bits. Inspired by [48] we tried two different guessing strategies. We guessed bits at the beginning or at the end of the initial state. It turned out that guessing the last bits is a better strategy. For the variant of Bivium B with internal state size 118 the average time to find the solution after we guessed the first 30 bits correct was 31635 sec while the average time for guessing the last 30 bits correct was just 6334 sec. Here we have an improvement by a factor of five. For that reason we will guess the last bits of the initial state in all further experiments. We can also see that for a variant of Bivium B with state size 118 we have to guess 30 variables. This means the problem has a high complexity.

Table 9.11 shows the results for Bivium B when we guess 50 or 55 bits. We can see that even though the running time of the optimization algorithm increases, the overall complexity decreases when we guess less bits. However, we can also observe that the variance of the results on Bivium B with 50 bits guessed is quite high. So far there is no explanation for the big differences in the running time but a general observation is that the fewer variables we fix the higher the standard deviation is.

We can determine the initial state for Bivium B in $2^{63.7}$ seconds (assuming that we run the tests for all guesses in parallel). Our simulations showed that we can search through 2^{24} keys in $2^{14.5}$ seconds. That means that the complexity of our approach corresponds to searching through $2^{73.2}$ keys. This is not a very impressive result

Table 9.8: Comparison Bivium A Step 3 and a small-scale variant of Bivium B.

Name	Bivium A Step 3	Bivium B (small scale)
state size	147	118
keystream bits required	196	158
variables	1894	1890
equalities	901	930
constraints	4027	3930

Table 9.9: Running time in seconds for some tests on Bivium A using IASC.

no.	+10 keystream bits	+59 keystream bits
1	243241	50395
2	211305	27853
3	6572	8912
4	35296	12545
5	9966	6760
6	29650	39950
7	52230	10596
8	183083	1050
9	130254	111693
av	100180	**27652**

compared to the result using Raddum splitting algorithm [90] which takes 2^{56} seconds corresponding to a search through $2^{69.3}$ keys or the even better result using MiniSAT [80] which takes $2^{42.7}$ seconds, corresponding to a search through 2^{56} keys, to attack Bivium. But it shows that mixed-integer programming has a certain potential for use in cryptanalysis.

9.2.4 Results Using the Integer Adapted Standard Conversion

Our second approach involves a different conversion method which leads to a different model of the state recovery problem of Bivium as a mixed-integer linear programming problem. The important difference are the constraints defining the feasible set. We describe the feasible set of the MIP by the constraints we get from using the conversion and linearization presented in Section 9.1.2. As objective function we use the sum over all but the variables introduced by the IASC, i.e., we sum over all binary variables. We restrict the initial state variables to be binary and the variables introduced by the IASC to their integer values. In Table 9.9 we can see that the running time for this approach is worse than for the approach using the standard conversion method even though using the IASC leads to a mixed-integer programming problem with less variables and constraints than the MIP which is obtained using the standard conversion method. (The dimensions of the two different MIPs are compared in Table 9.10.) One reason

Table 9.10: Parameters of the two approaches for Bivium A.

Method	IASC	SC
state size	177	177
variables	1773	2283
equalities	746	1086
constraints	2984	4854
variables with restrictions	517	177

for that might be that the parameters which yield good results in our first approach are not optimal for the approach using the IASC. Another more likely reason is that we have more variables with restrictions in the approach using the IASC, nearly three times as many as when using the standard conversion approach. This means that the algorithm has to consider more variables when it chooses the next branching variable. Some of these variables are not binary but integer-valued. That means there are even more possibilities to fulfill this equation and therefore the constraint is weaker. However, this approach is still interesting when we want to derive an MIP from a more complex equation system such as Bivium B or small-scale variants of AES. Here the lower complexity of the resulting constraints could be an advantage even if we get more variables with restrictions.

9.3 Possible Improvements through Extra Constraints

Fixing some variables to their correct values reduces the number of variables of the problem. Another possibility to simplify the problem is to reduce the size of the feasible region by adding additional constraints.

9.3.1 AND-gate Constraints

In the previous section we have seen that a mixed-integer programming problem corresponding to Bivium B is more difficult to solve than one corresponding to Bivium A. Thus, we have to simplify the problem and decrease its complexity. Our first idea to reduce the size of the feasible set is to use what we call AND-gate constraints. These are constraints which can easily be derived from a given instance of Bivium. The additional constraints are generated in the following way.

The quadratic Boolean update functions contain a quadratic term (an AND-gate) which we replace by a new variable during the linearization process. These AND-gates contain successive variables. This means if one AND-gate or one of the variables which represent the AND-gate is zero then also the following or the prior AND-gate/variable must be zero. This leads to the following deterministic constraint:

$$r_1 + r_3 - r_2 \leq 1$$

where $r_1 = x_1x_2, r_2 = x_2x_3$ and $r_3 = x_3x_4$ are variables that represent AND-gates.

Table 9.11: Timing results on Bivium B in seconds (*including cost for bit guessing). The numbers in the first row indicate how many bits are preassigned, 'AND-Gate' means we added the AND-Gate constraints. The rows labeled with 1 to 10 contain the run time in seconds excluding the cost for bit guessing.

	50 given	55 given	55+ And-gate
1	1073	104	382
2	1325	550	591
3	213	27	1711
4	97592	3818	742
5	79935	213	2194
6	31311	1308	1667
7	7642	452	2623
8	486	949	566
9	745	364	1734
10	8434	434	851
average *	$2^{63.7}$	$2^{65.2}$	$2^{65.5}$

Table 9.11 lists the running times for 10 instances of the problem as well as the average running time which is taken over a larger sample. As we can see the additional constraints do not improve the running time, actually they impair it. One reason could be that we increase the number of constraints too much, maybe adding just a few new constraints would improve the running time. However, we did not run tests to investigate this.

9.3.2 CNF Constraints

Another possibility is to add constraints which we derive from the conjunctive normal form of the Boolean equation describing the internal state of Bivium [86]. Given a Boolean function in algebraic normal form it is rather simple to convert it into conjunctive normal form using Algorithm 6. The size of the expression converted from ANF to CNF may be exponential.

Given a Boolean equation of the form

$$f(x_1, \ldots, x_n) = a$$

where f is a Boolean polynomial and $a \in \{FALSE, TRUE\}$ we convert

- $f(x_1, \ldots, x_n)$ into CNF if $a = TRUE$ and
- $\neg f(x_1, \ldots, x_n)$ into CNF if $a = FALSE$.

Note that the equation is satisfied if the corresponding term in CNF is true.

Considering a Boolean equation in conjunctive normal form we introduce a new inequality constraint for each clause. Each negated Boolean variable is converted into a negative integer variable and the OR operator is replaced by the addition over the

Algorithm 6 Converting ANF into CNF.

Require: f is a Boolean function in ANF

1: Eliminate exclusive-or $\oplus$ using the identity

$$a \oplus b \Leftrightarrow (a \vee b) \wedge (\neg a \vee \neg b)$$

2: Eliminate negation $\neg$ using DeMorgan's law

$$\neg(a \vee b) \Leftrightarrow \neg a \wedge \neg b$$

$$\neg(a \wedge b) \Leftrightarrow \neg a \vee \neg b$$

3: Distribute disjunction $\vee$ over conjunction $\wedge$

$$a \vee (b \wedge c) \Leftrightarrow (a \vee b) \wedge (a \vee c)$$

reals. The left-hand side of the inequality has to be chosen such that the clause is satisfied for all assignments of the variables to the values $\{0, 1\}$ for which the inequality holds. Given the following Boolean equation in CNF

$$(x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) = 1$$

yields the following two constraints

$$\begin{aligned} x + y + z &\geq 1 \\ -x + y - z &\geq -1 \end{aligned}$$

We focus on the keystream equation and add the constraints we obtain from the CNF of the keystream equation of Bivium to the corresponding MIP. The keystream equation of Bivium B in CNF and the resulting inequality constraints are given in the Appendix C. We performed limited tests on Bivium B with additional constraints where we considered three different approaches:

- We add all additional inequalities we can obtain from the keystream equations.
- We only add very few additional inequalities (e.g from the first 5-10 keystream equations).
- We add the inequality constraints which we obtain from the CNF of the keystream equations from the first 66 clockings. These are the keystream equations which only contain initial state bits.

The results are summarized in Table 9.12. The limited number of experiments suggest that using a few additional inequalities, which are derived by looking at the CNF, decreases the complexity of the attack. However, more tests are necessary to draw a firm conclusion.

Table 9.12: The complexity in seconds for solving an instance of the MIP corresponding to Bivium B where additional constraints obtained from the CNF of the keystream equations are used. The system is overdetermined by 59 additional clockings and the last 55 variables of the initial state are set to their correct values. The last row of the table contains the average complexity over the 10 instances.

	reference (no add. constraints)	all CNF constraints	few CNF constraints	CNF constraints of the first 66 equations
1	124	2731	57	393
3	318	594	703	261
4	2135	1613	942	4492
5	426	208	189	212
6	462	692	643	279
7	770	763	925	442
8	1450	697	283	3040
9	1061	1896	1227	1154
10	2011	3376	269	194
av.	973	1396	**582**	1163

9.4 Conclusion

We showed two ways of transforming Bivium into a mixed-integer linear programming problem. One way uses the standard conversion method to convert Boolean equations into equations over $\mathbb{R}$, the other way uses the integer adapted standard conversion to convert the Boolean equations into equations over $\mathbb{Z}$. These two methods are also applicable to any other Boolean equation system.

The best results for Bivium A are achieved by using the standard conversion method with an estimated time complexity of $2^{13.9}$ seconds or approximately 4.5 hours. Using the integer adapted standard conversion we get an attack with running time of $2^{14.76}$ seconds or approximately 8 hours. Solving the MIP which corresponds to state recovery of Bivium B converted using the standard conversion has an average time complexity of $2^{63.7}$ seconds which corresponds to a search through $2^{73.2}$ keys. Comparing the results on variants of Bivium B with a smaller state size to Bivium A shows that not only the increased number of variables in Bivium B but also the more complex structure of the equations are responsible for the increase of the running time. Moving from Bivium B to Trivium would double the number of variables in the corresponding MIP. Also, the keystream equation of Trivium involves six variables and hence is more complicated than the keystream equation of Bivium B. Therefore the mixed-integer programming in its current form as presented in this chapter can not be considered as a threat for Trivium.

Chapter 10

Hill Climbing Algorithms and Trivium

In this chapter we propose a new method to solve a certain class of systems of multivariate equations over the binary field. We show how heuristic optimization methods such as hill climbing algorithms and in particular simulated annealing can be relevant for solving such equation systems. A characteristic of equation systems that may be efficiently solvable by means of such algorithms is provided. As an example, we investigate equations induced by the problem of recovering the internal state of the stream cipher Trivium. We propose an improved variant of the simulated annealing method that seems to be well-suited for this type of equations and provide some experimental results.

The motivation behind this approach is that many cryptographic schemes can be represented as a system of multivariate non-linear equations, in such a way that solving this equation system recovers the secret key or initial state. This fact is exploited by the so-called algebraic attacks, which have received much attention in the recent years. In general, solving random systems of multivariate non-linear Boolean equations is an NP-hard problem [51]. However, when the system has a specific structure, we can hope that more efficient methods exist.

The method we propose here applies to sparse equation systems. The important additional requirement we make is that each variable appears only in a very limited number of equations. The equation system generated by the keystream generation algorithm of the stream cipher Trivium [43] satisfies those properties and will be examined in this paper as our main example. Our approach considers the problem of finding a solution for the system as an optimization problem and then applies an improved variant of simulated annealing to it. Simulated annealing works well for many difficult problem in combinatorial optimization.

The use of simulated annealing in the context of cryptology is not new. An attack on an identification scheme based on the permuted perceptron problem (PPP) was presented in [69]. An appropriate cost function was found which made it possible to solve the simpler perceptron problem as well as the PPP using a simulated annealing search. The attack showed that the recommended smallest parameters for the identification scheme are not secure. The same identification scheme was later subjected to an improved attack [32]. Simulated annealing was used to solve a related problem that had solutions highly correlated with the solution of the actual problem. Furthermore, timing analysis was applied where the search process was monitored and one could observe that some variables were stuck at correct values at an early state and never

changed again. These variables got stuck at their correct values, thus parts of the solution could be determined even if no complete solution was found. The two main properties that make an equation system suitable for simulated annealing are

1. sparsity of the equations,
2. locality of the variables, meaning that each variable only appears in a limited number of equations.

We will show that Trivium satisfies these properties and present an improved version of the simulated annealing algorithm.

10.1 Trivium as a Discrete Optimization Problem

To begin with we recall the discrete optimization problem (Definition 7.7). Given a finite set X of configurations we want to minimize the cost function. The cost function maps each configuration to a non-negative cost value. We want to find the configuration that yields minimal costs. In order to apply generalized hill climbing algorithms such as simulated annealing we additionally have to define a neighborhood function, which describes a set of neighbors for a given configuration $x \in X$.

We know that the state recovery problem of Trivium can be described as a system of 954 equations in 954 unknowns where the equations are of the form:

$$\begin{aligned} s_{66} \oplus s_{93} \oplus s_{91} \cdot s_{92} \oplus s_{171} &= s_{289} \\ s_{162} \oplus s_{177} \oplus s_{175} \cdot s_{176} \oplus s_{264} &= s_{290} \\ s_{243} \oplus s_{288} \oplus s_{286} \cdot s_{287} \oplus s_{69} &= s_{291} \\ s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288} &= z \end{aligned} \tag{10.1}$$

Furthermore, we know that the initial state bits together with the corresponding updated state bits satisfy all the generated equations (10.1). On the other hand, for a random point each equation is satisfied with probability $\frac{1}{2}$. It is obvious that a random point satisfies the linear equation with probability $\frac{1}{2}$. A quadratic equation is satisfied if the quadratic term and the linear part have the same value. In the Trivium system each variable appears at most once per equation. Therefore the probabilities for the quadratic term and the linear part of the equation are independent. Hence the probability that a random point fulfills a quadratic equation of the Trivium system is $\Pr[\text{quadratic term} = 0] \cdot \Pr[\text{linear part} = 0] + \Pr[\text{quadratic term} = 1] \cdot \Pr[\text{linear part} = 1] = \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{2}$.

If we consider the problem of solving the Trivium equation system as a discrete optimization problem which is suitable for hill climbing algorithms (cf. Section 7.2) the set of all possible assignments of the 954 variables $X = \{0,1\}^{954}$ is the set of possible configurations. As a cost function $f : X \to \mathbb{R}^+$ we count the number of not satisfied equations in the system. We know that the minimum of the cost function is 0 and that the initial state of the Trivium system is a configuration for which the cost function is minimal. There might be other optimal solutions, however, it is easy

to check whether the solution we found is the desired one. If a configuration is an optimal solution for the discrete optimization problem, it generates the same first 288 bits of keystream as the initial state we are looking for. But it is unlikely that the keystream will be the same for the following keystream bits. Therefore we can check whether a solution is the desired one by observing a few more keystream bits and comparing them to the keystream generated by the solution. In our experiments it is unlikely that multiple solutions occur because we set some of the variables to their correct values and therefore consider a highly overdetermined equation system.

In addition to the configuration set and the cost function we have to define a neighborhood function in order to use simulated annealing to solve the discrete optimization problem formulated above. We move from the current configuration x to a neighbor $y \in \eta(x)$ by flipping one bit.

$$\eta(x) = \{y \in X | \, \mathrm{hw}(x \oplus y) = 1\}$$

where hw denotes the hamming weight. That means every configuration has 954 neighbors, which is the smallest possible neighborhood. To sum up the state recovery problem of Trivium can be formulated as a discrete optimization problem defined as:

- $X = \{0,1\}^{954}$ is the configuration set.
- The cost function is the integer sum of the outcome of equations in the system evaluated at point $x \in X$ or in other words the costs are the number of not satisfied equations for the current configuration of the system.
- The neighborhood of a configuration $x \in X$ is defined as

$$\eta(x) = \{y \in X | hw(x \oplus y) = 1\}.$$

In the next section we analyze some of the properties of Trivium.

10.2 Properties of Trivium

Hill climbing algorithms are sensitive to the way the cost function changes when moving between configurations. The best results are obtained when a move from a configuration $x \in X$ to one of the neighbors $y \in \eta(x)$ does not change the value of the cost function too much.

