Tools in the Cryptanalysis of Symmetric Primitives

by

Tomislav Nad

A PhD Thesis
Presented to the Faculty of Computer Science in Partial Fulfillment of the Requirements for the PhD Degree

Assessors

Prof. Dr. Ir. Vincent Rijmen (TU Graz, Austria)
Prof. Dr. Gregor Leander (DTU, Denmark)

June 2012

Institute for Applied Information Processing and Communications (IAIK)
Faculty of Computer Science
Graz University of Technology, Austria

Abstract

In this thesis, we describe two different approaches to improve the cryptanalysis of symmetric primitives. In the first part we describe a new approach in cryptanalysis: numerical cryptanalysis. In this approach the cryptographic algorithm is represented as a system of equations over the reals and numerical methods and techniques are applied in order to solve such a system. We give a detailed analysis of each step of the approach and apply it to the stream cipher Trivium and to its reduced variants Bivium A and Bivium B. Although, we are not able to break Trivium or any of its variants, we show how techniques from numerical analysis can be used in cryptanalysis and discuss various future research directions.

In the second part of this thesis, we investigate two techniques based on differential cryptanalysis which are used in the cryptanalysis of ARX based hash functions. We improve and extend them leading to a new generation of tools which are used in attacks on several hash functions including SHA-2, HAS-160 or SIMD.

An application to SIMD-512 results in a distinguisher for the compression function. Due to this attack, the designers tweaked SIMD. However, we present a distinguisher for the full permutation and extend the attack to the full compression function of tweaked SIMD-512. An application to HAS-160, results in a semi-free-start collision for 65 (out of 80) steps which is so far the best known attack with practical complexity for HAS-160. The main idea of our attack is to combine both techniques by constructing two short characteristics which hold with high probability and connect them by a complex third characteristic using the non-linearity of the state update function. Finally, we apply our tools to SHA-256. We identify appearing problems and show how to overcome them. Important for the successful application of our tools to SHA-2, is the detection of contradictions for more conditions and the application of our advanced search strategy, which combines the search for differential characteristics with the search for a conforming message pair. We present a collision for 27 and a semi-free-start collision for 32 steps of SHA-256 with practical complexity. This significantly improves upon the best previously published (semi-free-start) collision attacks on SHA-256 for up to 24 steps.

Acknowledgements

First of all, I would like to thank my supervisor Vincent Rijmen for his support and trust in me. In particular, for his excellent scientific guidance during my PhD studies and for giving me free hand as much as possible. Many thanks also for providing interesting research topics and scientific discussions.

I would also like to thank Gregor Leander for being my external reviewer and examiner. I am grateful for his valuable comments and suggestions.

I would like to especially thank Florian Mendel, for sharing many research ideas, listening to my own ideas and for the numerous discussions on research, politics and life.

During my studies, I had the pleasure to work in and with a very successful team. I would like to thank my colleagues for their fruitful comments and discussions, but most importantly I would like to thank them for their friendships. Therefore, special thanks go to Mario Lamberger, Norbert Pramstaller, Christian Rechberger and Martin Schläffer. I also would like to thank my other colleagues from IAIK which were not directly involved in my research, but contributed in other ways positively to my working life. Thanks go to Michael Hutter, Mario Kirschbaum, Thomas Korak, Thomas Plos, Jörn-Marc Schmidt, Peter Teufl, Michaela Tretter-Dragovic and Erich Wenger. Furthermore, I would like to thank Kazumaro Aoki and Sebastiaan Indesteege who stayed with our group for several weeks or months during the years of my PhD studies.

Finally, I would like to thank all my friends for the welcome distractions in my spare time. Most of all, I want to thank my family for their endless support and my girlfriend for her understanding, patience and love.

Tomislav Nad
Graz, June 2012

Table of Contents

List of Tables

List of Figures

1 Overview

Cryptanalysis is the study of methods to analyse cryptographic primitives attempting to find weaknesses in these primitives. These weaknesses are then used to break the cryptographic algorithm. Breaking means bypassing the algorithm and breaking confidentiality, forging data or tampering an authentication process etc.. Cryptanalysis usually excludes methods that do not target weaknesses in the cryptographic algorithm such as bribery, physical coercion, burglary, keystroke logging, and social engineering, although these types of attacks are important. The methods and techniques in cryptanalysis have changed drastically through the history of cryptography, adapting to the increasing cryptographic complexity and increasing computational power. In the early 1990's first Biham and Shamir [BS92], and later Matsui [MY92] published two general techniques to cryptanalyse symmetric cryptographic algorithms: differential and linear cryptanalysis. These techniques have been successfully used to break many existing ciphers. Since the upcoming of these cryptanalytic methods, new design strategies have been proposed to resist this kind of analysis. However, a new type of attack could cause a complete breakdown of security.

The trend of developing cryptographic primitives seems to go to more complex designs. Especially, for hash functions a significant increase in the design complexity can be observed, like the transition from SHA-1 to SHA-2 or the design of several SHA-3 candidates. Due to this increased complexity, the cryptanalysis of hash functions has become a more challenging task and the development of new tools has become necessary.

In this thesis, we investigate two different approaches to improve the cryptanalysis of symmetric primitives. Although, both topics are part of cryptanalysis, the underlying techniques are quite different. Therefore, we split this thesis into two parts.

1.1 Numerical Methods in Cryptanalysis

Since the upcoming of linear and differential attacks, new design strategies have been proposed to resist these attacks. In the first part of this thesis, we investigate a new approach in cryptanalysis: numerical cryptanalysis.

Similar as in algebraic attacks, we represent the cryptographic algorithm as a system of equations. The system of equations is constructed such that there is a correspondence between its solutions and some secret information of the cryptographic primitive (for instance, the secret key of a block cipher). In our approach we apply methods and techniques from numerical analysis, which is a large and well-studied field of research. Many efficient algorithms exist to solve linear and non-linear systems of equations. We discuss how these methods can be used in the cryptanalysis of symmetric primitives. We give a detailed description of the technique and provide an analysis of each step of the approach. Finally, we apply the technique to the stream cipher Trivium and to its reduced variants Bivium A and Bivium B.

1.2 Tools in Differential Cryptanalysis

Differential cryptanalysis is one of the most important analysis techniques for symmetric primitives. It turned out to be of particular interest in the cryptanalysis of hash functions. Hash functions are an important cryptographic primitive and are used for data integrity, message authentication, digital signatures, password protection, pseudo-random number generation, key derivation, malicious code detection and in many other applications and cryptographic protocols.

While hash functions did not get a lot of attention by the cryptographic community, this changed with the breakthrough results of Wang et al. in 2004. Since then many attacks based on differential cryptanalysis have been presented for several well-known algorithms such as SHA-0, SHA-1 or MD5. The transition from SHA-1 to the SHA-2 family was proposed by the *National Institute of Standards and Technology* (NIST) as a first solution. As another consequence of these results NIST has initiated an open competition for a new hash function standard, called SHA-3.

In many designs the complexity has increased compared to previous hash functions. Larger states, more non-linear operations, more rounds and more complicated state updates are the consequence. Due to this increased complexity, the analysis of hash functions has become more difficult. Therefore, finding differential characteristics and conforming input pairs has become a more challenging task and the development of new tools has become necessary.

In the second part of this thesis, we analyse the most successful collision attacks on SHA-1. We describe two distinct techniques in detail. Furthermore, we improve and extend them leading to a new generation of tools which are used in attacks on several hash functions including SHA-2, HAS-160 or SIMD.

1.3 Main Contributions

In this thesis, we describe parts of the work done by the author during his PhD studies. The main contribution has been made in the development and improvement of new and existing tools for the cryptanalysis of symmetric primitives. Due to the increased design complexity of cryptographic primitives, the necessity for new tools has become apparent in the cryptographic community.

First a new technique which connects the research of numerical methods with the research of cryptanalysis has been investigated and new future research directions have been opened [LNR09a].

Due to the increased design complexity of many ARX based hash functions, automatic tools are needed in order to construct attacks. Therefore, the development of complex automated tools has been focused. The work has been build upon techniques in the cryptanalysis of SHA-1 and resulted in a new generation of tools which implementation has been partially published under the GPL-3.0 license [Nad10]. The new set of tools has been applied to several hash functions. The first application has resulted in the best attack with practical complexity on the Korean hash function standard HAS-160 [MNS11a]. Due to the importance of the hash function family SHA-2, the tools have been also applied to SHA-256 resulting in collision and semi-free-start collision attacks [MNS11b], which have significantly improved upon the best previously published (semi-free-start) collision attacks on SHA-256. Furthermore, the tools have been successfully applied to the ISO standards RIPEMD-128 and RIPEMD-160 [MNS12, MNSS12] and the SHA-3 candidate SIMD [MN09, MN11].

Additionally, the author has published practical collisions for the hash function Boole [MNS09] and has contributed to the compact implementation of lightweight block ciphers [EGG+12].

Part I

Numerical Methods in Cryptanalysis

2 Introduction

In the last years many researchers try to find new methods and techniques to analyse different types of cryptographic algorithms. Recent attacks are representing the cryptographic algorithm as a system of equations. The system of equations is constructed such that there is a correspondence between its solutions and some secret information of the cryptographic primitive (for instance, the secret key of a block cipher). As these systems are usually very sparse, over-defined, and structured, it is conjectured that they may be solved much faster than generic non-linear equation systems. One important feature is that the attacker needs only a low amount of data to set up a system of equations describing the cipher and determining the key (solutions). This makes such attacks more threatening than differential or linear cryptanalysis which typically require a large amount of data (known/chosen plainttext, ciphertexts) . A slightly different variant is to combine differential cryptanalysis with algebraic analysis, by solving algebraic relations arising from differential characteristics more efficiently.

These attacks mainly use algebraic methods (e.g. Gröbner Bases as in [BPW06]) or SAT solvers (cf. [EPV08]). In this thesis we study a different approach. In our approach we use techniques and methods from numerical analysis to solve systems of equations originating from a cryptographic algorithm.

2.1 Algebraic Cryptanalysis

Algebraic cryptanalysis received much attention, especially after it was proposed in [CP02] against the AES and Serpent block ciphers. In the recent years a lot of work has been done in this field [CM03, CB07, Alb08, CNO08, AC09, ACD+10].

Several algorithms have been proposed to solve system of equations in an algebraic field, e.g. the Buchberger algorithm, XL and variants [CKPS00, CP02, YCC04], F_4 [Fau99] and F_5 algorithm [Fau02]. Most of the attacks focus on the efficient computation of a Gröbner basis for the system considering structured properties of the targeted cipher. Gröbner bases are a proven tool for solving polynomial systems. A Gröbner basis for a system of polynomials is an equivalence system that has several desirable geometric and algorithmic properties that are not visible from the original system. This allows to efficiently compute the roots of a Gröbner basis and therefore the solutions of the original system of equations. The computation of a Gröbner basis can be easy for simple problems but the computational complexity increases exponentially with the complexity of the underlying problem. Hence, the worst case complexity of any algorithm that computes Gröbner bases in full generality must be high. The time and memory required to calculate a Gröbner basis depend very much on the structure of the original system of equations, like the variable ordering or monomial ordering. Thus, examples in three or four variables with polynomials of degree three or four may already fail to terminate in reasonable time or exceed available memory even on very fast machines. Therefore, it is important to analyse structural properties of the system and exploit them to decrease the computational complexity for specific problems as cryptographic algorithms.

2.2 Logical Cryptanalysis

Logical cryptanalysis has been introduced by Massacci and Marraro [MM00] as a general framework for encoding properties of cryptographic algorithms into SAT problems. The initial goal was to generate SAT benchmarks that are controllable and that share the properties of real-world problems and randomly generated problems. Since then several applications of SAT solvers in cryptanalysis have been described in the literature [FMM03, JJ05, MZ06]. Despite the initial goal in the last years researchers used SAT solvers to break ciphers or to improve existing cryptanalysis techniques by solving computationally expensive parts with SAT solvers [MZ06, BCJ07, MCP07, EPV08, SNC09]. The general approach is to convert a system of equations describing a cryptographic algorithm to Boolean expressions in Conjunctive Normal Form (CNF). Afterwards one uses off-the-shelf SAT-solvers to solve the SAT problem and therefore the original system of equations. SAT solvers have made a lot of progress in recent years, with both theoretical and practical improvements. SAT solvers are carefully designed to run on a large range of problems with no tuning required by users. Both in research and industry many problems are solved by mapping them to CNF and solving them using highly tuned SAT solvers. Usually, the mapping to CNF can lose much of the structure of the original problem. However, the performance of SAT solvers is often able to offset this loss of structural information. The majority of the state-of-the-art SAT solvers are based on the branch and backtracking algorithm called DLL algorithm [DLL62]. This algorithm searches for a solution by recursively choosing a variable and assign it to

one value and then to the other. At each stage of search a propagation step is performed, which attempts to imply the assignments to as many unassigned variables as possible based on the assignments made so far. In this stage also clauses which cannot be satisfied any more are detected and a backtracking process starts. The major problems for using SAT solvers in cryptanalysis are that the computational effort of SAT solvers increases exponentially with the size and the complexity of the underlying problem and as pointed out in [MM00] cryptographic algorithms provide hard and complex problems for SAT solvers.

2.3 Numerical Cryptanalysis

In this thesis we investigate a new approach in cryptanalysis. Again a system of equations representing a cryptographic algorithm is targeted. In our approach we apply methods and techniques from numerical analysis, which is a large and well-studied field of research. Many efficient algorithms exist to solve linear and non-linear systems of equations. Our approach and an application on the stream cipher Trivium has been published in [LNR09b].

A similar approach was investigated by Borghoff et.al. [BKS09] where a system of equations over $GF(2)$ is transformed in an optimization problem over the reals. Additionally, the authors published an approach using simulated annealing [BKM10] which is a probabilistic global optimization technique. Another approach we consider as part of numerical cryptanalysis was investigated by Tischhauser [Tis11] where the system of equations are modelled as a continuous optimisation problem where the equations are not continuously differentiable to avoid high degrees.

In Figure 2.1 our approach is outlined. From the system of equations over $GF(2)$ we use conversion methods to create an equivalent system over the reals. At this point one can apply numerical methods. The computed solution can be converted back to $GF(2)$ (with restrictions) which results in a solution for the original system.

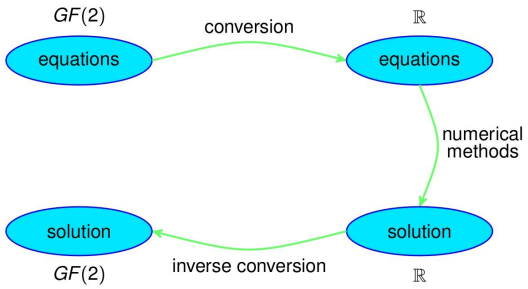


Figure 2.1: Basic approach of *Numerical Cryptanalysis*

Numerical solvers are methods to approximate solutions for equations and systems of equations. We are interested in the special case of solving non-

linear polynomial systems of equations. For such system a variety of different techniques and methods exist. A key advantage is that except for building the Boolean system of equations every step of our approach works fully automated.

2.4 Application of Numerical Cryptanalysis

In theory numerical cryptanalysis can be applied on any cryptographic primitive. However, we are targeting ciphers which equation structure offers low algebraic degree, low number of equations and simplicity. In general stream ciphers provide such properties compared to more "heavy" primitives like hash functions. A stream cipher is a symmetric key cipher, where the plaintext bits are combined with a pseudorandom bit stream (keystream). The keystream-bits are derived from a key and an initial value. Stream ciphers are in general fast and have limited or no error propagation, which makes them very useful for situations where transmission errors are highly probable or the plaintext is of unknown length. We apply our approach on Trivium [Rob08], which is recommended by the eStream project [ECR] and its reduced variants.

2.5 Outline

In Chapter 3 we present different methods to convert a Boolean equation to a polynomial over the reals. We analyse in detail the different conversion results and show their advantages and disadvantages according to the structure of the Boolean equation. We introduce two new conversion techniques in order to reduce the complex structure of the conversion results. In Chapter 4 we give a brief overview on numerical analysis and introduce the terminology and definitions which are necessary for the understanding of the numerical methods used in this thesis. Furthermore, we show which properties of the converted equations are coherent with desired properties in numerical analysis. Finally, we apply our approach on the stream cipher Trivium in Chapter 5. We define systems of equations for Trivium and two reduced variants, apply different conversion methods and use the presented numerical methods to search for a solution. In Chapter 6 we present a summary and conclude the first part of this thesis by discussing open problems and further research directions.

3

Conversion of Boolean Equations to the Real Domain

In cryptography typically non-linear Boolean equations appear. Sometimes they can be well enough linearised. Solvers for linear Boolean equations are well researched and methods are already available. Non-linear equations are rather difficult to solve, especially if the system consists of a high amount of unknowns and high degrees. In our approach we want to use numerical methods to solve such equations. Since these numerical solvers are defined to operate on real numbers, a conversion of Boolean equations to equations over the reals has to be done, which is the first step in our approach (see Section 2.3). Therefore, we need to choose an appropriate representation of Boolean equations as polynomials over the reals.

Many scientific fields work with the representation of Boolean functions as polynomials over the reals, e.g. optimization research, circuit complexity or machine learning. In operations research the problem to formulate logical conditions is about representation of logical conditions/equations as inequalities over the reals or integer numbers. In machine learning there are behaviours to learn, which can be expressed as logical conditions. The conditions have to be learned with sophisticated algorithms, where the representation of the logical equations is important. Furthermore, in the field of circuit complexity, the polynomial representation of Boolean equations is a basic problem.

For different types of applications different representations are used. In several publications [Bei93, BB99, NS94] four types occur consistently. These are

- Standard representation,
- Dual representation,

- Sign representation and
- Fourier representation.

Fourier representation is the most frequently used type in the theory of circuit complexity [Bei93], computational intelligence [Mon94] or theory of Boolean functions [KKL88]. There is no consensus on names for the different representations. The names used in this thesis are taken from [Bei93]. In order to represent Boolean functions (equations) as polynomials over the reals, the values for `true` and `false` have to be mapped to real numbers. From this mapping the resulting representation for the Boolean operators follows.

In this chapter we analyse different representations and conversion techniques. We provide the mathematical definitions of each representation and the needed lemmata for the Boolean operators. Each type of representation is analysed in order to derive properties which classifies the conversion methods. We show that one has a huge influence on the structure of the resulting equations over the reals, since each conversion method lead to a different system of polynomials with different complexity. Furthermore, we show how different properties can be exploited to decrease this complexity leading to new advanced conversion methods.

3.1 Conversion and Complexity

In the next sections we convert Boolean functions and equations, respectively. Therefore, we define Boolean functions in the following way.

Definition 3.1 (Boolean Function)**.** *A Boolean function is a function of the form* $f : B^k \to B$, *where* $B = \{\texttt{false}, \texttt{true}\}$ *is a Boolean domain and where* k *is a non-negative integer.*

Any Boolean function or equation can be converted into an equation over the reals, using one of the representation types presented in the following sections. The structure of the resulting polynomials depends on the Boolean function and on the chosen representation. Moreover, one can use additional techniques to reduce the complexity of the structure. We define two such techniques and name them *adapted conversion* and *splitting conversion*. In order to be able to give a general analysis of the conversion methods, we assume that the Boolean equations are in a normal form. There exist several different normal forms, which are used for different types of applications. The two basic normal forms are *Conjunctive Normal Form* and *Disjunctive Normal Form*.

Definition 3.2 (Conjunctive Normal Form)**.** *Let* f *be a Boolean function. We say that* f *is in Conjunctive Normal Form if it is a conjunction of clauses, where a clause is a disjunction of literals.*

Definition 3.3 (Disjunctive Normal Form)**.** *Let* f *be a Boolean function. We say that* f *is in Disjunctive Normal Form if it is a disjunction of clauses, where a clause is a conjunction of literals.*

In cryptography the *Algebraic Normal Form* is widely used since the the equations are mostly defined over $\mathbb{F}_2(\cong B)$.

Definition 3.4 (Algebraic Normal Form [OK94])**.** *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a function. We call the Algebraic Normal Form of f, the following expression of f as a polynomial*

$$f(x_1, \cdots, x_n) = \sum_{S \subseteq [n]} a_S \prod_{i \in S} x_i,$$

where $a_S \in \mathbb{F}_2$ and $[n] = \{1, \cdots, n\}$.

We choose a different notation to avoid using the same operators in Boolean and real equations.

Definition 3.5 (Algebraic Normal Form (Boolean notation))**.** *Let f be a Boolean function. The ANF of f in Boolean notation is the following expression.*

$$\bigoplus_{I \subseteq M} a_I \wedge (\bigwedge_{i \in I} x_i),$$

where $M = \{1, \cdots, n\}$ and $a_I \in \{\texttt{false}, \texttt{true}\}$. In this case a_I determines the existence of the corresponding conjunction.

In the next sections a Boolean system of equations in ANF will be converted into equations over the reals using the four different types of representation. Afterwards the resulting equations are analysed to show which type leads to the most simple equations over the reals.

An arbitrary Boolean system of equations in ANF using Definition 3.5 is defined as follows

$$\begin{pmatrix} \bigoplus_{I_1 \subseteq M} a_{I_1} \wedge (\bigwedge_{i \in I_1} x_i) \\ \vdots \\ \bigoplus_{I_m \subseteq M} a_{I_m} \wedge (\bigwedge_{i \in I_m} x_i) \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}, \tag{3.1}$$

for $b_i \in \{\texttt{false}, \texttt{true}\}$ and $i = 1, \ldots, m$. Converting a system of Boolean equations is done by converting each single equation separately.

3.1.1 Conversion Algorithm

The conversion of a Boolean equation using one of the representation types consists basically of two steps. First of all we have to convert all AND operations and afterwards each XOR operation. This order is given by the distributive law in Boolean algebra. The lemmata from the representation types are used to create an algorithm for the conversion. In order to specify the algorithm two functions are needed: $C_\wedge(a, b)$ and $C_\oplus(a, b)$. These two functions, representing the AND and XOR operations, are different for every type of representation

Algorithm 1 Conversion Algorithm

Require: Equation in ANF
1: Create a list A of all XOR operands
2: **for** $i = 1$ to $i =$sizeof(A) **do**
3: Create a list B of all variables in $A[i]$
4: **for** $i = 2$ to $i =$sizeof(B) **do**
5: $B[1] = C_\wedge(B[1], B[i])$
6: **end for**
7: $A[i] = B[1]$
8: **end for**
9: **for** $i = 2$ to $i =$sizeof A **do**
10: $A[1] = C_\oplus(A[1], A[i])$
11: **end for**
12: Print A[1]

and therefore are defined separately. The straightforward algorithm is given in Algorithm 1. The algorithm returns in polynomial time an equation over the reals representing the given Boolean equation. Basically the algorithm works for every representation despite of the different definitions of $C_\wedge$ and $C_\oplus$.

3.1.2 Complexity of Converted Equations

To analyse each conversion method a measurement for the complexity of the resulting system of equations is needed, to argue which type leads to "better" equations. In the context of numerical methods it is hard to define the difficulty of the system a priori. However, a rough classification can be done using the represented factors in this chapter. For example we assume that a Boolean equation converting to a polynomial over the reals resulting in a low monomial degree or sparsity grade, is less complex and therefore "easier" than a polynomial with high total and monomial degree. However, it is obvious that the difficulty of the system depends on more than these factors, like the dependencies between the equations of the system, but they are nevertheless useful for a rough estimation of the complexity and as we show in Chapter 5 the performance and accuracy changes according to these factors.

In the following we define these factors for single equations, since each equation of a system has to be converted separately. The following definitions are taken from [CLO07].

Definition 3.6 (Monomial)**.** *A monomial in $x_1, \cdots, x_n$ is a product of the form*

$$x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n},$$

where all of the exponents $\alpha_1, \cdots, \alpha_n$ are non-negative integers. The total degree of this monomial is defined as $|\alpha| = \alpha_1 + \cdots + \alpha_n$.

Definition 3.7 (Polynomial over the reals)**.** *A polynomial over the reals f in $x_1, \cdots, x_n$ with coefficients in $\mathbb{R}$ is a finite linear combination (with coefficients in $\mathbb{R}$) of monomials. We write a polynomial f in the form*

$$f = \sum_{\alpha} a_\alpha x^\alpha, a_\alpha \in \mathbb{R},$$

where the sum is over a finite number of n-tuples $\alpha = (\alpha_1, \cdots, \alpha_n)$. The set of all polynomials in $x_1, \cdots, x_n$ with coefficients in $\mathbb{R}$ is denoted by $\mathbb{R}[x_1, \cdots, x_n]$.

Definition 3.8. *Let $f = \sum_\alpha a_\alpha x^\alpha$ be a polynomial in $\mathbb{R}[x_1, \cdots, x_n]$.*

(i) We call a_α the coefficient of the monomial x^α.

(ii) If $a_\alpha \neq 0$, then we call $a_\alpha x^\alpha$ a term of f

(iii) The total degree of f, denoted $deg(f)$, is the maximum $|\alpha|$ such that a_α is nonzero.

Definition 3.9 (Multilinear Polynomials)**.** *We call a polynomial multilinear, if each variable in each monomial has a degree of at most* 1*.*

For the analysis of the conversion methods additional definitions are needed.

Definition 3.10 (Monomial Degree)**.** *Let f be a polynomial over the reals. The monomial degree of f, denoted $mdeg(f)$, is the number of monomials in f.*

Definition 3.11 (Sparsity Grade)**.** *Let S_r be a system of equations and f an equation in S_r. The Sparsity Grade of f, denoted $sg(f)$, is defined as*

$$sg(f) = \frac{mdeg(f)}{N},$$

where N is the number of different variables in S_r.

Boolean equations may have a special structure which influences the conversion result, e.g. same variables occur in different monomials. This can lead to a less complex polynomial over the reals. To include this observation in the analysis the following definition is necessary.

Definition 3.12 (Variable Sharing)**.** *Let f be a Boolean equation in ANF. If there exist I and J where a_I and a_J are true and $I \cap J \neq \{\}$, then f is called a variable sharing Boolean equation.*

These definitions include two important properties of an equation over the reals. Concerning numerical solvers both are important. The higher the total degree the more difficult a system of equations seems to be. The sparsity grade is a way to measure the sparsity of a system of equations. We speak of a sparse system if the sum of the sparsity grade over all equations is low. We assume the smaller this value the easier a solution can be determined. Obviously, these values depend mostly on the given equations, but the choice of the representation type, which is used for the conversion, can influence the results significantly. In the following sections the defined factors are used to analyse and afterwards to compare the four types of representation.

3.2 Conversion using Standard Representation

3.2.1 Standard Representation

Definition 3.13. *Let $f(x_1, \cdots, x_n)$ be a Boolean function, $x_i \in \{\texttt{true}, \texttt{false}\}$ for $i = 1, \cdots, n$ and $t : \{\texttt{false}, \texttt{true}\} \to \{0, 1\}$ with*

$$\begin{aligned} t(\texttt{false}) &= 0 \\ t(\texttt{true}) &= 1. \end{aligned}$$

$s(y_1, \cdots, y_n)$ is the standard representation of $f(x_1, \cdots, x_n)$, if

$$\forall x_i : s(t(x_1), \cdots, t(x_n)) = t(f(x_1, \cdots, x_n)).$$

Lemma 3.1. *Let $f(x_1, \cdots, x_n)$ be a Boolean function and $s(y_1, \cdots, y_n)$ its standard representation.*

a) $f(x_1, x_2) = x_1 \wedge x_2 \implies s(y_1, y_2) = y_1 \cdot y_2$, $y_i = t(x_i)$ *for* $i = 1, 2$.

b) $f(x_1, x_2) = x_1 \vee x_2 \implies s(y_1, y_2) = y_1 + y_2 - y_1 \cdot y_2$, $y_i = t(x_i)$ *for* $i = 1, 2$.

c) $f(x_1) = \neg x_1 \implies s(y_1) = 1 - y_1$, $y_1 = t(x_1)$

d) $f(x_1, x_2) = x_1 \oplus x_2 \implies s(y_1, y_2) = y_1 + y_2 - 2 \cdot y_1 \cdot y_2$, $y_i = t(x_i)$ *for* $i = 1, 2$.

Proof. The proof for Lemma 3.1 can be done by a truth table for each Boolean operator, to show that the standard representation has the same results as the Boolean function. For the truth tables the representation function s has to be evaluated, which results in the following tables.

x_1	x_2	$x_1 \wedge x_2$		y_1	y_2	$s(y_1, y_2)$
false	false	false		0	0	0
false	true	false		0	1	0
true	false	false		1	0	0
true	true	true		1	1	1

x_1	x_2	$x_1 \vee x_2$		y_1	y_2	$s(y_1, y_2)$
false	false	false		0	0	0
false	true	true		0	1	1
true	false	true		1	0	1
true	true	true		1	1	1

x_1	x_2	$x_1 \oplus x_2$		y_1	y_2	$s(y_1, y_2)$
false	false	false		0	0	0
false	true	true		0	1	1
true	false	true		1	0	1
true	true	false		1	1	0

x_1	$\neg x_1$		y_1	$s(y_1)$
`false`	`true`		0	1
`true`	`false`		1	0

Due to the truth tables $t(s(y_1, y_2)))$ and $t(s(y_1))$ respectively have the same values as the Boolean function for the specific operator. □

Any combination of the operators can be represented in standard form, since Lemma 3.1 can be used recursively in consideration of the distributive law in Boolean algebra.

Example 3.1. *For better understanding, a short example illustrates how a Boolean function is represented as a polynomial over the reals by using standard representation. Lets consider the following Boolean function:*

$$f(x_0, x_1, x_2, x_3) = x_3 \wedge x_1 \wedge x_0 \oplus x_3 \wedge x_1 \oplus x_3 \wedge x_0 \oplus x_3 \oplus x_1 \oplus x_0 \oplus \mathtt{true}.$$

By applying Lemma 3.1 recursively, under the consideration of the distributive law in Boolean algebra, the resulting polynomial over the reals using standard representation is

$$s(y_0, y_1, y_2, y_3) = 1 - y_0 - y_1 + 2y_0y_1 - y_3 + y_0y_3 + y_1y_3 - y_0y_1y_3.$$

An evaluation of $f(\mathtt{false}, \mathtt{true}, \mathtt{false}, \mathtt{true})$ *and* $s(0, 1, 0, 1)$ *leads to*

$$\begin{aligned} f(\mathtt{false}, \mathtt{true}, \mathtt{false}, \mathtt{true}) = {} & \mathtt{true} \wedge \mathtt{true} \wedge \mathtt{false} \oplus \mathtt{true} \wedge \mathtt{true} \oplus \\ & \mathtt{true} \wedge \mathtt{false} \oplus \mathtt{true} \oplus \\ & \mathtt{true} \oplus \mathtt{false} \oplus \mathtt{true} = \mathtt{false} \end{aligned}$$

and

$$s(0, 1, 0, 1) = 1 - 0 - 1 + 2 \cdot 0 \cdot 1 - 1 + 0 \cdot 1 + 1 \cdot 1 - 0 \cdot 1 \cdot 1 = 0.$$

The results of the evaluation of the Boolean function and the standard representation are equivalent.

3.2.2 Conversion

Using Algorithm 1, the definition of $C_\wedge$ and $C_\oplus$ follows directly from Lemma 3.1a) and Lemma 3.1d)

$$\begin{aligned} C_\wedge(a, b) &= a \cdot b, \\ C_\oplus(a, b) &= a + b - 2 \cdot a \cdot b. \end{aligned}$$

To analyse the conversion based on the standard representation it is sufficient to look only at one equation of system (3.1):

$$\bigoplus_{I \subseteq M} a_I \wedge \Big(\bigwedge_{i \in I} x_i\Big) = b. \tag{3.2}$$

According to Algorithm 1 the AND operators have to be converted first by applying $C_\wedge$ on

$$a_I \wedge \bigwedge_{i \in I} x_i.$$

In the standard representation the AND operator is converted into a multiplication and therefore a conversion results in

$$c(\vec{y}_I) := a_I \cdot \prod_{i \in I} y_i \text{ for } \vec{y}_I = (y_{i_1}, \cdots, y_{i_{|I|}}). \tag{3.3}$$

In the next step the XOR operations need to be converted

$$\bigoplus_{I \subseteq M} c(\vec{y}_I). \tag{3.4}$$

For a better understanding of this process we express Algorithm 1 as following recursive formula:

$$C_\oplus(c(\vec{y}_{I_1}), C_\oplus(c(\vec{y}_{I_2}), C_\oplus(\cdots, C_\oplus(c(\vec{y}_{I_{|\mathcal{P}(\mathcal{M})|-1}}), c(\vec{y}_{I_{|\mathcal{P}(\mathcal{M})|}}))))).$$

As an intermediate result the first two evaluations of $C_\oplus$ in the recursion result in

$$\begin{aligned}
C_\oplus(\cdots) &= t(a_{I_1}) \prod_{i \in I_1} y_i + C_\oplus(\cdots) - 2 \cdot t(a_{I_1}) \prod_{i \in I_1} y_i \cdot C_\oplus(\cdots) \\
&= t(a_{I_1}) \prod_{i \in I_1} y_i + t(a_{I_2}) \prod_{i \in I_2} y_i + C_\oplus(\cdots) \\
&\quad -2 \cdot t(a_{I_2}) \prod_{i \in I_2} y_i \cdot C_\oplus(\cdots) \\
&\quad -2 \cdot t(a_{I_1}) \prod_{i \in I_1} y_i \cdot t(a_{I_2}) \prod_{i \in I_2} y_i \\
&\quad -2t(a_{I_1}) \prod_{i \in I_1} y_i \cdot C_\oplus(\cdots) \\
&\quad +4 \cdot t(a_{I_1}) \prod_{i \in I_1} y_i \cdot t(a_{I_2}) \prod_{i \in I_2} y_i \cdot C_\oplus(\cdots).
\end{aligned}$$

The complete evaluation of the recursion leads to a polynomial over the reals representing the Boolean Equation (3.2) using standard representation

$$\begin{aligned}
&= \sum_{I \subseteq M} t(a_I) \prod_{i \in I} y_i - 2 \sum_{\substack{I,J \subseteq M \\ I \neq J}} t(a_I) t(a_J) \prod_{i \in I} y_i \prod_{j \in J} y_j \\
&\quad + 4 \sum_{\substack{I,J,K \subseteq M \\ I \neq J \neq K}} t(a_I) t(a_J) t(a_K) \prod_{i \in I} y_i \prod_{j \in J} y_j \prod_{k \in K} y_k - \cdots \\
&\quad + (-2)^{|\mathcal{P}(\mathcal{M})|-1} \prod_{I \subseteq M} t(a_I) \prod_{i \in I} y_i = t(b).
\end{aligned} \tag{3.5}$$

We denote Equation (3.5) as the converted polynomial over the reals p_s.

3.2.3 Analysis

Total Degree

The total degree of p_s depends on the number of variables in the Boolean equation. In the worst case we have $deg(p_s) = n$, due to Equation (3.3) where the maximum number of different variables multiplied can only be n. Higher degrees could occur during the conversion of the XOR operations, which is prevented by the multilinearity property ($x^l = x, \forall l \in \mathbb{N}$) of the standard representation. Hence, the total degree depends directly on $|\bigcup I|$ where a_I is equal `true`. Let k_t be the total degree of p_s then

$$k_t := |\bigcup_{\substack{I \subseteq M \\ a_I = \texttt{true}}} I|.$$

Monomial Degree

Equation (3.2) consists of $|\mathcal{P}(M)|$ different monomials. Obviously, the monomial degree for the converted equation is increased significantly. The monomial degree of p_s is determined by every possible product of two different monomials in (3.2), every possible product of three monomials etc.. The final maximum amount of monomials in equation (3.5) is

$$\sum_{i=1}^{|\mathcal{P}(M)|} \binom{|\mathcal{P}(M)|}{i},$$

if for all $I \in \mathcal{P}(M)$, a_I is equal `true`. Hence, the monomial degree directly depends on the number of coefficients a_I which are equal to `true`. Let k_s be defined as

$$k_s := \sum_{I \subseteq M} t(a_I),$$

then the monomial degree of p_r is

$$mdeg(p_s) = \sum_{i=1}^{k_s} \binom{k_s}{i}.$$

If the given Boolean equation is variable sharing, the monomial degree can decrease, since monomials can occur which just differ in the coefficients. The following example demonstrates how variable sharing effects the conversion.

Example 3.2. *The following Boolean function is converted using the standard representation.*

$$f = x_1 \oplus x_1 \wedge x_2$$

The resulting polynomial over the reals is given by

$$p_r = x_1 + x_1x_2 - 2x_1x_1x_2 = x_1 - x_1 \cdot x_2.$$

Due to the variable sharing property of f and the multilinearity property of the standard representation, the monomial degree in this example is decreased by one.

Example 3.3. *This example demonstrates that the standard conversion does not benefit from variable sharing, if the shared variable is part of a conjunction.*

$$f = x_1 \wedge x_2 \oplus x_1 \wedge x_3 \Longleftrightarrow p_r = x_1x_2 + x_1x_3 - 2 \cdot x_1x_2x_3$$

Removing the variable sharing property, the resulting polynomial has the same monomial degree.

$$f = x_1 \wedge x_2 \oplus x_4 \wedge x_3 \Longleftrightarrow p_r = x_1x_2 + x_4x_3 - 2 \cdot x_1x_2x_3x_4$$

The maximum monomial degree of a full Boolean equation in ANF (full means all a_I = `true`) is not that high as expected, since in such an equation variable sharing definitely occurs. Our experiments show that in such equations the monomial degree is decreased significantly.

3.3 Conversion using Dual Representation

3.3.1 Dual Representation

Definition 3.14. *Let $f(x_1, \cdots, x_n)$ be a Boolean function, $x_i \in \{\mathtt{true}, \mathtt{false}\}$ for $i = 1, \cdots, n$ and $t : \{\mathtt{false}, \mathtt{true}\} \to \{0, 1\}$ with*

$$\begin{aligned} t(\mathtt{false}) &= 1 \\ t(\mathtt{true}) &= 0. \end{aligned}$$

$s(y_1, \cdots, y_n)$ is the dual representation of $f(x_1, \cdots, x_n)$, if

$$\forall x_i : s(t(x_1), \cdots, t(x_n)) = t(f(x_1, \cdots, x_n)).$$

Lemma 3.2. *Let $f(x_1, \cdots, x_n)$ be a Boolean function and $s(y_1, \cdots, y_n)$ its dual representation.*

a) $f(x_1, x_2) = x_1 \wedge x_2 \Longrightarrow s(y_1, y_2) = y_1 + y_2 - y_1 \cdot y_2$, $y_i = t(x_i)$ *for* $i = 1, 2$.

b) $f(x_1, x_2) = x_1 \vee x_2 \Longrightarrow s(y_1, y_2) = y_1 \cdot y_2$, $y_i = t(x_i)$ *for* $i = 1, 2$.

c) $f(x_1) = \neg x_1 \Longrightarrow s(y_1) = 1 - y_1$, $y_1 = t(x_1)$.

d) $f(x_1, x_2) = x_1 \oplus x_2 \Longrightarrow s(y_1, y_2) = 1 - y_1 - y_2 + 2 \cdot y_1 \cdot y_2$, $y_i = t(x_i)$ *for* $i = 1, 2$.

The proof is done in the same way as the the proof of Lemma 3.1.

Example 3.4. *The same Boolean function is used as in Example 3.1.*

$$f(x_1, x_2, x_3, x_4) = x_3 \wedge x_1 \wedge x_0 \oplus x_3 \wedge x_1 \oplus x_3 \wedge x_0 \oplus x_3 \oplus x_1 \oplus x_0 \oplus \texttt{true}$$

The resulting polynomial over the reals using dual representation is

$$s(y_1, y_2, y_3, y_4) = y_0 + y_1 - y_0y_1 - y_0y_1y_3.$$

In this example we see already that the resulting polynomial has a less complex structure than using the standard representation, i.e. different representation types lead to a different structure of the resulting polynomial. Hence, the different conversion methods have a high influence on the resulting system over the reals.

3.3.2 Conversion

Using Algorithm 1, the definition of $C_\wedge$ and $C_\oplus$ follows directly from Lemma 3.2a) and Lemma 3.2d)

$$\begin{aligned} C_\wedge(a, b) &= a + b - a \cdot b, \\ C_\oplus(a, b) &= 1 - a - b + 2 \cdot a \cdot b. \end{aligned}$$

The conversion is done again only for one equation from system (3.1). Since Algorithm 1 works for all types of representation, the same procedure as in Section 3.2 has to be done.

$$\bigoplus_{I \subseteq M} a_I \wedge (\bigwedge_{i \in I} x_i) = b. \tag{3.6}$$

First the AND operations have to be converted according to $C_\wedge$.

$$\bigwedge_{i \in I} a_I \wedge x_i \tag{3.7}$$

In contrast to the standard representation the result is more complex. Therefore, we use the recursive representation of this part of the conversion process.

$$C_\wedge(a_I, C_\wedge(x_{i_1}, C_\wedge(x_{i_2}, C_\wedge(\cdots, C_\wedge(x_{i_{|I|}}, x_{i_{|I|+1}}))))), \tag{3.8}$$

For a better understanding the following intermediate result is obtained, evaluating two steps of the above recursion.

$$\begin{aligned} C_\wedge(\cdots) &= t(a_I) + C_\wedge(\cdots) - t(a_I) \cdot C_\wedge(\cdots) \\ &= t(a_I) + y_{i_1} + C_\wedge(\cdots) - y_{i_1} \cdot C_\wedge(\cdots) - t(a_I)y_{i_1} \\ &\quad -t(a_I)C_\wedge(\cdots) + t(a_I)y_{i_1}C_\wedge(\cdots) \end{aligned}$$

A complete evaluation of the recursion leads to following equation

$$\begin{aligned} c(\vec{y}_I) =& t(a_I)[1 - \sum_{i \in I} y_i + \sum_{\substack{i,j \in I \\ i<j}} y_iy_j - \sum_{\substack{i,j,k \in I \\ i<j<k}} y_iy_jy_k - \cdots + (-1)^{|I|} \prod_{i \in I} y_i] + \\ & \sum_{i \in I} y_i - \sum_{\substack{i,j \in I \\ i<j}} y_iy_j + \sum_{\substack{i,j,k \in I \\ i<j<k}} y_iy_jy_k - \cdots + (-1)^{|I|-1} \prod_{i \in I} y_i = t(b), \end{aligned} \tag{3.9}$$

where $\vec{y}_I = y_{i_1}, \cdots, y_{i_{|I|}}$. For better readability we write

$$\bigoplus_{I \subseteq M} c(\vec{y}_I). \tag{3.10}$$

The conversion for the XOR operations works in a similar way. According to Algorithm 1 the following expression is determined

$$C_\oplus(c(\vec{y}_{I_1}), C_\oplus(c(\vec{y}_{I_2}), C_\oplus(\cdots, C_\oplus(c(\vec{y}_{I_{|\mathcal{P}(\mathcal{M})|-1}}), c(\vec{y}_{I_{|\mathcal{P}(\mathcal{M})|}}))))).$$

After an evaluation of two steps of the above recursion we obtain

$$\begin{aligned} C_\oplus(\cdots) &= 1 - c(\vec{y}_{I_1}) - C_\oplus(\cdots) + 2 \cdot c(\vec{y}_{I_1}) \cdot C_\oplus(\cdots) \\ &= c(\vec{y}_{I_1}) + c(\vec{y}_{I_2}) + C_\oplus(\cdots) \\ &\quad -2 \cdot c(\vec{y}_{I_1}) \cdot c(\vec{y}_{I_2}) \\ &\quad -2 \cdot c(\vec{y}_{I_1}) \cdot C_\oplus(\cdots) \\ &\quad -2 \cdot c(\vec{y}_{I_2}) \cdot C_\oplus(\cdots) \\ &\quad +4 \cdot c(\vec{y}_{I_1}) \cdot c(\vec{y}_{I_2}) \cdot C_\oplus(\cdots). \end{aligned}$$

Finally, a full evaluation of $C_\oplus$ results in a polynomial over the reals representing the Boolean Equation (3.6) using dual representation

$$\begin{aligned} = & \sum_{i=1}^{|\mathcal{P}(\mathcal{M})|-1} (-1)^i + (-1)^{|\mathcal{P}(\mathcal{M})|-1} \sum_{I \subseteq M} c(\vec{y}_I) \\ & + 2 \cdot (-1)^{|\mathcal{P}(\mathcal{M})|} \sum_{\substack{I,J \subseteq M \\ I \neq J}} c(\vec{y}_I) c(\vec{y}_J) \\ & + 4 \cdot (-1)^{|\mathcal{P}(\mathcal{M})|-1} \sum_{\substack{I,J,K \subseteq M \\ I \neq J \neq K}} c(\vec{y}_I) c(\vec{y}_J) c(\vec{y}_K) \\ & + \cdots \\ & + 2^{|\mathcal{P}(\mathcal{M})|-1} \prod_{I \subseteq M} c(\vec{y}_I) = t(b). \end{aligned} \tag{3.11}$$

We denote Equation (3.11) as the converted polynomial over the reals p_d.

3.3.3 Analysis

Total Degree

The total degree of p_d depends on the last product in Equation (3.11) which is the largest one. Therefore, the last summand in Equation (3.9) defines the total degree of (3.9), which is $|I|$. Due to the last product in (3.11) the total degree of p_d can increase. However, the total degree is limited by the total number of variables in the equation because of the multilinearity property of the dual

representation. Hence, the total degree of p_d depends on the same factor as for the standard representation, which is

$$k_t := |\bigcup_{\substack{I \subseteq M \\ a_I = \texttt{true}}} I|.$$

Monomial Degree

Equation (3.6) contains $|\mathcal{P}(M)|$ different monomials. Similar to the standard representation, the converted equations consist of every possible product of Equation (3.9). Hence, there is a maximum of

$$\sum_{i=1}^{|\mathcal{P}(M)|} \binom{|\mathcal{P}(M)|}{i}$$

different products, if for all $I \in \mathcal{P}(M)$, a_I is equal `true`. Let k_s be defined as

$$k_s := \sum_{\substack{I \subseteq M \\ a_I = \texttt{true}}} 1,$$

then the number of products in Equation (3.11) is

$$\sum_{i=1}^{k_s} \binom{k_s}{i}.$$

Equation (3.9) has a similar structure and has

$$k_I = \sum_{i=1}^{|I|} \binom{|I|}{i}$$

different monomials for each $I \subseteq M$. Finally, the exact monomial degree of (3.11) is

$$mdeg(p_d) = \sum_{\substack{I \subseteq M \\ a_I = \texttt{true}}} k_I + \sum_{\substack{I,J \subseteq M \\ I \neq J \\ a_I, a_J = \texttt{true}}} k_I k_J + \sum_{\substack{I,J,K \subseteq M \\ I \neq J \neq K \\ a_I, a_J, a_K = \texttt{true}}} k_I k_J k_K + \cdots + \prod_{\substack{I \subseteq M \\ a_I = \texttt{true}}} k_I.$$

Note that the dual conversion leads to much larger polynomials over the reals compared to the standard representation. The effect of variable sharing is similar as for the Fourier or sign conversion, which are discussed in the subsequent sections. However, since the conversion of both the XOR and AND operation increases the monomial degree, the conversion results are significantly more complex than using the standard representation.

3.4 Conversion using Sign Representation

3.4.1 Sign Representation

Definition 3.15. *Let $f(x_1, \cdots, x_n)$ be a Boolean function, $x_i \in \{\texttt{true}, \texttt{false}\}$ for $i = 1, \cdots, n$ and $t : \{\texttt{false}, \texttt{true}\} \to \{-1, 1\}$ with*

$$\begin{aligned} t(\texttt{false}) &= -1 \\ t(\texttt{true}) &= 1. \end{aligned}$$

$s(y_1, \cdots, y_n)$ is the sign representation of $f(x_1, \cdots, x_n)$, if

$$\forall x_i : s(t(x_1), \cdots, t(x_n)) = t(f(x_1, \cdots, x_n)).$$

Lemma 3.3. *Let $f(x_1, \cdots, x_n)$ be a Boolean function and $s(y_1, \cdots, y_n)$ its sign representation.*

a) $f(x_1, x_2) = x_1 \wedge x_2 \Longrightarrow s(y_1, y_2) = \frac{1}{2}(-1 + y_1 + y_2 + y_1 \cdot y_2)$, $y_i = t(x_i)$ *for* $i = 1, 2$.

b) $f(x_1, x_2) = x_1 \vee x_2 \Longrightarrow s(y_1, y_2) = \frac{1}{2}(1 + y_1 + y_2 - y_1 \cdot y_2)$, $y_i = t(x_i)$ *for* $i = 1, 2$.

c) $f(x_1) = \neg x_1 \Longrightarrow s(y_1) = -y_1$, $y_1 = t(x_1)$.

d) $f(x_1, x_2) = x_1 \oplus x_2 \Longrightarrow s(y_1, y_2) = -y_1 \cdot y_2$, $y_i = t(x_i)$ *for* $i = 1, 2$.

Example 3.5. *The same Boolean function is used as in Example 3.1.*

$$f(x_1, x_2, x_3, x_4) = x_3 \wedge x_1 \wedge x_0 \oplus x_3 \wedge x_1 \oplus x_3 \wedge x_0 \oplus x_3 \oplus x_1 \oplus x_0 \oplus \texttt{true}$$

By applying Lemma 3.3 recursively as above, the polynomial over the reals using sign representation is

$$s(y_1, y_2, y_3, y_4) = \frac{1}{4}[1 - y_0 - y_1 - 3y_0y_1 + y_3 - y_0y_3 - y_1y_3 + y_0y_1y_3].$$

The reader may get the impression that this result is more complicated than in the standard or dual representation, but in the next sections the analysis will show that this is not always the case.

Remark 3.1. *A general property of the four representation types is the multilinearity. Concerning standard and dual representation the exponent of a variable is always equal to one ($x^2 = x$). More interesting is the multilinearity in the Fourier and sign representation since monomials can disappear ($x^2 = 1$).*

3.4.2 Conversion

Using Algorithm 1, the definition of $C_\wedge$ and $C_\oplus$ follows directly from Lemma 3.3a) and Lemma 3.3d).

$$\begin{aligned} C_\wedge(a,b) &= \frac{1}{2}(-1+a+b+a\cdot b), \\ C_\oplus(a,b) &= -a\cdot b. \end{aligned}$$

We consider again an arbitrary Boolean equation in ANF

$$\bigoplus_{I\subseteq M} a_I \wedge (\bigwedge_{i\in I} x_i) = b. \tag{3.12}$$

First the AND operations have to be converted by applying $C_\wedge$ on

$$\bigwedge_{i\in I} a_I \wedge x_i. \tag{3.13}$$

Therefore, we use the following recursion which is equivalent to Algorithm 1

$$C_\wedge(a_I, C_\wedge(x_{i_1}, C_\wedge(x_{i_2}, C_\wedge(\cdots, C_\wedge(x_{i_{|I|}}, x_{i_{|I|+1}}))))).$$

For a better understanding, the intermediate result after evaluating two steps of the above recursion results in

$$\begin{aligned} C_\wedge(\cdots) &= \frac{1}{2}\left(-1+t(a_I)+C_\wedge(\cdots)+t(a_I)\cdot C_\wedge(\cdots)\right) \\ &= \frac{1}{2}\Bigg(-1+t(a_I)+\frac{1}{2}(-1+y_1+C_\wedge(\cdots)+y_1\cdot C_\wedge(\cdots)) \\ &\quad +\frac{1}{2}(-t(a_I)+t(a_I)y_1+t(a_I)C_\wedge(\cdots)+t(a_I)y_1\cdot C_\wedge(\cdots))\Bigg) \\ &= \frac{1}{4}\left(-3+t(a_I)+y_1+C_\wedge(\cdots)+t(a_I)y_1+t(a_I)C_\wedge(\cdots)\right) \\ &\quad +y_1C_\wedge(\cdots)+t(a_I)y_1C_\wedge(\cdots)\right). \end{aligned}$$

By evaluating all $C_\wedge$ we obtain

$$\begin{aligned} c(\vec{y}_I) := &-1+\left(\frac{1}{2}\right)^{|I|-1} \\ &+\left(\frac{1}{2}\right)^{|I|-1} t(a_I)\left[1+\sum_{i\in I} y_i+\sum_{\substack{i,j\in I\\ i<j}} y_iy_j+\sum_{\substack{i,j,k\in I\\ i<j<k}} y_iy_jy_k+\cdots+\prod_{i\in I} y_i\right] \\ &+\left(\frac{1}{2}\right)^{|I|-1}\left[\sum_{i\in I} y_i+\sum_{\substack{i,j\in I\\ i<j}} y_iy_j+\sum_{\substack{i,j,k\in I\\ i<j<k}} y_iy_jy_k+\cdots+\prod_{i\in I} y_i\right], \end{aligned} \tag{3.14}$$

where $\vec{y}_I = y_{i_1}, \cdots, y_{i_{|I|}}$. For better readability we write

$$\bigoplus_{I \subseteq M} c(\vec{y}_I). \tag{3.15}$$

Again the XOR operations have to be converted next. According to following recursion

$$C_\oplus(c(\vec{y}_{I_1}), C_\oplus(c(\vec{y}_{I_2}), C_\oplus(\cdots, C_\oplus(c(\vec{y}_{I_{|\mathcal{P}(\mathcal{M})|-1}}), c(\vec{y}_{I_{|\mathcal{P}(\mathcal{M})|}})))))$$

we obtain a polynomial over the reals representing the Boolean Equation 3.12 using sign representation

$$\begin{aligned} C_{\oplus_1}(\cdots) =& - c(\vec{y}_{I_1}) \cdot C_{\oplus_2}(\cdots) \\ =& c(\vec{y}_{I_1}) c(\vec{y}_{I_2}) \cdot C_{\oplus_3}(\cdots) \\ =& \cdots \\ =& (-1)^{|\mathcal{P}(M)|-1} \prod_{I \subseteq M} c(\vec{y}_I) = t(b) \end{aligned} \tag{3.16}$$

We denote Equation (3.16) as the converted polynomial over the reals p_{si}.

3.4.3 Analysis

Total Degree

Since equation (3.16) consist only of one single product, we need to analyse Equation (3.14), where the product $\prod_{i \in I} y_i$ determines the total degree of (3.14), which is $|I|$. It follows from the single product in (3.16) that the total degree of p_{si} is equal to the total degree p_s or p_d, which is equal to the number of variables in the converted equation

$$k_t := |\bigcup_{\substack{I \subseteq M \\ a_I = 1}} I|.$$

Monomial Degree

Equation (3.12) consist of $|\mathcal{P}(M)|$ different monomials and Equation (3.16) consists of one single product. Hence, the number of monomials is determined by

$$\prod_{I \subseteq M} mdeg(c(\vec{y}_I)).$$

Since $c(\vec{y}_I)$ consists of each possible product of y_i, for $i \in I$ and $|I| > 1$, plus one constant, the number of monomials is

$$\sum_{i=1}^{|I|} \binom{|I|}{i} + 1.$$

The condition $|I| > 1$ follows from the fact that for $|I| = 1$ there is no conjunction and therefore no conversion is needed. However, only $c(\vec{y}_I)$ where $a_I = \texttt{true}$ contribute to the monomial degree. Therefore, the monomial degree of p_{si} is given by

$$mdeg(p_{si}) = \prod_{\substack{I \subseteq M \\ a_I = \texttt{true}}} \left(\sum_{i=1}^{|I|} \binom{|I|}{i} + 1 \right).$$

This is the exact number of monomials if for all $I \subseteq M$, $a_I = \texttt{true}$ and $\bigcap I = \{\}$. Due to the multilinearity property the monomial degree is decreased if the Boolean function is variable sharing. In contrast to the standard representation, the sign representation benefits from a different type of variable sharing. In this case the shared variable *has to be* a part of a conjunction. This derives from the fact, that only the AND conversion creates additional monomials.

Example 3.6. *The following Boolean function is converted using the sign representation.*

$$f = x_1 \wedge x_2 \oplus x_1 \wedge x_3$$

The resulting polynomial over the reals is given by

$$\begin{aligned} p_{si} &= \frac{1}{4}(-1 + x_1 + x_2 + x_1x_2 + x_3 - x_1x_3 \\ &\quad -x_2x_3 - x_1x_2x_3 + x_1 - x_1x_1 - x_2x_1 \\ &\quad -x_1x_2x_1 + x_3x_1 - x_1x_3x_1 - x_2x_3x_1 - x_1x_2x_3x_1) \\ &= \frac{1}{2}(-1 + x_1 - x_2x_3 - x_1x_2x_3) \end{aligned}$$

Due to the variable sharing property of f and the multilinearity property of the sign representation, the monomial degree in this example is decreased by a factor of 4. If the above requirement is not given, then the monomial degree is not decreased.

Example 3.7. *This example demonstrates that the sign conversion does not benefit from variable sharing, if the shared variable is not part of a conjunction.*

$$f = x_1 \oplus x_1 \wedge x_2 \Longleftrightarrow p_{si} = \frac{1}{2}(-1 + x_1 - x_2 - x_1x_2)$$

Removing the variable sharing property, the resulting polynomial has the same monomial degree (1 is interpreted as a monomial x^0).

$$f = x_1 \oplus x_3 \wedge x_2 \Longleftrightarrow p_{si} = \frac{1}{2}(x_1 - x_1x_2 - x_1x_3 - x_1x_2x_3)$$

Note that the observations for the variable sharing effect are also made for the Fourier conversion, which is introduced next.

3.5 Conversion using Fourier Representation

3.5.1 Fourier Representation

Definition 3.16. *Let $f(x_1, \cdots, x_n)$ be a Boolean function, $x_i \in \{\texttt{true}, \texttt{false}\}$ for $i = 1, \cdots, n$ and $t : \{\texttt{false}, \texttt{true}\} \to \{-1, 1\}$ with*

$$\begin{aligned} t(\texttt{false}) &= 1 \\ t(\texttt{true}) &= -1. \end{aligned}$$

$s(y_1, \cdots, y_n)$ is the Fourier representation of $f(x_1, \cdots, x_n)$, if

$$\forall x_i : s(t(x_1), \cdots, t(x_n)) = t(f(x_1, \cdots, x_n)).$$

Lemma 3.4. *Let $f(x_1, \cdots, x_n)$ be a Boolean function and $s(y_1, \cdots, y_n)$ its Fourier representation.*

a) $f(x_1, x_2) = x_1 \wedge x_2 \Longrightarrow s(y_1, y_2) = \frac{1}{2}(1 + y_1 + y_2 - y_1 \cdot y_2)$, $y_i = t(x_i)$ *for* $i = 1, 2$.

b) $f(x_1, x_2) = x_1 \vee x_2 \Longrightarrow s(y_1, y_2) = \frac{1}{2}(-1 + y_1 + y_2 + y_1 \cdot y_2)$, $y_i = t(x_i)$ *for* $i = 1, 2$.

c) $f(x_1) = \neg x_1 \Longrightarrow s(y_1) = -y_1$, $y_1 = t(x_1)$.

d) $f(x_1, x_2) = x_1 \oplus x_2 \Longrightarrow s(y_1, y_2) = y_1 \cdot y_2$, $y_i = t(x_i)$ *for* $i = 1, 2$.

Example 3.8. *Again the same Boolean function as in Example 3.1 is used.*

$$f(x_1, x_2, x_3, x_4) = x_3 \wedge x_1 \wedge x_0 \oplus x_3 \wedge x_1 \oplus x_3 \wedge x_0 \oplus x_3 \oplus x_1 \oplus x_0 \oplus \texttt{true}$$

$$s(y_1, y_2, y_3, y_4) = \frac{1}{4}[1 + y_0 + y_1 - 3y_0y_1 - y_3 - y_0y_3 - y_1y_3 - y_0y_1y_3].$$

3.5.2 Conversion

Using Algorithm 1, the definition of $C_\wedge$ and $C_\oplus$ follows directly from Lemma 3.4a) and Lemma 3.4d).

$$\begin{aligned} C_\wedge(a, b) &= \frac{1}{2}(1 + a + b - a \cdot b) \\ C_\oplus(a, b) &= a \cdot b \end{aligned}$$

We target again an arbitrary Boolean equation in ANF

$$\bigoplus_{I \subseteq M} a_I \left(\bigwedge_{i \in I} x_i\right) = b. \tag{3.17}$$

The AND operations have to be converted first by applying $C_\wedge$ on

$$\bigwedge_{i\in I} a_I \wedge x_i \tag{3.18}$$

Therefore, we use the following recursion

$$C_\wedge(a_I, C_\wedge(x_{i_1}, C_\wedge(x_{i_2}, C_\wedge(\cdots, C_\wedge(x_{i_{|I|}}, x_{i_{|I|+1}}))))). \tag{3.19}$$

The first two step results in

$$\begin{aligned} C_\wedge(\cdots) &= \frac{1}{2}\left(1 + t(a_I) + C_\wedge(\cdots) - t(a_I)\cdot C_\wedge(\cdots)\right) \\ &= \frac{1}{2}\Bigg(1 + t(a_I) + \frac{1}{2}(1 + y_1 + C_\wedge(\cdots) - y_1\cdot C_\wedge(\cdots)) \\ &\quad + \frac{1}{2}(-t(a_I) - t(a_I)y_1 - t(a_I)C_\wedge(\cdots) + t(a_I)y_1\cdot C_\wedge(\cdots))\Bigg) \\ &= \frac{1}{4}\left(3 + t(a_I) + y_1 + C_\wedge(\cdots) - t(a_I)y_1 - t(a_I)C_\wedge(\cdots)\right) \\ &\quad - y_1 C_\wedge(\cdots) + t(a_I)y_1 C_\wedge(\cdots)) \end{aligned}$$

and the full conversion of the AND operations leads to

$$\begin{aligned} c(\vec{y}_I) :=& 1 - \left(\frac{1}{2}\right)^{|I|-1} \\ &- \left(\frac{1}{2}\right)^{|I|-1} t(a_I)\left[1 + \sum_{i\in I} y_i - \sum_{\substack{i,j\in I\\ i<j}} y_i y_j + \sum_{\substack{i,j,k\in I\\ i<j<k}} y_i y_j y_k - \cdots + (-1)^{|I|-1}\prod_{i\in I} y_i\right] \\ &+ \left(\frac{1}{2}\right)^{|I|-1}\left[\sum_{i\in I} y_i - \sum_{\substack{i,j\in I\\ i<j}} y_i y_j + \sum_{\substack{i,j,k\in I\\ i<j<k}} y_i y_j y_k - \cdots + (-1)^{|I|-1}\prod_{i\in I} y_i\right], \end{aligned} \tag{3.20}$$

where $\vec{y}_I = y_{i_1}, \cdots, y_{i_{|I|}}$. For a better readability we write

$$\bigoplus_{I\subseteq M} c(\vec{y}_I). \tag{3.21}$$

Again the XOR operations have to be converted next according to the following recursion

$$C_\oplus(c(\vec{y}_{I_1}), C_\oplus(c(\vec{y}_{I_2}), C_\oplus(\cdots, C_\oplus(c(\vec{y}_{I_{|\mathcal{P}(\mathcal{M})|-1}}), c(\vec{y}_{I_{|\mathcal{P}(\mathcal{M})|}}))))),$$

we obtain a polynomial over the reals representing the Boolean Equation (3.17) using Fourier representation

$$\begin{aligned} C_{\oplus}(\cdots) =& c(\vec{y}_{I_1}) \cdot C_{\oplus}(\cdots) \\ =& c(\vec{y}_{I_1}) c(\vec{y}_{I_2}) \cdot C_{\oplus}(\cdots) \\ =& \cdots \\ =& \prod_{I \subseteq M} c(\vec{y}_I) = t(b). \end{aligned} \tag{3.22}$$

We denote Equation (3.22) as the converted polynomial over the reals p_f.