In our case we move from one configuration to another by flipping the value of a single variable. Each variable appears in at most 8 equations and in 6 equations on average, that means that we have a great locality of the variables. Thus, when moving to a neighbor of the current configuration the cost function will change by at most 8. Furthermore, each variable appears only once in an equation. Therefore changing the value of a single variable will change the value of the equation with probability 1 if the variable appears in a linear term and with probability $\frac{1}{2}$ if the variable appears in a quadratic term. In the latter case flipping the value of a variable will just change the outcome of the equation if the other variable in the quadratic term is assigned to '1'. If a variable appears in the maximum of eight equations it appears

Table 10.1: Change of the cost function when moving to a neighbor configuration: The first row denotes the number of preassigned bits we use to simulate different distances from the minimum. We count how often out of 10000 trials the cost function changes by 0 to 8 units. The last row gives us the average change of the cost function.

i	0	100	200	300	400	500	600	700	800	900	954
0	1714	1702	1685	1560	1309	1052	944	767	601	264	0
1	3253	3246	3297	3158	2641	2143	1856	1550	1120	389	34
2	2248	2235	2240	2241	1930	1720	1385	1172	937	1001	1062
3	1557	1571	1550	1659	1821	1757	1488	1278	1258	1515	1537
4	675	665	668	754	1024	1020	911	810	741	596	648
5	386	400	380	409	691	940	1088	1078	1024	1068	1160
6	127	128	130	164	409	916	1372	1630	1866	2002	2049
7	32	44	41	46	165	439	837	1352	1854	2297	2534
8	8	9	9	9	10	13	119	363	599	868	976
average	1.81	1.824	1.814	1.9	2.32	2.83	3.32	3.85	4.38	4.97	5.3

in two equations in the quadratic term only. The expected number of equations which change their outcome is 7. Additionally it is unlikely that flipping the value of a variable changes the outcome of all equations which contain this variable in the same direction or respectively it is unlikely that all equations which contain the variable have the same outcome for this configuration before the flip. (The case that a lot or even all equations have the same outcome will appear with higher probability the closer we are to the minimum.)

From these observations we infer that even if we move from a configuration x to one of its neighbors by flipping the value of a variable which appears in 8 equations we do not expect that the value of the cost function changes by 8 in almost all of the cases.

We confirmed this by the following experiment. We generated a Trivium system for a random key and calculated the cost function for a random starting point. Then we chose a neighbor configuration of our starting point and recorded the absolute value of the change in the cost function. To simulate being close to the minimum we set a number of bits to the correct solution but we allowed those bits to be flipped to move to a neighboring configuration. The results are summarized in Table 10.1.

10.3 Solving the Trivium Systems with Modified Simulated Annealing

The properties of the Trivium equation system suggest that it is possible to employ heuristic search methods such as simulated annealing to find a global minimum and thus recover the internal state of the cipher. We have the advantage of knowing that the minimum value is zero and therefore we have a stop criterion for the search.

Initial experiments with standard simulated annealing were not very encouraging.

As we knew the correct solution of the system we could observe that even after a long running time the algorithms would not approach the optimal solution. We simplified the problem by assigning some of the variables their correct values. To be able to solve the Trivium system in reasonable time, we needed to simplify the initial system by setting around 600 out of 954 variables to their correct values throughout the search.

The introduction of a parameter called nochangebound showed a significant improvement over the standard algorithms. The algorithm works as follows. As with standard simulated annealing, we randomly generate a neighbor. If the cost decreases, we accept this move. If not, instead of accepting the move with a probability related to the current temperature, we pick another neighbor and test that one. If after testing a certain number of neighbors we cannot find any cost decreasing move, we accept the deteriorating move with a certain probability, just as in the plain simulated annealing. The parameter nochangebound of this algorithm is the number of additional candidates to test before accepting a cost increase.

If the nochangebound is zero we get the plain simulated annealing, on the other hand, if we test all possible neighbors before accepting a deteriorating move, we obtain an algorithm that is equivalent to local search. We look for any possible decreasing move and follow it. That means, when we find a local minimum the algorithm gets stuck and enters a loop. After trying all possible neighbors the algorithm will finally accept the costs of an increasing candidate but in the next step we will always move back to the local minimum we found. Setting the parameter between those two extremes yields an intermediate algorithm.

In practice, we used a probabilistic variant of this approach that randomly selects neighbors until it finds one with smaller cost or it exceeds the number of tests specified as nochangebound. This algorithm is presented in Algorithm 7.

The proper choice of the nochangebound is critical for the efficiency of the simulated annealing variant. The relationship between the number of neighbors tested and the time it took to find a solution (measured in the number of neighbors tested) is presented in Fig 10.1. Values of nochangebound below 25 result in running times exceeding 2^{40} flips. It suggests that the nochangebound must not be too small.

10.4 Experimental Results

In this section we report results of our computational experiments with the basic equation system generated by the problem of recovering the internal state of Trivium. We took the fully determined system with 954 equations and 954 variables obtained after observing 288 bits of the keystream.

We made some comparisons between exponential and logarithmic cooling schedules (see Section 7.2 for details) and from our limited experience the logarithmic cooling schedule performed better in more cases, so we decided to pick that one for our further tests.

After a few trials we decided to use $\alpha = 35$ as the initial temperature. Too large α resulted in prolonged periods of almost-random walks where there was no clear sign of progress in terms of decrease of the cost function. Too small α caused the algorithm

Algorithm 7 Modified version of simulated annealing.

```
x_best ← x
T ← α                                          ▷ initial temperature parameter is α
k ← 0                                          ▷ set the outer loop counter
while T > 1 do
    for m = 0, ..., M − 1 do                   ▷ parameter M is the number of inner runs
        generate a neighbor y ∈ η(x) uniformly
        if f(y) < f(x) then                    ▷ if cost decreased
            x ← y                              ▷ accept the move
            if f(x) < f(x_best) then           ▷ found a new best value
                x_best ← x                     ▷ store the best configuration
                nc ← 0                         ▷ reset the neighbor counter
                if f(x_best) = 0 then          ▷ if we found a solution
                    return x_best              ▷ finish and return it
                end if
            end if
        else                                   ▷ the candidate cost is higher
            nc ← nc + 1
            if (nc > nochangebound) ∧ (exp((f(y) − f(x))/T > rnd[0, 1]) then
                x ← y                          ▷ accept the move
                nc ← 0                         ▷ reset the counter of tested neighbors
            end if
        end if
    end for
    k ← k + 1
    T = α/ log_2(k · M)                        ▷ Logarithmic cooling schedule
end while
```

to behave like a local search algorithm when the process was getting stuck in some shallow local minimum.

In order to simplify the problem we assigned some of the variables their correct values. For each number of bits of the state fixed to their correct values (preassigned) we ran ten identical tests with different random seeds testing various values of nochangebound parameter (from the set 100, 150, 175, 200, 250, 300). After the test batch finished, we picked the value of nochangebound that yielded the lowest search time. We managed to obtain optimal values of the parameter nochangebound for 200, 195, 190, 185, 180, 175 and 170 preassigned bits where we set the values of the first bits of the internal state. We used this optimal nochangebound to estimate the total complexity of the attack. The graph is presented in Fig. 10.2. The total complexity is the product of the number of guesses we would need to make ($2^{preassigned}$) multiplied by the experimentally obtained running time of the search for the solution. We take the complexity for the correct guess. For a wrong guess the algorithm will not find a solution with costs 0 and thus not terminate.

Figure 10.3 shows that the increase in the running time for the search procedure is less than the decrease in the costs for bit guessing, which means that the total running time for the attack decreases when we reduce the number of preassigned bits (cf. Figure 10.2).

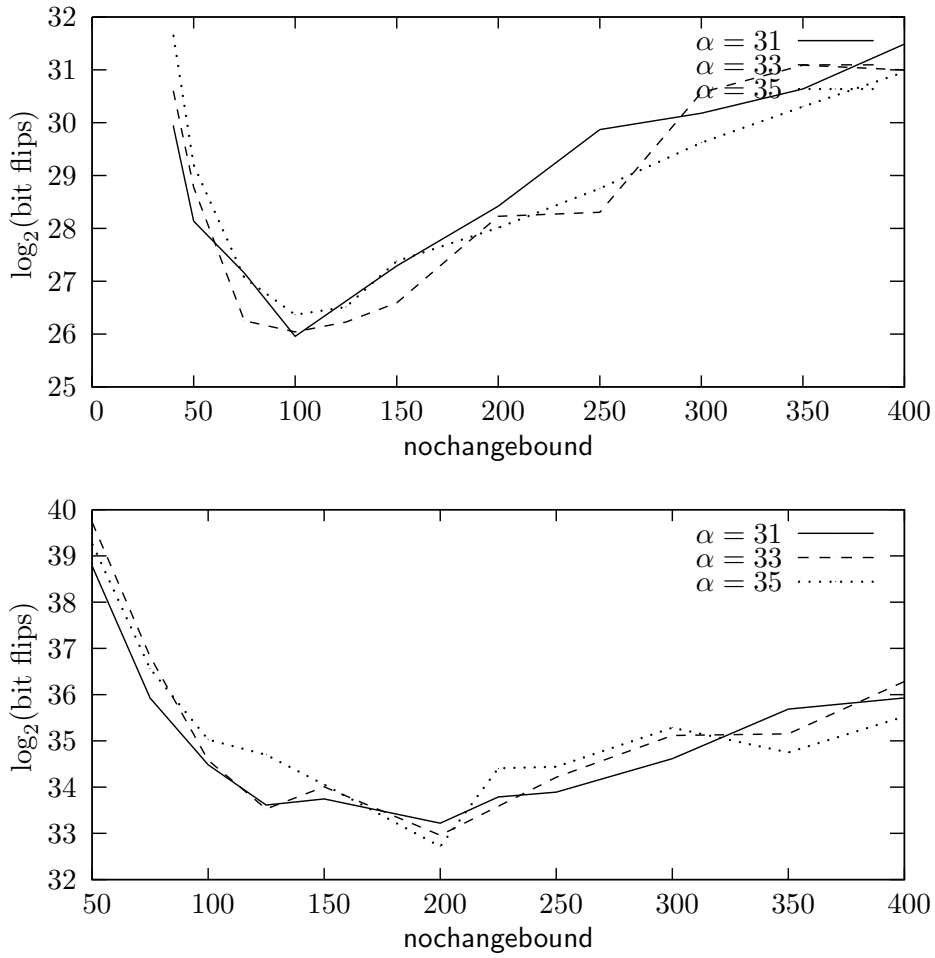


Figure 10.1: Influence of the nochangebound parameter on the efficiency of simulated annealing applied to the basic Trivium system for three values of initial temperature α. Other parameters are $M = 1024$ (cf. Alg. 7), averages are over 10 tests. In the top figure we guessed 200 first bits of the state, in the bottom one 180 bits.

If the curve goes down below the complexity level corresponding to 2^{80} key setups of Trivium, it would constitute a state-recovery attack. However, the problem is that due to limited computational power we were not able to gather enough results for values of *preassigned* smaller than 170. Our program running on 1.1GHz AMD Opteron 2354 was able to compute 2^{35} bit flips per hour and tests with $preassigned = 170$ required around $2^{38} \sim 2^{39}$ bit flips.

It seems that trying to extrapolate the running times is rather risky, since we do not have any analytical explanation of the complexities we get as is often the case with heuristic search methods. Applications of simulated annealing to other problems have shown that we might hit a wall for a certain problem size, meaning that the

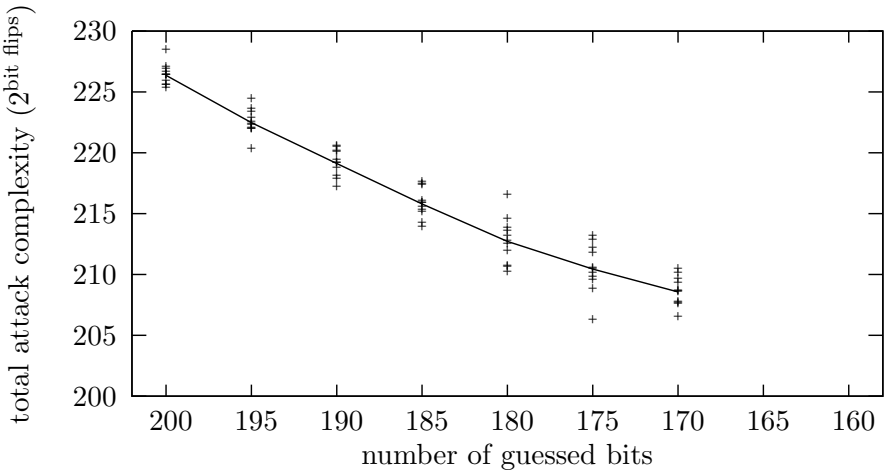


Figure 10.2: Running times of the attack based on modified simulated annealing depending on the number of guessed bits. The numbers on the vertical axis are base two logarithms of the total number of moves necessary to find the solution. Crosses represent results of single experiments, the line connects averages.

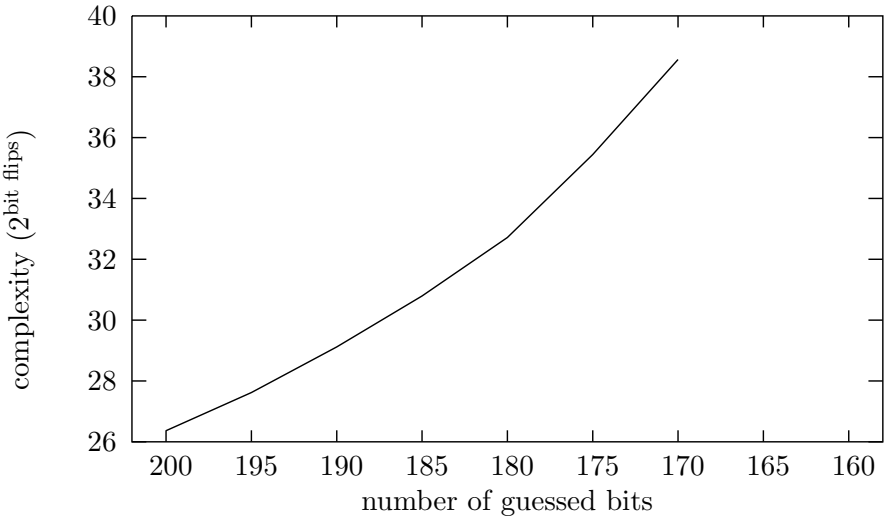


Figure 10.3: Running time on average of modified simulated annealing depending on the number of guessed bits. The running time does not include the time for bit guessing. The numbers on the vertical axis are base two logarithms of the number of moves necessary to find the solution given that the number of bits on the horizontal axis are preassigned.

problem is not solvable anymore instead of that it scales as expected. Therefore we do not claim anything about the feasibility of such an attack on the full Trivium. We can only conjecture that there might be a set of parameters for which such an attack may become possible.

Due to the computational complexity, our experimental results are so far based on only rather small samples of runs for the fixed set of parameters. Therefore, they cannot be taken as a rigorous statistical analysis but rather as a reconnaissance of the feasibility of this approach. However, we have noticed that for an overwhelming fraction of all the experiments, the running times for different runs with the same set of parameters do not deviate from the average exponent of the bit flips by more than ± 2, i.e., most of the experiments have the number of flips between 2^{avg-2} and 2^{avg+2}. Therefore, we believe that the results give some reasonable impression of the actual situation.

10.5 Some Variations

The previous section presented the set of our basic experiments. However, there is a multitude of possible variations of the basic setup which could possibly lead to better results.

10.5.1 Guessing Strategies

In order to lower the complexity of solving the equation system we set some of the variables to their correct values. However, the search complexity depends on which variables we choose.

We used different guessing strategies for pre-assigning variables and compared the influence on the running time of our algorithm. We used the following strategies to guess subsets of the state bits:

1. Select the first variables of the initial state.
2. Select the first variables of each register of the initial state.
3. Select the last variables of the initial state.
4. Select the last variables of each register of the initial state.
5. Select the most frequent variables. These are the variables which are introduced by the update function at the beginning of the keystream generation. We guess values for variable s_{289} and the consecutive ones in this case.
6. An adaptive pre-assignment strategy which is similar to the ThreeFour strategy in [47] (see Subsection 10.5.1).
7. Select the variables in such a way that the equation interdependence measure is minimal. (see Subsection 10.5.1).