3.5.3 Analysis

Due to the similarities between the Fourier and sign representation all derivations in Section 3.4.3 are also true for the Fourier representation. Hence, the total degree and monomial degree is the same if the Boolean equation is not variable sharing. If the variable sharing property is available, the same observations are made as for the sign representation.

Total Degree

The total degree is given by

$$k_t := |\bigcup_{\substack{I \subseteq M \\ a_I = \texttt{true}}} I|.$$

Monomial Degree

The monomial degree is given by

$$\prod_{\substack{I \subseteq M \\ a_I = \texttt{true}}} \left(\sum_{i=1}^{|I|} \binom{|I|}{i} + 1 \right),$$

for $|I| > 1$.

3.6 Comparison of Conversion Methods

We presented four different ways to convert any Boolean equation to a polynomial over the reals. The resulting polynomials look different but share the minimum amount of roots derived from the Boolean solutions of the original Boolean equation. As we have shown, one has a huge influence on the structure of the conversion results. The analysis of the conversion methods shows that the complexity of the structure of the resulting polynomials is increased significantly for all methods. The total degree is the same for all four, and depends

only on the number of occurring variables. However, the monomial degree is one factor were different methods deliver different results. If we assume that a lower monomial degree makes the solving of equations using numerical methods easier, then we would recommend the standard or Fourier conversion. The dual representation can be excluded from further consideration, since it delivers the highest monomial degree. The sign and Fourier representation are very similar and deliver the same monomial degree after the transformation. We prefer Fourier conversion, because the conversion for the XOR operations is slightly simpler (no sign alteration). If we reconsider the monomial degree using the standard representation

$$\sum_{i=1}^{k_s} \binom{k_s}{i}, \tag{3.23}$$

where k_s is the amount of $a_I = \texttt{true}$ and for the Fourier representation

$$\prod_{\substack{I \subseteq M \\ a_I = \texttt{true}}} \sum_{i=1}^{|I|} \binom{|I|}{i}, \tag{3.24}$$

one can see that both depend on the number of $a_I = \texttt{true}$ which is equivalent to the number of XOR operations in the Boolean equation. Additionally, the monomial degree for the conversion using Fourier representation depends on the number of AND operations. According to (3.23) and (3.24) an increase of XOR operations is a disadvantage for the standard representation. The number of AND operations can be expressed in the following way

$$\sum_{\substack{I \subseteq M \\ a_I = \texttt{true}}} (|I| - 1).$$

The following inequalities can be used as a rule of thumb for the decision in practice. If the condition

$$\sum_{\substack{I \subseteq M \\ a_I = \texttt{true}}} (|I| - 1) < k_s$$

holds then Fourier representation should be preferred. On the other hand if

$$\sum_{\substack{I \subseteq M \\ a_I = \texttt{true}}} (|I| - 1) \geq k_s$$

holds, then the standard representation results in a lower monomial degree. If the Boolean equation is variable sharing then the rule of thumb may change. In that case it depends how a variable is shared in one equation which is analysed in the next section. However, in practice it is useful to compute the polynomial over the reals using both conversion types and to compare the monomial degree, since the computational effort to do this is negligible.

Remark 3.2. *Note that the effect of the monomial degree on the solvability of the resulting system of equations over the reals is still an open problem. However, the monomial degree has a significant impact on the efficiency of numerical methods. Due to the fact that large equations are more difficult to handle. Storing and evaluating large equations and therefore large systems of equations can be costly. Especially, iterative numerical methods evaluate the system and the Jacobian matrix more than once per iteration. Hence, the performance of each iteration decreases with the size of the equations.*

Remark 3.3. *One disadvantage of all four conversion methods is that linear Boolean functions become non-linear function over the reals.As we show in Chapter 5, there are cases were we can prevent this.*

3.7 Variable Sharing

The variable sharing property decreases the monomial degree as shown in Sections 3.2.3 and 3.4.3. In the following section we analyse this property in more details. Due to the similarities between sign and Fourier conversion and the bad results for dual conversion, we focus on the standard and Fourier conversion. We consider again an arbitrary Boolean equation in ANF

$$\bigoplus_{I \subseteq M} a_I \wedge (\bigwedge_{i \in I} x_i) = b.$$

Without loss of generality let x_k be the shared variable (occurs in at least two monomials). Let be $\tilde{M} := \{I \in \mathcal{P}(M) : a_I = \texttt{true}\}$. We separate the monomials including x_k in two parts by splitting $\tilde{M}$ in $\tilde{M}_k = \{I \in \tilde{M} : k \in I\}$ and $\tilde{M}_{\bar{k}} = \{I \in \tilde{M} : k \notin I\}$ which results in

$$(\bigoplus_{K \in \tilde{M}_k} a_K \wedge (\bigwedge_{i \in K} x_i)) \oplus (\bigoplus_{I \in \tilde{M}_{\bar{k}}} a_I \wedge (\bigwedge_{i \in I} x_i)). \tag{3.25}$$

In the next step we factor out x_k:

$$(x_k \wedge (\bigoplus_{K \in \tilde{M}_k} a_K \wedge (\bigwedge_{i \in K \setminus \{k\}} x_i)) \oplus (\bigoplus_{I \in \tilde{M}_{\bar{k}}} a_I \wedge (\bigwedge_{i \in I} x_i)). \tag{3.26}$$

Under the consideration of the distributive law in Boolean algebra, we first convert the right part denoted by $c_{\bar{k}}(\vec{y})$, and then the left part denoted by $c_k(\vec{y})$ of Equation (3.26). Finally, the XOR combination of both results is converted

$$c_k(\vec{y}) \oplus c_{\bar{k}}(\vec{y}).$$

The affect of variable sharing on the monomial degree depends now on the chosen representation. Note that for more than one sharing variable the same technique is applied recursively under consideration of the distributive law in Boolean algebra.

3.7.1 Standard Conversion

As mentioned before, variable sharing can decrease the monomial degree of the resulting equations. The monomial degree of $c_{\bar{k}}(\vec{y})$ using standard representation (see Section 3.2.3) is given by

$$\sum_{i=1}^{|\tilde{M}_{\bar{k}}|} \binom{|\tilde{M}_{\bar{k}}|}{i}.$$

Considering Equation (3.26) the conjunction of x_k is converted to a multiplication and therefore does not increase the monomial degree. Hence, by factoring out x_k the monomial degree can only decrease if there is one constant monomial in

$$(\bigoplus_{K \in \tilde{M}_k} a_K \wedge (\bigwedge_{i \in K \setminus \{k\}} x_i).$$

In that case the monomial degree of $c_k(\vec{y})$ is reduced and is given by

$$\sum_{i=1}^{|\tilde{M}_k|-1} \binom{|\tilde{M}_k - 1|}{i}.$$

In the last step we convert the XOR combination

$$c_k(\vec{y}) \oplus c_{\bar{k}}(\vec{y}).$$

according to the conversion rule given by the standard representation:

$$C_{\oplus}(a, b) = a + b - 2 \cdot a \cdot b.$$

Hence, the monomial degree of the Equation (3.25) converted with one shared variable is given by

$$\sum_{i=1}^{|\tilde{M}_k|-1} \binom{|\tilde{M}_k - 1|}{i} \cdot \sum_{i=1}^{|\tilde{M}_{\bar{k}}|} \binom{|\tilde{M}_{\bar{k}}|}{i} + \sum_{i=1}^{|\tilde{M}_k|-1} \binom{|\tilde{M}_k - 1|}{i} + \sum_{i=1}^{|\tilde{M}_{\bar{k}}|} \binom{|\tilde{M}_{\bar{k}}|}{i}$$

which is less than the monomial degree without variable sharing

$$\sum_{i=1}^{|\tilde{M}|} \binom{|\tilde{M}|}{i}.$$

In other words, the standard conversion profits from variable sharing only if the shared variable is the only variable in at least one monomial.

3.7.2 Fourier Conversion

We use the Fourier representation to convert Equation (3.25) according to Section 3.5. The monomial degree of $c_{\bar{k}}(\vec{y})$ using Fourier representation is given by

$$\prod_{I \in \tilde{M}_{\bar{k}}} \left(\sum_{i=1}^{|I|} \binom{|I|}{i} + 1 \right),$$

for $|I| > 1$. To determine the monomial degree of $c_k(\vec{y})$ we first compute the monomial degree of the inner conjunctions

$$a_K \wedge (\bigwedge_{i \in K \setminus \{k\}} x_i))$$

which is given by

$$\sum_{i=1}^{|K|-1} \binom{|K|-1}{i} + 1,$$

for each $K \in \tilde{M}_k$ and for $|K| - 1 > 1$. The XOR operations are converted to multiplications and hence the monomial degree including the XOR conversion results in

$$\prod_{\substack{K \in \tilde{M}_k \\ |K|>2}} \left(\sum_{i=1}^{|K|-1} \binom{|K|-1}{i} + 1 \right).$$

Now we have one final conjunction to convert which results from factoring out the shared variable x_k (see Equation (3.26)). The rule for this conversion is given by

$$C_\wedge(a, b) = \frac{1}{2}(1 + a + b - a \cdot b)$$

which determines the monomial degree of $c_k(\vec{y})$:

$$2 \cdot \prod_{\substack{K \in \tilde{M}_k \\ |K|>2}} \left(\sum_{i=1}^{|K|-1} \binom{|K|-1}{i} + 1 \right).$$

Note that there is a special case if $\exists K \in \tilde{M}_k, |K| \leq 2$. In that case the monomial degree of $c_k(\vec{y})$ is increased by two. It follows that the monomial degree of Equation (3.25) converted with one shared variable is given by

$$\left(2 \cdot \prod_{\substack{K \in \tilde{M}_k \\ |K|>2}} \sum_{i=1}^{|K|-1} \binom{|K|-1}{i} + 1 \right) \cdot \left(\prod_{I \in \tilde{M}_{\bar{k}}} \sum_{i=1}^{|I|} \binom{|I|}{i} \right).$$

If we compare this result to the monomial degree without variable sharing

$$\prod_{I \subseteq \tilde{M}} \left(\sum_{i=1}^{|I|} \binom{|I|}{i} + 1 \right),$$

we see that the monomial degree depends on how often a shared variable occur in a conjunction. The higher the occurrence the lower the monomial degree. However, there is an exception if the shared variable is part of conjunctions with only one additional variable. In that case the monomial degree does not change.

3.7.3 Conversion Examples

The examples and conversion results shown in Table 3.1 should give a feeling about the total degree and monomial degree for different types of equations.

Table 3.1: Results for different examples.

	Standard	**Dual**	**Sign**	**Fourier**
f_1	$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \wedge x_6$			
Total degree	6	6	6	6
Monomial degree	31	63	4	4
f_2	$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \wedge x_5 \oplus x_6 \wedge x_7$			
Total degree	7	7	7	7
Monomial degree	31	127	16	16
f_3	$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \wedge x_5 \oplus x_6 \wedge x_7 \oplus x_8 \wedge x_9$			
Total degree	9	9	9	9
Monomial degree	63	512	64	64
f_4	$x_1 \oplus x_2 \oplus x_3 \oplus x_1 \wedge x_4 \oplus x_6 \wedge x_7 \oplus x_8 \wedge x_9$			
Total degree	8	8	8	8
Monomial degree	47	192	64	64
f_5	$x_1 \oplus x_2 \oplus x_3 \oplus x_1 \wedge x_4 \oplus x_1 \wedge x_7 \oplus x_8 \wedge x_9$			
Total degree	7	7	7	7
Monomial degree	39	128	16	16
f_6	$x_1 \oplus x_2 \oplus x_3 \oplus x_1 \wedge x_4 \oplus x_2 \wedge x_7 \oplus x_8 \wedge x_9$			
Total degree	7	7	7	7
Monomial degree	35	72	64	64

f_1 is not variable sharing and has more XOR than AND operations. f_2 has an additional AND operation. The results show that the monomial degree for the standard conversion does not change, but for the Fourier transformation it increases significantly. f_3 has a balanced amount of XOR and AND operations, hence the total degree and monomial degree are close. In f_4 we include variable sharing to the equation. This results in a significant decrease of the monomial degree for the standard and dual conversion. For f_5 a variable is shared such that the standard *and* Fourier conversion benefit from it. The Fourier conversion can compensate an AND operation, which decreases the monomial degree by a factor of 4. In f_6 more variables are shared such that the standard and dual conversion benefit from it. Therefore, the decrease factor is smaller than for f_5 using sign and Fourier transformation.

Remark 3.4. *Due to the fact that the conversion of a system of equations is done by converting each equation separately, we analysed the conversion methods only for one single equation. Although, considering the whole system, some beneficial effects may occur. For example a combination of converted equations may result in a less complex structure. However, this effects would occur independent of the chosen conversion methods and may already indicate a structural problem in the original Boolean system. For the targeted systems we can not observe such effects and therefore it does not influence the choice of the conversion method.*

Obviously, the structure of the resulting polynomials depends also on the actual used Boolean equations, why it is not possible to recommend one representation for all cases. However, the rule of thumb given in Section 3.6 should lead to a first choice of a conversion method. Due to the variable sharing property a closer look at the equations is inevitable.

3.8 Advanced Conversion Techniques

So far we influenced the structure of the resulting polynomials over the monomial degree and the variable sharing property. The total degree on the other hand is not changed by any type of representation and is considerable high. Since the total degree is a crucial factor in the solvability of non-linear systems of equations over the reals (see Chapter 4), it is important to keep this degree as low as possible. Therefore, we further develop the presented conversion methods such that we specifically influence the total degree of the resulting polynomials. We published both conversion methods in [LNR09b].

3.8.1 Adapted Standard Conversion

The main idea behind our *Adapted Standard Conversion* is to keep the structure of the equation when converting them to the real domain. This should result in equations over the reals with low monomial degrees and total degrees. The adapted standard conversion uses the standard representation, but with a different conversion algorithm. Instead of converting all operations recursively in consideration of the distributive law in Boolean algebra, the equation stays unchanged by introducing new variables and new equations. Note that this conversion method works for every type representation analogous.

Conversion

We consider again an arbitrary Boolean equation in ANF

$$f(\vec{x}) = \bigoplus_{I \subseteq M} a_I (\bigwedge_{i \in I} x_i) = b. \tag{3.27}$$

In the first step we compute the truth table of the Boolean function f and filter for the solutions of Equation (3.27). Let $S := \{\vec{x} : f(\vec{x}) = b\}$ be the set of solutions.

Next we keep the structure of the equation in the real domain, i.e. each AND operation becomes the multiplication and each XOR becomes an addition over the reals:

$$f_r(\vec{x}) = \sum_{I \subseteq M} t(a_I)(\prod_{i \in I} y_i) = t(b), \tag{3.28}$$

for $f_r : B^n \to \mathbb{R}$. We determine the set $S_r := \{f(\vec{y}) : \vec{y} = t(\vec{x}) \text{ for } \vec{x} \in S\}$ which contains all possible values of f_r at the solutions of (3.27). Since $|S_r| > 0$ we need to add equations such that the resulting system over the reals is equivalent to the Boolean system. Therefore, we introduce for each $c_i \in S_r$ the following equation and variable

$$d_i(\sum_{I \subseteq M} t(a_I)(\prod_{i \in I} y_i) - c_i) = 0, \tag{3.29}$$

where $d_i \in \mathbb{R}$. Finally, a last equation is added

$$\sum_{i=1}^{|S_r|} d_i = 1,$$

expressing that only one of the expressions between the brackets in (3.29) can be valid at the same time or that only one of the new variables should be equal to 1. The obvious solution is integer valued. Unfortunately, this construction also makes the existence of real-valued solutions more likely. Note that such a construction is common practice in operation research when modelling decision problems. In our case d_i decides which assignment of the variables corresponds to which equation.

The final result of converting the Boolean Equation (3.27) is the system

$$\begin{pmatrix} d_1(\sum_{I \subseteq M} t(a_I)(\prod_{i \in I} y_i) - c_1) \\ \vdots \\ d_{|S_r|}(\sum_{I \subseteq M} t(a_I)(\prod_{i \in I} y_i) - c_{|S_r|}) \\ \sum_{i=1}^{|S_r|} d_i \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{3.30}$$

Analysis

The adapted standard conversion is a trade-off approach. We trade for lower total and monomial degree and pay with more equations and variables. The cost depends only on $|S_r|$ and if its low we can improve the total and monomial degree significantly compared to any of the other conversion methods without paying to much. The total degree using this conversion method is effectively increased only by one, due to the multiplication of the new variables d_i. The monomial degree itself does not change. However, we need to introduce new equations such that the conversion result is consistent with the Boolean solutions. That leads to $|S_r| + 1$ additional equations and $|S_r|$ new variables. If this pays off depends on the actual Boolean equations.

3.8.2 Splitting Conversion

The *Splitting Conversion* aims also for a low total degree of the conversion result, but achieves it by a different approach. As we show in the four basic conversion methods the final total and monomial degree increases exponentially with the number of variables and monomials of the Boolean equation. As the name indicates we split the Boolean equation such that smaller equations need to be converted.

Conversion

We consider again an arbitrary Boolean equation in ANF

$$\bigoplus_{I \subseteq M} a_I (\bigwedge_{i \in I} x_i) = b. \tag{3.31}$$

Without loss of generality we set $\tilde{M} \subseteq \mathcal{P}(M)$ such that for all $J \in \tilde{M}$, $a_J = \texttt{true}$. We split $\tilde{M}$ into l parts. Hence, we split also Equation (3.31) into l equations

$$\left(\begin{array}{c} \bigoplus\limits_{I \subseteq \tilde{M}_1} a_I \wedge (\bigwedge_{i \in I} x_i) = z_1 \\ \bigoplus\limits_{I \subseteq \tilde{M}_2} a_I \wedge (\bigwedge_{i \in I} x_i) = z_2 \\ \vdots \\ \bigoplus\limits_{I \subseteq \tilde{M}_{l-1}} a_I \wedge (\bigwedge_{i \in I} x_i) = z_{l-1} \\ \bigoplus\limits_{I \subseteq \tilde{M}_l} a_I \wedge (\bigwedge_{j \in I} x_i) = \bigoplus_{i=1}^{l-1} z_i \oplus b \end{array} \right), \tag{3.32}$$

and introducing new variables $z_i \in \{false, true\}$, for $i = 1, \ldots l-1$. Now every equation in (3.32) is converted using any of the four basic conversion methods. Left-hand side and right-hand side of each equation are converted separately. The rule of thumb given in Section 3.6 can be applied on each equation for choosing an appropriate conversion method. Obviously, we are not allowed to mix conversion methods in this step.

Analysis

Each of the derivations made for the basic conversion methods holds also for the splitting conversion. Due to the splitting parameter $l > 0$ we can directly control the total degree of the result. Hence, the total degree of converted equation i is

$$k_i := |\bigcup_{I \subseteq \tilde{M}_i} I|.$$

The monomial degree of converted equation i using standard representation results in

$$\sum_{j=1}^{|\tilde{M}_i|} \binom{|\tilde{M}_i|}{j}$$

and for using Fourier representation it is

$$\prod_{I \subseteq \tilde{M}_i} \sum_{j=1}^{|I|} \binom{|I|}{j}.$$

The situation for the last equation on (3.32) is a little bit different, because the right-hand side consist of a linear combination of the introduced variables. Hence, for the last equation we obtain a total degree of

$$\max\{|\bigcup_{I \subseteq \tilde{M}_l} I|, l-1\}$$

and a monomial degree of

$$\max\{\sum_{j=1}^{|\tilde{M}_l|} \binom{|\tilde{M}_l|}{j}, \sum_{j=1}^{l-1} \binom{l-1}{j}\}$$

using standard representation. Since the right-hand side of the last equation is a linear function, the monomial degree is only slightly increased using Fourier representation

$$\prod_{I \subseteq \tilde{M}_l} \sum_{j=1}^{|I|} \binom{|I|}{j} + 1.$$

Overall, we can reduce the total degree and monomial degree significantly by the cost of $l-1$ additional equations and variables.

Even, if the splitting method seems to be more complicated, we show in Chapter 5 that it is especially useful when we deal with sparse Boolean equations.

3.9 Summary

Every cryptographic algorithm can be described as a system of Boolean equations. In order to be able to use numerical methods to solve such systems we have to convert the equations to equations over the real domain. By doing that we have to ensure that at least the set of Boolean solutions is represented in the reals, i.e. each solution of the Boolean system has at least one corresponding solution for the system over the reals. In several scientific fields similar problems occur for what different types are defined to represent a Boolean function/equation as a polynomial over the reals. Four types of representation occur consistently in the literature, namely standard, dual, Fourier and sign representation. Basically, the mapping of $\{\texttt{false}, \texttt{true}\}$ to real values determines the

conversion of the Boolean operators. In this chapter we analysed in detail each type of representation and defined criteria to classify them. Therefore, we focused on the structure of the resulting equations over the reals and defined the criteria monomial degree and total degree. We showed that one has a high influence on the structure of the conversion result, which is exceptional compared to usual use cases for numerical methods. Furthermore, we defined the property of variable sharing which has a high influence on the conversion result. We showed how this property affects the different types of representation. We also provided a rule of thumb helping to find the appropriate type of representation for specific equations.

However, all four types result in a relatively high total degree which has a crucial impact on the difficulty of the system over the reals. Therefore, we introduce two new conversion methods aiming especially for a low total degree and monomial degree for the resulting polynomials over the reals. Overall, we showed that by choosing the right representation and conversion method one can change significantly the structure of the system of equations over the reals and therefore its difficulty regarding the solvability.

4 Numerical Analysis

In this chapter we will give an overview on numerical analysis. It is difficult to give a general overview on numerical analysis since there are a large amount of different applications, problems, techniques and several sub-fields with numerous research results. Therefore, we will focus on methods for solving non-linear systems of polynomials which shrinks the number of usefull numerical methods. Additionally, we know that for our modelled problems solutions exists and we generally do not know much about them which further decreases the considerable techniques. In the following sections we will focus on those aspects of numerical analysis which are important for our approach. We will give an overview on numerical analysis, explain the basic concepts and introduce the most important methods and techniques.

4.1 Overview

Numerical analysis is the area of mathematics and computer science that creates, analyses, and implements algorithms for solving numerically the problems of continuous mathematics. Such problems originate generally from real-world applications of algebra, geometry and calculus, and they involve variables which vary continuously. These problems occur throughout the natural sciences, social sciences, engineering, medicine, and many more. In the last half-century the increasing power and availability of digital computers has raised the usage of mathematical models in science and engineering. Numerical analysis has been needed to solve these more detailed mathematical models of the world. The formal academic area of numerical analysis varies from quite theoretical mathematical studies to computer science issues. With the growth in importance of

using computers to carry out numerical methods in solving mathematical models of the world, an area known as scientific computing or computational science has taken shape during the 1980s and 1990s. This area looks at the use of numerical analysis from a computer science perspective. It is concerned with using the most powerful tools of numerical analysis, computer graphics, symbolic mathematical computations, and graphical user interfaces to make it easier for a user to set up, solve, and interpret complicated mathematical models of the real world.

Numerical analysis is used for a vast variety of problems and is concerned with all aspects of the numerical solution of a problem, from the theoretical development and understanding of numerical methods to their practical implementation. Numerical analysis consists of several sub-areas, but they share some common concerns, perspectives and mathematical methods. A brief overview is given in this section. Since numerical analysis is a huge field we do not claim completeness of this overview. Detailed information can be found in [Atk89, Spe93, Sch97, PR02, Deu04].

A well-known sub-field is interpolation theory which is the concept of choosing a function from a given class in such way that the graph passes through the given data points, hence interpolating the given data points by a continuous function, usually a polynomial. Interpolation is partially connected with approximation theory which covers the approximation of functions and methods. For evaluating a function on a computer, it is generally more efficient in space and time to have an analytic approximation rather than to store a table and use interpolation. Usually, polynomials as approximations to a given function are used.

Computations of eigenvalues and eigenvectors is another important field in numerical analysis. The problem of calculating eigenvalues and eigenvectors of a matrix occurs in a number of contexts in particularly in physics and engineering handling oscillation problems.

A major topic is the solving of differential and integral equations. Most mathematical models used in the natural sciences and engineering are based on ordinary differential equations, partial differential equations, and integral equations. The numerical methods for these equations are primarily of two types. The first type approximates the unknown function in the equation by a simpler function, often a polynomial or piecewise polynomial function, choosing it to satisfy the original equation approximately. Most numerical methods for solving differential and integral equations involve both approximation theory and the solution of quite large linear and non-linear systems.

Solving linear systems of equations is another large sub-field of numerical analysis. Such systems occur in a large number of areas, e.g. physics, biology, social science and many more. Because of the widespread importance of linear systems a lot of research has been devoted to their numerical solutions. Efficient and robust algorithms have been developed for the most common types of problems for linear systems.

The most difficult problems involve systems of non-linear equations. Methods to compute solutions for such systems have been studied for centuries and a large part of numerical analysis is devoted to its research. In cryptography the ability to compute solutions for non-linear systems efficiently is directly connected to the security of cryptographic algorithms. For solving non-linear equations numerically iterative methods are necessary which compute a solution as a limit value of a sequence of approximations. In general one can use two approaches. One operates directly on the system represented as a n-th dimensional function, the other reformulates the system as a unconstrained or constrained optimization problem. In this context direct and indirect methods, convergence, existence of solutions, local and global solutions are important and discussed in the following sections.

4.2 Newton Method and its Variants

Complex problems often involve the computation of solutions for non-linear equations or systems of non-linear equations. For this purpose iterative methods are necessary which compute a solution as the limit of a sequence of solution approximations [Sch97]. A general overview can be found for example in [QSS02] and detailed information can be found in [Spe93, Bjö96, Sch97, Deu04]. Finding a solution for a non-linear system of equations is the generalization of the root finding problem of a non-linear function extended to n-th dimension.

Let $F : \mathbb{R}^n \to \mathbb{R}^n$ be a system of n equations. Finding a solution for $F(x)$ means to find

$$x^* \text{ such that } F(x^*) = \vec{0}, \tag{4.1}$$

Let $C^k(D)$ be the set of all k-time ($k \geq 0$) continuous differentiable functions from D to $\mathbb{R}^n$. We assume that $F \in C^1(D)$. In order to be able to construct the most basic methods we need the following definitions.

Definition 4.1 (Jacobi Matrix)**.** *$J_F(x)$ is the Jacobi matrix associated with F at point $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ and is defined as*

$$(J_F(x))_{ij} = \left(\frac{\partial F_i}{\partial x_j}\right)(x),$$

for $i, j = 1, \ldots, n$.

Definition 4.2 (Lipschitz continuous)**.** *Let f be a function $f : A \to A$, A a closed subset of a $\mathbb{R}$. f is called Lipschitz continuous or is said to satisfy a Lipschitz condition, if*

$$\|f(x) - f(y)\| \leq \omega \cdot \|x - y\|, \forall x, y \in A.$$

f is called a contraction if $\omega < 1$. ω is called the Lipschitz constant.

One of the most well-known methods to solve a system of non-linear equations is the Newton method. The Newton method for the n-th dimensional case is

a direct extension of the basic Newton method for one dimensional real-valued functions. Given a system of equations $F \in C^1(D)$ the Newton method can be described as follows:

$$\begin{aligned} &\text{solve } J_F(x^k)\Delta x^k = -F(x^k), \\ &\text{set } x^{k+1} = x^k + \Delta x^k. \end{aligned} \tag{4.2}$$

Starting with an initial guess x^0 of the unknown solution x^*, this is repeated (for $k = 0, 1, 2, \dots$) until it converges to x^*. Numerical methods were the initial guess has to be close to the solution are called local methods. For global methods this is not necessary. The following theorems provide more details on the reason for this differentiation. The convergence of the Newton method depends heavily on the initial guess x^0 which is expressed in the following theorem [QSS02]:

Theorem 4.1. *Let $F : \mathbb{R}^n \to \mathbb{R}^n$ and $F \in C^1(D)$ on a convex set $D \subset \mathbb{R}^n$ which includes x^*. Assume that $J_F(x^*)$ is invertible and that there exist positive constants R, C and L such that $J_F^{-1}(x^*) \leq C$ and*

$$\|J_F(x) - J_F(y)\| \leq L\|x - y\| \qquad \forall x, y \in B(x^*; R).$$

Then there exists $r > 0$ such that for each $x^0 \in B(x^*; r)$ the sequence (4.2) is uniquely determined and converges towards x^*, whereby

$$\|x^{k+1} - x^*\|\| \leq CL\|x^k - x^*\|^2.$$

Theorem 4.1 proves quadratic convergence of the Newton method if x^0 is sufficient close to the solution x^* and the Jacobi matrix is not singular. Furthermore, solving the linear system (4.2) can be computational expensive for large n and if the Jacobi matrix is ill-conditioned achieving an exact solution can be very difficult. Therefore, several variants of the Newton method were constructed to handle ill-conditioned matrices and other problems. For more information we refer to [Deu04]. More insight in local Newton methods gives us the refined Newton-Mysovskikh theorem [DP92].

Theorem 4.2. *Let $F : \mathbb{R}^n \to \mathbb{R}^n$ and $FinC^1(D)$, where D is open and convex. Suppose that $J_F(x)$ is invertible for each $x \in D$. Assume that following affine covariant Lipschitz condition holds:*

$$\|J_F(x)^{-1}(J_F(y) - J_F(x))(y - x)\| \leq L\|y - x\|^2,$$

for $x, y \in D$. Let $F(x) = 0$ have a solution x^*. For the initial guess x^0 assume that $\overline{B}(x^*, \|x^0 - x^*\|) \subset D$ and that

$$L\|x^0 - x^*\| < 2.$$

Then the ordinary Newton iterates defined by (4.2) remain in the open ball $B(x^*, \|x^0 - x^*\|)$ and converge to x^* at an estimate rate

$$\|x^{k+1} - x^*\| \leq \frac{1}{2}L\|x^k - x^*\|^2.$$

Moreover, the solution x^* is unique in the open ball $B(x^*, 2/L)$.