Table 10.2: Running time for different pre-assignment strategies. nochangebound=110, 190 bits are preassigned, average taken over ten runs.

	first bits of the initial state	most frequent bits	first bit of every register	last bit of the initial state	last bit of each register
1	28.8	35.2	36.3	31.6	36.1
2	31.2	33.3	35.1	31.7	37.7
3	30.5	30.9	35.3	31.7	37.0
4	28.1	33.7	35.3	32.3	37.2
5	30.7	32.1	30.9	31.1	35.8
6	28.4	35.2	32.7	32.0	33.7
7	28.5	33.7	37.5	29.7	38.9
8	29.9	33.5	35.4	32.3	37.5
9	27.4	30.8	33.2	30.9	33.0
10	31.3	30.9	33.8	28.3	37.1
average	**29.5**	33.0	34.5	31.2	36.4

It turns out that the best guessing strategy of the ones we tested is to guess the first bits of the initial state. In addition to a pre-assignment of variables we can determine the value of further variables by considering the linear and quadratic equations (see below). We use this technique in the adaptive pre-assignment strategy. However we tested this strategy using the nochangebound which has been determined for the number of guess bits and the optimal nochangebound for the number of actually fixed, pre-assigned bits might be different.

The Adaptive Pre-assignment Strategy

In this pre-assignment strategy we use the fact that assigning 5 of the variables in a linear equation will uniquely determine the 6th variable. Starting with an arbitrary linear equation we guess and pre-assign 5 of the 6 variables, determine the value of the remaining variable and assign this to its value. We know that a large fraction of the variables appear in two linear equations. So in the next round of pre-assignment we pick an equation in which at least one variable is already assigned. That means we only have to guess at most 4 variable to get one for free. We continue until we have made the maximum number of guesses or we cannot find an equation in which one variable is already assigned. In the latter case we just have to pick an equation without preassigned variables and run the algorithm again until we made the maximum number of guesses.

The advantage of this pre-assignment strategy is that we can assign much more variables than we actually have to guess. Table 10.3 gives us an impression of this advantage.

The disadvantage is that instead of making the equations sparser we fix some equations to be zero. This means that there are less equations left which contain free

variables but the maximum number of equations in which a variables appears is still 8. Therefore a variable influences a higher percentage of equations.

Additionally to this adaptive pre-assignment strategy we can also use the quadratic equations to determine the value of variables.

If we want to use a linear equation to determine the value of a variable the values of 5 of the 6 variables must be known. Then the value of the remaining variable is unique. If there are less than 5 variables assigned we cannot say anything about the remaining variables. These are '0' or '1' with probability $\frac{1}{2}$, independent of the values of the assigned variables. If we use a quadratic equation to determine the value of unassigned variables we have to look at different cases. A quadratic equation contains 6 variables, 4 of them appear in linear terms and two in the quadratic term. We have to consider how many variables are already assigned, at which positions they are, and what value they have:

- 5 variables in the equation are preassigned
  - If both of the variables in the quadratic term are preassigned we can determine the value of the remaining variable.
  - If only one of the variables in the quadratic term is preassigned we distinguish between the following two cases
    1. The preassigned variable in the quadratic term has the value '1'. Then we can determine the value of the remaining variable in the quadratic term.
    2. The preassigned variable in the quadratic term has the value '0'. Then we cannot make a statement on the value of the remaining variable.
- 4 variables in the equation are preassigned.
  - If both variables in the quadratic term are preassigned no further assignment of other variables in the equation is possible
  - If one variable in the quadratic term is preassigned we have to distinguish between the following two cases
    1. The preassigned variable in the quadratic term is '0'. Then we can determine the value of the variable which is unassigned and appears

Table 10.3: The table shows how many bits additional to the guessed bits can be assigned using the adaptive pre-assignment strategy.

# guessed bits	# assigned bits	additional assigned bits in %
5	6	20%
50	66	32%
100	135	35%
200	281	40.5%

in a linear term but we cannot determine the second variable in the quadratic term.

2. The preassigned variable in the quadratic term is '1'. Then no further assignment is possible.

– None of the variables in the quadratic term is preassigned i.e all variables of the linear terms are preassigned. In this case we can set both variables in the quadratic term to '1' if and only if the sum of the variables which appear in the linear terms is 1.

This enables us to significantly increase the number of assigned variables after we pre-assigned a sufficient number of bits. Unfortunately, it does not improve the complexity of the simulated annealing search.

Minimizing Equation Interdependency

If all the equations used different sets of variables, it would be trivial to solve the system by a simple local search. However, variables appear in many equations and changing the value of one of them influences other equations at the same time. This suggests the idea of guessing (pre-assigning) bits to minimize the number of variables shared by many equations and thus reduce the degree of mutual relationships between equations.

Capturing this intuition more formally, let E_i be an equation and let $\mathcal{V}(E_i)$ denote the set of *not preassigned* variables that appear in the equation. We can define the measure of interdependence of two equations E_i, E_j as

$$IntrDep(E_i, E_j) = |\mathcal{V}(E_i) \cap \mathcal{V}(E_j)|.$$

If the measure is zero, the equations use different variables and we can call them separated. Note that pre-assigning any bit that is used by both equations decreases the value of interdependence.

To capture the notion of equations interdependence $IntrDep(E)$ in the whole system of Trivium equations E, the following measure could be used

$$IntrDep(E) = \sum_{e,g \in E, e \neq g} |\mathcal{V}(e) \cap \mathcal{V}(g)|^2 \ . \tag{10.2}$$

We used the sum of squares to prefer systems with more equations with only few active (non-preassigned) variables over less equations that have more active variables, but it is possible to use an alternative measure without the squares,

$$IntrDep(E) = \sum_{e,g \in E, e \neq g} |\mathcal{V}(e) \cap \mathcal{V}(g)| \ . \tag{10.3}$$

The algorithm for pre-assigning bits to minimize the above measure is rather simple. We start with computing the initial interdependence of the system. Then, we temporarily pick a free variable and assign it to compute the new interdependence of

the system. If this value is smaller than the current record, we remember it as a new record. After we test all possible candidates, we pick the record one and assign it for good. We repeat this procedure until we get the required number of preassigned bits.

Algorithm 8 Bit pre-assignment minimizing equation interdependence.

```
compute the initial interdependence IntrDep(E) of the system E
while number of preassigned variables ≤ desired number do
    for each free variable do
        assign the variable temporarily
        compute the new interdependence IntrDep(E) of the system
        if IntrDep(E) ≤ current record then
            record the current record and remember the best candidate so far
        end if
    end for
    assign the best candidate
end while
```

Using Algorithm 8 with measure (10.2) to preassign 200 bits in order to minimize the equations interdependence yields the following variable indices:

```
289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
307 308 309 313 314 315 319 320 321 325 326 327 331 332 333 337 338 339
343 344 345 349 350 351 355 356 357 361 362 363 367 368 369 373 374 375
379 380 381 385 386 387 391 392 393 397 398 399 403 404 405 409 410 411
415 416 417 421 422 423 427 428 429 433 434 435 439 440 441 445 446 447
451 452 453 457 458 459 463 464 465 469 470 471 475 476 477 481 482 483
487 488 489 493 494 495 499 500 501 505 506 507 511 512 513 517 518 519
523 524 525 529 530 531 535 536 537 541 542 543 547 548 549 553 554 555
559 560 561 565 566 567 571 572 573 577 578 579 583 584 585 589 590 591
595 596 597 601 602 603 607 608 609 613 614 615 619 620 621 625 627 631
633 637 639 643 645 649 651 655 657 661 663 667 669 673 675 679 685 691
697 700
```

Applying the minimization strategy (10.3) gives a different set of 200 preassigned bits. This strategy simply fixes 200 consecutive bits starting at position 289. Theses are the variables which are newly introduced and appear in the maximal number of 8 equations, meaning that this strategy is the same as Strategy 5 and does not have to be investigated again.

We performed an experiment that compared the results of the reference pre-assignment strategy fixing the first 190 bits of the state with the pre-assigning strategy where we minimize the interdependence using measure (10.2). The results presented in Table 10.4 are interesting. It seems that minimizing the interdependence using the square of sums as measurements worsen the running time. One possible explanation is that for such systems a different value of the nochangebound should be used.

Table 10.4: Running times for bit pre-assignment strategies minimizing equation interdependence. Parameter values $\alpha = 33$, $M = 1024$, nochangebound $= 110$ were used.

strategy:	reference	Case 1
	29.49	> 39.3
	28.36	38.2
	30.17	38.1
	29.95	39.7
	29.53	39.3
	29.49	36.9
	28.94	40.8
	30.66	
	27.76	
	29.08	
avg:	29.34	38.9

10.5.2 Overdetermined System of Equations

A fully-determined non-linear Boolean equation system might have more than one solution, which means that the corresponding discrete optimization problem has more than one global minimum. Additionally we expect that the discrete optimization problem contains several local minima, this we also observe during our experiments. When we consider an overdetermined equation system we expect that there exists a unique solution, which means that the corresponding discrete optimization problem has only one global minimum. Furthermore we hope, that a discrete optimization problem derived from an overdetermined Boolean equation system possesses less or at least more shallow minima, so that it is easier for a search algorithm to find the optimum because the algorithm does not get stuck in local optima.

The first approach is to consider overdetermined equation systems of Trivium which are obtained by running the keystream generator further. After the first 288 clockings we have obtained a fully-determined system describing the internal state of Trivium. Each further clocking yields three additional equations and two unknowns. Initial experimental results showed that overdetermined systems do not lead to a better complexity of the attack. We infer that the better properties such as a unique minimum etc. are offset by the larger size of the system, because we do not only add equations but also unknowns.

A second approach is to add new equations to the system without increasing the number of variables. This can be done by adding what we call mixed quadratic equations [76]. The quadratic equations in the Trivium system are all of the form

$$s_{p_0} = s_{p_1} + s_{p_2} + s_{p_3} \cdot s_{p_4} + s_{p_5}$$

where the indices $p_0, \ldots, p_5$ are defined by the clocking process. A condition for new equations added to the system is that they are zero for the initial state of Trivium,

so that the global optimum of the corresponding discrete optimization problem is still zero.

We consider the following two equations taken from the set of quadratic Trivium equations

$$s_{p_0} = s_{p_1} + s_{p_2} + s_{p_3} \cdot s_{p_4} + s_{p_5} \tag{10.4}$$
$$s_{q_0} = s_{q_1} + s_{q_2} + s_{q_3} \cdot s_{q_4} + s_{q_5} \tag{10.5}$$

If the set $\{p_3, p_4\} \cap \{q_3, q_4\} =: \{w\}$ contains exactly one element, then we multiply equation (10.4) with s_a where $a := \{q_3, q_4\} \setminus \{w\}$ and equation (10.5) by s_b where $b = \{p_3, p_4\} \setminus \{w\}$. This yields two cubic equations that share the same term of degree three. By adding these equations together we obtain a new quadratic equation of the form

$$s_a \cdot (s_{p_0} + s_{p_1} + s_{p_2} + s_{p_5}) + s_b \cdot (s_{q_0} + s_{q_1} + s_{q_2} + s_{q_5}) = 0, \tag{10.6}$$

which we referred to as a mixed quadratic equation. We can obtain 663 new equations of this form from the basic Trivium system that way. The set of quadratic terms in those equations are mutually disjoint, meaning that there is no way of reducing the density of any equation by linear combinations. Each equation consists of 8 quadratic terms in 9 variables. There are 9 instead of 10 variables in each equation because in equation (10.6) either s_a equals one of the s_{q_i}'s or s_b is equal to one of the s_{p_i}'s.

These results can be explained using an alternative description of Trivium utilizing three registers and recurrence relations as shown by Bernstein [11]. In this description the three registers that form the internal state of Trivium are called register x, y and z, respectively, and the Trivium equations are of the form

$$o_n = z_{n-66} + z_{n-111} + x_{n-66} + x_{n-93} + y_{n-69} + y_{n-84},$$

where o_n denotes the keystream bit and

$$\begin{aligned} x_n &= z_{n-66} + z_{n-111} + z_{n-110} \cdot z_{n-109} + x_{n-69}, \\ y_n &= x_{n-66} + x_{n-93} + x_{n-92} \cdot x_{n-91} + y_{n-78}, \\ z_n &= y_{n-69} + y_{n-84} + y_{n-83} \cdot y_{n-82} + z_{n-87}. \end{aligned} \tag{10.7}$$

Considering the equations in (10.7) the only way that two equations can share a variable in the quadratic term is to take two equations describing the same register in two consecutive clocks $n, n+1$. As an example we consider the two equations

$$x_n = z_{n-66} + z_{n-111} + z_{n-110} \cdot z_{n-109} + x_{n-69}, \tag{10.8}$$
$$x_{n+1} = z_{n-65} + z_{n-110} + z_{n-109} \cdot z_{n-108} + x_{n-68}. \tag{10.9}$$

These two equations share the common variable z_{n-109} in their quadratic terms. Thus the multiplier for Equation (10.8) is z_{n-108}, while the multiplier for Equation (10.9) is the variable z_{n-110}, which also appears as a linear term in Equation (10.8). Hence the associated mixed quadratic equation contains 9 variables. Furthermore, in 288 clockings of the keystream generator 666 quadratic equations are generated, 222 for

each register (Remember that we do not consider the quadratic equations generated in the last 66 clockings of the algorithm). For each two consecutive quadratic equations in each register we obtain a mixed quadratic equation, meaning we obtain 221 mixed quadratic equations for each register, which yields a total of 663 mixed quadratic equations.

Initial tests showed that adding all possible mixed quadratic equations to the basic Trivium system impairs the running time of the simulated annealing search. We ran the tests for the same parameters as we used for the basic Trivium system. One possible explanation is that we modify the cost function by adding more equations and that for this new cost function the old parameters such as initial temperature, nochangebound and cooling schedule are not optimal anymore, because the average cost and the variance of the cost are significantly different from the original system.

A direction for further research is adding a subset of the mixed quadratic equations instead of all. This would still give us the benefits of an overdetermined system but not change the cost function too dramatically. The main questions are how many and which equations should be added to the system. However, no experiments have been done so far.

10.5.3 Variable Persistence

In [69] simulated annealing is applied to the permuted perceptron problem. An idea used in that work is to find a correlation between local minima of the cost function. In [69] the simulating annealing search is run t times until it found a configuration with low costs. These solutions where recorded and after t runs a set I of variables was identified, where a variable had the same sign for all recorded solutions. In the next runs of the algorithms the starting point of search was chosen such that all variables in I where fixed to their corresponding signs, while all remaining variables where chosen uniformly at random.

This gives rise to the assumption that there is a correlation between bits of the local and global minima and that bits of the global minimum can be identified by generating solutions for local minima and choosing the majority value of bits in the different solution as solution value of the global minimum.

We did some experiments that investigated if configurations of local minima, meaning the state we obtained after a long cooling run, have variables correlated with the global minimum state. In our limited experiments with the basic Trivium system we did not observe such correlations. In our experiments we considered sets that consists of 40 configurations with costs between 35 and 45. For each position we calculate the majority and set the bit to the value of the majority. Comparing this vector to the solution showed that their hamming distance is approximately the same as the hamming weight of a random point and the solution. The reasons for this might be that the sample size is too small, the local minima we consider with cost between 35-45 do not exhibit a correlations or a simple majority is not a strong enough condition.

We could also observe from our preliminary experiments that for a local minimum with a cost value around 40 the current solution still had a large hamming distance

to the known optimal solution.

10.6 Conclusion and Future Directions

In this chapter we presented a new way of approaching the problem of solving systems of sparse multivariate Boolean equations with a large variable locality. We represent them as a discrete optimization problem where the cost function counts the number of not satisfied equations and then we apply a simulated annealing variant.

We showed that such systems may be relevant in cryptography by giving an example of the system generated by the problem of recovering the internal state of the stream cipher Trivium.

Our experimental results were focused on the Trivium system and they seem to be promising but for now they do not seem to pose any real threat to the security of this algorithm. The attack complexity which we can confirm is 2^{210} bit flips which is equivalent to 2^{203} evaluations of the equation system. Hence this complexity is much worse than exhaustive search.

There are many open problems in this area, the most obvious ones are the selection of better parameters of the search procedures and analytically estimating the possible complexity of such algorithms. However, this is a time consuming process.

The other interesting direction seems to be the investigation of alternative cost functions. In all our experiments we use the simplest measure counting the number of not satisfied equations. However, many results in heuristic search literature suggest that the selection of a suitable cost function may dramatically change the efficiency of a search. The question of determining whether in our case there exist measures better than the ones we used is still open.

Furthermore, we can investigate the role, the choice of the neighborhood plays. In our experiments we use the smallest possible neighborhood. Considering a larger neighborhood might lead to a faster improvements in the costs but a larger neighborhood comes also for the expense that we might have to consider more neighbors in each step. Also an adaptive choice of the neighbors instead of random generation might improve the algorithm. Here we could exploit the special structure of the problem we are looking at instead of applying a purely generic algorithm.