Theorem 4.2 presents an appropriate local convergence condition of the form

$$\|x^0 - x^*\| < 2/L.$$

Under this condition Newton methods are guaranteed to converge. The computational complexity of such problems is priori bounded in terms of the computational complexity of solving linear problems of the same structure. In contrast under condition

$$\|x^0 - x^*\| \gg 2/L$$

Newton methods will not provide guaranteed convergence. Therefore, the computational complexity cannot be bounded a priori. Such problems are often called highly non-linear. For large systems it is hard to prove Lipschitz continuity and even harder to compute the actual Lipschitz constant L, which is required to determine a reasonable starting value. For very low dimensions, e.g. 3, it is possible to find a starting value through geometric interpretation, but for high dimensional problems (like in the case of cryptography) geometric interpretation is infeasible.

For a general mapping F, a globalization of the Newton methods must be constructed. Only globally convergent methods are capable of using an arbitrary starting point (under restrictions). However, all iterative solvers have in common that the starting point has a high influence on the convergence. In our case we know that the solution is in a specific sub domain (see Chapter 3) and therefore we can restrict the start value to this domain. By guessing the values of variables, which is a common approach in cryptanalysis, we can shrink this domain even more. Guessing values means that an exhaustive search on a specific number of variables is done. First the values for a number of variables are guessed. Then the attack is applied. If the attack succeeds, the guess was correct, otherwise the attack with a new guess is applied again. Note that even if we are in the real domain we consider only two possibilities for each variable, depending on the conversion method. With the increase of dimension both restrictions to the search area lose in value. The difficulty of the start value determination with increasing dimension is often called "curse of dimensionality". If the available information is not good enough to provide good starting points the best one can do is to choose random values normally distributed in the search domain.

As the need for global methods is further accentuated in Chapter 5 we use different globalization concepts to overcome the problem of finding a good initial guess. Such concepts are steepest descent methods, damping strategies or trust-region methods. The concept of trust-region is utilized in several different numerical methods and is nowadays one of the most used concepts. One method using trust-region is called *Interior Reflective Newton Method* by Coleman and Li [CL96] which has additional properties which are useful for our approach. Also the Levenberg-Marquardt method can be viewed as using a trust-region approach. The Gauss-Newton method can be extended by a damping strategy. These methods are explained in Section 4.4. Most of the advanced numerical methods which provide global convergence are designed for minimization

problems. These methods usually merge into a local Newton variant for faster convergence.

4.3 Minimization Problem

The problem of solving a system of equations can either be directly addressed, as the Newton method does or indirectly by transforming the problem in an minimization problem. First problem (4.1) is reformulated to an *unconstrained minimization problem* by introducing a *objective function* $f : \mathbb{R}^n \to \mathbb{R}$.

Definition 4.3 (Unconstrained Minimization Problem)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be the objective function, The unconstrained minimization problem is defined as the problem of finding*

$$x^* = \min_{x \in \mathbb{R}^n} f(x).$$

A typical example is to determine the optimal distribution of n resources $x_1, x_2, \ldots, x_n$. In general there will exist restrictions for the solution space such that the solution is in a subspace $D \subset \mathbb{R}^n$. Furthermore, such a problem may have equality and inequality constraints, turning the initial problem in a *constrained optimization problem* which leads us to the following definition.

Definition 4.4 (Constrained Minimization Problem)**.** *Let $D \subseteq \mathbb{R}^n$, $f : D \to \mathbb{R}$ be the objective function, $g : D \to \mathbb{R}^m$ be the inequality constraints and $h : D \to \mathbb{R}^p$ be the equality constraints. The constrained minimization problem is defined as the problem of finding*

$$x^* = \min_{x \in \Omega} f(x),$$

where $\Omega := \{x \in D : g(x) \geq 0, h(x) = 0\}$.

Solving a system of non-linear equations and constrained or unconstrained minimization are strongly connected. Let F_i be the components of F, then the point x^*, which is a solution of (4.1), is a minimum of the function

$$f(x) = \|F(x)\|_2^2 = \sum_{i=1}^{n} F_i^2(x). \tag{4.3}$$

Conversely, under the assumption of differentiability of f, setting the partial derivatives of f in a point x^* to zero leads to a system of non-linear equations. Therefore, every system of non-linear equations can be connected with a eligible minimization problem and vice-versa. Basically any algorithm for minimization problems can be used for non-linear systems of equations. However, there exist several special algorithms for solving *least-squares* problems as given in Equation (4.3), e.g. the Levenberg-Marquardt and Gauss-Newton method.

In our approach solving a system of non-linear equations via a constrained minimization problem has some advantages. As shown in Chapter 3 the Boolean values `false` and `true` are mapped to real values. These values span a hypercube over the reals. Hence, the location of the solution can be reduced to this

hyper-cube, which can be expressed in a constrained minimization problem by defining upper and lower bounds on the variables. An additional advantage is that also overdetermined systems can now be handled, which can increase the accuracy and robustness of numerical methods, if the existence of a solution can still be guaranteed. Furthermore, any useful foreknowledge can be expressed as additional linear or non-linear constraints, e.g. if it is known that keys are in a specific subspace.

In the context of a minimization problem we need to differ between a global and local minimum.

Definition 4.5 (Minimum of a Function)**.** *Let $K(x^*, \epsilon) = \{x : \|x - x^*\| < \epsilon\}$ be the neighbourhood around x^* of function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$. Then x^* is called a local minimum if $f(x^*) \leq f(x)$ for all $x \in K(x^*, \epsilon)$ holds and a global minimum if $f(x^*) \leq f(x)$ for all $x \in D$ holds.*

A solution x^* for the system (4.1) is a global minimum of (4.3). A method providing global convergence does not guarantee to converge to the global minimum of a function, and therefore to the exact solution of the system of equations. Almost all practical numerical algorithms for minimization problems guarantee only to compute a local minimum of a function. The efficient and guaranteed computation of the global minimum is a hard problem and part of global optimization research. However, the ability to compute a solution for a system of non-linear equations via minimization depends strongly on their structure.

4.4 Numerical Methods Used in This Thesis

From the vast pool of available numerical methods we choose three different methods for the experimental results which provide global convergence. The classical methods are *Levenberg-Marquardt* and *Gauss-Newton*. The *Interior Reflective Newton* method is a more advanced method which is more robust than the classic methods and handles upper and lower bounds to the variables. This is especially useful in our case as all variables in our system of equations is lower and upper bounded (see Chapter 3). As shown in the results for Trivium (see Chapter 5) all methods get stuck in local minima. Therefore, we need to consider also algorithms for global minimization. We choose the DIRECT algorithm which as well handles upper and lower bounded variables. All chosen methods are well understood with detailed analysis and efficient implementations available.

4.4.1 Gauss-Newton Method

The Gauss-Newton method for the problem (4.3) is based on a sequence of linear approximations of $F(x)$ by

$$\tilde{F}(x) = F(x^k) + J_F(x^k)(x - x^k). \tag{4.4}$$

If x^k denotes the current approximation, then the next search direction Δx is a solution to the linear least-squares problem

$$\min_{\Delta x} \|\tilde{F}(x)\|_2^2 = \|F(x^k) + J_F(x^k)\Delta x\|_2^2 \tag{4.5}$$

for $\Delta x \in \mathbb{R}^n$. To compute the minimum of (4.5) one has to compute the gradient and set it to zero which results in the following linear system of equations

$$J_F(x^k)^T J_F(x^k)\Delta x = -J_F(x^k)^T F(x^k).$$

This system can be solved in different ways resulting in different variants of the Gauss-Newton method. The new approximation x^{k+1} is given then by

$$x^{k+1} = x^k + \Delta x,$$

resulting in the basic Gauss-Newton method. The basic Gauss-Newton method provides only local convergence. There are different globalization concepts which can be used to extend this method. One of them introduces a damping factor λ_k and changes the computation of the iterates to

$$x^{k+1} = x^k + \lambda_k \Delta x.$$

The damping factor λ_k has to be chosen carefully. A common way is the Armijo-Goldstein [Bjö96, OR00] step length principle, where λ_k is set to the largest number in the sequence of $1, \frac{1}{2}, \frac{1}{4}, \ldots$ for which the inequality

$$\|F(x^k)\|_2^2 - \|F(x^k + \lambda_k \Delta x)\|_2^2 \geq \tfrac{1}{2}\lambda_k \|J_F(x^k)\Delta x\|_2^2$$

holds. Another way to determine a appropriate damping factor is to set it to the solution to the one-dimensional minimization problem

$$\min_{\lambda} \|F(x^k + \lambda \Delta x)\|_2^2.$$

4.4.2 Levenberg-Marquardt Method

The Levenberg-Marquardt method is another algorithm to solve the least-squares problem (4.3) and is seen as a trust-region extension for the Gauss-Newton method, resulting in a more robust algorithm which handles 'bad' initial guesses better. A detailed analysis of the this method can be found in [Mor78, Deu04].

The Levenberg-Marquardt method is again based on a sequence of linear approximations of $F(x)$ by Equation(4.4). A correction vector (or search direction) Δx^k is determined by the constrained quadratic minimization problem

$$\min_{\Delta x} \|\tilde{F}(x)\|_2^2 = \|F(x^k) + J_F(x^k)\Delta x\|_2^2$$

subject to the constraint

$$\|\Delta x\|_2^2 \leq \delta$$

in terms of some prescribed parameter $\delta > 0$ which limits the size of Δx and ensures that (4.5) is reduced in each step. The set of feasible vectors Δx where $\|\Delta x\|_2 \leq \delta$ can be thought of as a region of trust for the linear model (4.4) which is why this method is seen as a trust-region extension of Gauss-Newton. The trust region constraint may be treated by the introduction of a Lagrange multiplier $p \geq 0$ subject to

$$p(\|\Delta x\|_2^2 - \delta^2) = 0,$$

which leads to the equivalent unconstrained quadratic optimization problem

$$\min_{\Delta x}\{\|F(x^k) + J_F(x^k)\Delta x\|_2^2 + p\|\Delta x\|_2^2\}. \tag{4.6}$$

To compute the minimum of (4.6) the gradient is computed and set equal to zero which results in the Levenberg-Marquardt method:

$$(J_F(x^k)^T J_F(x^k) + pI)\Delta x = -J_F(x^k)^T F(x^k),$$

$$x^{k+1} = x^k + \Delta x.$$

Depending on the strategy to choose the parameter p or equivalently the parameter δ different variants of this method can be defined. However, for any parameter $p > 0$ the matrix $(J_F(x^k)^T J_F(x^k) + pI)$ is non-singular, even when the the Jacobian matrix itself if singular. Nevertheless, the above iteration may also converge to 'small' gradients, since for singular $J(x^k)$ the right-hand side also degenerates.

4.4.3 Interior Reflective Newton Method

The *interior reflective Newton* method was invented by Coleman and Li [CL96]. It is well studied and efficient implementations are available. Furthermore, a detailed convergence analysis [Tho94] is available which is very useful for a better understanding of the algorithm. The interior reflective Newton method is designed to handle bound constrained minimization problems

$$\min_x f(x), l \leq x \leq u,$$

where $x, l, u \in \mathbb{R}^n$. A special and important property of this methods is that all iterates stay between user defined upper and lower bounds. As shown in Chapter 3 all variables in our equations have such known bounds. This makes the analysis of the system easier, since we only have to consider the hypercube formed by the bound and additionally it is ensured that bad properties outside of this cube do not influence the algorithm's convergence. The interior reflective Newton method achieves global convergence using the concept of trust regions. It increases the robustness of the method if the starting point is far from the solution and also handles the case when the Jacobi matrix is singular or ill-conditioned. As for the Levenberg-Marquardt method (4.6) a merit function is

used to decide if the next iterate x^{k+1} is better or worse than the current iterate x^k. Therefore, the increment $\Delta x_k = x_{k+1} - x_k$ is a solution to this standard quadratic sub-problem with a bound on the step:

$$\min_{\Delta x}\{\nabla f(x_k)^T \Delta x + \tfrac{1}{2}\Delta x^T \nabla^2 f(x_k)\Delta x\}, \tag{4.7}$$

subject to

$$\|D(x_k)\Delta x\|_2 \leq d_k.$$

D is a scaling matrix and d_k is a positive scalar representing the trust region size. Solving Problem (4.7) in a reliable and efficient way is a non-trivial task. Coleman and Li [CL96] proposed a quadratic model and scaling matrix such that there is no need to handle the upper and lower bound explicitly. They first introduced the following definition:

Definition 4.6. *The vector $v(x) \in \mathbb{R}^n$ is defined:*

1. *if $\nabla f(x)_i < 0$ and $u_i < \infty$ then $v_i = x_i - u_i$.*
2. *if $\nabla f(x)_i \geq 0$ and $l_i > -\infty$ then $v_i = x_i - l_i$.*
3. *if $\nabla f(x)_i < 0$ and $u_i = \infty$ then $v_i = -1$.*
4. *if $\nabla f(x)_i \geq 0$ and $u_i - = \infty$ then $v_i = 1$.*

The scaling matrix is defined as $D(x) = diag(|v(x)|^{-\frac{1}{2}})$.

They extend the *standard* quadratic model (4.7) to

$$\begin{aligned} &\min_{\Delta x}\{\psi(\Delta x) := \nabla f(x_k)^T \Delta x + \tfrac{1}{2}\Delta x^T(\nabla^2 f(x_k) + C(x_k))\Delta x\} \\ &\text{s.t. } \|D(x_k)\Delta x\|_2 \leq d_k, \end{aligned} \tag{4.8}$$

where $C(x) = D(x)diag(\nabla f(x))\nabla^2 |v(x)|D(x)$. In each iteration a solution Δx_k to the subproblem (4.8) is computed. To ensure a sufficient decrease of the objective function the following condition is determined:

$$p_k = \frac{f(x_k + \Delta x_k) - f(x_k) + \frac{1}{2}\Delta x_k^T C(x_k)\Delta x_k}{\psi(\Delta x_k)} > \mu$$

for some constant $\mu > 0$. If this conditions holds set $x_{k+1} = x_k + \Delta x_k$, otherwise set $x_{x+1} = x_k$. At the end of each iteration the trust region size d_k is updated according to the following rules:

1. if $p_k \leq \mu$ then set $d_{k+1} \in (0, \gamma_1 d_k]$.
2. if $p_k \in (\mu, \eta)$ then set $d_{k+1} \in [\gamma_1 d_k, d_k]$.
3. if $p_k \geq \eta$ and $d_k > \lambda$ then set $d_{k+1} \in [\gamma_1 d_k, d_k]$ or $[d_k, \gamma_2 d_k]$.
4. if $p_k \geq \eta$ and $d_k \leq \lambda$ then set $d_{k+1} \in [d_k, \gamma_2 d_k]$.

$0 < \mu < \eta 1$, $\gamma_1 < 1 < \gamma_2$ and $0 < \lambda$ are given parameters of the algorithm.

For more information on this method and for a detailed convergence analysis we refer to [CL96].

4.4.4 DIRECT algorithm

Apart from global convergence we also need global optimization techniques to find a solution for the system. Therefore, we used the DIRECT (DIviding RECTangles) algorithm by Jones *et al.* [JPS93]. It is a deterministic sampling method designed for finding the global minima for bound constrained optimization problems and does not need a starting point. It requires no knowledge of the objective function gradient. Instead, the algorithm samples points in the domain, and uses the information it has obtained to decide where to search next. It is guaranteed to converge to the global optimal function value, if the objective function is continuous or at least continuous in the neighbourhood of a global optimum. This guarantee can be given because if the number of iterations goes towards infinity the set of sampled points form a dense subset of the unit hypercube. Unfortunately, this global property may come at the expense of a large and exhaustive search over the domain.

The strengths of DIRECT lie in the balanced effort it gives to local and global searches, and the few parameters it requires to run. It was designed to overcome some of the problems in the minimization of Lipschitz continuous functions. As mentioned in Section 4.2 the Lipschitz constant of a Lipschitz continuous function can not be determined or reasonable estimated for highly non-linear problems. Many simulations with industrial applications may not even be Lipschitz continuous throughout their domains. Even if the Lipschitz constant can be estimated, a poor choice can lead to poor results.

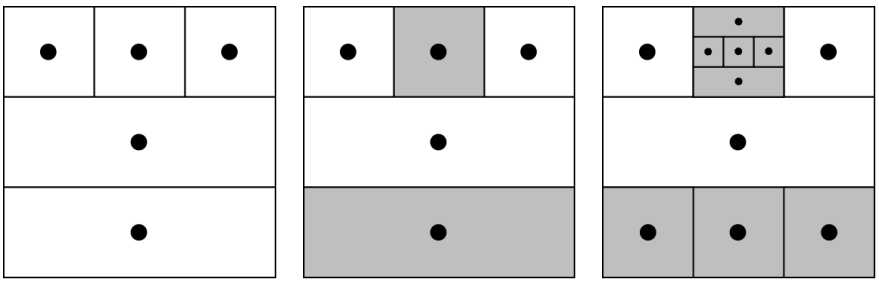
At the beginning DIRECT transforms the domain of the problem into the unit hypercube

$$\Omega = \{x \in \mathbb{R}^n : 0 \leq x_i \leq 1\}.$$

The algorithm operates in this space except for calls of the function which should be optimized. DIRECT proceeds by partitioning this space into rectangles, where each rectangle has a sampled point in the centre (see Figure 4.1(a) for $n = 2$). Each iteration begins witch a selection of one or more of current rectangles for further search (see Figure 4.1(b)). The second step is to evaluate the function at new points in the selected rectangles and subdivide them such that each new point becomes the centre of a new sub-rectangle (see Figure 4.1(c)). In the selection process rectangle r is selected if there exists an f^* satisfying

$$\begin{gathered}(f_r - f^*)/\delta_r \leq (f_s - f^*)/\delta_s, \ \forall s \neq r \\ f^* \leq f_{min} - \epsilon,\end{gathered}$$

where f_r and f_s is the function value at the centre of rectangle r and s respectively. $\delta > 0$ is called the accuracy. After the selection process, new points are sampled in the chosen rectangles and used to create new sub-rectangles. Therefore, the function is evaluated at the points $c \pm \delta \cdot e_i$ for all $i \in I$ where I is the set of dimensions corresponding to the sides of the rectangle with maximum side length, δ is one third of this length, c the centre of the rectangle and e_i is the i-th unit vector. The algorithm chooses to leave the best (lowest) function



(a) Start (b) Select (c) Sample and divide

Figure 4.1: Example iteration of the algorithm.

values in the largest space by computing

$$w_i = \min\{f(c + \delta \cdot e_i), f(c - \delta \cdot e_i)\}$$

and splitting the rectangle containing c into thirds along the dimension in I with the smallest w_i such that $c \pm \delta e_i$ are the centres of the new sub-rectangles. This dimension is deleted from I. This process continues until the set I is empty. The algorithm now begins its loop of identifying potentially optimal hyper-cubes, dividing these cubes appropriately, and sampling at their centres until an iteration limit is reached.

4.5 Boolean Conversion and Numerical Methods

No matter which conversion method we choose, the resulting polynomials over the reals have desirable properties considering numerical methods:

- Multilinearity
- Small coefficients
- Continuous differentiability

An important fact is that the values for the variables depend on the type of conversion. If we consider the Fourier representation from Section 3.5.1 then the conversion of the arbitrary system (3.1) results in

$$F(y) = \begin{pmatrix} \prod_{I_1 \subseteq M} c(\vec{y}_{I_1}) - t(b_1) \\ \vdots \\ \prod_{I_n \subseteq M} c(\vec{y}_{I_n}) - t(b_n) \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{4.9}$$

where $c(\vec{y}_{I_i})$ is defined in (3.20) and $y = (y_1, \cdots, y_n)$. The Boolean domain $\{\texttt{false}, \texttt{true}\}$ is mapped to $\{1, -1\}$. This leads to the fact that $F(y)$ is restricted

to the set $D := \{1, -1\}^n$. Hence, the resulting function F can be considered as

$$F : D \subset \mathbb{R}^n \to \mathbb{R}^n.$$

D describes a convex hyper cube, which is an important prerequisite for the convergence of numerical methods. Since $c(\vec{y}_{I_i})$ for $i = 1, \cdots, n$ is continuously differentiable on D, $F(y)$ is also continuously differentiable on D. Furthermore, the Jacobian of (4.9) can be analytically determined, which increases the robustness and accuracy of numerical methods.

4.5.1 Starting Point

For local convergent methods a starting point is needed, which is "sufficiently close" to the solution y^*. Unfortunately, it is computational infeasible to check the Lipschitz continuity or to compute the Lipschitz constant for large n, such that a good starting point can be determined. In practice this property is assumed to be available. Since the Lipschitz constant is not available it can only be chosen randomly without further knowledge. However, we know that the solution y^* is a corner of the hyper cube D. A good starting point would be a point near the solution corner, which implicates that we already know the solution. We can choose a different corner, where only few coordinates (variables) differ from the solution. Another possibility is to test each of 2^n corners which is equal to exhaustive search. An additional method for the choice, beside the random option, is to set the centre of the hyper cube as starting point. Our experiments showed that this approach is useful for small systems of equations. Even for global convergent methods these observations are useful to determine the starting point.

4.5.2 Existence and Uniqueness of Solutions

An important question is how we interpret the solution of the converted system as a solution for the Boolean system. From the definitions of the four representation types we know that the Boolean domain is mapped to appropriate numbers. According to Lemma 3.1 and the equivalent lemmata for the other representations, the converted equations have equivalent solutions. This leads to the following corollary.

Corollary 4.1. *Let $F_b(x)$ be a system of Boolean equations, $F_r(y)$ the resulting system using one of the conversion methods in Chapter 3 and x^* a solution of $F_b(x) =$ `false`. Then $y^* = t(x^*)$ is a solution of $F_r(y) = 0$, where t is the Boolean domain mapping from the definitions of the representation types.*

Now we have an important fact available: If the Boolean system has a solution, then there exists a solution for the converted system, which is a corner of the hyper cube D. However, it is possible that additional real-valued solutions exist which cannot be directly converted back to the Boolean domain. Hence, we can guarantee *existence*, but not *uniqueness* of solutions.

4.6 Summary

In this chapter, we gave an overview on numerical analysis. Since numerical analysis is a broad research field we focused on iterative methods for solving non-linear systems of equations. We introduced the necessary terminology and definitions. First of all we distinguished between local and global convergent methods. For local convergent methods a 'good' starting point has to be provided, i.e. a point already close to the solution. As it is shown in Chapter 5, we cannot provide such a point. This is not necessary for global convergent methods, which, under specific prerequisites, converge even when started far away from the solution. Hence, we considered only global convergent methods and presented three methods in detail, namely Gauss-Newton, Levenberg-Marquardt and interior reflective Newton method. They use different concepts to achieve global convergence.

Modern iterative methods, as the interior reflective Newton method, are designed for minimization problems, since solving a system of equations is strongly related to it. Hence, we introduce the minimization problem and define local and global minima. A point is a local minimum of a function if it is only minimal within a specific area. A point is a global minimum if the function at this point is minimal overall points in the domain. Although, iterative methods for minimization problems provide global convergence, they can also converge to local minima. Usually, a local minimum is not a solution of the system of equations. Finding a global minimum for non-linear functions is a hard problem and a whole research field (global optimization) is dedicated to it. Nevertheless, we used one method for global minimization, called DIRECT. The strengths of DIRECT lie in the balanced effort it gives to local and global searches, and the few parameters it requires to run.

Finally, we worked out properties of the converted equations (see Chapter 3) with respect to numerical analysis. No matter which conversion method we choose, the resulting polynomials over the reals have desirable properties considering numerical methods. Foremost, all polynomials are continuous differentiable. Second, if a solution for the Boolean system of equations exists (and in the case of cryptographic primitives it does), we can guarantee that the converted system has at least the same amount of solutions. However, it is possible that additional real-valued solutions exist which cannot be directly converted back to the Boolean domain. Hence, we can guarantee *existence*, but not *uniqueness* of solutions.

5 Application to Trivium

In this chapter we apply our approach on the stream cipher Trivium and two reduced variants – called Bivium A and Bivium B. Trivium is recommended by the eStream project [Rob08] in the hardware category. We first set up a system of equations describing the internal state of the cipher and convert it into a system over the reals. We apply four different conversion methods, standard, Fourier, adapted standard and splitting conversion (see Chapter 3). Hence, we obtain different systems of equations over the reals for Trivium and its variants. Finally, we apply numerical methods presented in Chapter 4 and discuss the results.

5.1 Earlier Work on Trivium

Several papers have been proposed about cryptanalytic results on Trivium. Khazaei and Hassanzadeh [KH05] showed that Trivium is strong against the linear sequential circuit approximation attack in spite of the extra simplicity of its output function and next-state function. Turan and Kara [TK07] define the initialization step of Trivium as an 8-round function and try to attack the initialization with a smaller number of rounds. Maximov and Biryukov [MB07] developed two attacks on Trivium with decreased complexity compared to the work of Raddum [Rad07]. Raddum developed a new technique to solve systems of equations and applied these to the equation system representing Trivium and the reduced variants. He successfully broke Bivium A and B, but with high complexity for the second variant. The full version of Trivium resists his attack. McDonald *et al.* attacked Bivium with MiniSat in [MCP08], by transforming the equations into a satisfiability problem. They estimate the complexity for the at-

tack on Bivium B to be about 2^{52} operations. This algebraic attack recovers the private key after observing only 1770 bits of keystream. Bivium A is completely broken requiring only 177 bits of the keystream. Eibach *et al.* [EPV08] attacked also Bivium B with SAT solvers. Fischer *et al.* [FKM08] can successfully recover the initial key by using statistical distinguishers with complexity about 2^{55} if the number of iterations in the initialization step is reduced down to 672. Dinur and Shamir [DS09] proposed a new method in cryptanalysis, called the cube attack. Using this technique, they can successfully recover the initial key with complexity about 2^{45} if the iterations in the initialization step are reduced down to 767.

5.2 Trivium

Trivium was designed by Christophe De Cannière in 2005 [Rob08, Can06]. Trivium is a synchronous stream cipher and generates up to 2^{64} bits of keystream from an 80-bit secret key (K) and an 80-bit initial value (IV). It consists of two phases: First the internal state of the cipher is initialized using the key and the IV, then the state is repeatedly updated and used to generate keystream-bits. The internal state consist of a 288-bit register with a nonlinear feedback. Figure 5.1 shows a schematic view of the construction.

5.2.1 Initialization Phase

The internal state is represented by a 288-bit register $\vec{s}$. In the initialization step an 80-bit key and an 80-bit initial vector are loaded into the register (see Equation 5.1). Then, the state is rotated over four full cycles (without generating any keystream), where one cycle is the execution of the algorithm 288 times.

$$\begin{aligned}(s_1, s_2, \cdots, s_{93}) &\leftarrow (K_1, K_2, \cdots, K_{80}, 0, \cdots, 0)\\ (s_{94}, s_{95}, \cdots, s_{177}) &\leftarrow (IV_1, IV_2, \cdots, IV_{80}, 0, \cdots, 0)\\ (s_{178}, s_{95}, \cdots, s_{288}) &\leftarrow (0, \cdots, 0, 1, 1, 1)\end{aligned} \tag{5.1}$$

5.2.2 Keystream Generation

After the initialization phase, Trivium starts to generate keystream-bits. As illustrated in Figure 5.1, Trivium consists only of XOR and AND operations. The loop in (5.2) represents the keystream generation algorithm, where in each iteration one keystream-bit z_i is generated.

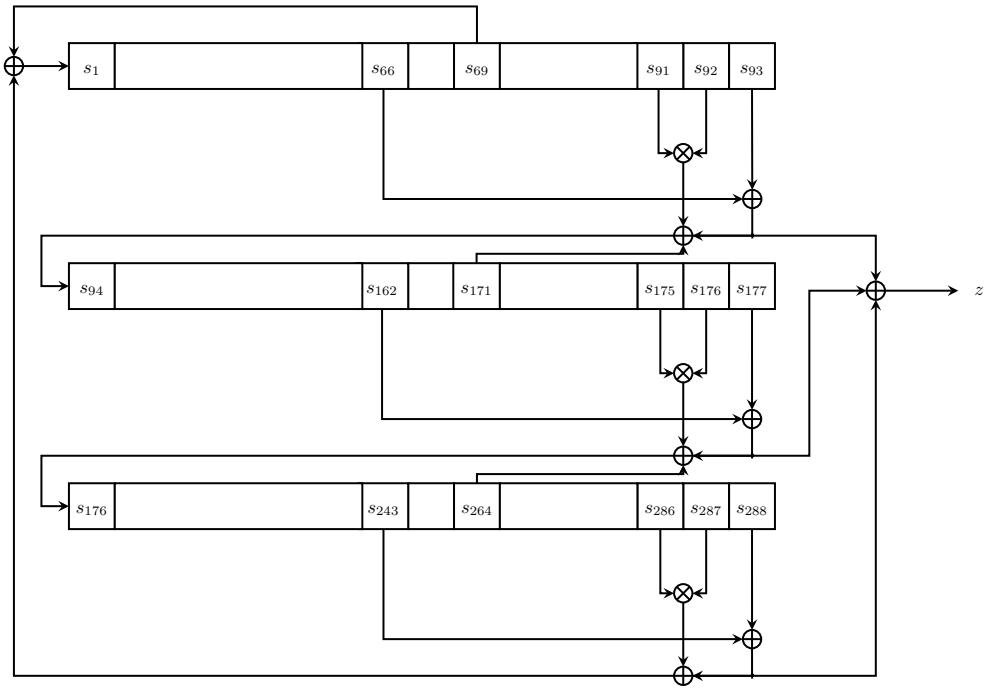


Figure 5.1: Schematics of Trivium.

For i=1 to N do

$$
\begin{aligned}
t_1 &\leftarrow s_{66} \oplus s_{93} \\
t_2 &\leftarrow s_{162} \oplus s_{177} \\
t_3 &\leftarrow s_{243} \oplus s_{288} \\
z_i &\leftarrow t_1 \oplus t_2 \oplus t_3 \\
t_1 &\leftarrow t_1 \oplus s_{91} \wedge s_{92} \oplus s_{171} \\
t_2 &\leftarrow t_2 \oplus s_{175} \wedge s_{176} \oplus s_{264} \\
t_3 &\leftarrow t_3 \oplus s_{286} \wedge s_{287} \oplus s_{69} \\
(s_1, s_2, \cdots, s_{93}) &\leftarrow (t_3, s_1, s_2, \cdots, s_{92}) \\
(s_{94}, s_{95}, \cdots, s_{177}) &\leftarrow (t_1, s_{94}, s_{95}, \cdots, s_{176}) \\
(s_{178}, s_{95}, \cdots, s_{288}) &\leftarrow (t_2, s_{178}, s_{95}, \cdots, s_{287})
\end{aligned}
\tag{5.2}
$$

end for

5.2.3 Bivium A and Bivium B

In addition to Trivium we apply our approach on two reduced variants. In [Can06] the designers describe the basic construction of the Trivium design which consists of a 288-bit register separated in three parts. Raddum [Rad07] formalized their description and introduced two reduced variants, called Bivium A and Bivium B. Bivium A is constructed by removing the third part of the register and adapting the keystream generator algorithm accordingly. The internal state has a length of 177 bits. In the initialization phase the third step of (5.1) is omitted and the keystream generation algorithm is changed to:

For i=1 to N do

$$
\begin{aligned}
t_1 &\leftarrow s_{66} \oplus s_{93} \\
t_2 &\leftarrow s_{162} \oplus s_{177} \\
z_i &\leftarrow t_2 \\
t_1 &\leftarrow t_1 \oplus s_{91} \wedge s_{92} \oplus s_{171} \\
t_2 &\leftarrow t_2 \oplus s_{175} \wedge s_{176} \oplus s_{69} \\
(s_1, s_2, \cdots, s_{93}) &\leftarrow (t_2, s_1, s_2, \cdots, s_{92}) \\
(s_{94}, s_{95}, \cdots, s_{177}) &\leftarrow (t_1, s_{94}, s_{95}, \cdots, s_{176})
\end{aligned}
\tag{5.3}
$$

end for

The specification of Bivium B is equal to Bivium A except the keystream-bit computation $z_i \leftarrow t_2$ is changed to $z_i \leftarrow t_1 \oplus t_2$.

5.3 Constructing Systems of Equations

For our approach we need a representation of the cipher as a system of equations. In this section we show how each system is constructed. In order to get a solvable system we apply a known-plaintext attack to compute the keystream-bits of the stream cipher. In a known-plaintext scenario, where the plaintext and corresponding ciphertext is known to the adversary, one can compute the keystream. Using this information the adversary tries to recover the initial key. If the initial key is successfully recovered, one can reproduce any sequence of keystream-bits. This should be difficult for a strong cipher.

5.3.1 System for Trivium

In the same way as in [Rad07] the system can be derived from the keystream generation algorithm (5.2). We denote the register bits $s_1, \cdots, s_{288}$ as the variables of the system. In each step we get four equations. One for the keystream-bit (5.4) and three for the register feedback (5.5).

$$s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288} = z_i \tag{5.4}$$

To keep the system sparse and to avoid long equations, new variables are introduced at each iteration.

$$\begin{aligned} s_{66} \oplus s_{93} \oplus s_{91} \wedge s_{92} \oplus s_{171} &= s_{289} \\ s_{162} \oplus s_{177} \oplus s_{175} \wedge s_{176} \oplus s_{264} &= s_{290} \\ s_{243} \oplus s_{288} \oplus s_{286} \wedge s_{287} \oplus s_{69} &= s_{291} \end{aligned} \tag{5.5}$$

To get a fully determined system, we need to clock Trivium 288 times, which is equivalent to producing 288 keystream-bits. The last $3 \cdot 66$ equations can be dropped since the introduced variables are not used in any keystream-bit equation [Rad07]. Hence, the resulting system over the Boolean domain has 954 equations and variables. Note that, due to the known-plaintext attack we know the keystream-bits z_i.

Finding a solution of the system is equivalent to reconstruct the internal state. Once the state is known one can clock the cipher backwards to reconstruct the secret key. Note, that it is possible to find more than one solution by solving the system, which does not lead to the correct key. However, in the attack we assume that for a strong cipher the amount of solutions is rather limited. To overcome this problem, one can generate more keystream-bits to get an overdetermined system of equations.

5.3.2 System for Bivium B

In the same way we construct the system of equations for Bivium B. Therefore, we use the keystream generation algorithm (5.3) with

$$z_i \leftarrow t_1 \oplus t_2.$$

We denote the register bits $s_1, \cdots, s_{177}$ as the variables of the system. In each step we get now three equations. One for the keystream-bit (5.6) and two for the register feedback (5.7).

$$s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} = z_i. \tag{5.6}$$

To keep the system sparse and to avoid long equations, new variables are introduced at each iteration.

$$\begin{aligned} s_{66} \oplus s_{93} \oplus s_{91} \wedge s_{92} \oplus s_{171} &= s_{178} \\ s_{162} \oplus s_{177} \oplus s_{175} \wedge s_{176} \oplus s_{69} &= s_{179} \end{aligned} \tag{5.7}$$

To get a fully determined system, we need to clock Bivium B 177 times, which is equivalent to producing 177 keystream-bits. The last $2 \cdot 66$ equations can be dropped since the introduced variables are not used in any keystream-bit equation. Hence, the resulting system has 399 equations and variables.

5.3.3 System for Bivium A

Bivium A represents the least complex problem. The equations are derived from (5.3). We denote the register bits $s_1, \cdots, s_{177}$ as the variables of the system. In each step we get now three equations. One for the keystream-bit (5.8) and two for the register feedback (5.9).

$$s_{162} \oplus s_{177} = z_i \tag{5.8}$$

To keep the system sparse and to avoid long equations, new variables are introduced at each iteration.

$$\begin{aligned} s_{66} \oplus s_{93} \oplus s_{91} \wedge s_{92} \oplus s_{171} &= s_{178} \\ s_{162} \oplus s_{177} \oplus s_{175} \wedge s_{176} \oplus s_{69} &= s_{179} \end{aligned} \tag{5.9}$$

To get a fully determined system, we need to clock Bivium A 177 times, which is equivalent to producing 177 keystream-bits. The last $2 \cdot 66$ equations can be dropped since the introduced variables are not used in any keystream-bit equation. Additionally, 72 more equations and variables can be dropped since only t_2 is used for the keystream-bit computation. In total we get 327 equations in 327 variables.

5.4 Conversion to the Real Domain

The second step in our approach consists of the conversion of each equation of the targeted system to an equation over the real domain. In Chapter 3 we presented several methods for the conversion. According to our rule of thumb in Section 3.6, Fourier or sign conversion should be preferred, since the equations for Trivium and its reduced variants consist of more XOR than AND operations. Nevertheless, we also use the standard conversion to have a comparison of both methods. Due to the similarities of sign and Fourier conversion we omit the use of sign conversion as mentioned in Section 3.6. Since the total degree of the converted equations is high we apply the advanced conversion techniques: adapted standard and splitting conversion. Hence, we focus on four different methods. In the following we give a detailed overview of the conversion results of the systems constructed in Section 5.3. We denote r_i as the variables over the reals corresponding to the Boolean variables s_i.

5.4.1 Conversion of Trivium Equations

The conversion of a Boolean equation to an equation over the reals depends only on its structure. The system constructed in Section 5.3.1 consists of two types of equations. The linear equations for the keystream-bit computation (5.4) define Type I and are of the form:

$$s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288} = z_i. \tag{5.10}$$

The remaining non-linear equations define type II and are of the form:

$$s_{66} \oplus s_{93} \oplus s_{91} \wedge s_{92} \oplus s_{171} \oplus s_{289} = \texttt{false}. \tag{5.11}$$

Without loss of generality we convert these representative equations using the four conversion methods.

Standard Conversion

In order to emphasize the different results for different conversion methods, we present the whole converted equations despite the bad readability. Applying the

conversion method described in Section 3.2 on Equation (5.10) results in

$$
\begin{aligned}
&r_{162} + r_{177} - 2r_{162}r_{177} + r_{243} - 2r_{162}r_{243} - 2r_{177}r_{243} + 4r_{162}r_{177}r_{243} \\
&+ r_{288} - 2r_{162}r_{288} - 2r_{177}r_{288} + 4r_{162}r_{177}r_{288} - 2r_{243}r_{288} + 4r_{162}r_{243}r_{288} \\
&+ 4r_{177}r_{243}r_{288} - 8r_{162}r_{177}r_{243}r_{288} + r_{66} - 2r_{162}r_{66} - 2r_{177}r_{66} \\
&+ 4r_{162}r_{177}r_{66} - 2r_{243}r_{66} + 4r_{162}r_{243}r_{66} + 4r_{177}r_{243}r_{66} \\
&- 8r_{162}r_{177}r_{243}r_{66} - 2r_{288}r_{66} + 4r_{162}r_{288}r_{66} + 4r_{177}r_{288}r_{66} \\
&- 8r_{162}r_{177}r_{288}r_{66} + 4r_{243}r_{288}r_{66} - 8r_{162}r_{243}r_{288}r_{66} \\
&- 8r_{177}r_{243}r_{288}r_{66} + 16r_{162}r_{177}r_{243}r_{288}r_{66} + r_{93} - 2r_{162}r_{93} \\
&- 2r_{177}r_{93} + 4r_{162}r_{177}r_{93} - 2r_{243}r_{93} + 4r_{162}r_{243}r_{93} + 4r_{177}r_{243}r_{93} \\
&- 8r_{162}r_{177}r_{243}r_{93} - 2r_{288}r_{93} + 4r_{162}r_{288}r_{93} + 4r_{177}r_{288}r_{93} \\
&- 8r_{162}r_{177}r_{288}r_{93} + 4r_{243}r_{288}r_{93} - 8r_{162}r_{243}r_{288}r_{93} \\
&- 8r_{177}r_{243}r_{288}r_{93} + 16r_{162}r_{177}r_{243}r_{288}r_{93} - 2r_{66}r_{93} \\
&+ 4r_{162}r_{66}r_{93} + 4r_{177}r_{66}r_{93} - 8r_{162}r_{177}r_{66}r_{93} + 4r_{243}r_{66}r_{93} \\
&- 8r_{162}r_{243}r_{66}r_{93} - 8r_{177}r_{243}r_{66}r_{93} + 16r_{162}r_{177}r_{243}r_{66}r_{93} \\
&+ 4r_{288}r_{66}r_{93} - 8r_{162}r_{288}r_{66}r_{93} - 8r_{177}r_{288}r_{66}r_{93} \\
&+ 16r_{162}r_{177}r_{288}r_{66}r_{93} - 8r_{243}r_{288}r_{66}r_{93} + 16r_{162}r_{243}r_{288}r_{66}r_{93} \\
&+ 16r_{177}r_{243}r_{288}r_{66}r_{93} - 32r_{162}r_{177}r_{243}r_{288}r_{66}r_{93} = t(z_i).
\end{aligned}
$$

Applying the conversion method described in Section 3.2 on Equation (5.11) results in

$$
\begin{aligned}
&r_{171} + r_{289} - 2r_{171}r_{289} + r_{66} - 2r_{171}r_{66} - 2r_{289}r_{66} + 4r_{171}r_{289}r_{66} \\
&+ r_{91}r_{92} - 2r_{171}r_{91}r_{92} - 2r_{289}r_{91}r_{92} + 4r_{171}r_{289}r_{91}r_{92} \\
&- 2r_{66}r_{91}r_{92} + 4r_{171}r_{66}r_{91}r_{92} + 4r_{289}r_{66}r_{91}r_{92} \\
&- 8r_{171}r_{289}r_{66}r_{91}r_{92} + r_{93} - 2r_{171}r_{93} - 2r_{289}r_{93} \\
&+ 4r_{171}r_{289}r_{93} - 2r_{66}r_{93} + 4r_{171}r_{66}r_{93} + 4r_{289}r_{66}r_{93} \\
&- 8r_{171}r_{289}r_{66}r_{93} - 2r_{91}r_{92}r_{93} + 4r_{171}r_{91}r_{92}r_{93} \\
&+ 4r_{289}r_{91}r_{92}r_{93} - 8r_{171}r_{289}r_{91}r_{92}r_{93} + 4r_{66}r_{91}r_{92}r_{93} \\
&- 8r_{171}r_{66}r_{91}r_{92}r_{93} - 8r_{289}r_{66}r_{91}r_{92}r_{93} + 16r_{171}r_{289}r_{66}r_{91}r_{92}r_{93} = 0.
\end{aligned}
$$

Comparing the result of type I and type II equations, we see that the monomial degree increases exponentially with the number of XOR operations, as shown in Chapter 3. The monomial degree for type I equations is 63 and 31 for type II equations. The total degree is for both cases 6. Both factors can be considered as relatively high.

Fourier Conversion

Using the Fourier conversion results in a less complex structure as shown in the following. Applying the conversion method described in Section 3.5 on Equa-

tion (5.10) results in

$$r_{66}r_{93}r_{162}r_{177}r_{243}r_{288} = t(z_i).$$

Applying the conversion method described in Section 3.5 on Equation (5.11) results in

$$\begin{aligned}&\tfrac{1}{2}r_{171}r_{289}r_{66}r_{93} + \tfrac{1}{2}r_{171}r_{289}r_{66}r_{91}r_{93}\\ +&\tfrac{1}{2}r_{171}r_{289}r_{66}r_{92}r_{93} - \tfrac{1}{2}r_{171}r_{289}r_{66}r_{91}r_{92}r_{93} = 1.\end{aligned}$$

Compared to the standard representation the resulting equations are significantly less complex. Hence, the monomial degree for type I equations is 1 and 4 for type II equations. However, the total degree is 6 and still considerable high.

Adapted Standard Conversion

In order to use the method described in Section 3.8.1 for type I equations, we need to determine the truth table of the Boolean function

$$f(\vec{s}) = s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288}$$

and compute the solution set $S := \{\vec{s} : f(\vec{s}) = z_i\}$. Next we evaluate the real function

$$f_r(\vec{r}) = r_{66} + r_{93} + r_{162} + r_{177} + r_{243} + r_{288}$$

for $\vec{r} \in S_r := \{f(\vec{r}) : \vec{r} = t(\vec{s}) \text{ for } \vec{s} \in S\}$. Let $z_i = \texttt{false}$ and therefore $t(z_i) = 0$. Since Equation (5.10) is linear over B, S_r consist of all possible even numbers of the sum of six variables and therefore $S_r = \{0, 2, 4, 6\}$. Hence, Equation (5.10) is converted to

$$\begin{aligned} d_1(r_{66} + r_{93} + r_{162} + r_{177} + r_{243} + r_{288}) &= 0\\ d_2(r_{66} + r_{93} + r_{162} + r_{177} + r_{243} + r_{288} - 2) &= 0\\ d_3(r_{66} + r_{93} + r_{162} + r_{177} + r_{243} + r_{288} - 4) &= 0\\ d_4(r_{66} + r_{93} + r_{162} + r_{177} + r_{243} + r_{288} - 6) &= 0\\ d_1 + d_2 + d_3 + d_4 &= 1 \end{aligned} \tag{5.12}$$

where $d_i \in \mathbb{R}$ for $i = 1, \ldots 4$. The case $z_i = \texttt{true}$ works analogous except that $S_r = \{1, 3, 5\}$ and therefore one equation and one variable less is generated. Let x denote the number of $z_i = \texttt{true}$.

The conversion of type II equations are done in the same way. We determine the truth table of the Boolean function

$$f(\vec{s}) = s_{66} \oplus s_{93} \oplus s_{91} \wedge s_{92} \oplus s_{171} \oplus s_{289}$$

and compute the solution set $S := \{\vec{s} : f(\vec{s}) = \texttt{false}\}$. Then we evaluate the real function

$$f_r(\vec{r}) = r_{66} + r_{93} + r_{91} \cdot r_{92} + r_{171} + r_{289}.$$

In that case $S_r = \{0, 2, 4\}$ and the conversion of Equation (5.11) results in

$$\begin{aligned} d_5(r_{66} + r_{93} + r_{91} \cdot r_{92} + r_{171} + r_{289}) &= 0 \\ d_6(r_{66} + r_{93} + r_{91} \cdot r_{92} + r_{171} + r_{289} - 2) &= 0 \\ d_7(r_{66} + r_{93} + r_{91} \cdot r_{92} + r_{171} + r_{289} - 4) &= 0 \\ d_5 + d_6 + d_7 &= 1 \end{aligned} \tag{5.13}$$

where $d_i \in \mathbb{R}$ for $i = 5, \ldots, 7$. Using the adapted standard conversion we decrease both the monomial and total degree by the cost of additional equations and variables. The Boolean system has 954 equations and variables. The converted system over the reals has $288 \cdot 5 - x$ equations derived from type I and $222 \cdot 4 \cdot 3$ equations derived from type II. We introduce $288 \cdot 4 - x$ and $222 \cdot 3 \cdot 3$ new variables. Hence, the system consists of $4104 - x$ equations in $4104 - x$ variables. The total degree of converted type I equations is two and converted type II equations have total degree of three. The monomial degree is at most 7.

Splitting Conversion

We first convert type I equations using the method from Section 3.8.2. Therefore, we determine $\tilde{M}$ for Equation (5.10) which is $\tilde{M} = \{66, 93, 162, 177, 243, 288\}$. Our goal is to achieve a total degree of two for each converted equation. Hence, we partition $\tilde{M}$ in $\tilde{M}_1 = \{66, 93\}$, $\tilde{M}_2 = \{162, 177\}$ and $\tilde{M}_3 = \{243, 288\}$. Next we split Equation (5.10) accordingly:

$$\begin{aligned} s_{66} \oplus s_{93} &= u_1 \\ s_{162} \oplus s_{177} &= u_2 \\ s_{243} \oplus s_{288} &= u_3 \\ u_1 \oplus u_2 &= u_3 \oplus z_i \end{aligned} \tag{5.14}$$

for $u_1, u_2, u_3 \in B$. The right-hand side and left-handside of each equation in (5.14) is now separately converted using standard or Fourier conversion. Note both conversion methods result in equations with total degree of two. However, the structure is less complex using Fourier conversion. Hence, Equation (5.10) is converted to

$$\begin{aligned} r_{66}r_{93} - v_1 &= 0 \\ r_{162}r_{177} - v_2 &= 0 \\ r_{243}r_{288} - v_3 &= 0 \\ v_1v_2 - v_3t(z_i) &= 0 \end{aligned} \tag{5.15}$$

where $v_1, v_2, v_3 \in \mathbb{R}$ correspond to u_1, u_2, u_3. Note that $t(z_i)$ is a constant and does not contribute to the total degree.

The conversion of Equation (5.11) is done in the same way. We determine $\tilde{M} = \{66, 93, 91, 92, 171, 289\}$ and partition it to $\tilde{M}_1 = \{66, 93\}$, $\tilde{M}_2 = \{91, 92\}$

and $\tilde{M}_3 = \{171, 289\}$. Hence, we split Equation (5.11) to

$$\begin{aligned} s_{66} \oplus s_{93} &= u_1 \\ s_{91} \wedge s_{92} &= u_3 \\ s_{171} \oplus s_{289} &= u_1 \oplus u_3. \end{aligned} \tag{5.16}$$

The conversion result of Equation (5.11) is

$$\begin{aligned} r_{66}r_{93} - v_1 &= 0 \\ \tfrac{1}{2}(1 + r_{91} + r_{92} - r_{91}r_{92}) - v_3 &= 0 \\ r_{171}r_{289} - v_1v_3 &= 0. \end{aligned} \tag{5.17}$$

As we see the monomial and total degree are reduced significantly compared to the previous conversions. The total degree is at most 2. The monomial degree for most equations 2 expect for the second equation in (5.16) where it is 5. However, we again increase the size of the system over the reals in terms of number of equations and variables. The resulting system has 2484 equations and 2484 variables.

5.4.2 Conversion of Bivium B Equations

The equations representing Bivium B are converted in the same way as done for Trivium, but the resulting system over the reals has less equations and variables. We again differ between two types of equations. The equations for the keystream-bit computation (5.6) define Type I equations and are of the form:

$$s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} = z_i \tag{5.18}$$

The remaining equations define type II equations and are of the form:

$$s_{66} \oplus s_{93} \oplus s_{91} \wedge s_{92} \oplus s_{171} \oplus s_{178} = \texttt{false}. \tag{5.19}$$

Without loss of generality we convert these representative equations using the four conversion methods.

Standard Conversion

Applying the conversion method described in Section 3.2 on Equation (5.18) results in the following equation over the reals:

$$\begin{aligned} &r_{162} + r_{177} - 2r_{162}r_{177} + r_{66} - 2r_{162}r_{66} - 2r_{177}r_{66} + 4r_{162}r_{177}r_{66} \\ &+ r_{93} - 2r_{162}r_{93} - 2r_{177}r_{93} + 4r_{162}r_{177}r_{93} - 2r_{66}r_{93} \\ &+ 4r_{162}r_{66}r_{93} + 4r_{177}r_{66}r_{93} - 8r_{162}r_{177}r_{66}r_{93} = t(z_i) \end{aligned} \tag{5.20}$$

Applying the conversion method described in Section 3.2 on Equation (5.19) results in the following equation over the reals:

$$\begin{aligned}
&r_{171} + r_{178} - 2r_{171}r_{178} + r_{66} - 2r_{171}r_{66} - 2r_{178}r_{66} + 4r_{171}r_{178}r_{66} \\
&+ r_{91}r_{92} - 2r_{171}r_{91}r_{92} - 2r_{178}r_{91}r_{92} + 4r_{171}r_{178}r_{91}r_{92} \\
&- 2r_{66}r_{91}r_{92} + 4r_{171}r_{66}r_{91}r_{92} + 4r_{178}r_{66}r_{91}r_{92} \\
&- 8r_{171}r_{178}r_{66}r_{91}r_{92} + r_{93} - 2r_{171}r_{93} - 2r_{178}r_{93} \\
&+ 4r_{171}r_{178}r_{93} - 2r_{66}r_{93} + 4r_{171}r_{66}r_{93} \\
&+ 4r_{178}r_{66}r_{93} - 8r_{171}r_{178}r_{66}r_{93} - 2r_{91}r_{92}r_{93} \\
&+ 4r_{171}r_{91}r_{92}r_{93} + 4r_{178}r_{91}r_{92}r_{93} - 8r_{171}r_{178}r_{91}r_{92}r_{93} \\
&+ 4r_{66}r_{91}r_{92}r_{93} - 8r_{171}r_{66}r_{91}r_{92}r_{93} \\
&- 8r_{178}r_{66}r_{91}r_{92}r_{93} + 16r_{171}r_{178}r_{66}r_{91}r_{92}r_{93} = 0.
\end{aligned} \tag{5.21}$$

The number of equations and variables does not change. The monomial degree is 15 for converted type I equations and 31 for converted type II equations. Due to the simplified keystream generation the total degree is decreased to 4 for converted type I equations and is still 6 for type II.

Fourier Conversion

Applying the conversion method described in Section 3.5 on Equation (5.18) results in

$$r_{162}r_{177}r_{66}r_{93} = t(z_i), \tag{5.22}$$

and applying the same method on quation (5.19), results in the following equation over the reals:

$$\begin{aligned}
&\tfrac{1}{2}r_{171}r_{178}r_{66}r_{93} + \tfrac{1}{2}r_{171}r_{178}r_{66}r_{91}r_{93} \\
&+ \tfrac{1}{2}r_{171}r_{178}r_{66}r_{92}r_{93} - \tfrac{1}{2}r_{171}r_{178}r_{66}r_{91}r_{92}r_{93} = 1.
\end{aligned} \tag{5.23}$$

Due to the amount of XOR operations, the results are more promising. The monomial degrees are 1 and 4, respectively. The total degrees are not changed.

Adapted Standard Conversion

In order to use the method described in Section 3.8.1 for type I equations, we need to determine the truth table of the Boolean function

$$f(\vec{s}) = s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177}$$

and compute the solution set $S := \{\vec{s} : f(\vec{s}) = z_i\}$. Next we evaluate the real function

$$f_r(\vec{r}) = r_{66} + r_{93} + r_{162} + r_{177}$$

for $\vec{r} \in S_r := \{f(\vec{r}) : \vec{r} = t(\vec{s}) \text{ for } \vec{s} \in S\}$. Let $z_i = \texttt{false}$ and therefore $t(z_i) = 0$. Due to the XOR operation, S_r consists of all possible even numbers of the sum of

four variables and therefore $S_r = \{0, 2, 4\}$. Hence, Equation (5.18) is converted to

$$\begin{aligned} d_1(r_{66} + r_{93} + r_{162} + r_{177}) &= 0 \\ d_2(r_{66} + r_{93} + r_{162} + r_{177} - 2) &= 0 \\ d_3(r_{66} + r_{93} + r_{162} + r_{177} - 4) &= 0 \\ d_1 + d_2 + d_3 &= 1 \end{aligned} \tag{5.24}$$

where $d_i \in \mathbb{R}$ for $i = 1, \ldots 3$. The case $z_i = \texttt{true}$ works analogous except that $S_r = \{1, 3\}$ and therefore one equation and one variable less is generated. Let x denote the number of $z_i = \texttt{true}$.

The conversion of type II equations are done in the same way. We determine the truth table of the Boolean function

$$f(\vec{s}) = s_{66} \oplus s_{93} \oplus s_{91} \wedge s_{92} \oplus s_{171} \oplus s_{178}$$

and compute the solution set $S := \{\vec{s} : f(\vec{s}) = \texttt{false}\}$. Then we evaluate the real function

$$f_r(\vec{r}) = r_{66} + r_{93} + r_{91} \cdot r_{92} + r_{171} + r_{178}.$$

In that case $S_r = \{0, 2, 4\}$ and the conversion of Equation (5.19) results in

$$\begin{aligned} d_4(r_{66} + r_{93} + r_{91} \cdot r_{92} + r_{171} + r_{178}) &= 0 \\ d_5(r_{66} + r_{93} + r_{91} \cdot r_{92} + r_{171} + r_{178} - 2) &= 0 \\ d_6(r_{66} + r_{93} + r_{91} \cdot r_{92} + r_{171} + r_{178} - 4) &= 0 \\ d_4 + d_5 + d_6 &= 1 \end{aligned} \tag{5.25}$$

where $d_i \in \mathbb{R}$ for $i = 4, \ldots 6$. The converted system over the reals has $177 \cdot 4 - x$ equations derived from type I and $111 \cdot 4 \cdot 2$ equations derived from type II. We introduce $177 \cdot 3 - x$ and $111 \cdot 3 \cdot 2$ new variables. Hence, the system consists of $1596 - x$ equations in $1596 - x$ variables. The total degree of converted type I equations is 2 and converted type II equations have total degree of 3. The monomial degree is at most 6.

Splitting Conversion

We first determine $\tilde{M}$ for Equation (5.18) which is $\tilde{M} = \{66, 93, 162, 177, \}$. We aim for a total degree of two for each converted equation. Hence, we partition $\tilde{M}$ in $\tilde{M}_1 = \{66, 93\}$ and $\tilde{M}_2 = \{162, 177\}$. Since we have only two partitions we do not need new variables. Instead we split Equation (5.18) in the following obvious way:

$$s_{66} \oplus s_{93} = s_{162} \oplus s_{177} \oplus z_i. \tag{5.26}$$

Note that if we split in the same way as done for Trivium, the resulting system would be larger. The right-hand side and left-hand side of Equation (5.26) are now separately converted using any basic conversion method. For the same

reasons as for Trivium we choose Fourier conversion. Hence, Equation (5.18) is converted to

$$r_{66}r_{93} - r_{162}r_{177}t(z_i) = 0 \tag{5.27}$$

Note that $t(z_i)$ is a constant and does not contribute to the total degree.

The conversion of Equation (5.19) is done in the same way and the result is not different from Trivium. We determine $\tilde{M} = \{66, 93, 91, 92, 171, 178\}$ and partition it to $\tilde{M}_1 = \{66, 93\}$, $\tilde{M}_2 = \{91, 92\}$ and $\tilde{M}_3 = \{171, 178\}$. Hence, we split Equation (5.19) to

$$\begin{aligned} s_{66} \oplus s_{93} &= u_1 \\ s_{91} \wedge s_{92} &= u_2 \\ s_{171} \oplus s_{178} &= u_1 \oplus u_2. \end{aligned} \tag{5.28}$$

The conversion result of Equation (5.19) is

$$\begin{aligned} r_{66}r_{93} - v_1 &= 0 \\ \tfrac{1}{2}(1 + r_{91} + r_{92} - r_{91}r_{92}) - v_2 &= 0 \\ r_{171}r_{178} - v_1v_2 &= 0, \end{aligned} \tag{5.29}$$

for $v_1, v_2 \in \mathbb{R}$. The total degree of the resulting equations is 2. The monomial degree is except for one equation 2. The resulting system has 843 equations in 843 variables. Compared to Trivium, we deal with a much smaller system of equations.

5.4.3 Conversion of Bivium A Equations

We again differ between two types of equations. The equations for the keystream-bit computation (5.8) define Type I and are of the form:

$$s_{162} \oplus s_{177} = z_i. \tag{5.30}$$

Type II equations are the same as for Bivium B:

$$s_{66} \oplus s_{93} \oplus s_{91} \wedge s_{92} \oplus s_{171} \oplus s_{178} = \texttt{false}. \tag{5.31}$$

Due to the simplified keystream-bit computation we get a third type of equations:

$$s_{175} \wedge s_{176} \oplus s_{69} \oplus s_{179} = z_i \tag{5.32}$$

Without loss of generality we convert these representative equations using the four conversion methods.

Standard Conversion

Applying the conversion method described in Section 3.2 on Equation (5.30) results in the following equation over the reals:

$$r_{162} + r_{177} - 2r_{162}r_{177} = 0. \tag{5.33}$$

Applying the conversion method described in Section 3.2 on Equation (5.31) results in the following equation over the reals:

$$\begin{aligned}
&r_{171} + r_{178} - 2r_{171}r_{178} + r_{66} - 2r_{171}r_{66} - 2r_{178}r_{66} + 4r_{171}r_{178}r_{66} \\
&+ r_{91}r_{92} - 2r_{171}r_{91}r_{92} - 2r_{178}r_{91}r_{92} + 4r_{171}r_{178}r_{91}r_{92} \\
&- 2r_{66}r_{91}r_{92} + 4r_{171}r_{66}r_{91}r_{92} + 4r_{178}r_{66}r_{91}r_{92} \\
&- 8r_{171}r_{178}r_{66}r_{91}r_{92} + r_{93} - 2r_{171}r_{93} - 2r_{178}r_{93} \\
&+ 4r_{171}r_{178}r_{93} - 2r_{66}r_{93} + 4r_{171}r_{66}r_{93} \\
&+ 4r_{178}r_{66}r_{93} - 8r_{171}r_{178}r_{66}r_{93} - 2r_{91}r_{92}r_{93} \\
&+ 4r_{171}r_{91}r_{92}r_{93} + 4r_{178}r_{91}r_{92}r_{93} - 8r_{171}r_{178}r_{91}r_{92}r_{93} \\
&+ 4r_{66}r_{91}r_{92}r_{93} - 8r_{171}r_{66}r_{91}r_{92}r_{93} \\
&- 8r_{178}r_{66}r_{91}r_{92}r_{93} + 16r_{171}r_{178}r_{66}r_{91}r_{92}r_{93} = 0.
\end{aligned} \tag{5.34}$$

The conversion of type III equations results in

$$\begin{aligned}
&r_{175}r_{176} + r_{179} - 2r_{175}r_{176}r_{179} + r_{69} \\
&- 2r_{175}r_{176}r_{69} - 2r_{179}r_{69} + 4r_{175}r_{176}r_{179}r_{69} = t(z_i).
\end{aligned} \tag{5.35}$$

The monomial degree of Equation (5.33) is only 3 and the total degree 2. For Equation (5.34) we obtain the same results as for Bivium B. For Equation (5.35) the monomial degree is 7 and the total degree 4.