Chapter 11

Conclusion

The objective of this thesis was the cryptanalysis of symmetric encryption schemes. We worked in two very different research topics. One topic was the security of block ciphers with secret components where we considered the block cipher C2 and Maya, both examples of designs with secret S-boxes. The other topic was the cryptanalysis by means of numerical methods where we investigated the applicability of optimization methods in cryptanalysis. Here we focused on the stream cipher Trivium due to its elegant description as a sparse non-linear Boolean equation system.

Block ciphers are usually built from a round function in an iterative way. While a single round can often be easily broken, a repeated use of the round function obscures patterns and structures, which could be used to break the cipher. On the one hand each additional round strengthens the cipher in terms of security but on the other hand it also adds to the complexity of the encryption. Thus, the designer of a block cipher has to find the right trade-off between performance and security. One idea is to reduce the number of rounds while at the same time strengthening the round function by using secret components. For example the S-boxes could be kept secret and considered as part of the secret key. If the S-boxes are unknown also their differential and linear behavior is unknown to an adversary. That suggests that differential and linear cryptanalysis, two of the most powerful cryptanalytic techniques, become infeasible. However, our analysis showed that secret S-boxes do not yield an impressive improvement of the security. We can recover the S-boxes of Maya with a practical complexity. Actually, an extrapolation of the attack complexity by means of a mathematical model suggests that more rounds of a cipher with secret but randomly chosen S-boxes can be attacked than of a cipher like PRESENT where the S-box is public but chosen carefully to exhibit good differential and linear properties. As our attack exploits weak differential properties, ciphers whose S-boxes are strong towards differential cryptanalysis are not affected by it. However, the set of non-equivalent strong S-boxes is very small and thus this restriction would not allow a huge key. We conjecture that the use of secret randomly chosen S-boxes does not allow us to decrease the number of rounds compared to ciphers with publicly known but strong S-boxes (at least not significantly).

In the case of C2 where a single 8-bit application-dependent S-box is used we can find a 5-round differential independent of the S-box which can be used to mount a boomerang. A further observation is that the security of the cipher almost entirely relies on the modular addition. For a linearized version of the cipher, where the modular addition is replaced by XOR, we can find a 9-round differential with probability one independent of the S-box, meaning that this version can be easily broken. From

these observations we infer that to achieve an impact on the security an encryption algorithm using a secret S-box has to be designed carefully.

The second direction of research in this thesis was inspired by the fact that many cryptographic algorithms can be described as non-linear multivariate Boolean equation systems. Solving the equation system recovers the secret key or initial state. However, solving a non-linear Boolean equation system is in general a difficult problem. Nevertheless, there are specially tailored algorithms which can solve such systems faster than exhaustive search by exploiting the special structure of the equation system. Examples for such algorithms are the XL and the XSL algorithm which operate over $\mathbb{F}_2$. However, in general algorithms for solving non-linear multivariate equation systems are better developed for the continuous problems. Thus our idea is to lift the problem of solving an equation system over the binary field to the reals and use optimization methods in order to solve such systems. We focus on the stream cipher Trivium because it has a very sparse and simple description as a quadratic Boolean equation system.

One approach is to convert the Boolean equations into equations over the reals. Solving the resulting non-linear equation system by means of continuous methods for zero finding or non-linear optimization leads to fractional solutions which cannot be converted back into a Boolean solution. Therefore, we restrict the solution space to integer-valued solutions and formulate the problem as a mixed-integer linear programming problem. The commercial solver Cplex was able to solve the mixed-integer programming problems corresponding to the Trivium small scale variants Bivium A and B faster than exhaustive key search. However, in the current form mixed-integer programming is not a threat to Trivium.

We focused on mixed-integer linear programming as mature and proven solvers exist for such problems. However, linearizing the equations increases the number of constraints and variables significantly which makes the problem more difficult. Integer programming is a still developing field of research with solvers for the linear case being improved and also new algorithms for mixed-integer non-linear programming being developed. A future direction of research is to consider for example Trivium as a mixed-integer non-linear programming problem and to investigate if the smaller problem size compensates for the difficulties caused by the non-linearity of the constraints.

In the second approach we used the Boolean equation system to define a discrete optimization problem. In a discrete optimization problem we consider a finite set of configurations and assign to each configuration a cost value. We apply simulated annealing as an example of a neighborhood search algorithm to the problem. If we consider the Trivium equation system as a random Boolean system we can solve it faster than exhaustive search, namely in 2^{203} evaluations of the system. However, as the complexity of recovering the key is upper bounded by 2^{80} key set-ups this attack does not pose a threat to Trivium. It is important to note that the neighborhood defined in the simulated annealing algorithm is the smallest possible neighborhood and does not incorporate the specific structure of the Trivium system. A future direction of research is to more carefully design a neighborhood, such that the neighbors are not

chosen at random but in a way that neighbors which are likely to improve the costs are considered first. Also an alternative definition of the cost function might improve the attack.

While simulated annealing was considered in the context of cryptology before, mixed-integer programming is to our knowledge an entirely new approach in cryptanalysis. Even though neither mixed-integer linear programming nor simulated annealing pose a threat to an existing real-life cipher our analysis shows the potential of both approaches in cryptanalysis and we believe that this thesis contains an interesting starting point for further research on the use of optimization methods in cryptology.

Appendix A

Five-round Characteristics for C2

We list some 5-round characteristics for the block cipher C2 together with their estimate and actual probabilities. As the 5-round characteristic used in the boomerang attack on C2 in Chapter 5 this characteristic where found while searching for code words with low hamming weights. All characteristics contain 15 ones. However, their estimated and actually probabilities vary depending on if there is a difference in the most significant bits. We determined they actual probability choosing 2^{24} random plaintexts and keys. In the first column the input and output difference of the characteristic is given. The second column contains the theoretical estimate of the probability of the differential and the third column the experimentally obtain probability.

Differential	th. prob.	exp. prob.
(0x00020800 0x80200100) → (0x80200100 0x00020800)	2^{-12}	$2^{-11.17}$
(0x04100000 0x40020100) → (0x40020100 0x04100000)	2^{-21}	$2^{-18.17}$
(0x08200000 0x80040200) → (0x80040200 0x08200000)	2^{-17}	$2^{-14.14}$
(0x40000010 0x08040100) → (0x08040100 0x40000010)	2^{-21}	$2^{-16.74}$
(0x80000020 0x10080200) → (0x10080200 0x80000020)	2^{-19}	$2^{-15.73}$
(0x00000104 0x80401000) → (0x80401000 0x00000104)	2^{-17}	$2^{-13.81}$
(0x00000041 0x20100400) → (0x20100400 0x00000041)	2^{-21}	$2^{-17.7}$
(0x80040200 0x00000200) → (0x08200000 0xC8260A10)	2^{-22}	$2^{-17.98}$
(0x00000082 0x40200800) → (0x40200800 0x00000082)	2^{-21}	$2^{-17.54}$
(0x80200100 0x80000000) → (0x00020800 0x84320982)	2^{-18}	$2^{-15.8}$
(0x40020100 0x00000100) → (0x04100000 0x64130508)	2^{-26}	$2^{-20.83}$

Appendix B

A Short Analysis of the Trivium Equation System

As explained in Section 3.4 the internal state of the stream cipher Trivium [43] can be described as a system of sparse, non-linear Boolean equations where the equations are of the form

$$\begin{aligned} s_{66} \oplus s_{93} \oplus s_{91} \cdot s_{92} \oplus s_{171} &= s_{289} \\ s_{162} \oplus s_{177} \oplus s_{175} \cdot s_{176} \oplus s_{264} &= s_{290} \\ s_{243} \oplus s_{288} \oplus s_{286} \cdot s_{287} \oplus s_{69} &= s_{291} \\ s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288} &= z \end{aligned}$$

In each clocking of the algorithm we introduce three new variables, one for each updated state bit and obtain one linear and three quadratic equations. After observing 288 keystream bits we obtain a system of 954 equations in 954 unknowns. The system consist of 288 linear and 666 quadratic equations. For the last 66 clockings of the algorithms we do neither derive the quadratic equations nor introduce new variables since each newly updated state bit is not use for at least 66 clockings of the algorithms.

Each of the equations contains exactly 6 variables. In each clocking all of the 6 variable contained in the keystream equation are also contained as linear terms in quadratic equations which are derived from the state update functions. Each two that belong to the same register appear together in the same quadratic equation.

It is easy to obtain an overdetermined system by observing more than 288 keystream bits. In each further clocking of the algorithm we introduce three new variables and obtain four equations.

For any size of the system each variable appears in at most two linear and 6 quadratic equations, in average a variable appears in 6 equations. If a variable appears in the maximum of 8 equations it appears twice in the quadratic term of a quadratic equation. We list the variables and the indices of the equations in which they appear in Table B.1

In Subsection 10.5.2 we explained how to obtain additional mixed quadratic equations from a given Trivium system. We can obtain $q - 3$ mixed quadratic equations where q is the number of quadratic equations in the given equation system. Each mixed quadratic equations consists of exactly 8 quadratic terms in 9 variables. Each variable appears in at most 9 mixed quadratic equations.

A similar analysis holds for Bivium B. After observing 177 keystream bits we can establish a fully determined system of 177 linear and 222 quadratic equations in 399

unknowns. Each linear equation consists of exactly four and each quadratic equation of 6 variables. Each variable appears in at most 8 equations.

In the case of Bivium A the analysis is slightly more complicated. The linear equation in Bivium A depends only directly on bits of the second register. The quadratic equations are the same as for Bivium B. That means that the each linear equations contains two variables and each quadratic equation 6 variables. Each variable from the second register appears in a maximum of 8 equations, two linear and 6 quadratic, while each variable from the first register appears in at most 6 quadratic and no linear equations.

Table B.1: Index Table: This table contains the indices of the linear and quadratic equations in which a single variable appears.

var	linear		quadratic						var	linear		quadratic					
1	65	92	195	206	270	273	276	-1	2	64	91	192	203	267	270	273	-1
3	63	90	189	200	264	267	270	-1	4	62	89	186	197	261	264	267	-1
5	61	88	183	194	258	261	264	-1	6	60	87	180	191	255	258	261	-1
7	59	86	177	188	252	255	258	-1	8	58	85	174	185	249	252	255	-1
9	57	84	171	182	246	249	252	-1	10	56	83	168	179	243	246	249	-1
11	55	82	165	176	240	243	246	-1	12	54	81	162	173	237	240	243	-1
13	53	80	159	170	234	237	240	-1	14	52	79	156	167	231	234	237	-1
15	51	78	153	164	228	231	234	-1	16	50	77	150	161	225	228	231	-1
17	49	76	147	158	222	225	228	-1	18	48	75	144	155	219	222	225	-1
19	47	74	141	152	216	219	222	-1	20	46	73	138	149	213	216	219	-1
21	45	72	135	146	210	213	216	-1	22	44	71	132	143	207	210	213	-1
23	43	70	129	140	204	207	210	-1	24	42	69	126	137	201	204	207	-1
25	41	68	123	134	198	201	204	-1	26	40	67	120	131	195	198	201	-1
27	39	66	117	128	192	195	198	-1	28	38	65	114	125	189	192	195	-1
29	37	64	111	122	186	189	192	-1	30	36	63	108	119	183	186	189	-1
31	35	62	105	116	180	183	186	-1	32	34	61	102	113	177	180	183	-1
33	33	60	99	110	174	177	180	-1	34	32	59	96	107	171	174	177	-1
35	31	58	93	104	168	171	174	-1	36	30	57	90	101	165	168	171	-1
37	29	56	87	98	162	165	168	-1	38	28	55	84	95	159	162	165	-1
39	27	54	81	92	156	159	162	-1	40	26	53	78	89	153	156	159	-1
41	25	52	75	86	150	153	156	-1	42	24	51	72	83	147	150	153	-1
43	23	50	69	80	144	147	150	-1	44	22	49	66	77	141	144	147	-1
45	21	48	63	74	138	141	144	-1	46	20	47	60	71	135	138	141	-1
47	19	46	57	68	132	135	138	-1	48	18	45	54	65	129	132	135	-1
49	17	44	51	62	126	129	132	-1	50	16	43	48	59	123	126	129	-1
51	15	42	45	56	120	123	126	-1	52	14	41	42	53	117	120	123	-1
53	13	40	39	50	114	117	120	-1	54	12	39	36	47	111	114	117	-1
55	11	38	33	44	108	111	114	-1	56	10	37	30	41	105	108	111	-1
57	9	36	27	38	102	105	108	-1	58	8	35	24	35	99	102	105	-1
59	7	34	21	32	96	99	102	-1	60	6	33	18	29	93	96	99	-1
61	5	32	15	26	90	93	96	-1	62	4	31	12	23	87	90	93	-1
63	3	30	9	20	84	87	90	-1	64	2	29	6	17	81	84	87	-1
65	1	28	3	14	78	81	84	-1	66	0	27	0	11	75	78	81	-1
67	26	-1	8	72	75	78	-1	-1	68	25	-1	5	69	72	75	-1	-1
69	24	-1	2	66	69	72	-1	-1	70	23	-1	63	66	69	-1	-1	-1
71	22	-1	60	63	66	-1	-1	-1	72	21	-1	57	60	63	-1	-1	-1
73	20	-1	54	57	60	-1	-1	-1	74	19	-1	51	54	57	-1	-1	-1
75	18	-1	48	51	54	-1	-1	-1	76	17	-1	45	48	51	-1	-1	-1
77	16	-1	42	45	48	-1	-1	-1	78	15	-1	39	42	45	-1	-1	-1
79	14	-1	36	39	42	-1	-1	-1	80	13	-1	33	36	39	-1	-1	-1
81	12	-1	30	33	36	-1	-1	-1	82	11	-1	27	30	33	-1	-1	-1
83	10	-1	24	27	30	-1	-1	-1	84	9	-1	21	24	27	-1	-1	-1
85	8	-1	18	21	24	-1	-1	-1	86	7	-1	15	18	21	-1	-1	-1
87	6	-1	12	15	18	-1	-1	-1	88	5	-1	9	12	15	-1	-1	-1
89	4	-1	6	9	12	-1	-1	-1	90	3	-1	3	6	9	-1	-1	-1
91	2	-1	0	3	6	-1	-1	-1	92	1	-1	0	3	-1	-1	-1	-1
93	0	-1	0	-1	-1	-1	-1	-1	94	68	83	205	231	244	247	250	-1
95	67	82	202	228	241	244	247	-1	96	66	81	199	225	238	241	244	-1
97	65	80	196	222	235	238	241	-1	98	64	79	193	219	232	235	238	-1
99	63	78	190	216	229	232	235	-1	100	62	77	187	213	226	229	232	-1
101	61	76	184	210	223	226	229	-1	102	60	75	181	207	220	223	226	-1
103	59	74	178	204	217	220	223	-1	104	58	73	175	201	214	217	220	-1
105	57	72	172	198	211	214	217	-1	106	56	71	169	195	208	211	214	-1
107	55	70	166	192	205	208	211	-1	108	54	69	163	189	202	205	208	-1
109	53	68	160	186	199	202	205	-1	110	52	67	157	183	196	199	202	-1
111	51	66	154	180	193	196	199	-1	112	50	65	151	177	190	193	196	-1
113	49	64	148	174	187	190	193	-1	114	48	63	145	171	184	187	190	-1
115	47	62	142	168	181	184	187	-1	116	46	61	139	165	178	181	184	-1
117	45	60	136	162	175	178	181	-1	118	44	59	133	159	172	175	178	-1
119	43	58	130	156	169	172	175	-1	120	42	57	127	153	166	169	172	-1
121	41	56	124	150	163	166	169	-1	122	40	55	121	147	160	163	166	-1
123	39	54	118	144	157	160	163	-1	124	38	53	115	141	154	157	160	-1

Continued on next page

Table B.1 – continued from previous page

var	linear		quadratic						var	linear		quadratic					
125	37	52	112	138	151	154	157	-1	126	36	51	109	135	148	151	154	-1
127	35	50	106	132	145	148	151	-1	128	34	49	103	129	142	145	148	-1
129	33	48	100	126	139	142	145	-1	130	32	47	97	123	136	139	142	-1
131	31	46	94	120	133	136	139	-1	132	30	45	91	117	130	133	136	-1
133	29	44	88	114	127	130	133	-1	134	28	43	85	111	124	127	130	-1
135	27	42	82	108	121	124	127	-1	136	26	41	79	105	118	121	124	-1
137	25	40	76	102	115	118	121	-1	138	24	39	73	99	112	115	118	-1
139	23	38	70	96	109	112	115	-1	140	22	37	67	93	106	109	112	-1
141	21	36	64	90	103	106	109	-1	142	20	35	61	87	100	103	106	-1
143	19	34	58	84	97	100	103	-1	144	18	33	55	81	94	97	100	-1
145	17	32	52	78	91	94	97	-1	146	16	31	49	75	88	91	94	-1
147	15	30	46	72	85	88	91	-1	148	14	29	43	69	82	85	88	-1
149	13	28	40	66	79	82	85	-1	150	12	27	37	63	76	79	82	-1
151	11	26	34	60	73	76	79	-1	152	10	25	31	57	70	73	76	-1
153	9	24	28	54	67	70	73	-1	154	8	23	25	51	64	67	70	-1
155	7	22	22	48	61	64	67	-1	156	6	21	19	45	58	61	64	-1
157	5	20	16	42	55	58	61	-1	158	4	19	13	39	52	55	58	-1
159	3	18	10	36	49	52	55	-1	160	2	17	7	33	46	49	52	-1
161	1	16	4	30	43	46	49	-1	162	0	15	1	27	40	43	46	-1
163	14	-1	24	37	40	43	-1	-1	164	13	-1	21	34	37	40	-1	-1
165	12	-1	18	31	34	37	-1	-1	166	11	-1	15	28	31	34	-1	-1
167	10	-1	12	25	28	31	-1	-1	168	9	-1	9	22	25	28	-1	-1
169	8	-1	6	19	22	25	-1	-1	170	7	-1	3	16	19	22	-1	-1
171	6	-1	0	13	16	19	-1	-1	172	5	-1	10	13	16	-1	-1	-1
173	4	-1	7	10	13	-1	-1	-1	174	3	-1	4	7	10	-1	-1	-1
175	2	-1	1	4	7	-1	-1	-1	176	1	-1	1	4	-1	-1	-1	-1
177	0	-1	1	-1	-1	-1	-1	-1	178	65	110	197	259	326	329	332	-1
179	64	109	194	256	323	326	329	-1	180	63	108	191	253	320	323	326	-1
181	62	107	188	250	317	320	323	-1	182	61	106	185	247	314	317	320	-1
183	60	105	182	244	311	314	317	-1	184	59	104	179	241	308	311	314	-1
185	58	103	176	238	305	308	311	-1	186	57	102	173	235	302	305	308	-1
187	56	101	170	232	299	302	305	-1	188	55	100	167	229	296	299	302	-1
189	54	99	164	226	293	296	299	-1	190	53	98	161	223	290	293	296	-1
191	52	97	158	220	287	290	293	-1	192	51	96	155	217	284	287	290	-1
193	50	95	152	214	281	284	287	-1	194	49	94	149	211	278	281	284	-1
195	48	93	146	208	275	278	281	-1	196	47	92	143	205	272	275	278	-1
197	46	91	140	202	269	272	275	-1	198	45	90	137	199	266	269	272	-1
199	44	89	134	196	263	266	269	-1	200	43	88	131	193	260	263	266	-1
201	42	87	128	190	257	260	263	-1	202	41	86	125	187	254	257	260	-1
203	40	85	122	184	251	254	257	-1	204	39	84	119	181	248	251	254	-1
205	38	83	116	178	245	248	251	-1	206	37	82	113	175	242	245	248	-1
207	36	81	110	172	239	242	245	-1	208	35	80	107	169	236	239	242	-1
209	34	79	104	166	233	236	239	-1	210	33	78	101	163	230	233	236	-1
211	32	77	98	160	227	230	233	-1	212	31	76	95	157	224	227	230	-1
213	30	75	92	154	221	224	227	-1	214	29	74	89	151	218	221	224	-1
215	28	73	86	148	215	218	221	-1	216	27	72	83	145	212	215	218	-1
217	26	71	80	142	209	212	215	-1	218	25	70	77	139	206	209	212	-1
219	24	69	74	136	203	206	209	-1	220	23	68	71	133	200	203	206	-1
221	22	67	68	130	197	200	203	-1	222	21	66	65	127	194	197	200	-1
223	20	65	62	124	191	194	197	-1	224	19	64	59	121	188	191	194	-1
225	18	63	56	118	185	188	191	-1	226	17	62	53	115	182	185	188	-1
227	16	61	50	112	179	182	185	-1	228	15	60	47	109	176	179	182	-1
229	14	59	44	106	173	176	179	-1	230	13	58	41	103	170	173	176	-1
231	12	57	38	100	167	170	173	-1	232	11	56	35	97	164	167	170	-1
233	10	55	32	94	161	164	167	-1	234	9	54	29	91	158	161	164	-1
235	8	53	26	88	155	158	161	-1	236	7	52	23	85	152	155	158	-1
237	6	51	20	82	149	152	155	-1	238	5	50	17	79	146	149	152	-1
239	4	49	14	76	143	146	149	-1	240	3	48	11	73	140	143	146	-1
241	2	47	8	70	137	140	143	-1	242	1	46	5	67	134	137	140	-1
243	0	45	2	64	131	134	137	-1	244	44	-1	61	128	131	134	-1	-1
245	43	-1	58	125	128	131	-1	-1	246	42	-1	55	122	125	128	-1	-1
247	41	-1	52	119	122	125	-1	-1	248	40	-1	49	116	119	122	-1	-1
249	39	-1	46	113	116	119	-1	-1	250	38	-1	43	110	113	116	-1	-1
251	37	-1	40	107	110	113	-1	-1	252	36	-1	37	104	107	110	-1	-1
253	35	-1	34	101	104	107	-1	-1	254	34	-1	31	98	101	104	-1	-1
255	33	-1	28	95	98	101	-1	-1	256	32	-1	25	92	95	98	-1	-1
257	31	-1	22	89	92	95	-1	-1	258	30	-1	19	86	89	92	-1	-1
259	29	-1	16	83	86	89	-1	-1	260	28	-1	13	80	83	86	-1	-1
261	27	-1	10	77	80	83	-1	-1	262	26	-1	7	74	77	80	-1	-1
263	25	-1	4	71	74	77	-1	-1	264	24	-1	1	68	71	74	-1	-1
265	23	-1	65	68	71	-1	-1	-1	266	22	-1	62	65	68	-1	-1	-1
267	21	-1	59	62	65	-1	-1	-1	268	20	-1	56	59	62	-1	-1	-1
269	19	-1	53	56	59	-1	-1	-1	270	18	-1	50	53	56	-1	-1	-1
271	17	-1	47	50	53	-1	-1	-1	272	16	-1	44	47	50	-1	-1	-1
273	15	-1	41	44	47	-1	-1	-1	274	14	-1	38	41	44	-1	-1	-1
275	13	-1	35	38	41	-1	-1	-1	276	12	-1	32	35	38	-1	-1	-1
277	11	-1	29	32	35	-1	-1	-1	278	10	-1	26	29	32	-1	-1	-1
279	9	-1	23	26	29	-1	-1	-1	280	8	-1	20	23	26	-1	-1	-1
281	7	-1	17	20	23	-1	-1	-1	282	6	-1	14	17	20	-1	-1	-1
283	5	-1	11	14	17	-1	-1	-1	284	4	-1	8	11	14	-1	-1	-1
285	3	-1	5	8	11	-1	-1	-1	286	2	-1	2	5	8	-1	-1	-1
287	1	-1	2	5	-1	-1	-1	-1	288	0	-1	2	-1	-1	-1	-1	-1
289	69	84	0	208	234	247	250	253	290	66	111	1	200	262	329	332	335

Continued on next page

Table B.1 – continued from previous page

var	linear		quadratic						var	linear		quadratic					
291	66	93	2	198	209	273	276	279	292	70	85	3	211	237	250	253	256
293	67	112	4	203	265	332	335	338	294	67	94	5	201	212	276	279	282
295	71	86	6	214	240	253	256	259	296	68	113	7	206	268	335	338	341
297	68	95	8	204	215	279	282	285	298	72	87	9	217	243	256	259	262
299	69	114	10	209	271	338	341	344	300	69	96	11	207	218	282	285	288
301	73	88	12	220	246	259	262	265	302	70	115	13	212	274	341	344	347
303	70	97	14	210	221	285	288	291	304	74	89	15	223	249	262	265	268
305	71	116	16	215	277	344	347	350	306	71	98	17	213	224	288	291	294
307	75	90	18	226	252	265	268	271	308	72	117	19	218	280	347	350	353
309	72	99	20	216	227	291	294	297	310	76	91	21	229	255	268	271	274
311	73	118	22	221	283	350	353	356	312	73	100	23	219	230	294	297	300
313	77	92	24	232	258	271	274	277	314	74	119	25	224	286	353	356	359
315	74	101	26	222	233	297	300	303	316	78	93	27	235	261	274	277	280
317	75	120	28	227	289	356	359	362	318	75	102	29	225	236	300	303	306
319	79	94	30	238	264	277	280	283	320	76	121	31	230	292	359	362	365
321	76	103	32	228	239	303	306	309	322	80	95	33	241	267	280	283	286
323	77	122	34	233	295	362	365	368	324	77	104	35	231	242	306	309	312
325	81	96	36	244	270	283	286	289	326	78	123	37	236	298	365	368	371
327	78	105	38	234	245	309	312	315	328	82	97	39	247	273	286	289	292
329	79	124	40	239	301	368	371	374	330	79	106	41	237	248	312	315	318
331	83	98	42	250	276	289	292	295	332	80	125	43	242	304	371	374	377
333	80	107	44	240	251	315	318	321	334	84	99	45	253	279	292	295	298
335	81	126	46	245	307	374	377	380	336	81	108	47	243	254	318	321	324
337	85	100	48	256	282	295	298	301	338	82	127	49	248	310	377	380	383
339	82	109	50	246	257	321	324	327	340	86	101	51	259	285	298	301	304
341	83	128	52	251	313	380	383	386	342	83	110	53	249	260	324	327	330
343	87	102	54	262	288	301	304	307	344	84	129	55	254	316	383	386	389
345	84	111	56	252	263	327	330	333	346	88	103	57	265	291	304	307	310
347	85	130	58	257	319	386	389	392	348	85	112	59	255	266	330	333	336
349	89	104	60	268	294	307	310	313	350	86	131	61	260	322	389	392	395
351	86	113	62	258	269	333	336	339	352	90	105	63	271	297	310	313	316
353	87	132	64	263	325	392	395	398	354	87	114	65	261	272	336	339	342
355	91	106	66	274	300	313	316	319	356	88	133	67	266	328	395	398	401
357	88	115	68	264	275	339	342	345	358	92	107	69	277	303	316	319	322
359	89	134	70	269	331	398	401	404	360	89	116	71	267	278	342	345	348
361	93	108	72	280	306	319	322	325	362	90	135	73	272	334	401	404	407
363	90	117	74	270	281	345	348	351	364	94	109	75	283	309	322	325	328
365	91	136	76	275	337	404	407	410	366	91	118	77	273	284	348	351	354
367	95	110	78	286	312	325	328	331	368	92	137	79	278	340	407	410	413
369	92	119	80	276	287	351	354	357	370	96	111	81	289	315	328	331	334
371	93	138	82	281	343	410	413	416	372	93	120	83	279	290	354	357	360
373	97	112	84	292	318	331	334	337	374	94	139	85	284	346	413	416	419
375	94	121	86	282	293	357	360	363	376	98	113	87	295	321	334	337	340
377	95	140	88	287	349	416	419	422	378	95	122	89	285	296	360	363	366
379	99	114	90	298	324	337	340	343	380	96	141	91	290	352	419	422	425
381	96	123	92	288	299	363	366	369	382	100	115	93	301	327	340	343	346
383	97	142	94	293	355	422	425	428	384	97	124	95	291	302	366	369	372
385	101	116	96	304	330	343	346	349	386	98	143	97	296	358	425	428	431
387	98	125	98	294	305	369	372	375	388	102	117	99	307	333	346	349	352
389	99	144	100	299	361	428	431	434	390	99	126	101	297	308	372	375	378
391	103	118	102	310	336	349	352	355	392	100	145	103	302	364	431	434	437
393	100	127	104	300	311	375	378	381	394	104	119	105	313	339	352	355	358
395	101	146	106	305	367	434	437	440	396	101	128	107	303	314	378	381	384
397	105	120	108	316	342	355	358	361	398	102	147	109	308	370	437	440	443
399	102	129	110	306	317	381	384	387	400	106	121	111	319	345	358	361	364
401	103	148	112	311	373	440	443	446	402	103	130	113	309	320	384	387	390
403	107	122	114	322	348	361	364	367	404	104	149	115	314	376	443	446	449
405	104	131	116	312	323	387	390	393	406	108	123	117	325	351	364	367	370
407	105	150	118	317	379	446	449	452	408	105	132	119	315	326	390	393	396
409	109	124	120	328	354	367	370	373	410	106	151	121	320	382	449	452	455
411	106	133	122	318	329	393	396	399	412	110	125	123	331	357	370	373	376
413	107	152	124	323	385	452	455	458	414	107	134	125	321	332	396	399	402
415	111	126	126	334	360	373	376	379	416	108	153	127	326	388	455	458	461
417	108	135	128	324	335	399	402	405	418	112	127	129	337	363	376	379	382
419	109	154	130	329	391	458	461	464	420	109	136	131	327	338	402	405	408
421	113	128	132	340	366	379	382	385	422	110	155	133	332	394	461	464	467
423	110	137	134	330	341	405	408	411	424	114	129	135	343	369	382	385	388
425	111	156	136	335	397	464	467	470	426	111	138	137	333	344	408	411	414
427	115	130	138	346	372	385	388	391	428	112	157	139	338	400	467	470	473
429	112	139	140	336	347	411	414	417	430	116	131	141	349	375	388	391	394
431	113	158	142	341	403	470	473	476	432	113	140	143	339	350	414	417	420
433	117	132	144	352	378	391	394	397	434	114	159	145	344	406	473	476	479
435	114	141	146	342	353	417	420	423	436	118	133	147	355	381	394	397	400
437	115	160	148	347	409	476	479	482	438	115	142	149	345	356	420	423	426
439	119	134	150	358	384	397	400	403	440	116	161	151	350	412	479	482	485
441	116	143	152	348	359	423	426	429	442	120	135	153	361	387	400	403	406
443	117	162	154	353	415	482	485	488	444	117	144	155	351	362	426	429	432
445	121	136	156	364	390	403	406	409	446	118	163	157	356	418	485	488	491
447	118	145	158	354	365	429	432	435	448	122	137	159	367	393	406	409	412
449	119	164	160	359	421	488	491	494	450	119	146	161	357	368	432	435	438
451	123	138	162	370	396	409	412	415	452	120	165	163	362	424	491	494	497
453	120	147	164	360	371	435	438	441	454	124	139	165	373	399	412	415	418
455	121	166	166	365	427	494	497	500	456	121	148	167	363	374	438	441	444