Fourier Conversion

Applying the conversion method described in Section 3.5 on Equation (5.30) results in

$$r_{162}r_{177} = t(z_i), \tag{5.36}$$

and applying the same method on Equation (5.31), results in the following equation over the reals:

$$\begin{aligned}
&\tfrac{1}{2}r_{171}r_{178}r_{66}r_{93} + \tfrac{1}{2}r_{171}r_{178}r_{66}r_{91}r_{93} \\
&+ \tfrac{1}{2}r_{171}r_{178}r_{66}r_{92}r_{93} - \tfrac{1}{2}r_{171}r_{178}r_{66}r_{91}r_{92}r_{93} = 1.
\end{aligned} \tag{5.37}$$

Fourier conversion of Equation (5.32) results in the following equation over the reals:

$$\tfrac{1}{2}r_{179}r_{69} + \tfrac{1}{2}r_{175}r_{179}r_{69} + \tfrac{1}{2}r_{176}r_{179}r_{69} - \tfrac{1}{2}r_{175}r_{176}r_{179}r_{69} = 1. \tag{5.38}$$

The result for type II equation does not change compared to Bivium B. However, converted type I equations consists only of one monomial with a degree of two and the converted type III equations have a monomial degree and total degree of 4.

Adapted Standard Conversion

Contrary to Trivium and Bivium B we can preserve the linearity of type I equations. Hence, Equation (5.30) is converted to

$$s_{162} \oplus s_{177} = z_i \rightarrow \begin{cases} r_{162} - r_{177} = 0 & \text{if } z_i = \texttt{false} \\ r_{162} + r_{177} - 1 = 0 & \text{if } z_i = \texttt{true}. \end{cases}$$

We refer to Section 5.4.2 for the conversion of type II equations. The conversion of type III equation is done in a similar way as for type II equations. We determine the truth table of the Boolean function

$$f(\vec{s}) = s_{175} \wedge s_{176} \oplus s_{69} \oplus s_{179}$$

and compute the solution set $S := \{\vec{s} : f(\vec{s}) = z_i\}$. Then we evaluate the real function

$$f_r(\vec{r}) = r_{175} \cdot r_{176} + r_{69} + r_{179}.$$

For $z_i = \texttt{false}$, $S_r = \{0, 2\}$ and the conversion of Equation (5.32) results in

$$\begin{aligned} d_1(r_{175} \cdot r_{176} + r_{69} + r_{179}) &= 0 \\ d_2(r_{175} \cdot r_{176} + r_{69} + r_{179} - 2) &= 0 \\ d_1 + d_2 &= 1 \end{aligned} \tag{5.39}$$

where $d_1, d_2 \in \mathbb{R}$. The case $z_i = \texttt{false}$ is done analogous.

Finally, the converted system over the reals has 177 equations derived from type I, 432 equations derived from type II and 126 equations derived from type III. We introduce 408 new variables. Hence, the system consists of 735 equations in 735 variables. The total degree of converted type I equations is one, converted type II and III equations have total degree of three. The monomial degree is at most 6.

Splitting Conversion

Due to the simplicity of type I equations for Bivium A, we keep the linearity after the conversion. We choose again the Fourier representation and want to preserve the linearity. Hence, we convert type I equations in the following way

$$s_{162} \oplus s_{177} = z_i \rightarrow \begin{cases} r_{162} - r_{177} = 0 & \text{if } z_i = \texttt{false} \\ r_{162} + r_{177} = 0 & \text{if } z_i = \texttt{true}. \end{cases}$$

The conversion of type II equations and the result are exactly the same as for Bivium B (see Section 5.4.2). The conversion of type III equations is slightly less complex. We split Equation (5.32) in the following way:

$$s_{175} \wedge s_{176} = s_{69} \oplus s_{179} \oplus z_i. \tag{5.40}$$

The right-hand side and left-hand side of Equation (5.40) are separately converted. Hence, Equation (5.32) is converted to

$$\tfrac{1}{2}(1 + r_{175} + r_{176} - r_{175}r_{176}) - r_{69}r_{179}t(z_i) = 0. \tag{5.41}$$

The total degree of all equations is at most 2. Converted type I equations have a monomial degree of 2. Converted type II equations have a monomial degree of 2 and 5, respectively. The result for type III equations is 5. The resulting system has 543 equations in 543 variables. Hence, this conversion leads to a far less complex system over the reals where the equations have at most degree 2 and 32% of the equations are even linear.

5.4.4 Comparison of the Results

In Table 5.1 we summarize the conversion results. Since the monomial degree and total degree is different for each type of equations, we present the maximum values in the table. We constructed 12 different systems of equations which differ in size and complexity. The systems for Trivium are the largest ones. Bivium A and Bivium B differ in the keystream-bit computation, which is slightly simplified for Bivium A. Therefore, the number of equations and variables using adapted standard conversion and splitting conversion is significantly reduced. The system for Bivium A using splitting conversion seems to be the easiest one in terms of size, monomial degree and total degree. Due to the simplified keystream-bit computation the trade-off between number of equations/variables and total degree, made in the splitting conversion, turns out to be better for Bivium A. Although, the systems resulting from standard and Fourier conversion are smaller in size, the total degree is considerable high and present therefore a more difficult problem for numerical methods as shown in Section 5.5.

Table 5.1: Comparison of conversion results. *mdeg* and *tdeg* are the maximum values of the monomial and total degree for the converted equations. # stands for the number of variables and equations.

	Trivium			**Bivium B**			**Bivium A**		
	mdeg	*tdeg*	#	*mdeg*	*tdeg*	#	*mdeg*	*tdeg*	#
Standard	63	6	954	31	6	399	31	6	327
Fourier	4	6	954	4	6	399	4	6	327
ASC	7	3	4104	6	3	1596	6	3	735
Splitting	5	2	2484	5	2	843	5	2	543

5.5 Numerical Results

In Chapter 4 we give an overview on numerical analysis and describe four different iterative methods in detail. In Section 5.4 we constructed four systems of equations for each variant of Trivium. Therefore, we define an experimental

setting where each of the iterative methods is applied on each system of equations using three different types of initial points. Furthermore, we precompute a solution using a random key, to evaluate the quality of the solutions over the reals. As described in Section 4.3 the numerical methods converge either to a local or global minimum or do not converge at all. Therefore, we evaluate the obtained results by computing the objective function value (see Section 4.3) at the found local or global minimum. If this value is equal to zero we have found a solution for the system of equations. Unfortunately, not all of the systems can be solved using the chosen methods, instead only local solutions are computed. For the solvable systems real-valued solutions are found which cannot be directly converted back to the Boolean domain. To determine the quality of the found local or global solutions we compute the Hamming distance between the precomputed solution and the found local or global solutions. In this case the Hamming distance is the amount of elements which are different between two vectors. For real-valued solutions we apply two different rounding strategies. Finally, we run each experiment 100 times and present the average values in this thesis.

5.5.1 Initial Points and Rounding Strategies

As mentioned in Chapter 4, the initial point for iterative methods has a high influence on their behaviour. Therefore, we used different strategies for the initial guess.

- Random in $\{0,1\}^n$, $\{-1,1\}^n$
- Random in $[0,1]^n$, $[-1,1]^n$
- Part of the precomputed solution used in the starting point

We have two types of randomness. First points in $\{0,1\}^n$ (or $\{-1,1\}^n$ for Fourier representation) are chosen. Secondly, we set the starting point to a random point in $[0,1]^n$ (or $[-1,1]^n$). Additionally, we use part of the precomputed solution in the initial guess, to see how "close" the guess has to be for convergence to the solution. If the number of values used from the precomputed solution is low, one can use a guess and determine attack to determine the correct values without knowing the solution.

Guessing Bits

Guess and determine attacks are commonly used in cryptanalysis. By guessing a few bits, which for our experiments means that we replace some variables with their correct values, we simplify the system before the conversion. By this reduction of the dimension, we also simplify the problem in the real domain. To get a good reduction with as few as possible guessed values, we have to choose the bits wisely. To achieve this goal we compute for each variable its distribution in the system, i.e. in how many equations a specific variable occurs. We see for

example that variables from s_{94} up to s_{162} occur in 7 equations. If we guess such a variable we can simplify 7 equations in one step.

The second usage of guessing bits is to determine a good starting point as mentioned above. In that case the system is not simplified but better starting information for a numerical method is provided. Since during the system creation new variables are introduced, more variables can be guessed than the number of internal state bits.

Rounding Strategies

In order to determine how close a found minimum is to the wanted solution, we use the Hamming distance between the precomputed solution and the found local or global minimum. If the local or global minima contain real-valued values we round them using two different rounding strategies. The first strategy is commonly known as round to nearest integer. The second strategy is called round to nearest bound. Let $x \in \mathbb{R}$ and $l, u \in \mathbb{R}$ then round to nearest bound is defined as

$$round(x) = \begin{cases} l & \text{if } |x-l| < |x-u| \\ u & \text{otherwise.} \end{cases}$$

With the second strategy a found local or global minimum can directly be mapped back to the Boolean domain. The lower bound l and upper bound u are given by the type of representation.

5.5.2 Results for Trivium

The system of equations for Trivium represent the largest problems. This is especially noticed in the running time of the iterative methods where one iteration can take up to several minutes on a standard PC. In Table 5.2 we present the results for the various systems of Trivium. We do not obtain any results using the DIRECT method, since it is not feasible to apply it on such large systems. Moreover, as mentioned in Chapter 4 iterative methods can run into difficulties if the Jacobian matrix is singular or ill-conditioned. Levenberg-Marquardt and the interior reflective Newton method can handle such cases. However, the Gauss-Newton method does not converge in such a case. In our experiments we encounter ill-conditioned Jacobian matrices regularly using the Gauss-Newton method. Therefore, the Gauss-Newton method does not converge in many runs. If it converges then the reached objective function value is only slightly improved. This emphasises the necessity of more robust methods like the Levenberg-Marquardt and interior reflective Newton method. Both methods find in most cases only a local minimum. The system using the adapted standard conversion can be solved by the Levenberg-Marquardt method and the system derived from the splitting conversion can be solved by both methods. Unfortunately, all found solutions are real-valued and cannot be mapped back to the Boolean domain.

Table 5.2: Results for Trivium using Gauss-Newton (GN), Levenberg-Marquardt (LM) and interior reflective Newton method (IRN) for each conversion type. The DIRECT method cannot handle such a large system. The columns contain the average objective function value $f(x^*)$ at the converged point x^*.

Conversion	GN	LM	IRN	DIRECT
Fourier	893.05	164.74	242.15	–
Standard	222.72	44.28	65.71	–
ASC	519.25	0	8.36	–
Splitting	1131.34	0	0	–

In Table 5.3 we show the Hamming distance between the local/global minima and the precomputed solution using the first rounding strategy. Using the Fourier conversion and Gauss-Newton method results in local minima where many variables are over the bounds which results in a high Hamming distance. Similar are the results for the Levenberg-Marquardt method where in average one third of the variables is in $(-0.5, 0.5)$. Better results are obtained by the interior reflective Newton method. In this case the Hamming distance is approximately one half of the total amount of variables in the system. However, this is also the expected Hamming distance of a random point in the Boolean domain converted to the reals. Hence, we cannot extract any useful information of these local minima. Similar observations are made for the standard conversion.

Different results are obtained for the system based on the adapted standard conversion. There the Hamming distance for all numerical methods is less than one half of the total amount of variables. In the case of interior reflective Newton method it is even less than one third. It is interesting to note that the results for all decision variables (see Section 3.8.1) are integer-valued (0 or 1) (except for Gauss-Newton method). In the precomputed solution the ratio between ones and zeros for the decision variables is 1:3 which is almost matched by the found minima, explaining the lower Hamming distance. However, the other variables are not between the bounds defined by the type of representation, with the obvious exception of the interior reflective Newton method. The results of this method are in between the bound but all variables are real-valued.

The high Hamming distance for the splitting conversion is explained by the fact that many variables of the found solutions are close to 0 using interior reflective Newton method or not in between bounds in the case of Levenberg-Marquardt method. In that case the first rounding strategy may be not the best choice.

In Table 5.4 we present the Hamming distance using the second rounding strategy. This rounding strategy basically moves the found minimum to the nearest corner of the hyper-cube defined by the type of representation. We see that almost all distances are close to one half of the total amount of variables except for the adapted standard conversion due to the decision variables.

Table 5.3: Average Hamming distance between the precomputed solution and the found minima using the first rounding strategy.

Conversion	GN	LM	IRN
Fourier	715	848	487
Standard	483	636	469
ASC	1634	1627	1207
Splitting	1862	2113	2342

Table 5.4: Average Hamming distance between the precomputed solution and the found minima using the second rounding strategy.

Conversion	GN	LM	IRN
Fourier	487	482	470
Standard	488	461	453
ASC	1347	1355	1840
Splitting	1222	1073	1034

In the final experiment we determine how close the initial point has to be such that the numerical method converges to the wanted solution. Therefore, we set $x\%$ of the variables to the precomputed solution. In Table 5.5 the results are presented. Due to the structural problems in the system of equations, leading to ill-conditioned Jacobian matrices, the results of the Gauss-Newton method are not useful. For the simplified systems, where we replaced the guessed variables with their correct values before the conversion, we get similar results.

Table 5.5: Amount of correct variables in the initial point such that the numerical method converges to the wanted solution.

Conversion	GN	LM	IRN
Fourier	–	80%	55%
Standard	–	80%	55%
ASC	–	45%	70%
Splitting	–	90%	85%

Overall, a solution for the Boolean system could not be found. We see that we get closer to 0 for the systems with low degrees, even if they have more equations and variables. It seems that using the advanced conversion techniques results in systems which have more solutions beside the wanted one and these solutions are real-valued. So far we could not find any relations between a real-valued solution and the Boolean solution. Using one of the rounding strategies reveals that the found local and global minima are not better than a random

point in the Boolean domain. Finally, both random initial guess strategies lead to the same results.

5.5.3 Results for Bivium B

In Table 5.6 we present the results for the various systems of Bivium B. Due to the reduced size of the systems the reached minimum function values are lower compared to Trivium. The found local minima of the systems derived by Fourier and standard conversion are real-valued and not in between the given bounds, except for the minima computed by the interior reflective Newton method. Its local minima are also mostly integer-valued which shows that this method tends towards a corner of the hyper-cube.

Again two systems can be solved but only the solutions computed by the interior reflective Newton method are in between the given bounds. The solutions for the system derived by the adapted standard conversion are real-valued, except the decision variables which are all integer-valued. It seems that the additional equations and variables in this conversion method can be handled well. The solutions for the system using splitting conversion are real-valued. The solutions of the Levenberg-Marquardt method contain a lot of zeros resulting in a high Hamming distance using the first rounding strategy (see Table 5.7). Contrary, the interior reflective Newton method computes solutions containing a lot of values equal to 0 and 0.5.

Regarding the DIRECT method and Gauss-Newton method we encounter the same issues as for Trivium.

Table 5.6: Results for Bivium B using Gauss-Newton (GN), Levenberg-Marquardt (LM) and interior reflective Newton method (IRN) for each conversion type. The DIRECT method cannot handle such a large system. The columns contain the average objective function value $f(x^*)$ at the converged point x^*.

Conversion	GN	LM	IRN	DIRECT
Fourier	393.42	64.71	103.94	–
Standard	97.78	16.24	24.90	–
ASC	125.90	0	6.03	–
Splitting	266.82	0	0	–

In Table 5.7 we see that the first rounding strategy results in a high Hamming distance, due to the mentioned values of the local and global minima.

The results for the second rounding strategy are shown in Table 5.8. They reveal that the local and global minima are as good as a random guess in the Boolean domain converted to the reals since the Hamming distance is approximately half the total amount of variables. The exception is again the system derived from the adapted standard conversion, due to the different ratio between the appearance of ones and zeros for the solution of the decision variables.

Table 5.7: Average Hamming distance between the precomputed solution and the found minima using the first rounding strategy.

Conversion	GN	LM	IRN
Fourier	308	358	209
Standard	202	275	197
ASC	766	703	1580
Splitting	673	737	733

Table 5.8: Average Hamming distance between the precomputed solution and the found minima using the second rounding strategy.

Conversion	GN	LM	IRN
Fourier	201	204	195
Standard	2022	201	200
ASC	557	525	500
Splitting	404	373	371

Table 5.9: Amount of correct variables in the initial point such that the numerical method converges to the wanted solution.

Conversion	GN	LM	IRN
Fourier	–	75%	60%
Standard	–	70%	60%
ASC	–	40%	55%
Splitting	–	90%	90%

In Table 5.9 we show how much of the precomputed solution is needed such that the wanted solution is found by the numerical methods. All results are considerably high.

In the end the Boolean system could not be solved by any of the numerical methods. The obtained results are very similar to Trivium. Furthermore, the real-valued solutions for the solved systems cannot be mapped to the Boolean domain nor any significant relation between these solutions and the Boolean solution can be found. Once again both random initial guess strategies lead to the same result.

5.5.4 Results for Bivium A

Bivium A represents the easiest of all three problems and is already broken [Rad07] with practical complexity. Hence, its results are of special interest. In Table 5.10 we present the results for the various systems of Bivium A. The

systems for Bivium A have the same structural problem which cannot be handled well by the Gauss-Newton method. We again analyse the results using Fourier and standard conversion first. As for Bivium B the interior reflective Newton method converges almost to a corner of the hyper-cube. Only few variables are left real-valued. In contrast the local minima computed by the Levenberg-Marquardt method are significantly far away from the bounds.

Looking at the results for the adapted standard converted system we see again that the decision variables are integer-valued, for both numerical methods. For the variables which are also existing in the Boolean system we obtain different results. The solutions computed by Levenberg-Marquardt method are again not in between the bounds, but using the interior reflective Newton method they are. Furthermore, some of the variables are even integer-valued.

Finally, the results for the system derived by the splitting conversion are similar to Trivium and Bivium B. Levenberg-Marquardt computes real-valued solutions which are not in between the bounds. The interior reflective Newton method seems to converge to an area around the centre of the hyper cube. Hence, the Hamming distance presented in Table 5.11 using the first rounding strategy is very high. The second rounding strategy reveals more information as shown in Table 5.12. If we move the local and global minima to the nearest corner, the result is again as good as a random guess in the Boolean domain. However, it is worth to notice that the rounded points are different from the initial guess.

Table 5.10: Results for Bivium A using Gauss-Newton (GN), Levenberg-Marquardt (LM) and interior reflective Newton method (IRN) for each conversion type. The DIRECT method cannot handle such a large system. The columns contain the average objective function value $f(x^*)$ at the converged point x^*.

Conversion	GN	LM	IRN	DIRECT
Fourier	338.27	41.70	66.84	–
Standard	77.95	10.62	17.05	–
ASC	440.36	0	1.44	–
Splitting	63.72	0	0	–

Table 5.11: Average Hamming distance between the precomputed solution and the found minima using the first rounding strategy.

Conversion	GN	LM	IRN
Fourier	252	282	170
Standard	167	217	197
ASC	370	305	254
Splitting	465	476	465

Table 5.12: Average Hamming distance between the precomputed solution and the found minima using the second rounding strategy

Conversion	GN	LM	IRN
Fourier	167	160	160
Standard	161	164	163
ASC	370	275	252
Splitting	274	284	288

In Table 5.13 we see again that the part of the precomputed solution needs to be large such that the numerical methods converge to this solution. Even if we assign the correct values for all variables representing the internal state bits, the algorithms find only local minima. That means the objective function has a lot of stationary points beside the one we are looking for. For the simplified systems, where we replaced the guessed variables with their correct values before the conversion, we get similar results.

Table 5.13: Amount of correct variables in the initial point such that the numerical method converges to the wanted solution.

Conversion	GN	LM	IRN
Fourier	–	80%	55%
Standard	–	80%	55%
ASC	–	45%	70%
Splitting	–	85%	85%

The DIRECT algorithm does not need an initial guess since it is an deterministic algorithm. In our experiments the algorithm did not converge to any point, so we stopped the algorithm after three days. Other methods with global optimization techniques may be more successful.

For Bivium A we draw the same conclusions as for Bivium B. Although Bivium A is the easiest of the three problems and already broken by other techniques, the chosen numerical algorithms cannot find a solution which corresponds to the Boolean one. However, in this thesis we covered only a fraction of all numerical methods and we do not exclude the possibility that an other numerical algorithm maybe better suited for these problems, especially considering that new methods for different kind of problems are showing up constantly.

5.6 Summary

In this chapter we applied our approach on the stream cipher Trivium and on its reduced variants Bivium A and Bivium B. Trivium is recommended by the eStream project [Rob08]. We first constructed the Boolean system of equations

describing the internal state of the cipher for each of the variants. Afterwards, each of these Boolean systems are converted to the real domain using four conversion methods discussed in Chapter 3. These methods are standard conversion, Fourier conversion, adapted standard conversion and splitting conversion. Each of the resulting systems over the reals differ in size, monomial degree and total degree. Therefore, they represent different difficulties for the numerical methods. In the next step we used four different iterative numerical algorithms to search for a solution of each of the systems. The algorithms are discussed in Chapter 4 and have different properties. The first three algorithms provide global convergence. This is necessary since we are not able to provide an initial guess which is already close to the solution. In our experimental setting we defined three different strategies for the initial point. In the first two we have chosen either a random point in the hyper-cube or a random corner of the hyper-cube. The hyper-cube is defined by the chosen type of representation. The outcome for both strategies is the same. In order to find out how close the starting point has to be to the solution, we used part of the precomputed solution for the starting point and assigned the other variables to random values. We showed that the size of this part is considerable high for all systems and numerical algorithms.

The results of the numerical algorithms differ for each system. The Levenberg-Marquardt and interior reflective Newton method can handle the constructed systems best. The Gauss-Newton method is not robust enough since it cannot handle ill-conditioned Jacobian matrices well. Due to the structure of the systems the Jacobian matrix is often ill-conditioned and therefore the Gauss-Newton method does not converge most of the time. Contrary, the other two methods converge for every initial point. Unfortunately, for most of the systems they converge to a local minimum which is not a solution. Only the systems derived by our advanced conversion techniques can be solved (for any variant of Trivium). We analysed the found local and global minima. The solutions for the solved systems are real-valued. Since real-valued minima cannot be directly converted back to the Boolean domain we defined two rounding strategies. The first one rounds real values to the nearest integer which tells us how far a point from a corner of the hyper-cube is. The results of the Levenberg-Marquardt method are often not located in the hyper-cube, in contrast to the interior reflective Newton method where per design all points are within the hyper-cube. The second rounding strategy moves the point to the nearest corner of the hyper-cube. In that way we see that the global minima as well as the local minima are as good as a random guess in the Boolean domain. We were not able to find any other relations between these minima and the wanted Boolean solution. However, we cannot exclude the possibility that there is one and that another numerical algorithm may reveal some information about the Boolean solution.

The approach of using numerical methods in cryptanalysis including an application on Bivium A has been published in [LNR09b]. Note, that the results in [LNR09b] has been improved in this thesis.

6 Conclusions

Since the upcoming of linear and differential attacks, new design strategies have been proposed to resist these attacks. A disadvantage of such designs is, that a new type of attack could cause a complete breakdown of security. In the first part of this thesis we investigated a new approach in cryptanalysis. In this approach the cryptographic algorithm is represented as a system of equations. The system of equations is constructed such that there is a correspondence between its solutions and some secret information of the cryptographic primitive (for instance, the secret key of a block cipher). We apply methods and techniques from numerical analysis, which is a large and well-studied field of research, to solve this system. We use conversion methods to create an equivalent system over the real domain from the system of equations over the Boolean domain. At this point one can apply numerical solvers. The computed solution can be converted back to the Boolean domain (with restrictions) which results in a solution for the original system. Numerical solvers are methods to approximate solutions for equations and systems of equations. We are interested in the special case of solving non-linear polynomial systems of equations. For such system a variety of different techniques and methods exist. A key advantage is that except for building the Boolean system of equations every step of our approach works fully automated.

In order to be able to use numerical methods to solve such systems we have to convert the equations to equations over the real domain. By doing that we have to ensure that at least the set of Boolean solutions is represented in the reals, i.e. each solution of the Boolean system has at least one corresponding solution for the system over the reals. In Chapter 3 we presented four basic conversion methods which convert a Boolean equation to a polynomial over the reals depending on the representation of the values `false` and `true`. The main

problem using these techniques is the high degree of the resulting polynomial. A system consisting of polynomials with hight degrees is usually more difficult to solve than polynomials with low degrees. Therefore, we developed two advanced conversion techniques naming them adapted standard conversion and splitting conversion. Both methods were published in [LNR09b]. Furthermore, we gave a detailed analysis of each conversion method and define criteria to classify them. We focused on the structure of the resulting equations over the reals and defined the criteria monomial degree and total degree. We showed that one has a high influence on the structure of the conversion result, which is exceptional compared to usual use cases for numerical methods. Furthermore, we defined the property of variable sharing which has a high influence on the conversion result. We showed how this property affects the different types of representation. We also provided a rule of thumb helping to find the appropriate type of representation for specific equations. Overall, we showed that by choosing the right representation and conversion method one can change significantly the structure of the system of equations over the reals and therefore its difficulty regarding the solvability.

In Chapter 4 we gave an overview on numerical analysis. Since numerical analysis is a broad research field we focused on iterative methods for solving non-linear systems of equations. We described three global convergent methods in detail, namely Gauss-Newton, Levenberg-Marquardt and interior reflective Newton method. Global convergent methods are necessary as we cannot provide a starting point for the iterative methods which is already close to the solution. Modern iterative methods, as the interior reflective Newton method, are designed for minimization problems, since solving a system of equations is strongly related to it. In this context the notion of global and local minima are important. A solution for a system of equations is a global minimum. In general finding a global minimum for non-linear functions is a hard problem and a whole research field (global optimization) is dedicated to it. Nevertheless, we described and used one method for global minimization, called DIRECT. Furthermore, we worked out properties of the converted equations with respect to numerical analysis. No matter which conversion method we choose, the resulting polynomials over the reals have desirable properties considering numerical methods. Foremost, all polynomials are continuous differentiable. Second, if a solution for the Boolean system of equations exists (and in the case of cryptographic primitives it does), we can guarantee that the converted system has at least the same amount of solutions. However, it is possible that additional real-valued solutions exist. Hence, we cannot guarantee *uniqueness* but *existence* of solutions.

In Chapter 5 we applied our approach to the stream cipher Trivium and on its reduced variants Bivium A and Bivium B. We first constructed the Boolean system of equations describing the internal state of the cipher for each of the variants. Afterwards, each of these Boolean systems are converted to the real domain using four conversion methods from Chapter 3. With the help of these examples, we showed that the structure of the system over the reals can significantly change using different conversion methods. In the next step we set

up an experimental setting to apply all four numerical algorithms and evaluate their results. We showed that the results of the different numerical methods differ significantly from each other. Furthermore, we showed that the interior reflective Newton and Levenberg-Marquardt converge to local minima for most systems. However, the systems derived by our advanced conversion techniques were solved (for any variant of Trivium). Unfortunately, the solutions for these systems are real-valued which cannot be directly converted back to the Boolean domain. We defined two rounding strategies to extract more information from these solutions. We demonstrated that moving the solutions to the nearest corner of the given hyper-cube reveals that the global minima as well as the local minima are as good as a random guess in the Boolean domain. We were not able to find any other relations between these minima and the wanted Boolean solution. However, in this thesis we covered only a fraction of all numerical methods and we do not exclude the possibility that an other numerical algorithm maybe better suited for these problems, especially considering that new methods for different types of problems are showing up constantly. The approach of using numerical methods in cryptanalysis including an application on Bivium A has been published in [LNR09b].

Although, we were not able to break Trivium or any of its variants we showed how techniques from numerical analysis can be used in cryptanalysis. Furthermore, various future research directions are possible. One direction should investigate different ways of modelling the numerical problem. A possibility is to change the objective function to something different (e.g. Hamming weight of the state) and use the given equations as constraints. Furthermore, any obtained side-channel information can be added as equality or inequality constraints which can lead the numerical methods to the wanted solution. The meaning of local minima over the reals in the Boolean domain is another open problem as well as the meaning of real-valued solutions. Overall, the field of numerical analysis and optimization research offers great tools and possibilities which can be of use in the cryptanalysis of symmetric primitives.

Part II

Tools in Differential Cryptanalysis

7 Introduction

In the early 1990's Biham and Shamir [BS92], published a general technique for the cryptanalysis of symmetric primitives, called differential cryptanalysis. The basic idea is to study how differences in an input affect the differences at the output. Biham and Shamir applied this technique on the block cipher DES. Differential cryptanalysis is a statistical attack which studies the propagation of differences through all transformations of the cipher. Since then differential cryptanalysis turned out to be one of the most powerful techniques to analyse block ciphers, hash functions and stream ciphers. Attacks based on differential cryptanalysis are dedicated attacks, usually exploiting the internal structure of a design. The propagation of differences through the cipher transformations is usually predicted over multiple rounds. The differences in the input, intermediate values and output is called a differential characteristic. The probability of a characteristic is the fraction of conforming input pairs, which result in the differences of a characteristic. In a differential attack an adversary tries to find characteristics with high probability. For a high probability characteristic many conforming pairs exist which makes it easier to find such pairs.

In the last years the concept of differential cryptanalysis has been developed further. Foremost, different types of differences have been defined to cope with different operations in the targeted algorithms, e.g. XOR differences [BS92], modular differences [Dob98], signed-bit differences [WY05, WYY05b] or truncated differences [Knu94]. Furthermore, new type of attacks based on differential cryptanalysis have been proposed, like impossible differential attacks [BKR97, BBS99], linear-differential attacks [CJ98] or the boomerang attack [Wag99].

Differential cryptanalysis turned out to be of particular interest in the cryptanalysis of hash functions where it become one of the most important tech-

niques. Cryptographic hash functions are a security-critical building block for e-commerce and e-government systems. For example, when a document is signed by means of a digital signature (electronic signature), firstly hash functions are used to compress the document to a "fingerprint". For performance reasons, the 'raw' signature using asymmetric techniques like RSA, DSA or ECDSA, is made on the fingerprint of the document only. For security reasons, it is of utmost importance that no two documents can be created which result in the same fingerprint. When this happens, this is called a collision. While the existence of collisions cannot be avoided, due to the nature of the compression functions used, the design goal of a cryptographic hash function is to make it infeasible to construct such collisions. While hash functions did not get a lot of attention by the cryptographic community, this changed with the breakthrough results of Wang et al. in 2004. Since then many attacks based on differential cryptanalysis have been presented for several well-known algorithms such as SHA-0, SHA-1 or MD5. The transition from SHA-1 to the SHA-2 family was proposed by the *National Institute of Standards and Technology* (NIST) as a first solution [Nat08]. Since then more and more companies and organization are migrating to the SHA-2 family. As another consequence of these results NIST has initiated an open competition for a new hash function standard, called SHA-3 [Nat07]. In November 2008, round one has started and in total 51 out of 64 submissions have been accepted. In December 2009 the 14 round 2 candidates and in December 2010 the final five were announced. NIST will select a winner in 2012.

Since the upcoming of differential cryptanalysis and other cryptanalytic methods, new design strategies have been proposed to resist this kind of analysis. Furthermore, in many designs the complexity has increased compared to previous ones. Larger states, more non-linear operations, more rounds and more complicated state updates are the consequence. This can be especially observed for hash functions, like the transition from SHA-1 to SHA-2. Due to this increased complexity, the analysis of hash functions has become more difficult. Therefore, finding differential characteristics and conforming input pairs has become a more challenging task and the development of new tools has become necessary.