Continued on next page

Table B.1 – continued from previous page

var	linear		quadratic						var	linear		quadratic					
457	125	140	168	376	402	415	418	421	458	122	167	169	368	430	497	500	503
459	122	149	170	366	377	441	444	447	460	126	141	171	379	405	418	421	424
461	123	168	172	371	433	500	503	506	462	123	150	173	369	380	444	447	450
463	127	142	174	382	408	421	424	427	464	124	169	175	374	436	503	506	509
465	124	151	176	372	383	447	450	453	466	128	143	177	385	411	424	427	430
467	125	170	178	377	439	506	509	512	468	125	152	179	375	386	450	453	456
469	129	144	180	388	414	427	430	433	470	126	171	181	380	442	509	512	515
471	126	153	182	378	389	453	456	459	472	130	145	183	391	417	430	433	436
473	127	172	184	383	445	512	515	518	474	127	154	185	381	392	456	459	462
475	131	146	186	394	420	433	436	439	476	128	173	187	386	448	515	518	521
477	128	155	188	384	395	459	462	465	478	132	147	189	397	423	436	439	442
479	129	174	190	389	451	518	521	524	480	129	156	191	387	398	462	465	468
481	133	148	192	400	426	439	442	445	482	130	175	193	392	454	521	524	527
483	130	157	194	390	401	465	468	471	484	134	149	195	403	429	442	445	448
485	131	176	196	395	457	524	527	530	486	131	158	197	393	404	468	471	474
487	135	150	198	406	432	445	448	451	488	132	177	199	398	460	527	530	533
489	132	159	200	396	407	471	474	477	490	136	151	201	409	435	448	451	454
491	133	178	202	401	463	530	533	536	492	133	160	203	399	410	474	477	480
493	137	152	204	412	438	451	454	457	494	134	179	205	404	466	533	536	539
495	134	161	206	402	413	477	480	483	496	138	153	207	415	441	454	457	460
497	135	180	208	407	469	536	539	542	498	135	162	209	405	416	480	483	486
499	139	154	210	418	444	457	460	463	500	136	181	211	410	472	539	542	545
501	136	163	212	408	419	483	486	489	502	140	155	213	421	447	460	463	466
503	137	182	214	413	475	542	545	548	504	137	164	215	411	422	486	489	492
505	141	156	216	424	450	463	466	469	506	138	183	217	416	478	545	548	551
507	138	165	218	414	425	489	492	495	508	142	157	219	427	453	466	469	472
509	139	184	220	419	481	548	551	554	510	139	166	221	417	428	492	495	498
511	143	158	222	430	456	469	472	475	512	140	185	223	422	484	551	554	557
513	140	167	224	420	431	495	498	501	514	144	159	225	433	459	472	475	478
515	141	186	226	425	487	554	557	560	516	141	168	227	423	434	498	501	504
517	145	160	228	436	462	475	478	481	518	142	187	229	428	490	557	560	563
519	142	169	230	426	437	501	504	507	520	146	161	231	439	465	478	481	484
521	143	188	232	431	493	560	563	566	522	143	170	233	429	440	504	507	510
523	147	162	234	442	468	481	484	487	524	144	189	235	434	496	563	566	569
525	144	171	236	432	443	507	510	513	526	148	163	237	445	471	484	487	490
527	145	190	238	437	499	566	569	572	528	145	172	239	435	446	510	513	516
529	149	164	240	448	474	487	490	493	530	146	191	241	440	502	569	572	575
531	146	173	242	438	449	513	516	519	532	150	165	243	451	477	490	493	496
533	147	192	244	443	505	572	575	578	534	147	174	245	441	452	516	519	522
535	151	166	246	454	480	493	496	499	536	148	193	247	446	508	575	578	581
537	148	175	248	444	455	519	522	525	538	152	167	249	457	483	496	499	502
539	149	194	250	449	511	578	581	584	540	149	176	251	447	458	522	525	528
541	153	168	252	460	486	499	502	505	542	150	195	253	452	514	581	584	587
543	150	177	254	450	461	525	528	531	544	154	169	255	463	489	502	505	508
545	151	196	256	455	517	584	587	590	546	151	178	257	453	464	528	531	534
547	155	170	258	466	492	505	508	511	548	152	197	259	458	520	587	590	593
549	152	179	260	456	467	531	534	537	550	156	171	261	469	495	508	511	514
551	153	198	262	461	523	590	593	596	552	153	180	263	459	470	534	537	540
553	157	172	264	472	498	511	514	517	554	154	199	265	464	526	593	596	599
555	154	181	266	462	473	537	540	543	556	158	173	267	475	501	514	517	520
557	155	200	268	467	529	596	599	602	558	155	182	269	465	476	540	543	546
559	159	174	270	478	504	517	520	523	560	156	201	271	470	532	599	602	605
561	156	183	272	468	479	543	546	549	562	160	175	273	481	507	520	523	526
563	157	202	274	473	535	602	605	608	564	157	184	275	471	482	546	549	552
565	161	176	276	484	510	523	526	529	566	158	203	277	476	538	605	608	611
567	158	185	278	474	485	549	552	555	568	162	177	279	487	513	526	529	532
569	159	204	280	479	541	608	611	614	570	159	186	281	477	488	552	555	558
571	163	178	282	490	516	529	532	535	572	160	205	283	482	544	611	614	617
573	160	187	284	480	491	555	558	561	574	164	179	285	493	519	532	535	538
575	161	206	286	485	547	614	617	620	576	161	188	287	483	494	558	561	564
577	165	180	288	496	522	535	538	541	578	162	207	289	488	550	617	620	623
579	162	189	290	486	497	561	564	567	580	166	181	291	499	525	538	541	544
581	163	208	292	491	553	620	623	626	582	163	190	293	489	500	564	567	570
583	167	182	294	502	528	541	544	547	584	164	209	295	494	556	623	626	629
585	164	191	296	492	503	567	570	573	586	168	183	297	505	531	544	547	550
587	165	210	298	497	559	626	629	632	588	165	192	299	495	506	570	573	576
589	169	184	300	508	534	547	550	553	590	166	211	301	500	562	629	632	635
591	166	193	302	498	509	573	576	579	592	170	185	303	511	537	550	553	556
593	167	212	304	503	565	632	635	638	594	167	194	305	501	512	576	579	582
595	171	186	306	514	540	553	556	559	596	168	213	307	506	568	635	638	641
597	168	195	308	504	515	579	582	585	598	172	187	309	517	543	556	559	562
599	169	214	310	509	571	638	641	644	600	169	196	311	507	518	582	585	588
601	173	188	312	520	546	559	562	565	602	170	215	313	512	574	641	644	647
603	170	197	314	510	521	585	588	591	604	174	189	315	523	549	562	565	568
605	171	216	316	515	577	644	647	650	606	171	198	317	513	524	588	591	594
607	175	190	318	526	552	565	568	571	608	172	217	319	518	580	647	650	653
609	172	199	320	516	527	591	594	597	610	176	191	321	529	555	568	571	574
611	173	218	322	521	583	650	653	656	612	173	200	323	519	530	594	597	600
613	177	192	324	532	558	571	574	577	614	174	219	325	524	586	653	656	659
615	174	201	326	522	533	597	600	603	616	178	193	327	535	561	574	577	580
617	175	220	328	527	589	656	659	662	618	175	202	329	525	536	600	603	606
619	179	194	330	538	564	577	580	583	620	176	221	331	530	592	659	662	665
621	176	203	332	528	539	603	606	609	622	180	195	333	541	567	580	583	586

Continued on next page

Table B.1 – continued from previous page

var	linear		quadratic						var	linear		quadratic						
623	177	222	334	533	595	662	665	-1	624	177	204	335	531	542	606	609	612	
625	181	196	336	544	570	583	586	589	626	178	223	337	536	598	665	-1	-1	
627	178	205	338	534	545	609	612	615	628	182	197	339	547	573	586	589	592	
629	179	224	340	539	601	-1	-1	-1	630	179	206	341	537	548	612	615	618	
631	183	198	342	550	576	589	592	595	632	180	225	343	542	604	-1	-1	-1	
633	180	207	344	540	551	615	618	621	634	184	199	345	553	579	592	595	598	
635	181	226	346	545	607	-1	-1	-1	636	181	208	347	543	554	618	621	624	
637	185	200	348	556	582	595	598	601	638	182	227	349	548	610	-1	-1	-1	
639	182	209	350	546	557	621	624	627	640	186	201	351	559	585	598	601	604	
641	183	228	352	551	613	-1	-1	-1	642	183	210	353	549	560	624	627	630	
643	187	202	354	562	588	601	604	607	644	184	229	355	554	616	-1	-1	-1	
645	184	211	356	552	563	627	630	633	646	188	203	357	565	591	604	607	610	
647	185	230	358	557	619	-1	-1	-1	648	185	212	359	555	566	630	633	636	
649	189	204	360	568	594	607	610	613	650	186	231	361	560	622	-1	-1	-1	
651	186	213	362	558	569	633	636	639	652	190	205	363	571	597	610	613	616	
653	187	232	364	563	625	-1	-1	-1	654	187	214	365	561	572	636	639	642	
655	191	206	366	574	600	613	616	619	656	188	233	367	566	628	-1	-1	-1	
657	188	215	368	564	575	639	642	645	658	192	207	369	577	603	616	619	622	
659	189	234	370	569	631	-1	-1	-1	660	189	216	371	567	578	642	645	648	
661	193	208	372	580	606	619	622	625	662	190	235	373	572	634	-1	-1	-1	
663	190	217	374	570	581	645	648	651	664	194	209	375	583	609	622	625	628	
665	191	236	376	575	637	-1	-1	-1	666	191	218	377	573	584	648	651	654	
667	195	210	378	586	612	625	628	631	668	192	237	379	578	640	-1	-1	-1	
669	192	219	380	576	587	651	654	657	670	196	211	381	589	615	628	631	634	
671	193	238	382	581	643	-1	-1	-1	672	193	220	383	579	590	654	657	660	
673	197	212	384	592	618	631	634	637	674	194	239	385	584	646	-1	-1	-1	
675	194	221	386	582	593	657	660	663	676	198	213	387	595	621	634	637	640	
677	195	240	388	587	649	-1	-1	-1	678	195	222	389	585	596	660	663	-1	
679	199	214	390	598	624	637	640	643	680	196	241	391	590	652	-1	-1	-1	
681	196	223	392	588	599	663	-1	-1	682	200	215	393	601	627	640	643	646	
683	197	242	394	593	655	-1	-1	-1	684	197	224	395	591	602	-1	-1	-1	
685	201	216	396	604	630	643	646	649	686	198	243	397	596	658	-1	-1	-1	
687	198	225	398	594	605	-1	-1	-1	688	202	217	399	607	633	646	649	652	
689	199	244	400	599	661	-1	-1	-1	690	199	226	401	597	608	-1	-1	-1	
691	203	218	402	610	636	649	652	655	692	200	245	403	602	664	-1	-1	-1	
693	200	227	404	600	611	-1	-1	-1	694	204	219	405	613	639	652	655	658	
695	201	246	406	605	-1	-1	-1	-1	696	201	228	407	603	614	-1	-1	-1	
697	205	220	408	616	642	655	658	661	698	202	247	409	608	-1	-1	-1	-1	
699	202	229	410	606	617	-1	-1	-1	700	206	221	411	619	645	658	661	664	
701	203	248	412	611	-1	-1	-1	-1	702	203	230	413	609	620	-1	-1	-1	
703	207	222	414	622	648	661	664	-1	704	204	249	415	614	-1	-1	-1	-1	
705	204	231	416	612	623	-1	-1	-1	706	208	223	417	625	651	664	-1	-1	
707	205	250	418	617	-1	-1	-1	-1	708	205	232	419	615	626	-1	-1	-1	
709	209	224	420	628	654	-1	-1	-1	710	206	251	421	620	-1	-1	-1	-1	
711	206	233	422	618	629	-1	-1	-1	712	210	225	423	631	657	-1	-1	-1	
713	207	252	424	623	-1	-1	-1	-1	714	207	234	425	621	632	-1	-1	-1	
715	211	226	426	634	660	-1	-1	-1	716	208	253	427	626	-1	-1	-1	-1	
717	208	235	428	624	635	-1	-1	-1	718	212	227	429	637	663	-1	-1	-1	
719	209	254	430	629	-1	-1	-1	-1	720	209	236	431	627	638	-1	-1	-1	
721	213	228	432	640	-1	-1	-1	-1	722	210	255	433	632	-1	-1	-1	-1	
723	210	237	434	630	641	-1	-1	-1	724	214	229	435	643	-1	-1	-1	-1	
725	211	256	436	635	-1	-1	-1	-1	726	211	238	437	633	644	-1	-1	-1	
727	215	230	438	646	-1	-1	-1	-1	728	212	257	439	638	-1	-1	-1	-1	
729	212	239	440	636	647	-1	-1	-1	730	216	231	441	649	-1	-1	-1	-1	
731	213	258	442	641	-1	-1	-1	-1	732	213	240	443	639	650	-1	-1	-1	
733	217	232	444	652	-1	-1	-1	-1	734	214	259	445	644	-1	-1	-1	-1	
735	214	241	446	642	653	-1	-1	-1	736	218	233	447	655	-1	-1	-1	-1	
737	215	260	448	647	-1	-1	-1	-1	738	215	242	449	645	656	-1	-1	-1	
739	219	234	450	658	-1	-1	-1	-1	740	216	261	451	650	-1	-1	-1	-1	
741	216	243	452	648	659	-1	-1	-1	742	220	235	453	661	-1	-1	-1	-1	
743	217	262	454	653	-1	-1	-1	-1	744	217	244	455	651	662	-1	-1	-1	
745	221	236	456	664	-1	-1	-1	-1	746	218	263	457	656	-1	-1	-1	-1	
747	218	245	458	654	665	-1	-1	-1	748	222	237	459	-1	-1	-1	-1	-1	
749	219	264	460	659	-1	-1	-1	-1	750	219	246	461	657	-1	-1	-1	-1	
751	223	238	462	-1	-1	-1	-1	-1	752	220	265	463	662	-1	-1	-1	-1	
753	220	247	464	660	-1	-1	-1	-1	754	224	239	465	-1	-1	-1	-1	-1	
755	221	266	466	665	-1	-1	-1	-1	756	221	248	467	663	-1	-1	-1	-1	
757	225	240	468	-1	-1	-1	-1	-1	758	222	267	469	-1	-1	-1	-1	-1	
759	222	249	470	-1	-1	-1	-1	-1	760	226	241	471	-1	-1	-1	-1	-1	
761	223	268	472	-1	-1	-1	-1	-1	762	223	250	473	-1	-1	-1	-1	-1	
763	227	242	474	-1	-1	-1	-1	-1	764	224	269	475	-1	-1	-1	-1	-1	
765	224	251	476	-1	-1	-1	-1	-1	766	228	243	477	-1	-1	-1	-1	-1	
767	225	270	478	-1	-1	-1	-1	-1	768	225	252	479	-1	-1	-1	-1	-1	
769	229	244	480	-1	-1	-1	-1	-1	770	226	271	481	-1	-1	-1	-1	-1	
771	226	253	482	-1	-1	-1	-1	-1	772	230	245	483	-1	-1	-1	-1	-1	
773	227	272	484	-1	-1	-1	-1	-1	774	227	254	485	-1	-1	-1	-1	-1	
775	231	246	486	-1	-1	-1	-1	-1	776	228	273	487	-1	-1	-1	-1	-1	
777	228	255	488	-1	-1	-1	-1	-1	778	232	247	489	-1	-1	-1	-1	-1	
779	229	274	490	-1	-1	-1	-1	-1	780	229	256	491	-1	-1	-1	-1	-1	
781	233	248	492	-1	-1	-1	-1	-1	782	230	275	493	-1	-1	-1	-1	-1	
783	230	257	494	-1	-1	-1	-1	-1	784	234	249	495	-1	-1	-1	-1	-1	
785	231	276	496	-1	-1	-1	-1	-1	786	231	258	497	-1	-1	-1	-1	-1	
787	235	250	498	-1	-1	-1	-1	-1	788	232	277	499	-1	-1	-1	-1	-1	