In this thesis we investigate how recent attacks on SHA-1 can be applied on its successor SHA-256 and other similar hash functions. Therefore, we improve and extend existing techniques leading to a new generation of tools which are used in attacks on several hash functions including SHA-2, HAS-160 or SIMD.

7.1 Cryptanalysis of SHA-1

The work on hash functions by Dobbertin [Dob98, Dob97], Chabaud and Joux [CJ98] and Biham et al. [BCJ+05] and the further advances in the security analysis of the MD4 [WY05] hash function were a wake-up call. The breakthrough results of Wang et al. [WY05, WYY05b] on MD5 and SHA-1 have shocked the cryptographic community and were the starting point for an unprecedented activity in the research on the security of hash functions in both academia and industry. Since then the results by Wang et al. were improved by

others. The cryptanalysis of SHA-1 includes different techniques from coding theory or techniques used in the cryptanalysis of block ciphers. New techniques were developed like message modification or automated search for differentials. Even if SHA-1 is considered as broken, it is still used in a variety of applications. Therefore, researchers are still interested in SHA-1 and try to provide practical collisions for more steps. The currently best results on SHA-1 are by De Cannière et al. [DMR07] who presented a collision for 70 steps and Grechnikov and Adinetz [GA11], who used the same techniques with minor improvements and more computational power to present collision for 75 steps out of 80 steps of SHA-1.

Overall two distinct methods where most successful. Using techniques from coding theory to find differential characteristics in a linearised model of the hash function such that they hold with high probability for the original hash function is the first one. The second technique automatically searches for complex characteristics based on the concept of generalized conditions.

7.2 Outline

In Chapter 8 we give a definition of cryptographic hash functions and their security requirements. Next we introduce the notation and definitions for differential cryptanalysis. In Chapter 9 we explain in detail the two most successful attacks on SHA-1 and describe the concept of the two distinct techniques. The first technique linearises a hash function and applies algorithms from coding theory to find differential characteristics which hold with high probability for the original hash function. The second technique is based on the concept of generalized conditions. Around this concept an automatic search algorithm is constructed which searches for complex differential characteristics. We extend and develop tools for both techniques and use them to construct attacks on different hash functions in the subsequent chapters.

In Chapter 10 we present two attacks on SIMD using the coding theory approach. SIMD is one of the round 2 candidates of the SHA-3 competition. Due to our first attack, the designers modified the specification of SIMD. However, we present another attack on the modified version, using the same basic technique under a different attack setting.

In Chapter 11 we attack the Korean hash function standard HAS-160 combining both tools. We first construct two differential characteristics using the first approach and connect them through a characteristics using the automatic search algorithm. We present a conforming message pair resulting in the currently best attack in terms number of steps with practical complexity.

In Chapter 12 we describe the extensions and modification needed such that we can successfully apply the search algorithm on the hash function standard SHA-256 leading to the best attacks on SHA-256 with practical complexity.

In Chapter 13 we present a summary and conclude the second part of the thesis by discussing open problems and further research directions.

8 Notation and Definitions

In this chapter, we introduce cryptographic hash functions and their basic security requirements. We describe the Merkle-Damgård design principle and describe different types of collisions for hash functions and their building blocks. Furthermore, we give the notation and definitions necessary for differential cryptanalysis and introduce the concept of generalized conditions used in the most recent attacks on SHA-1.

8.1 Cryptographic Hash Functions

Cryptographic hash functions are an important symmetric primitive in cryptography. They are used in numerous applications, like digital signatures, password protection, random number generation, key derivation, integrity protection, malicious code detection, message authentication, and many more. A cryptographic hash function H is an algorithm that maps a message string m of arbitrary length to a fixed-length hash value $h = H(m)$ of n bits. The hash value h can be seen as a digital fingerprint of the message m. In general it should be difficult to find two distinct messages resulting in the same hash value. A cryptographic hash function should be efficiently computable and each hash value should be a unique representation of the message. Due to the fact that the input message can be of arbitrary length, the existence of two different messages resulting in the same hash value cannot be prevented. Hence, the purpose of a hash function is not to prevent the existence of such colliding messages, but to ensure that it is computationally infeasible to find them.

8.1.1 Security Requirements

Since cryptographic hash functions are used in many applications with different requirements, many different properties are needed. The three basic security requirements are the following:

- *Collision resistance*: it should be computationally infeasible to find two messages m and m^* with $m \neq m^*$, such that they have the same hash value $H(m) = H(m^*)$.
- *Second preimage resistance*: for a given message m, it should be computationally infeasible to find a second message m^* with $m \neq m^*$, such that they have the same hash value $H(m) = H(m^*)$.
- *Preimage resistance*: for a given hash value h, it should be computationally infeasible to find any message m, such that it has the given hash value $H(m) = h$.

A detailed treatment of these requirements can be found in [RS04]. The basic three requirements are usually set in relation to the bit length n of the hash value. For any hash function, we can always find preimages or second preimages by testing approximately 2^n random input messages. Due to the birthday paradox finding collisions requires only $2^{n/2}$ calls to the hash function. Therefore, the hash size n is usually chosen large enough to make such generic attacks computationally infeasible. Since these generic attacks work for any hash function, a cryptographic hash function is said to be ideal if the generic bounds hold. There are other important properties. In the recent years non-random properties of hash functions were targeted. In such attacks an adversary utilizes specific properties of a hash function to define a distinguishing property such that one can distinguish the output of a hash function from a random function. To formalize this and other properties the random oracle models has been introduced [BR93]. A random oracle is a function which outputs a random hash value for any given input message. If the same message is used again, it outputs the previously used corresponding hash value. Therefore, we introduce another security requirement for hash functions:

- *randomness*: it should be computationally infeasible to create statistical irregularities for the distribution of the output.

Due to the limited internal state of a practical hash function it can never be a random oracle. However, it should be infeasible to distinguish a hash function from such a random oracle up to the generic bound for any attack.

8.1.2 The Merkle-Damgård Design Principle

The hash functions analysed in this thesis are iterated hash functions following the Merkle-Damgård design principle [Dam89, Mer89]. In order to compute the hash value h the message m is first split into t message blocks of b bits

each. To ensure that the message length is a multiple of b bits, an unambiguous padding method is applied. Then each message block is processed by iterating the compression function f t times resulting in the final hash value h. To be more precise, let $h : (0,1)^* \to (0,1)^n$ be an iterated hash function based on a compression function $f : (0,1)^n \times (0,1)^b \to (0,1)^n$ and $m = M_1 \| M_2 \| \cdots \| M_t$ be a t-block message (after padding). Then the hash value h is computed as follows (see Figure 8.1):

$$\begin{aligned} H_0 &= IV, \\ H_j &= f(H_{j-1}, M_j) \quad \text{for } 0 < j \leq t, \\ H_{t+1} &= g(H_t). \end{aligned} \tag{8.1}$$

The n-bit variable H_j is called the (intermediate) chaining value and is initialized with a predefined n-bit initial value IV. The variable $h = H_{t+1}$ is called the hash value. The function g is called the output transformation. However, in most hash function designs, like SHA-2, the output transformation is the identity mapping and we get $h = H_{t+1} = H_t$.

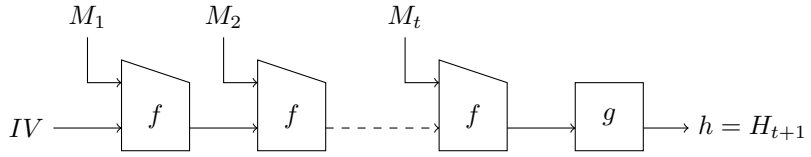


Figure 8.1: Outline of the Merkle-Damgård design principle.

Merkle-Damgård provide a proof showing that if the compression function f is collision resistant then the hash function H is also collision resistant, which is a major advantage of this construction. In order to achieve this, messages are preprocessed using a technique called Merkle-Damgård strengthening. It specifies an unambiguous padding method which includes the binary representation of the message length. The existence of proofs which reduce properties of the hash function to properties of the compression function has directed the cryptanalytic attention to the compression function as well. Although, a collision attack on the compression function rarely leads to a collision attack on the hash function, such proofs are not applicable any more.

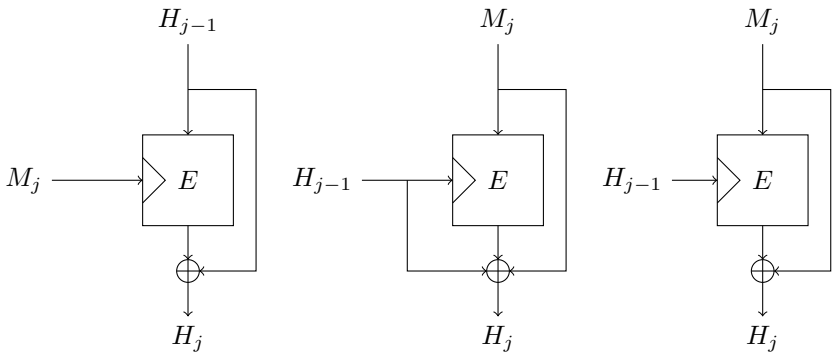
8.1.3 Compression Function Constructions

The most used hash functions today are block cipher based hash functions, where the compression function consists of a block cipher in a special mode of operation. There are several advantages using such constructions. Block ciphers are well studied, good implementations exists and if a certain block cipher is already available in an application, the effort to add a hash function based on this cipher is low. In [PGV93], Preneel et al. studied constructions for compression functions based on a block ciphers. Later, Black et al. [BRS02] provided security proofs in

the ideal cipher model. The three most popular modes are Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO) and Miyaguchi-Preneel (MP) [MvOV97]. For a block cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ with a key of size k bits and a block size of n bits, the three modes are defined as follows:

$$\begin{aligned} H_j &= E(M_j, H_{j-1}) \oplus H_{j-1} && \text{Davies-Meyer (DM)} \\ H_j &= E(H_{j-1}, M_j) \oplus H_{j-1} \oplus M_j && \text{Miyaguchi-Preneel (MP)} \\ H_j &= E(H_{j-1}, M_j) \oplus M_j && \text{Matyas-Meyer-Oseas (MMO)} \end{aligned}$$

In Figure 8.2 the three modes are illustrated.



(a) Davies-Meyer (b) Miyaguchi-Preneel (c) Matyas-Meyer-Oseas

Figure 8.2: The three most common constructions for block cipher based hash functions.

One disadvantage of constructing hash functions from block ciphers is that usually, the block size of the cipher needs to be quite large to prevent generic attacks. Furthermore, depending on the construction mode an adversary may have full control over the key. Results considering this for the security analysis have been presented in [KR07, RP94]. Therefore, dedicated hash functions have been designed with the explicit purpose of hashing. The most important ones are MD5 [Riv92], SHA-1 [Nat95] and SHA-2 [Nat02]. Also the round 2 SHA-3 candidate SIMD [LBF09a] and the Korean hash function standard HAS-160 [Tel08] are in this category. Although, these hash functions are called dedicated hash functions their construction is also based on block ciphers specially designed for hashing. They operate in in Davies-Meyer mode, except SIMD which is using a different mode (see Chapter 10).

8.1.4 ARX Hash Functions

The most commonly used hash functions today are also ARX hash functions, e.g. MD5, SHA-1 or SHA-256. Furthermore, in the SHA-3 competition several hash functions of these category have been proposed, e.g. Skein [FLS+11],

Blake [AHMP11], SIMD [LBF09a] or CubeHash [Ber09]. The operations in such a hash functions consist of modular additions, rotations and XORs. ARX hash functions following the design principles of MD4 [Riv90] include also one or more Boolean functions. An additional property shared among these hash functions is the usage of a message expansion. Such a function takes as input a message block and extends it to a larger size.

The ARX design has several advantages, like fast performance on various platforms or compact implementations. However, it is difficult to show any bounds regarding linear and differential cryptanalysis. Furthermore, finding differential characteristics in such a construction can be hard. Hence, the necessity of advanced and automatic tools is increased. Note that, all hash function analysed in this thesis are ARX hash functions.

8.2 Types of Collisions

In the last decade new types of collisions have been in the focus of research. For each type the definition of a collision is weakened, giving the cryptanalyst more freedom in the analysis of hash functions and its underlying compression function. In the following, we define various forms of collision resistance which are first mentioned in[LM92]. They are important since every property of a practical hash function which deviates from the ideal model of a hash function can be seen as a weakness.

Inputs m and m^* with $m \neq m^*$ collide for a given hash function H if $H(m) = H(m^*)$. Thus we define the notion of collision resistance:

Definition 8.1 (Collision Resistance)**.** *A hash function H is collision resistant if it is computationally infeasible to find any two distinct inputs m, m^* which hash to the same output, i.e., $H(m) = H(m^*)$.*

The best generic algorithm to find collisions for H is to generate t messages and to search for colliding outputs in all $H(m_i)$ for $1 \leq i \leq t$. For a hash function with a digest size of n bits, the expected value for t to find a collision is

$$\mathbb{E}(t) \approx 2^{n/2}. \tag{8.2}$$

Definition 8.2 (k-near Collision)**.** *Given a hash function H and two distinct messages m and m^*. If the Hamming weight of the XOR-difference between $H(m)$ and $H(m^*)$* is *k, we have a k-near collision.*

The expected value of the number of messages t_k needed to find a k-near collision for hash functions of n bit output size is (cf. [BC04])

$$\mathbb{E}(t_k) \approx \frac{2^{n/2}}{\sqrt{\binom{n}{k}}}, \tag{8.3}$$

If the IVs are not specified in advance, one has more freedom. Hence, colliding hash values might be easier to find if we can freely choose the IV. We denote by

$H(IV, m)$ an iterated hash function as in (8.1) where the initial value is set to a value of our choice.

Definition 8.3 (Semi-free-start Collision of a Hash Function)**.** *For every given 4-tuple (m, m^*, IV, IV^*) with $IV = IV^*$ and $m \neq m^*$ which satisfies the equality*

$$H(IV, m) = H(IV^*, m^*) \tag{8.4}$$

is a semi-free-start collision for the hash function H.

Colliding hash values might be easier to find if we allow differences both in the messages and in the IVs.

Definition 8.4 (Free-start Collision of a Hash Function)**.** *For every given 4-tuple (m, m^*, IV, IV^*) with $IV \neq IV^*$ which satisfies the equality*

$$H(IV, m) = H(IV^*, m^*) \tag{8.5}$$

is a free-start collision for the hash function H.

The definition of a collision for a compression function is slightly different, since an adversary has always the possibility to modifiy both inputs, the message and the chaining input.

Definition 8.5 (Collision of a Compression Function)**.** *For every given 4-tuple (m, m^*, IV, IV^*) with $IV \neq IV^*$ or $m \neq m^*$ which satisfies the equality*

$$f(IV, m) = f(IV^*, m^*) \tag{8.6}$$

is a collision for the compression function f.

8.3 Differences, Characteristics and Probabilities

In the following sections we introduce the notion and definitions used in differential cryptanalysis. We define various types of differences, define the notion of differentials and characteristics, and show how the probabilities for both are determined.

8.3.1 Types of Differences

Definition 8.6 (XOR Difference)**.** *To denote differences between $X \in \{0,1\}^n$ and $X^* \in \{0,1\}^n$ with respect to the XOR operation we use*

$$\Delta X = X \oplus X^* \in \{0,1\}^n.$$

Definition 8.7 (Modular Difference)**.** *For differences between $X \in \{0,1\}^n$ and $X^* \in \{0,1\}^n$ with respect to modular addition, we use the bijection*

$$\begin{aligned} i: \{0,1\}^n &\longrightarrow \mathbb{Z}_{2^n} = \{0,1,\ldots,2^n-1\} \\ X = (x_{n-1},\ldots,x_0) &\longmapsto x_{n-1}2^{n-1} + \cdots + 2x_1 + x_0. \end{aligned}$$

Then, we denote by

$$\delta X = X - X^*$$

the modular difference $i^{-1}(i(X) - i(X^*)) \in \{0,1\}^n$.

One of the contributions of [WLF+05] was the introduction of *signed-bit differences.*

Definition 8.8 (Signed-bit Difference)**.** *For differences between $X \in \{0,1\}^n$ and $X^* \in \{0,1\}^n$, we denote by*

$$\Delta_s X = X - X^* = (y_{n-1},\ldots,y_0)$$

the signed-bit difference where $y_j = X_j - X_j^* \in \{-1,0,1\}$ for $0 \leq j < n$.

These signed-bit differences reflect the fact that in a dedicated attack, not only differences play a role but also the actual values of the bits. In other words, a given signed-bit difference defines a subset of all message pairs $(X, X^*) \in \{0,1\}^n \times \{0,1\}^n$, that satisfy the specified difference conditions.

Inspired by these signed-bit differences, [DR06] came up with the so called *generalized conditions* for differences, where all 16 possible conditions on a pair of bits are taken into account. Table 8.1 lists all these possible conditions and introduces notations for the various cases.

Definition 8.9 (Generalized Conditions for Differences)**.** *Let $X \in \{0,1\}^n$ and $X^* \in \{0,1\}^n$, then the notation*

$$\nabla X = [c_{n-1},\ldots,c_0],$$

where c_i denotes one of the conditions of Table 8.1 for the i-th bit, defines a subset of pairs $(X, X^*) \in \{0,1\}^n \times \{0,1\}^n$ that conforms to the specified conditions.

For example in [DR06], all pairs of 8-bit words X and X^* that satisfy

$$\{(X, X^*) \in \{0,1\}^n \times \{0,1\}^n \mid X_7 \cdot X_7^* = 0, X_i = X_i^* \text{ for } 2 \leq i \leq 5, X_1 \neq X_1^*, X_0 = X_0^*\},$$

are written in the form

$$\nabla X = \texttt{[7?----x0]}$$

and the generalized condition of bit i is denoted by ∇X_i. For the remainder of this thesis if we mention differences we usually mean bitwise XOR differences. It is explicitly mentioned if other differences are used.

Table 8.1: Notation for possible generalized conditions on a pair of bits.

(X_i, X_i^*)	$(0,0)$	$(1,0)$	$(0,1)$	$(1,1)$
?	✓	✓	✓	✓
-	✓	-	-	✓
x	-	✓	✓	-
0	✓	-	-	-
u	-	✓	-	-
n	-	-	✓	-
1	-	-	-	✓
#	-	-	-	-
3	✓	✓	-	-
5	✓	-	✓	-
7	✓	✓	✓	-
A	-	✓	-	✓
B	✓	✓	-	✓
C	-	-	✓	✓
D	✓	-	✓	✓
E	-	✓	✓	✓

8.3.2 Differential Characteristics and Probabilities

Definition 8.10. *A* differential *[LM92] of a function $B : \{0,1\}^n \to \{0,1\}^n$ is a pair $(\Delta X, \Delta Y) \in \{0,1\}^n \times \{0,1\}^n$. We call ΔX the* input difference *and ΔY the* output difference*. The* differential probability *$DP_B(\Delta X, \Delta Y)$ of a differential $(\Delta X, \Delta Y)$ with respect to B is defined as*

$$DP_B(\Delta X, \Delta Y) = 2^{-n} \# \{X \in \{0,1\}^n | B(X \oplus \Delta X) = B(X) \oplus \Delta Y\}.$$

Analogously, we can define differential and differential probability if the function $B[k](x)$ is parametrized by a key k.

Definition 8.11. *Let $B[k](x)$ denote a function composed of r steps $B^i[k^i](x)$ parametrized by r subkeys $k^0, k^1, \ldots k^{r-1} \in \{0,1\}^l$:*

$$B[k](x) = \left(B^{r-1}[k^{r-1}] \circ \cdots \circ B^0[k^0]\right)(x).$$

A characteristic *through $B[k](x)$ is a vector $Q = (\Delta X^0, \Delta X^1, \ldots, \Delta X^r)$ with $\Delta X^i \in \{0,1\}^n$ for $i = 0, \ldots, r$. A characteristic $Q = (\Delta X^0, \Delta X^1, \ldots, \Delta X^r)$ is* in *a differential $(\Delta X, \Delta Y)$ if $\Delta X^0 = \Delta X$ and $\Delta X^r = \Delta Y$. If we now consider the following set of equations*

$$\begin{aligned} B^0[k^0](X \oplus \Delta X^0) &= B^0[k^0](X) \oplus \Delta X^1 \\ &\vdots \\ \left(B^{r-1}[k^{r-1}] \circ \cdots \circ B^0[k^0]\right)(X \oplus \Delta X^0) &= \left(B^{r-1}[k^{r-1}] \circ \cdots \circ B^0[k^0]\right)(X) \\ &\quad \oplus \Delta X^r, \end{aligned} \tag{8.7}$$

then the parameterized differential probability $DP_B[k](Q)$ *of a characteristic* Q *with respect to* $B[k](x)$ *is defined as*

$$DP[k](Q) = 2^{-n} \# \{X \in \{0,1\}^n \mid X \text{ satisfies (8.7)}\}.$$

To bring the definitions in context with hash fucntion we adapt Definition 8.11. The subkeys in Definition 8.11 correspond to expanded message blocks, and therefore, differences in the messages will lead to differences in the keys. From a block-cipher point of view this corresponds to the related-key setting. So let us assume we have a vector

$$\Delta K = (\Delta k^0, \ldots, \Delta k^{r-1}) \in (\{0,1\}^l)^r \tag{8.8}$$

of (round-key) differences. Let $M \in \{0,1\}^b$ be a message block and let the injective function $\mathrm{ME} : \{0,1\}^b \to (\{0,1\}^l)^r$ be the message expansion of the underlying hash or compression function. We introduce the set

$$S(\Delta K, \Delta M) = \{M \in \{0,1\}^b \mid \mathrm{ME}(M) \oplus \mathrm{ME}(M \oplus \Delta M) = \Delta K\}. \tag{8.9}$$

Definition 8.12. *Let* $f(x,m)$ *be a Davies-Meyer compression function based on* $B[k](x)$ *as in Definition 8.11. A* characteristic *through* $f(x,m)$ *is a vector*

$$Q = (\Delta M; \Delta K; \Delta X^0, \Delta X^1, \ldots, \Delta X^r, \Delta X^{r+1})$$

with $\Delta X^i \in \{0,1\}^n$ *for* $i = 0, \ldots, r+1$ *and* ΔK *as in* (8.8)*. A characteristic* Q *is* in *a differential* $(\Delta M; \Delta K; \Delta X, \Delta Y)$ *if* $\Delta X^0 = \Delta X$ *and* $\Delta X^{r+1} = \Delta Y$ *and* $\mathrm{ME}(M) \oplus \mathrm{ME}(M \oplus \Delta M) = \Delta K$*. With the notation*

$$B[k \oplus \Delta K](x) = \left(B^{r-1}[k^{r-1} \oplus \Delta k^{r-1}] \circ \cdots \circ B^0[k^0 \oplus \Delta k^0]\right)(x)$$

we are now considering the equations

$$\begin{aligned} B^0[k^0](X \oplus \Delta X^0) &= B^0[k^0 \oplus \Delta k^0](X) \oplus \Delta X^1 \\ &\vdots \\ B[k](X \oplus \Delta X^0) &= B[k \oplus \Delta K](X) \oplus \Delta X^r \\ B[k](X \oplus \Delta X^0) \boxplus (X \oplus \Delta X^0) &= (B[k \oplus \Delta K](X) \boxplus X) \oplus \Delta X^{r+1}. \end{aligned} \tag{8.10}$$

Here, $\boxplus : \{0,1\}^n \to \{0,1\}^n$ *defines the feed-forward in the Davies-Meyer construction. Then, for a characteristic as above we can define* $DP(Q)$ *as*

$$\begin{aligned} &DP(Q) = 2^{-(n+b)}. \\ &\#\left\{M \in \{0,1\}^b, X \in \{0,1\}^n \mid X \text{ satisfies (8.10) and } M \in S(\Delta K, \Delta M)\right\}. \end{aligned} \tag{8.11}$$

A right pair for such a characteristic defines a free-start collision.

Note that the differential analysis of block ciphers and hash functions differ in the aspect that in the block cipher case, we are interested in the output of the most probable characteristic whereas in the hash function case, we are mostly interested in a collision producing characteristic.

L- and NL-Characteristics

In this thesis, we use two different techniques to construct differential characteristics. In order to distinguish between them, we introduce the notion of L-characteristics and NL-characteristics.

Definition 8.13 (L-Characteristics)**.** *Let H be a hash function consisting of non-linear operations and LH be a linearised model of the hash function H, that is all non-linear operations are approximated by linear operations. Then we call a differential characteristic for LH an L-characteristic.*

Definition 8.14 (NL-Characteristics)**.** *Let H be a non-linear hash function. A characteristic for H is called NL-characteristic.*

The advantage of L-characteristics is that they can be found easily, by solving a set of linear equations. However, the quality of such an L-characteristic highly depends on the number of (bit)-conditions that have to be fulfilled in order to guarantee that the it holds also in the original function.

Correlation between Collision Types and Characteristics

In order to put Definition 8.12 of a differential characteristic in context to the different types of collisions, we briefly discuss the attack scenarios in the following.

- Collision
  In the case of a collision attack on a hash function, the difference in the chaining input and in the output is zero. Hence, $\Delta X^0 = 0$ and $\Delta X^{r+1} = 0$ have to hold. Furthermore, in a collision attack the chaining input is fixed to a predefined IV. By this additional condition the differential probability of a characteristic is affected, since the amount of possible inputs (available freedom) is reduced.

- Near-Collision
  For a near-collision $\Delta X^0 = 0$ and $\Delta X^{r+1} \neq 0$ holds (where ΔX^{r+1} is of low Hamming weight). In this scenario the chaining input is also fixed. The effects are the same as mentioned above.

- Free-Start Collision
  In contrast to a collision attack the chaining inputs contain differences. Therefore, $\Delta X^0 \neq 0$ and $\Delta X^{r+1} = 0$ have to hold. In this case the chaining inputs can be freely chosen and do not change the differential probability given in Definition 8.12.

- Free-Start Near-Collision
  A free-start near-collision does not restrict the location of differences, i.e. differences in the chaining inputs and in the outputs are allowed. Hence, $\Delta X^0 \neq 0$ and $\Delta X^{r+1} \neq 0$ holds for a characteristic of a such a collision. Again the chaining input can be freely chosen.

8.4 Summary

In this chapter, we introduced on cryptographic hash functions and their basic security requirements. We described the Merkle-Damgård design principle which is used by all hash functions analysed in this thesis. Furthermore, we explain the most common modes to construct a hash function out of a block cipher.

The most common tool in the cryptanalysis of hash functions is differential cryptanalysis. This technique, originally invented in the analysis of block ciphers, can be used for attacks on hash functions. Most collision attacks on hash functions are differential attacks.

We introduced different types of collisions and the important terms used in differential cryptanalysis, like differential characteristic and differential probability. In this context, we distinguished between two types of differential characteristics, depending on which technique is used to construct them. This differentiation is important for the subsequent chapters. Finally, we set different types of collision in context with differential cryptanalysis.

9 Cryptanalysis of ARX Based Hash Functions

In this chapter, we review two analysis methods for cryptographic hash functions that have led, when combined, to the most successful collision attacks on SHA-1 [WYY05b, DR06, DMR07, GA11]. Both methods are based on differential cryptanalysis and aim to find differential characteristics with high probability. However, both methods utilize different techniques to achieve this goal, and can be successfully combined as shown for SHA-1 [WYY05b, DR06]. Furthermore, these techniques are especially applicable to ARX based hash functions. We adapt and extend the ideas which results in two distinct tools which search completely automatic for differential characteristics.

9.1 Attacks on SHA-1

In this section, we give an overview of the basic attack strategy of the recent collision attacks on SHA-1. In the subsequent sections, we will in detail describe the underlying techniques and our contributions.

9.1.1 First attack on full SHA-1

The major breakthrough in the cryptanalysis of SHA-1 was achieved by Wang et al. First announced at Crypto 2004 in the rump session and then presented at Crypto 2005, Wang et al. [WYY05b] broke the full SHA-1 hash function. Using signed bit differences and message modification techniques, they mount an attack with total complexity of 2^{69}. At the same conference they announced

that the complexity can be improved to 2^{63}. Furthermore, they provided an actual collision for 58 out of 80 steps. Mendel et al. [MPRR06b] computed the success probability of the attack which is $2^{-64.57}$. For the collision attack on SHA-1, Wang et al. use basically the following strategy (see Figure 9.1):

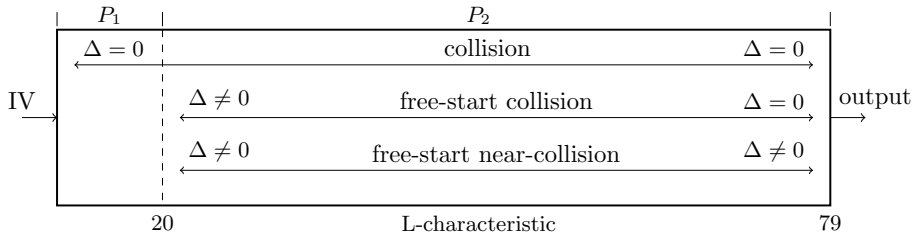


Figure 9.1: General strategy of Wang et al.'s attack on SHA-1.

- Split the 80 steps into two parts denoted by P_1 and P_2.
  - P_1 consists of the steps at the start of the hash function, for which it is possible to efficiently solve the equations imposed by the characteristic, by using methods like for instance so-called basic message modification and advanced message modification also introduced by Wang et al. [WYY05b].
  - P_2 consists of the remaining steps. Since, the conditions imposed by the characteristic through P_1 have no impact, due to message modification, the attack complexity is determined by P_1.
- Determine a low-weight L-characteristic, that leads to a free-start collision in P_2.
- Find an NL-characteristic in P_1, such that the zero difference in the state variables at the start of P_1 is transformed into the desired difference in the state variables at the beginning of P_2. Hence, it becomes possible to turn the free-start collision for P_2 into a collision for SHA-1.

Furthermore, in the attack on SHA-1 Wang et al. used multi-block messages. First, a pair of message blocks is determined, which leads to a free-start near-collision in P_2. As before, the NL-characteristic is used to turn the free-start near-collision collision for P_2 into a near-collision for SHA-1. Second, a pair of message blocks is determined, which results in a collision for SHA-1, when concatenated with the first pair of message blocks. The second pair of message blocks uses the same L-characteristic as the first one. The NL-characteristic is slightly changed because it no longer starts from a zero input difference in the state variables. The fact that it is easier to find a near-collision than a collision was observed already by Biham and Chen in [BC04]. The technique can of course be extended to messages consisting of 3 or more blocks, but this does not improve the complexity of the attack.

In summary, the attack constructs an L-characteristic which holds with high probability and a complex NL-characteristic which connects the L-characteristic with the zero difference in the chaining input. To construct L-characteristics Wang et al. extended their approach for SHA-0 [WYY05c]. However, at the same time Rijmen and Oswald [RO05] applied and optimized the attack of Chabaud and Joux [CJ98] to SHA-1 which offers a more convenient method to find L-characteristics with high probability. The work was further improved by Pramstaller et al. [PRR05]. This approach is discussed in detail in Section 9.2.