Continued on next page

Table B.1 – continued from previous page

var	linear		quadratic						var	linear		quadratic					
789	232	259	500	-1	-1	-1	-1	-1	790	236	251	501	-1	-1	-1	-1	-1
791	233	278	502	-1	-1	-1	-1	-1	792	233	260	503	-1	-1	-1	-1	-1
793	237	252	504	-1	-1	-1	-1	-1	794	234	279	505	-1	-1	-1	-1	-1
795	234	261	506	-1	-1	-1	-1	-1	796	238	253	507	-1	-1	-1	-1	-1
797	235	280	508	-1	-1	-1	-1	-1	798	235	262	509	-1	-1	-1	-1	-1
799	239	254	510	-1	-1	-1	-1	-1	800	236	281	511	-1	-1	-1	-1	-1
801	236	263	512	-1	-1	-1	-1	-1	802	240	255	513	-1	-1	-1	-1	-1
803	237	282	514	-1	-1	-1	-1	-1	804	237	264	515	-1	-1	-1	-1	-1
805	241	256	516	-1	-1	-1	-1	-1	806	238	283	517	-1	-1	-1	-1	-1
807	238	265	518	-1	-1	-1	-1	-1	808	242	257	519	-1	-1	-1	-1	-1
809	239	284	520	-1	-1	-1	-1	-1	810	239	266	521	-1	-1	-1	-1	-1
811	243	258	522	-1	-1	-1	-1	-1	812	240	285	523	-1	-1	-1	-1	-1
813	240	267	524	-1	-1	-1	-1	-1	814	244	259	525	-1	-1	-1	-1	-1
815	241	286	526	-1	-1	-1	-1	-1	816	241	268	527	-1	-1	-1	-1	-1
817	245	260	528	-1	-1	-1	-1	-1	818	242	287	529	-1	-1	-1	-1	-1
819	242	269	530	-1	-1	-1	-1	-1	820	246	261	531	-1	-1	-1	-1	-1
821	243	-1	532	-1	-1	-1	-1	-1	822	243	270	533	-1	-1	-1	-1	-1
823	247	262	534	-1	-1	-1	-1	-1	824	244	-1	535	-1	-1	-1	-1	-1
825	244	271	536	-1	-1	-1	-1	-1	826	248	263	537	-1	-1	-1	-1	-1
827	245	-1	538	-1	-1	-1	-1	-1	828	245	272	539	-1	-1	-1	-1	-1
829	249	264	540	-1	-1	-1	-1	-1	830	246	-1	541	-1	-1	-1	-1	-1
831	246	273	542	-1	-1	-1	-1	-1	832	250	265	543	-1	-1	-1	-1	-1
833	247	-1	544	-1	-1	-1	-1	-1	834	247	274	545	-1	-1	-1	-1	-1
835	251	266	546	-1	-1	-1	-1	-1	836	248	-1	547	-1	-1	-1	-1	-1
837	248	275	548	-1	-1	-1	-1	-1	838	252	267	549	-1	-1	-1	-1	-1
839	249	-1	550	-1	-1	-1	-1	-1	840	249	276	551	-1	-1	-1	-1	-1
841	253	268	552	-1	-1	-1	-1	-1	842	250	-1	553	-1	-1	-1	-1	-1
843	250	277	554	-1	-1	-1	-1	-1	844	254	269	555	-1	-1	-1	-1	-1
845	251	-1	556	-1	-1	-1	-1	-1	846	251	278	557	-1	-1	-1	-1	-1
847	255	270	558	-1	-1	-1	-1	-1	848	252	-1	559	-1	-1	-1	-1	-1
849	252	279	560	-1	-1	-1	-1	-1	850	256	271	561	-1	-1	-1	-1	-1
851	253	-1	562	-1	-1	-1	-1	-1	852	253	280	563	-1	-1	-1	-1	-1
853	257	272	564	-1	-1	-1	-1	-1	854	254	-1	565	-1	-1	-1	-1	-1
855	254	281	566	-1	-1	-1	-1	-1	856	258	273	567	-1	-1	-1	-1	-1
857	255	-1	568	-1	-1	-1	-1	-1	858	255	282	569	-1	-1	-1	-1	-1
859	259	274	570	-1	-1	-1	-1	-1	860	256	-1	571	-1	-1	-1	-1	-1
861	256	283	572	-1	-1	-1	-1	-1	862	260	275	573	-1	-1	-1	-1	-1
863	257	-1	574	-1	-1	-1	-1	-1	864	257	284	575	-1	-1	-1	-1	-1
865	261	276	576	-1	-1	-1	-1	-1	866	258	-1	577	-1	-1	-1	-1	-1
867	258	285	578	-1	-1	-1	-1	-1	868	262	277	579	-1	-1	-1	-1	-1
869	259	-1	580	-1	-1	-1	-1	-1	870	259	286	581	-1	-1	-1	-1	-1
871	263	278	582	-1	-1	-1	-1	-1	872	260	-1	583	-1	-1	-1	-1	-1
873	260	287	584	-1	-1	-1	-1	-1	874	264	279	585	-1	-1	-1	-1	-1
875	261	-1	586	-1	-1	-1	-1	-1	876	261	-1	587	-1	-1	-1	-1	-1
877	265	280	588	-1	-1	-1	-1	-1	878	262	-1	589	-1	-1	-1	-1	-1
879	262	-1	590	-1	-1	-1	-1	-1	880	266	281	591	-1	-1	-1	-1	-1
881	263	-1	592	-1	-1	-1	-1	-1	882	263	-1	593	-1	-1	-1	-1	-1
883	267	282	594	-1	-1	-1	-1	-1	884	264	-1	595	-1	-1	-1	-1	-1
885	264	-1	596	-1	-1	-1	-1	-1	886	268	283	597	-1	-1	-1	-1	-1
887	265	-1	598	-1	-1	-1	-1	-1	888	265	-1	599	-1	-1	-1	-1	-1
889	269	284	600	-1	-1	-1	-1	-1	890	266	-1	601	-1	-1	-1	-1	-1
891	266	-1	602	-1	-1	-1	-1	-1	892	270	285	603	-1	-1	-1	-1	-1
893	267	-1	604	-1	-1	-1	-1	-1	894	267	-1	605	-1	-1	-1	-1	-1
895	271	286	606	-1	-1	-1	-1	-1	896	268	-1	607	-1	-1	-1	-1	-1
897	268	-1	608	-1	-1	-1	-1	-1	898	272	287	609	-1	-1	-1	-1	-1
899	269	-1	610	-1	-1	-1	-1	-1	900	269	-1	611	-1	-1	-1	-1	-1
901	273	-1	612	-1	-1	-1	-1	-1	902	270	-1	613	-1	-1	-1	-1	-1
903	270	-1	614	-1	-1	-1	-1	-1	904	274	-1	615	-1	-1	-1	-1	-1
905	271	-1	616	-1	-1	-1	-1	-1	906	271	-1	617	-1	-1	-1	-1	-1
907	275	-1	618	-1	-1	-1	-1	-1	908	272	-1	619	-1	-1	-1	-1	-1
909	272	-1	620	-1	-1	-1	-1	-1	910	276	-1	621	-1	-1	-1	-1	-1
911	273	-1	622	-1	-1	-1	-1	-1	912	273	-1	623	-1	-1	-1	-1	-1
913	277	-1	624	-1	-1	-1	-1	-1	914	274	-1	625	-1	-1	-1	-1	-1
915	274	-1	626	-1	-1	-1	-1	-1	916	278	-1	627	-1	-1	-1	-1	-1
917	275	-1	628	-1	-1	-1	-1	-1	918	275	-1	629	-1	-1	-1	-1	-1
919	279	-1	630	-1	-1	-1	-1	-1	920	276	-1	631	-1	-1	-1	-1	-1
921	276	-1	632	-1	-1	-1	-1	-1	922	280	-1	633	-1	-1	-1	-1	-1
923	277	-1	634	-1	-1	-1	-1	-1	924	277	-1	635	-1	-1	-1	-1	-1
925	281	-1	636	-1	-1	-1	-1	-1	926	278	-1	637	-1	-1	-1	-1	-1
927	278	-1	638	-1	-1	-1	-1	-1	928	282	-1	639	-1	-1	-1	-1	-1
929	279	-1	640	-1	-1	-1	-1	-1	930	279	-1	641	-1	-1	-1	-1	-1
931	283	-1	642	-1	-1	-1	-1	-1	932	280	-1	643	-1	-1	-1	-1	-1
933	280	-1	644	-1	-1	-1	-1	-1	934	284	-1	645	-1	-1	-1	-1	-1
935	281	-1	646	-1	-1	-1	-1	-1	936	281	-1	647	-1	-1	-1	-1	-1
937	285	-1	648	-1	-1	-1	-1	-1	938	282	-1	649	-1	-1	-1	-1	-1
939	282	-1	650	-1	-1	-1	-1	-1	940	286	-1	651	-1	-1	-1	-1	-1
941	283	-1	652	-1	-1	-1	-1	-1	942	283	-1	653	-1	-1	-1	-1	-1
943	287	-1	654	-1	-1	-1	-1	-1	944	284	-1	655	-1	-1	-1	-1	-1
945	284	-1	656	-1	-1	-1	-1	-1	946	-1	-1	657	-1	-1	-1	-1	-1
947	285	-1	658	-1	-1	-1	-1	-1	948	285	-1	659	-1	-1	-1	-1	-1
949	-1	-1	660	-1	-1	-1	-1	-1	950	286	-1	661	-1	-1	-1	-1	-1
951	286	-1	662	-1	-1	-1	-1	-1	952	-1	-1	663	-1	-1	-1	-1	-1
953	287	-1	664	-1	-1	-1	-1	-1	954	287	-1	665	-1	-1	-1	-1	-1

Appendix C

CNF for the Bivium B Keystream Equation

The keystream equation of Bivium B is of the form

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 = z$$

where $z \in \{0, 1\}$ is the keystream bit. Then we consider the Boolean polynomial

$$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4.$$

If $z = 1$ we convert $f(x_1, x_2, x_3, x_4)$ into CNF and if $z = 0$ we convert $\neg f(x_1, x_2, x_3, x_4)$. One the one hand, we obtain the following conjunctive normal form

$$\begin{aligned} CNF(f) = &(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4) \wedge \\ &(x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge \\ &(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge \\ &(\neg x_1 \vee x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3 \vee x_4) \end{aligned} \tag{C.1}$$

if $z = 1$. This leads to the following 8 inequality constraints:

$$\begin{aligned} x_1 - x_2 + x_3 - x_4 &\geq -1, \\ -x_1 - x_2 - x_3 - x_4 &\geq -3, \\ x_1 + x_2 - x_3 - x_4 &\geq -1, \\ -x_1 + x_2 - x_3 + x_4 &\geq -1, \\ x_1 - x_2 - x_3 + x_4 &\geq -1, \\ -x_1 - x_2 + x_3 - x_4 &\geq -1, \\ -x_1 + x_2 + x_3 - x_4 &\geq -1, \\ x_1 + x_2 + x_3 + x_4 &\geq 1. \end{aligned}$$

On the other hand, if $z = 0$, we get

$$\begin{aligned} CNF(\neg f) = &(\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge \\ &(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3 \vee x_4) \wedge \\ &(x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge \\ &(\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3 \vee \neg x_4) \end{aligned} \tag{C.2}$$

From this Boolean equation in CNF we derive the following 8 inequality constraints:

$$\begin{aligned}
-x_1 + x_2 - x_3 - x_4 &\geq -2, \\
x_1 - x_2 + x_3 + x_4 &\geq 0, \\
-x_1 - x_2 - x_3 + x_4 &\geq -2, \\
-x_1 + x_2 + x_3 + x_4 &\geq 0, \\
x_1 - x_2 - x_3 - x_4 &\geq -2, \\
x_1 + x_2 - x_3 + x_4 &\geq 0, \\
-x_1 - x_2 + x_3 - x_4 &\geq -2, \\
x_1 + x_2 + x_3 - x_4 &\geq 0.
\end{aligned}$$

This means every key stream equation yields 8 additional linear inequality constraints containing precisely 4 variables.

Bibliography

[1] 802.11i-2004 - ieee standard for local and metropolitan area networks - specific requirements-part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications: Amendment 6: Medium access control (mac) security enhancements.

[2] Distributed C2 brute force attack : Status page. web page, `http://www.marumo.ne.jp/c2/bf/status.html`. accessed on 12/02/2009.

[3] The eSTREAM project. `http://www.ecrypt.eu.org/stream/`.

[4] ILOG CPLEX. http://www.ilog.com/products/cplex/.

[5] C2 Block Cipher Specification, Revision 1.0. `http://www.4Centity.com`, 2003. used to be available online from 4C Entity, can be downloaded e.g. from: `http://edipermadi.files.wordpress.com/2008/08/cryptomeria-c2-spec.pdf`.

[6] 4C Entity. Wikipedia article, `http://en.wikipedia.org/wiki/4C_Entity`, accessed on 11/02/2009.

[7] 4C Entity. C2 facsimile s-box. `http://www.4centity.com/docs/C2_Facsimile_S-Box.txt`.

[8] Armknecht, F., and Krause, M. Algebraic attacks on combiners with memory. In *Advances in Cryptology – CRYPTO 2003* (2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 162–175.

[9] Babbage, S. A space/time tradeoff in exhaustive search attacks on stream ciphers. In *European Covention on Security and Detection, IEEE Conference Publication No. 408* (1995).

[10] Beigel, R. The polynomial method in circuit complexity. *Structure in Complexity Theory Conference* (1993), 82–95.

[11] Bernstein, D. A reformulation of Trivium. ECRYPT forum post, `http://www.ecrypt.eu.org/stream/phorum/read.php?1,448):`, 2006.

[12] Biham, E., and Biryukov, A. How to Strengthen DES Using Existing Hardware. In *Advances in Cryptology – ASIACRYPT '94, Proceedings* (1995), J. Pieprzyk and R. Safavi-Naini, Eds., vol. 917 of *Lecture Notes in Computer Science*, Springer, pp. 398–412.

[13] Biham, E., Biryukov, A., and Shamir, A. Miss in the middle attacks on IDEA and khufu. In *Fast Software Encryption - FSE '99* (1999), L. R. Knudsen, Ed., vol. 1636 of *Lecture Notes of Computer Science*, Springer, pp. 124– 138.

[14] Biham, E., and Dunkelman, O. Cryptanalysis of the a5/1 gsm stream cipher. In *Progress in Cryptology - Indocrypt 2000* (2000), B. Roy and E. Okamoto, Eds., vol. 1977 of *Lecture Notes in Computer Science*, pp. 43–51.

[15] Biham, E., Dunkelman, O., and Keller, N. The rectangle attack - rectangling the serpent. In *Advances in Cryptology – EUROCRYPT 2001* (2001), B. Pfitzmann, Ed., vol. 2045 of *Lecture Notes in Computer Science*, Springer, pp. 340–357.

[16] Biham, E., and Shamir, A. Differential cryptanalysis of the full 16-round DES. In *Advances in Cryptology – CRYPTO '92* (1992), E. F. Brickel, Ed., vol. 740 of *Lecture Notes in Computer Science*, pp. 487 – 496.

[17] Biham, E., and Shamir, A. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993. ISBN: 0-387-97930-1, 3-540-97930-1.

[18] Biryukov, A., and Shamir, A. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *Advances in Cryptology - ASIACRYPT 2000* (2000), T. Okamoto, Ed., vol. 1976 of *Lecture Notes of Computer Science*, pp. 1–13.

[19] Biryukov, A., and Shamir, A. Structural Cryptanalysis of SASAS. In *Advances in Cryptology – EUROCRYPT 2001, Proceedings* (2001), B. Pfitzmann, Ed., vol. 2045 of *Lecture Notes in Computer Science*, Springer, pp. 394–405.

[20] Biryukov, A., Shamir, A., and Wagner, D. Real time cryptanalysis of a5/1 on a pc. In *Fast Software Encryption - FSE 2001* (2001), M. Matsui, Ed., vol. 1978 of *Lecture Notes in Computer Science*, Springer, pp. 37 – 44.

[21] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsø, C. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems – CHES 2007, Proceedings* (2007), P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *Lecture Notes in Computer Science*, Springer, pp. 450–466.

[22] Borghoff, J., Knudsen, L. R., Leander, G., and Matusiewicz, K. Cryptanalysis of C2. In *Advances in Cryptology – CRYPTO 2009,* (2009), S. Halevi, Ed., vol. 5677 of *Lecture Notes in Computer Science*, Springer, pp. 250–266.

[23] Borghoff, J., Knudsen, L. R., Leander, G., and Matusiewicz, K. Cryptanalysis of C2 – extended abstract. Pre-proceedings of WEWoRC 2009, Graz, 2009.

[24] Borghoff, J., Knudsen, L. R., Leander, G., and Thomsen, S. S. Cryptanalysis of PRESENT-like ciphers with secret s-boxes. submitted to FSE 2011, 2010.

[25] Borghoff, J., Knudsen, L. R., and Matusiewicz, K. Analysis of Trivium by a simulated annealing variant. Proceedings of Ecrypt II Workshop on Tools for Cryptanalysis 2010, Egham, 2010.

[26] Borghoff, J., Knudsen, L. R., and Matusiewicz, K. Hill climbing algorithms and Trivium. to appear in Proceedings of Selected Areas in Cryptology 2010, Lecture notes in Computer Science, 2010.

[27] Borghoff, J., Knudsen, L. R., and Stolpe, M. Bivium as a mixed-0-1 programming problem – extended abstract. Pre-proceedings of WEWoRC 2009, Graz, 2009.

[28] Borghoff, J., Knudsen, L. R., and Stolpe, M. Bivium as a mixed-integer linear programming problem. In *Cryptography and Coding, 12th IMA International Conference* (2009), M. G. Parker, Ed., vol. 5921 of *Lecture Notes of Computer Science*, Springer, pp. 133–152.

[29] Cerny, V. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications 45* (1985), 41 – 51.

[30] Chinneck, John, W. Practical optimization: A gentle introduction. available under `http://www.sce.carleton.ca/faculty/chinneck/po.html`.

[31] Cho, J. Linear Cryptanalysis of Reduced-Round PRESENT. In *Topics in Cryptology - CT-RSA 2010* (2010), J. Pieprzyk, Ed., vol. 5985 of *Lecture Notes in Computer Science*, Springer, pp. 302–317.

[32] Clark, J. A., and Jacob, J. L. Fault injection and a timing channel on an analysis technique. In *Advances in Cryptology – EUROCRYPT 2002* (2002), L. R. Knudsen, Ed., vol. 2332 of *Lecture Notes of Computer Science*, Springer, pp. 181–196.

[33] Contini, S., and Yin, Y. L. Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In *Advances in Cryptology - ASIACRYPT 2006* (2006), X. Lai and K. Chen, Eds., vol. 4284 of *Lecture Notes in Computer Science*, Springer, pp. 37–53.

[34] Coppersmith, D. The data encryption standard (DES) and its strength against attacks. *IBM Journal of Research and Development 38*, 3 (1994), 243–250.

[35] Corless, R. M., Gonnet, G. H., Hare, D., and Jeffery, D. J. Lambert's W function in Maple. Maple Technical Newsletter 9, 1993.

[36] Courtois, N., Klimov, E., Patarin, J., and Shamir, A. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology – EUROCRYPT 2000* (2000), B. Preenel, Ed., vol. 1807 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 392–407.

[37] Courtois, N., and Meier, W. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology – EUROCRYPT 2003* (2003), E. Biham, Ed., vol. 2656 of *Lecture Notes of Computer Science*, Springer, pp. 644–644.

[38] Courtois, N. T., and Pieprzyk, J. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology - ASIACRYPT 2002* (2002), Y. Zheng, Ed., vol. 2501 of *Lecture Notes of Computer Science*, Springer, pp. 267–287.

[39] Dantzig, G. Linear programming. *Operation Research 50*, 1 (2002), 42 – 47. 50th Anniversary Issue.

[40] Dantzig, G., and Thapa, M. *Linear Programming: Introduction*. Springer, 1997.

[41] De Cannière, C., Biryukov, A., and Preneel, B. An introduction to block cipher cryptanalysis. *Proceedings of the IEEE 94*, 2 (February 2006), 346 – 356.

[42] De Cannière, C., and Preneel, B. Trivium - a stream cipher construction inspired by block cipher design principles. estream, ecrypt stream cipher. Tech. rep., of Lecture Notes in Computer Science, 2005.

[43] De Cannière, C., and Preneel, B. Trivium specifications. *eSTREAM, ECRYPT Stream Cipher Project* (2006).

[44] Deuflhard, P. *Newton Methods for Nonlinear Problems - Affine Invariance and Adaptive Algorithms*. Springer, 2004.

[45] Dinur, I., and Shamir, A. Cube attacks on tweakable black box polynomials. In *Advances in Cryptology - EUROCRYPT 2009* (2009), A. Joux, Ed., vol. 5479 of *Lecture Notes in Computer Science*, pp. 278–299.

[46] Eglese, R. Simulated annealing: A tool for operation research. *European Journal of Operational Research 46* (1990), 271–281.

[47] Eibach, T., Pilz, E., and Steck, S. Comparing and optismising two generic attacks on Bivium. In *Preproceedings of SASC 2008* (2008), C. De Cannière, Ed., pp. 57–68.

[48] Eibach, T., Pilz, E., and Völkel, G. Attacking bivium using SAT solvers. In *Theory and applications of satisfiability testing - SAT '08* (2008), H. Kleine Büning and X. Zhao, Eds., vol. 4996 of *Lecture Notes in Computer Science*, Springer, pp. 63–76.

[49] Feistel, H. Cryptography and computer privacy. *Scienctific America 288* (1973), 15–23.

[50] Feller, W. *An introduction to probability theory and its applications. Vol I*, 3rd ed. Wiley, 1968.

[51] Fraenkel, A. S., and Yesha, Y. Complexity of solving algebraic equations. *Information Processing Letters 10*, 4/5 (1980), 178–179.

[52] Geman, S., and Geman, D. Stochastic relaxation, Gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 6*, 6 (1984), 721–741.

[53] Golic, J. Cryptanalysis of alleged A5 stream cipher. In *Advances in Cryptology - EUROCRYPT 1997* (1997), W. Fumy, Ed., vol. 1233 of *Lecture Notes in Computer Science*, Springer, pp. 239–255.

[54] Golic, J. D. Linear cryptanalysis of stream ciphers. In *Fast Software Encryption - FSE '94* (1995), B. Preneel, Ed., vol. 1008 of *Lecture Notes in Computer Science*, Springer, pp. 154–169.

[55] Gomathisankaran, M., and Lee, R. B. Maya: A Novel Block Encryption Function. International Workshop on Coding and Cryptography 2009, Proceedings. Available: `http://palms.princeton.edu/system/files/maya.pdf` (2010/02/14).

[56] Gomory, R. An algorithm for mixed integer problem. Tech. Rep. RM-2597, The Rand Corporation, 1960.

[57] Hell, M., Johnassen, T., and Meier, W. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing 2*, 1 (2007), 86 – 93.

[58] Hellman, M. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory IT-26*, 4 (July 1980), 401–406.

[59] Hillier, Frederick, S., and Lieberman, Gerald, J. *Introduction to Operations Research*, 8th ed. McGraw-Hill, 2005.

[60] Hoos, H. The international conferences on theory and applications of satisfiability testing (sat). Web portal `http://www.satisfiability.org/`.

[61] Johnson, A. W., and Jacobson, S. H. A class of convergent generalized hill climbing algorithms. *Applied Mathematics and Computation 125*, 2-3 (2002), 359–373.

[62] Joux, A. *Algorithmic Cryptanalysis*. CRC Press LLC, Taylor and Francis Group, 2009.

[63] Kelsey, J., Kohno, T., and Schneier, B. Amplified boomerang attacks against reduced-round MARS and Serpent. In *Fast Software Encryption – FSE 2000* (2000), B. Schneier, Ed., vol. 1978 of *Lecture Notes in Computer Science*, Springer, pp. 75–93.

[64] Kerckhoffs, A. La cryptographie militaire. *Journal des sciences militaires IX* (Januar, February 1883), 5 – 83, 161 – 191.

[65] Khazaei, S. Re: A reformuation of TRIVIUM. Posted on the eSTREAM Forum, 2006. `http://www.ecrypt.eu.org/stream/phorum/read.php?1,448`.

[66] Kirkpatrick, S., Gelatt, Jr., C., and Vecchi, M. Optimization by simulated annealing. *Science 200* (1983), 671–680.

[67] Knudsen, L. R. *Block Ciphers - Analysis, Design and Applications.* PhD thesis, Department of Computer Science, University of Aarhus, November 1994.

[68] Knudsen, L. R. Truncated and higher order differentials. In *Fast Software Ecncrytion - Second International Workshop 1994* (1995), B. Preenel, Ed., vol. 1008 of *Lecture Notes of Computer Science*, Springer, pp. 196–211.

[69] Knudsen, L. R., and Meier, W. Cryptanalysis of an identification scheme based on the permuted perceptron problem. In *Advances in Cryptology - EUROCRYPT '99* (1999), J. Stern, Ed., vol. 1592 of *Lecture Notes in Computer Science*, Springer, pp. 363–374.

[70] Knudsen, L. R., and Robshaw, M. J. B. *The Block Cipher Companion.* Springer, 2011.

[71] Lai, X. *Communication and Cryptography - Two sides of the tapestry.* Kluwer Academic Publishers, 1992, ch. Higher order derivatives and differential cryptanalysis, pp. 227 – 233.

[72] Lai, X. On the design and security of block ciphers. In *ETH Series in Information Processing*, J. Massey, Ed., vol. 1. Hartung-Gorre Verlag, Konstanz,, 1992.

[73] Lai, X., Massey, J. L., and Murphy, S. Markov ciphers and differential cryptanalysis. In *Advances in Cryptology – EUROCRYPT '91* (1991), D. Davies, Ed., vol. 547 of *Lecture Notes of Computer Sciences*, Springer, pp. 17–38.

[74] Lipmaa, H., and Moriai, S. Efficient algorithms for computing differential properties of addition. In *Fast Software Encryption – FSE 2001* (2002), M. Matsui, Ed., vol. 2355 of *Lecture Notes in Computer Science*, Springer, pp. 35–45.

[75] Matsui, M. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - EUROCRYPT 93* (1994), T. Helleseth, Ed., vol. 765 of *Lecture Notes in Computer Science*, Springer.

[76] Matusiewicz, K. Personal communication.

[77] Matusiewicz, K. A note on conversions booleans - reals. Personal Communication, 2008.

[78] Maximov, A. *Some words on the cryptanalysis of stream ciphers*. PhD thesis, University of Lund, 2006.

[79] Maximov, A., and Biryukov, A. Two trivial attacks on Trivium. In *Selected Areas in Cryptography – SAC 2007* (2007), C. Adams and M. Wiener, Eds., vol. 4876 of *Lecture Notes of Computer Science*, Springer, pp. 36–55.

[80] McDonald, C., Charnes, C., and Pieprzyk, J. An algebraic analysis of trivium ciphers based on the boolean satisfiability problem. Cryptology ePrint Archive, Report 2007/129, 2007. `http://eprint.iacr.org/2007/129`.

[81] Mendel, F., Pramstaller, N., Rechberger, C., and Rijmen, V. Analysis of step-reduced SHA-256. In *Fast Software Encryption – FSE 2006* (2006), M. Robshaw, Ed., vol. 4047 of *Lecture Notes in Computer Science*, Springer, pp. 126–143.

[82] Menezes, A., Oorschot, P. v., and Vanstone, S. *Handbook of Applied Cryptography*. CRC Press LLC, 1997.

[83] Merkle, R. C. Fast Software Encryption Functions. In *Advances in Cryptology – CRYPTO '90, Proceedings* (1991), A. Menezes and S. A. Vanstone, Eds., vol. 537 of *Lecture Notes in Computer Science*, Springer, pp. 476–501.

[84] Murphy, S. The effectiveness of the linear hull effect. Tech. Rep. RHUL-MA-2009-19, Department of Mathematics, Royal Holloway, University of London, `http://www.ma.rhul.ac.uk/static/techrep/2009/RHUL-MA-2009-19.pdf`, 2009.

[85] Murphy, S. The return of the boomerang. Tech. Rep. RHUL-MA-2009-20, Department of Mathematics, Royal Holloway, University of London, `http://www.rhul.ac.uk/mathematics/techreports`, 2009.

[86] Nad, T. Personal communication, 2009.

[87] Nad, T., Lamberger, M., and Rijmen, V. Numerical solvers and cryptanalysis. *Journal of mathematical cryptology 3*, 3 (2009), 249 –263.

[88] National Bureau of Standards. Data encryption standard. Federal Information Processing Standard (FIPS), January 1977.

[89] Potra, F. A., and Wright, S. J. Interior-point methods. *Journal of Computational and Applied Mathematics 124*, 1 – 2 (2000), 281 – 302.

[90] Raddum, H. Cryptanalytic results on Trivium. eSTREAM report 2006/039, 2006. http://www.ecrypt.eu.org/stream/triviump3.html.

[91] Robshaw. Stream cipers. Tech. Rep. TR-701, RSA Laboratories, a division of RSA Data Security, Inc, July 1995.

[92] Schneier, B. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). In *Fast Software Encryption 1993, Proceedings* (1994), R. J. Anderson, Ed., vol. 809 of *Lecture Notes in Computer Science*, Springer, pp. 191–204.

[93] Schneier, B. *Applied Cryptography*, 2nd ed. John Wiley & Sons, Inc, 1996.

[94] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., and Ferguson, N. Twofish: A 128-Bit Block Cipher. Submitted as candidate for AES. Available: `http://www.schneier.com/paper-twofish-paper.pdf` (2010/02/05).

[95] Shannon, C. Communication theory of secrecy systems. *Bell System Technical Journal 28* (1949), 656 – 715.

[96] Standaert, F.-X., Piret, G., and Quisquater, J.-J. Cryptanalysis of block ciphers: A survey. Tech. Rep. CG-2003/2, UCL Crypto Group, 2003.

[97] Stinson, D. *Cryptography - Theory and Practice*, 3rd ed. Chapman & Hall/CRC, 2002.

[98] Vielhaber, M. Breaking ONE.FIVIUM by AIDA an algebraic IV differential attack. Cryptology ePrint Archive, Report 2007/413, 2007.

[99] Wagner, D. The boomerang attack. In *Fast Software Encryption - FSE 1999* (1999), L. R. Knudsen, Ed., vol. 1636 of *Lecture Notes in Computer Science*, Springer, pp. 156–170.

[100] Weinmann, R.-P. Algebraic S-Box recovery: the case of Cryptomeria. Presentation at Echternach Seminar on Symmetric Cryptography, 11/01/2008, Echternach, Luxembourg, available from `wiki.uni.lu/esc/docs/rpw_friday_algebraic_sbox_recovery.pdf`.

[101] Wolsey, L. A., and Nemhauser, G. L. *Integer and Combinatorial Optimization*. Wiley-Interscience, November 1999.

[102] Wu, H., and Preneel, B. Differential cryptanalysis of the stream ciphers py, py6 and pypy. In *Advance in Cryptology - EUROCRYPT 2007* (2007), M. Naor, Ed., vol. 4515 of *Lecture Notes in Computer Science*, Springer, pp. 276–290.

Index

AIDA, *see* cube attack
algebraic attack, 65–66
ANF, *see* algebraic normal form
attack
    adaptively chosen plaintext, 14
    chosen ciphertext, 14
    chosen plaintext, 14
    ciphertext only, 14
    known plaintext, 14

Bivium, 37–39
    Bivium A, 37
    Bivium B, 37
Boolean function, 127
Boolean polynomial, 128
boomerang attack, 55–57
    amplified boomerang, 56
bounding, 116
branch and bound, 115–121
branching, 115–116
branching variable, 116
break
    global deduction, 13
    information deduction, 13
    local deduction, 13
    total break, 13

C2, 19–20
characteristic
    differential, 48
      iterative, 53
    linear, 61
      iterative, 62
cipher
    Feistel, 17–18
    iterated, 17
    SP-network, 21–22
CNF, *see* conjunctive normal form
complexity
    data, 13
    processing, 13
    storage, 13
configuration acceptance probability, 123
configuration generation probability, 123
confusion, 12
constraint, 109
conversion method
    adapted standard, 133
    dual, 129–130
    Fourier, 130–131
    integer adapted standard, 135–136
    sign, 130
    standard, 129
cooling schedule, 124
    exponential, 125
    logarithmic, 125
cryptanalysis
    differential, 46–55
cryptographically secure pseudorandom, 26
Cryptomeria, *see* C2
cube attack, 57
cutting-plane, 121–122

difference, 46
difference distribution table, 47
differential, 51–52
    higher order, *see* cube attack
diffusion, 12
distinguishing attack, 13, 35

exhaustive search, 12

fathoming, 116–117
feasibility problem, 112
feasible set, 112

hill climbing algorithm, 123
hypothesis of stochastic equivalence, 52

incumbent, 117
integer programming problem, 113

integrality restriction, 109
IP, *see* integer programming problem

Kerckhoffs' principle, 9
keystream generator, 27

LFSR, *see* linear feedback shift register
linear approximation table, 60
linear complexity, 31–32
linear feedback shift register, 30–31
    maximum-length, 31
linear hull, 64–65
linear masks, 60
linear programming problem, 112
LP, *see* linear programming problem
LP-relaxation, 112–113

Markov chain, 51
Markov cipher
    differential cryptanalysis, 50
    linear cryptanalysis, 64
master polynomial, 58
maximum
    global, 109
    local, 109
maxterm, 58
minimal cover, 121
minimum
    global, 109
    local, 109
MIP, *see* mixed-integer linear programming problem
mixed-integer linear programming problem, 111
    feasible, 112
    infeasible, 112
    unbounded, 112
monomial degree, 131

neighborhood function, 123
neighborhood search algorithm, *see* hill climbing algorithm
NFSR, *see* non-linear feedback shift register
non-linear feedback shift register, 34
normal form
    algebraic, 128
    conjunctive, 127–128
    disjunctive, 127–128

objective function, 109, 112
objective value, 109
one-time pad, 11
optimal point, 112
optimization problem, 109, 112
    constrained, 110
    discrete, 123

perfect secrecy, 11
Piling-up Lemma, 63–64
PRESENT, 22–23
pruning, 117

rectangle attack, *see also* boomerang attack
representation
    dual, 129
    Fourier, 130
    sign, 130
    standard, 128–129
resynchronization attack, 35
right pair, 53

security
    computational, 12
    unconditional, 11
simulated annealing, 124–125
stream cipher
    synchronous, 27
    self-synchronizing, 28
superpoly, 58
symmetric encryption, 8–9

temperature, 124, 125
three-round test, 73–75
transition probability, 123
Trivium, 35–40

variable dichotomy, 115

wrong pair, 53