To construct a NL-characteristic, Wang et al. exploit the non-linearity of the state update. Unfortunately, Wang et al. do not describe in detail how to construct this NL-characteristic. But they show by an example collision for 58 steps that the non-linearity can be exploited. However, De Cannière and Rechberger [DR06] presented later an automatic search algorithm to find such NL-characteristics.

9.1.2 Automatic Search

The collision attack on SHA-1 by Wang et al. used complex characteristics which were manually constructed. De Cannière and Rechberger [DR06] were the first who presented a method to find for NL-characteristics in an automatic way. As a proof of concept they showed a two-block collision for 64-step SHA-1 based on a new characteristic. At that time this was the highest number of steps for which a SHA-1 collision was published.

In order to reflect that both the differences and the actual values of bits are important for the attack, they introduced generalized conditions for differential characteristics (see Section 8.3). In this concept they allowed characteristics to impose arbitrary conditions on the pairs of bits. Using these generalized conditions they built an algorithm to search for NL-characteristics for SHA-1.

The idea behind the heuristic algorithm is rather simple, but works very well on SHA-1. The basic strategy of the attack is the same as Wang et al.'s attack on SHA-1 (see Section 9.1.1) and is illustrated in Figure 9.2. The N steps of

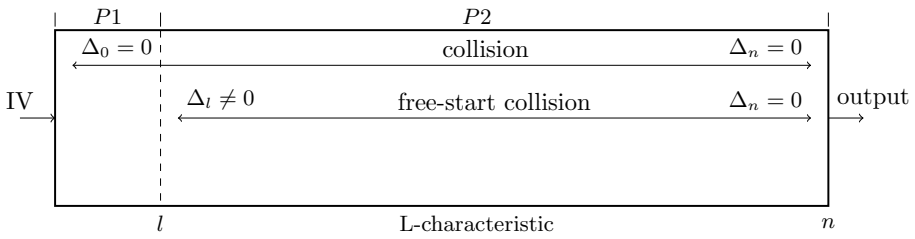


Figure 9.2: General attack strategy. Note that in all cases the message difference is non-zero.

the hash function are split into two groups, which are denoted by P_1 and P_2. P_1 consists of the steps at the start of the hash function. P_2 consists of the

remaining steps. Finding a good characteristic is then divided into two phases. First, a sparse L-characteristic, which can start with any difference in the state variables, but ends in a zero-difference in the last step, i.e. leads to a free-start collision in P_2, is determined. Such characteristics can easily be found with the techniques described in Section 9.2. In the case of SHA-1, L-characteristics for the linearised state update (omitting the message expansion) were searched. Actually, it is sufficient to search only in the state word A, since A is the only state word updated in one iteration. Once such a suitable differential characteristic is found, the corresponding message difference needs to be determined. Since these characteristics are using XOR differences, they need to be converted to the representation based on generalized conditions. Table 9.1 is an example of a characteristic using generalized conditions, which is used as input for the non-linear search algorithm. ∇A_i and ∇W_i represent the state words and expanded message words, respectively. The state words for step -4 to 0 represent the IV of SHA-1 and are therefore already fixed at the beginning. The state words from step 1 to 11 are free of conditions, which is imposed by `'?'`. Hence, step -4 to 11 define P_1. For the notation of generalized conditions see Section 8.3.1.

In the second phase the search algorithm proposed by De Cannière and Rechberger is used to find conditions for the words of P_1 such that the zero difference in the state variables at the start of P_1 is transformed into the desired difference in the state variables at the start of P_2. Hence, it becomes possible to turn the free-start collision for P_2 into a collision. The basic idea of the search algorithm is to randomly pick an unrestricted bit position denoted by `'?'` and impose a zero-difference denoted by `'-'`. Afterwards, it is calculated how this condition propagates. Table 9.2 illustrates the propagation after the imposition of a zero difference on a unrestricted bit. If an inconsistency occurs the algorithm backtracks to an earlier state of the search. This is repeated until all unrestricted bits are eliminated. This algorithm is the largest part of the work factor in this approach. Using this techniques and further optimizations the authors were able to find NL-characteristics for SHA-1 efficiently. In a further improvement they do not restrict the output of the L-characteristic to be zero. In a two block approach the search for two different NL-characteristics connecting the same L-characteristic in each block. The output differences are then cancelled out by the feed-forward after the second block.

Later they improved their attack and presented a collision for 70 steps [DMR07]. Recently, Grechnikov and Adinetz [GA11] published on the ePrint archive a further improvement of the automatic collision search. They managed to construct collisions for 75 steps of SHA-1. This is so far the highest number of steps for a practical SHA-1 collision.

9.2 Finding L-Characteristics

In this section, we give a general decription of how L-characteristics can be constructed for any hash functions using techniques from coding theory. Due to the increasing complexity in the design of hash functions, especially noticeable

Table 9.1: Characteristic for SHA-1 using generalized conditions [DR06]. Starting point of the non-linear search algorithm.

i	∇A_i	∇W_i
-4:	00001111010010111000011111000011	
-3:	01000000110010010101000111011000	
-2:	01100010111010110111001111111010	
-1:	11101111110011011010101110001001	
0:	01100111010001010010001100000001	-xx----------
1:	????????????????????????????????	xxx----------x-x-x-
2:	????????????????????????????????	--x----------x----xx
3:	????????????????????????????????	x-xx----------x------
4:	????????????????????????????????	xx-x----------x-x--xx
5:	????????????????????????????????	xx-x----------x---x-
6:	????????????????????????????????	--x----------
7:	????????????????????????????????	-xx----------xx--x-
8:	????????????????????????????????	-xx----------x----xx
9:	????????????????????????????????	--x----------x------
10:	????????????????????????????????	xxx----------x----x-
11:	????????????????????????????????	-xx----------x-
12:	x----------	x----------x
13:	x----------	----------x-----
14:	----------	----------xx
15:	x----------xx	-x----------x-x--x-
16:	----------x-	-x----------x------
17:	x----------x-	xxx----------x-x--x-
18:	----------	x-x----------
19:	----------x-	x----------x------
	...	...
49:	----------x-	----------x------
50:	----------	x----------x-
51:	----------	----------
52:	----------	x----------
53:	----------	x----------
54:	----------	----------
	...	...
60:	----------	----------
61:	----------	----------
62:	----------	----------
63:	----------	----------
64:	----------	

Table 9.2: Example for the propagation of the nonlinear search algorithm [DR06].

i	∇A_i	∇W_i
-4:	00001111010010111000011111000011	
-3:	01000000110010010101000111011000	
-2:	01100010111010110111001111111010	
-1:	11101111110011011010101110001001	
0:	01100111010001010010001100000001	-xx----------
1:	??x----------	xxx----------x-x-x-
2:	????????????????????????????????x-	--x----------x----xx
3:	????????????????????????????????	x-xx----------x------
	...	...

in the SHA-3 competition, the necessity of automated tools is apparent. Hence, we present the first publicly available toolbox which is used to search automated for L-characteristics that hold with high probability. Although, the techniques presented in this section are known, even for recently designed hash functions as SIMD, weaknesses can be shown using the techniques presented in this section.

Various problems arising in cryptanalysis of hash functions can be linked to problems in (linear) coding theory [PRR05, RO05, MN09, MN11]. As observed by Rijmen and Oswald [RO05], all differential characteristics for a linearised hash function can be seen as the codewords of a linear code. The probability that the characteristic holds in the original hash function is related to the Hamming weight of the characteristic. In general, a differential characteristic with low Hamming weight has a higher probability than one with a high Hamming weight (see Section 9.2.1). Finding a characteristic with high probability (low Hamming weight) is related to finding a low-weight codeword in linear codes. Therefore, we can use algorithms from coding theory to search for codewords with low Hamming weight, which can be used to construct high probability characteristics for (parts of) the hash function. This is an essential part of our attacks on SIMD (see Chapter 10) and HAS-160 (see Chapter 11). The main strategy in this approach can be summarized as follows:

1. Construct a linear approximation of the target hash function.
2. Construct a generator matrix for the corresponding linear code.
3. Search for characteristics for the linearised hash function.
4. Identify conditions such that the differential holds for the real hash function.

A linear approximation of a hash function is constructed by replacing all non-linear operations by linear ones. Depending on the non-linear operations, different approximations may be useful. Therefore, the chosen approximation depends on the hash function. However, ARX based hash functions have one non-linear operation in common, which is the modular addition.

9.2.1 Approximation of Modular Additions

As mentioned before it is assumed that a differential characteristic with low Hamming weight has a higher probability than one with a high Hamming weight. The probability that a differential characteristic is followed, is determined by the differences that are input to each of the non-linear functions that were approximated using a linear operation. It is well known that the differential behaviour of modular addition can be approximated quite well, by that of XOR when the Hamming weight of the input difference, ignoring the most significant bit, is small [CJ98, LM01, PRR05, RO05, IP09, MN09, MN11]. Based on the analysis for the differential probability of modular addition by Lipmaa and Moriai [LM01] it follows that the Hamming weight of the input differences for each modular

addition can be used to approximate the success probability of the differential characteristic.

9.2.2 Construction of a Generator Matrix

In coding theory, a linear code of length m and rank k is a linear subspace $C \leq \mathbb{F}_q^m$ with dimension k of the vector space $\mathbb{F}_q^m$. $\mathbb{F}_q$ is a finite field with q elements. A linearised model of a hash function describes a linear binary code $C \leq \mathbb{F}_2^m$. A codeword of this code represents a differential characteristic for the hash function. A bit with value 1 denotes a difference between the corresponding pair. The Hamming weight of a codeword is used to approximate the differential probability. Hence, we want to find codewords with a low Hamming weight.

A generator matrix $G_{k\times m}$ is a basis of a linear code. Such a basis generates all possible codewords. In order to construct a generator matrix for a linearised hash function, we first need to determine which values used in the hash computation should be included in the linear code. First of all each input which introduces differences need to be included, i.e. message input and in the case of free-start collisions also the chaining input. Next all intermediate values which are input to a non-linear operation in the original hash function need to be included, since the amount differences in these inputs should be as low as possible. As we show in Chapter 10 and Chapter 11 sometimes not all inputs need to be stored. Let H be an iterative hash function model. In the following we show how a generator matrix for the linear code described by LH is constructed. Without loss of generality we consider one block of the hash computation.

Let $\Delta W \in \{0,1\}^k$ be the differences in all inputs bit-wise concatenated, $\Delta Z \in \{0,1\}^l$ the differences in all intermediated values bit-wise concatenated and $\Delta O \in \{0,1\}^n$ the output difference of $HL(\Delta W)$. A codeword cw of the linear code is defined as

$$cw := (\Delta W, \Delta Z, \Delta O).$$

In order to construct a generator matrix we compute cw_i for $\Delta W = e_i$ where e_i is the i-th unit vector, for $i = 1, \ldots, k$. The generator matrix is then given by

$$G_{k\times(k+l+n)} := \begin{pmatrix} cw_1 \\ \vdots \\ cw_k \end{pmatrix}. \tag{9.1}$$

Note that, the resulting generator matrix is systematic. A systematic generator matrix has the form

$$G_{k\times m} = [I_k|B],$$

where I_k is the identity matrix of dimension k. In the following we also need the definition of a check matrix. The check matrix is derived from the generator matrix as follows:

$$H = [B^T|I_{m-k}].$$

A check matrix is used to test if a codeword c belongs to the corresponding linear code, since $H \cdot c = 0$ holds for any codeword of the corresponding linear code.

In coding theory there are different applications where it is important to construct codewords with a low Hamming weight. Although determining the minimum weights of linear binary code is an NP-complete problem [BMvT78], the general problem of finding a codeword of a weight bounded to a given value is not proved to be NP-hard. Therefore, different probabilistic algorithms has been developed to search for such codewords. These algorithms are using the generator matrix or the check matrix for the search. By applying such a search algorithm to linear code described by the linear model of a hash function, we search for differential characteristics with high probabilities.

Collision Producing Characteristics

In order to search for differential characteristics which produce a collision, i.e. $\Delta O = \vec{0}$, we need to shorten the linear code. A shortened code of a linear code C is the set of all codewords of C which are zero at a fixed coordinate with that coordinate deleted. Those codewords in C with 1 at that coordinate are removed from C. The generator matrix of the shortened code is derived from the generator matrix (10.6). First Gaussian elimination is used to remove all elements equal to 1 from the columns corresponding to ΔO. Next the row and the column with the remaining 1 are removed from the matrix. Hence, the length and dimension of the code is reduced by one for each output bit resulting in a generator matrix with dimension $k - n$ and length $k + l$.

9.2.3 Algorithms for Low Hamming Weight Search

In the following, we briefly discuss three algorithms which can be used to search for a low Hamming weight codeword in a linear code.

Canteaut's and Chabaud's Algorithm

Leon published [Leo88] a probabilistic algorithm which was later improved by Chabaud [Cha94] and led finally to the efficient algorithm proposed by Canteaut and Chabaud [CC98]. This iterative algorithm basically looks for small Hamming weight codewords in a smaller code. Such a codeword is considered as a good candidate for a low Hamming weight codeword for the whole code. Given a systematic generator matrix, the algorithm randomly selects σ columns of it and splits the selection in two sub matrices of equal size. By computing all linear combination of p rows (usually 2 or 3) for each sub matrix and storing their weight, the algorithm searches for a collision of both weights which allow to search for codewords of $2p$. Then two randomly selected columns are interchanged, followed by one Gaussian elimination step. This procedure is repeated until a sufficiently small Hamming weight is found.

One should note that the parameter σ has an upper bound regarding the implementation aspect. By using a preallocated table for all possible p linear combinations of the first submatrix, the search for a collision of weights is significantly improved. Therefore, the parameter is limited by the amount of available

memory. However, Canteaut's and Chabaud do recommend a value of 20 for the codes used in [CC98], which is small enough, but it is unclear if it is the best choice for codes considered here.

Stern's Algorithm

Stern's algorithm [Ste88] uses the check matrix of a linear code, rather than the generator matrix as Canteaut and Chabaud do. Nevertheless it is easy to convert a generator matrix to a check matrix and vice versa. Let be H the $(n-k) \times n$ check matrix for a $[n, k]$ code over $\mathbb{F}_2$. The algorithm selects $(n-k)$ columns of H. A subset Z of l columns out of the $(n-k)$ columns is randomly selected. The remaining k columns of H are partitioned into two sets where each column is chosen independently and uniformly to join one of the two sets. The algorithm then searches for codewords that have p nonzero bits in both sets, zero nonzero bits in Z and $w-2p$ nonzero bits in the remaining columns, where $w \geq 0$ is an input chosen by the user. If there are no such codewords, the algorithm starts again at this beginning.

Ball-Collision Algorithm

Bernstein et al. [BLP10] proposed recently a new algorithm. Their algorithm operates on a given check matrix. The algorithm selects a random information set in the check matrix and then searches for vectors having a particular and complicated pattern. Due to the more complex process we refer to [BLP10] for a detailed description of the algorithm.

9.2.4 The CodingTool Library

In the subsequent chapters we show for two hash functions that the coding theoretic approach can be successfully used to find efficiently L-characteristics with high probability. This L-characteristics are used in the construction of distinguishers, near-collisions or collisions. We developed a toolbox which purpose is to search efficiently and automated for L-characteristics of a linearised model of any hash function [Nad10]. Due to the increasing complexity in the design of hash functions, especially noticeable in the SHA-3 competition, the necessity of an automated tool was apparent.

The CodingTool library is a new collection of tools for using techniques from coding theory in cryptanalysis. It is the first tool of this kind publicly available and published under the GPL 3.0 license. The core part is an implementation of the probabilistic algorithm from Canteaut and Chabaud [CC98] to search for code words with low Hamming weight. Additional functionalities like code shortening, code puncturing or adding a weight to each bit of a codeword are implemented. Furthermore, the library provides data structures to assist the user in creating a linear code for a specific problem. An easy to use interface to the provided algorithms, powerful data structures and a command line parser reduces the implementation work of a cryptanalyst to a minimum. Beside the

existing functionality, the library can be extended very easily. A possible improvement is the implementation of faster search algorithms or the improvement of the existing one.

9.3 Finding NL-characteristics

In Section 9.1.2 we describe the technique used by De Cannière and Rechberger to construct NL-characteristics for SHA-1. Unfortunately, in [DR06] there are not many details on how the propagation of conditions is done although this is a crucial part of the algorithm.

In this section, we generalize the approach, investigate the search algorithm and show our improvements leading to the best attacks with practical complexities for HAS-160 and SHA-256 (see Chapter 11 and Chapter 12). Most of our improvements to this technique have been developed during the analysis of SHA-2. Unfortunately, the approach of Cannière and Rechberger on SHA-1 cannot directly be applied to SHA-2. We have observed several problems in finding valid differential characteristics for SHA-2. We have identified these problems and solved them efficiently. Our extensions and improvements for this approach are published in [MNS11b] and [MNS11a].

For a more detailed description, we divide the technique into four parts. In the first part a starting point for the search algorithm is constructed. Next a search strategy is defined. We show a more general view on the strategy and present a new and more efficient strategy. The third part consists of the consistency checks performed during the search. Finally, the last part deals with the efficient propagation of conditions.

9.3.1 Determining a Starting Point

In the first part of the thesis, we show that a good starting point is important for the convergence of an iterative numerical method. This is also the case for the non-linear search algorithm described in this chapter. As shown in Section 9.1.2, for SHA-1 the starting point is defined by a certain L-characteristic. As we show in Chapter 11 a starting point can also consist of two different L-characteristics which are connected through the NL-characteristics. Furthermore, in Chapter 12 we need to extend the general strategy in order to successfully apply the technique on SHA-256. There we do not use L-characteristics as starting point. Hence, determining a good starting point depends on the targeted hash function why we refer to the corresponding chapters for more details.

9.3.2 Search Strategy

In general, our search technique can be divided into three parts: decision, deduction and backtracking. Note that the same separation is done in many other fields, like SAT solvers [GPFW96]. The first aspect of our search strategy is the decision, where we decide which bit is chosen and which condition is imposed

at its position. In the deduction part we compute the propagation of the imposed condition and check for contradictions. If a contradiction occurs we need to backtrack and undo decisions, which is the third part of the search strategy. A basic search strategy to find differential characteristics has been described in [DR06] and works as follows.

Let U be the set of all `'?'` and `'x'`, then repeat the following until U is empty.

Decision

1. Pick randomly a bit in U.
2. Impose a `'-'` for a `'?'` or randomly a sign (`'u'` or `'n'`) for `'x'`.

Deduction

3. Compute the propagation.
4. If a contradiction is detected start backtracking, else go to step 1.

Backtracking

5. Jump back to the point where the last sign was imposed and make a different decision and go to step 1.

Unfortunately, this strategy does not lead to any valid characteristics for SHA-256. In all cases at least one of the checks described in Section 9.3.3 failed. The reason for this is that conditions which are not covered by generalized conditions appear much more often than in SHA-1. Since more advanced checks are too expensive, we have developed a more sophisticated search strategy to find valid differential characteristics.

Two-Bit Conditions

Apart from generalized conditions, additional conditions on more than a single bit are present in a differential characteristic. Especially, conditions on two bits are needed such that a differential path is valid. These two-bit conditions have already been used by Wang et al. in their attacks on the members of the MD4 family [WLF+05]. Such two-bit conditions occur mostly in the propagation of differences through Boolean functions, like used in HAS-160 or SHA-2. The following example illustrates such conditions.

Example 9.1. *Let f be the majority function defined as follows:*

$$f(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z).$$

If an input difference in x should result in a zero output difference of $f(x, y, z)$, the remaining values y and z should be equal. Hence, we get a condition on two bits: $y = z$.

Similar conditions occur in various Boolean functions. Two-bit conditions are not covered by generalized conditions and thus, not shown in the characteristics given in [DR06]. However, two-bit conditions may lead to additional inconsistencies. The structure of such inconsistencies depends on the actual used Boolean functions. Therefore, we refer to Chapter 12.4.3 for more details.

Even if we include two-bit conditions to the approach, the above search strategy does not lead to valid characteristics for SHA-256, since more complex conditions occur. Since it is not feasible to track all possible conditions during a search, we developed a new search strategy.

Advanced Search Strategy

In our approach we already determine some message bits during the search for a differential characteristic. Generally speaking, we are combining the search for a conforming message pair with the search for a differential characteristic. In doing so we consider those bits much earlier, which are involved in many relations with other bits. This way, we can detect invalid characteristics at an early stage of the search. However, this should not be done too early to not restrict the message freedom too much. In addition, we are remembering critical bits during the search to improve the backtracking and speed-up the search process. In the following, we describe the used search strategy in more detail.

In general we have two phases in our search strategy where different bits are chosen (guessed) and we switch between these two dynamically. Phase 1 can be described as follows:

Let U be the set of all `'?'` and `'x'`.

Repeat the following until U is empty:

Decision

1. Pick randomly a bit in U.
2. Impose a `'-'` for a `'?'` or randomly a sign (`'u'` or `'n'`) for `'x'`.

Deduction

3. Compute the propagation as described in Section 9.3.4.
4. If a contradiction is detected start backtracking, else apply the additional checks of Section 9.3.3.
5. Continue with step 1 if all checks passed, if not start backtracking.

Backtracking

6. If the decision bit is `'x'` try the second choice for the sign or if the decision bit is `'?'` impose a `'x'`.
7. If still a contradiction occurs mark bit as critical by including it in set C.

8. Jump back until all critical bits in C can be resolved.
9. Continue with step 1.

Note that, the additional checks in step 4 are optional and a trade-off between number of checks and speed has to be done. The additional steps in the backtracking process improve the search speed significantly and prevent that critical bits result in a contradiction again.

Once phase 1 is finished (U is empty) we continue with phase 2 which can be summarized as follows.

Let U' be the set of all `'-'` with many two-bit conditions.

Repeat the following until U' is empty:

Decision

1. Pick randomly a bit in U'.
2. Impose randomly a `'0'` or `'1'`.

Deduction

3. Compute the propagation as described in Section 9.3.4.
4. If a contradiction is detected start backtracking, else apply additional checks from Section 9.3.3.
5. Continue with step 1 if all checks passed, if not start backtracking.

Backtracking

6. Try the second choice of the decision bit.
7. If still a contradiction occurs mark bit as critical.
8. Jump back until all critical bits can be resolved.
9. If necessary jump back to phase 1, otherwise continue with step 1.

Choosing a decision bit with many two-bit conditions ensures that bits which influence a lot of other bits are chosen first. Therefore, many other bits propagate by defining the value of a single bit. We want to note that due to step 9, we actually switch quite often between both phases in our search.

Additionally, we restart the search from scratch after a certain amount of contradictions or iterations to terminate branches which appear to be stuck because of exploring a search space far from a solution.

Note that, depending on the set U' the algorithm can run until a complete conforming message pair is found.

Table 9.3: Search parameters for the algorithm.

Parameters
size limit for $\|C\|$
number of two-bit conditions for a bit in U'
number of contradictions before a restart from scratch

Search Parameters

Our search strategy has several configurable parameters. In Table 9.3 these parameters are listed.
The number of critical bits which need to be resolved simultaneously is the first parameter. In our experiments the number of iterations until a solution is found changes with the parameter changing. If it is too small the convergence to a solution is slow. On the other hand if the parameter is too large the search is more likely to be directed to invalid characteristics. Hence, this parameter has to be chosen carefully. The second parameter controls which bits are included in U' in phase two. One bit can be involved in several two-bit conditions. The parameter specifies the minimum number of two-bit conditions of one bit such that it is included in C'. The value vary for different hash functions, but at least those bits with the maximum number of two-bit conditions are included. The last parameter defines the number of iterations after the search is restarted from scratch.

Note that the appropriate values for the parameters depend on the hash function and on the starting point as well. Hence, the values are determined empirically for each application.

9.3.3 Consistency Checks

To avoid inconsistent differential characteristics, we have evaluated a number of checks to detect contradictions as early and efficiently as possible. Note that a test which is able to detect many contradictions is usually also less efficient. However, also a simple test may detect a contradiction at a later point in the search. Depending on the target hash function the number of complex conditions can be high and hence they can be difficult to detect. Therefore, a trade-off has to be made.

Two-Bit Condition Check

Two-bit conditions are linear conditions in $\mathbb{F}_2$ since such conditions can only be either equal ($y = z$) or non-equal ($y \neq z$). Contradictions in two-bit conditions are efficiently detected by determining all two-bit conditions, setting up a linear system of equations and checking if the system is solvable by computing the rank.

Single-Bit Condition Check

A quite expensive test is to check for every bit restricted to `'-'` or `'x'` whether both possible cases (`'0'` and `'1'`, or `'n'` and `'u'`) are indeed still valid. If both choices for a single bit are invalid we know that the whole characteristic is impossible. Note that a similar check has been done by De Cannière and Rechberger in their attack on SHA-1. Of course these tests can be extended to other generalized conditions as well. However, it turned out to be more efficient to apply this check only rarely and only to specific conditions during the search. Furthermore, we have improved the speed of this complete test by applying it only to bits which are restricted by two-bit conditions.

Complete Condition Check on a Set of Bits

Since even the complete condition check is not able to detect many contradictions, we have analysed different variants of setting all possibilities for all or selected combinations of 2, 3 or 4 bits. Such tests indeed detect more impossible characteristics but are very inefficient to compute and thus, cannot be effectively used during the search for differential characteristics.

9.3.4 Efficient Condition Propagation

The efficient propagation of new conditions is crucial for the performance of the algorithm, since it is the most often needed operation in the search algorithm. Due to the nature of the search algorithm where changes to the characteristic (using generalized conditions) are done on bit-level, we perform the propagation of conditions also on bit-level. At the beginning of the search every bit has at least one of the 16 generalized conditions (see Table 8.1). During the search we impose conditions on specific bits. These bits are inputs or outputs of functions. If a bit in the output is changed then all bits which are used to determine this output bit are updated. We call such a set of bits a bit-slice. If the changed bit is an input of a function then all other bits of the corresponding bit-slice are updated. The following example illustrates this process.

Example 9.2 (Condition Propagation)**.** *Let $f : \mathbb{F}_{32}^3 \to \mathbb{F}_{32}$ be the Boolean IF function operating on 32-bit words and defined as follows:*

$$f(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) = o.$$

Then the output bit o_i depends on $\{x_i, y_i, z_i\}$ and the set $\{x_i, y_i, z_i, o_i\}$ forms a bit-slice. If the generalized condition ∇x_i changes then the conditions of the set $\{\nabla x_i, \nabla y_i, \nabla z_i, \nabla o_i\}$ are updated.

In our approach the update process is done exhaustively by computing all possible conditions of a bit-slice. This seems at first to be inefficient but we are using three techniques to significantly speed up the process. However, the update process for a modular addition is done in a slightly different way. The bit-slices of a modular addition contain also input carry and output carry. Hence, the

bit-slices are connected through the carry bits. If the condition for a carry bit changes then the connected bit-slice is updated as well. Furthermore, the whole update process is iterative and updates bits until conditions do not change any more.

Split-Up

The first technique is to split the step update of a hash function into smaller functions such that the number of inputs of one function is low. How this is done best depends on the actual hash function. The following example illustrates this step.

Example 9.3 (Split Up a Function)**.** *Let* $f : \mathbb{F}_{32}^3 \rightarrow \mathbb{F}_{32}$ *be defined as follows:*

$$f(x, y, z, u, v, w) = ((x \oplus y \oplus z) + u + v + w).$$

Then we split f *into following sub-functions:*

$$\begin{aligned} f_0(x, y, z) &= x \oplus y \oplus z = t \\ f_1(u, v, w, t) &= u + v + w + t \end{aligned}$$

In Example 9.3 a function performing a XOR of three inputs added to the result of an addition of three other inputs is split up in one sub-function performing the XOR operation and introducing a new state word, and a second function for the addition of four inputs. In that way we reduce the computational complexity of one propagation step. The drawback of this method is that we lose the relation between the sub-functions compared to a combined propagation.

Caching

The second technique is caching. By using a cache in the update process, a significant speed-up is achieved. A cache is a component that transparently stores data so that future requests for that data can be delivered faster. The data that is stored within a cache are generalized conditions for a group of bits before the update process and the result after the update process. Now before an update of a group of bits is done exhaustively, the update result is requested from the cache. If the cache contains the data (cache hit), no further computation has to be done. Otherwise (cache miss) the update is done as explained above and the result is added to the cache.

Note that, for functions with a low amount of inputs we can precompute a table with all possibilities and therefore omitting the exhaustive computation completely for this function.

Generalized Conditions and Linear Functions

The third speed-up is achieved from a special treatment of generalized conditions and linear functions. The generalized conditions in Table 8.1 are linear except for four conditions (‘7’,‘B’,‘D’ and ‘E’). In Table 9.4 we extend the previous

Table 9.4: Linear generalized conditions.

$(\mathbf{x_i^*}, \mathbf{x_i})$	$(\mathbf{0},\mathbf{0})$	$(\mathbf{1},\mathbf{0})$	$(\mathbf{0},\mathbf{1})$	$(\mathbf{1},\mathbf{1})$	**linear**
?	✓	✓	✓	✓	
-	✓	-	-	✓	$x_i + x_i^* = 0$
x	-	✓	✓	-	$x_i + x_i^* = 1$
0	✓	-	-	-	$x_i = 0, x_i^* = 0$
u	-	✓	-	-	$x_i = 0, x_i^* = 1$
n	-	-	✓	-	$x_i = 1, x_i^* = 0$
1	-	-	-	✓	$x_i = 1, x_i^* = 1$
#	-	-	-	-	
3	✓	✓	-	-	$x_i = 0$
5	✓	-	✓	-	$x_i^* = 0$
7	✓	✓	✓	-	
A	-	✓	-	✓	$x_i^* = 1$
B	✓	✓	-	✓	
C	-	-	✓	✓	$x_i = 1$
D	✓	-	✓	✓	
E	-	✓	✓	✓	

table by the corresponding linear equation. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be a linear function with $n > m$. Such a linear function can be described by a matrix $L \in \mathbb{F}_2^{n\times(n+m)}$ with the property

$$y = f(x) \Leftrightarrow L \begin{bmatrix} x \\ y \end{bmatrix} = 0. \tag{9.2}$$

Since we are considering pair of bits, we also get

$$y^* = f(x^*) \Leftrightarrow L \begin{bmatrix} x^* \\ y^* \end{bmatrix} = 0. \tag{9.3}$$

Now we merge (9.2) and (9.3) into one matrix L':

$$\begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix} \begin{bmatrix} x \\ y \\ x^* \\ y^* \end{bmatrix} = L' \begin{bmatrix} x \\ y \\ x^* \\ y^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{9.4}$$

L' is precomputed and modified during the update process. If the generalized conditions of one or more bits are changed which correspond to a linear function, the update is done as follows:

1. Back-substitute each linear generalized condition of the changed bits into L'.
2. Use Gauss-Jordan elimination to turn L' again in reduced row-echelon form.

3. Check each row if it represents a linear generalized condition.
4. Check each linear combination of corresponding two rows if it represents a linear generalized condition.
5. Set the linear generalized condition for the corresponding bit.
6. For non-linear generalized conditions use the bit-wise update approach.

Using this update process many bits are updated at once which results in a performance improvement. Furthermore, there are linear functions where the bit-slice update approach does not reveal all new propagated conditions. The problem is solved using this method. We refer to Section 12.5.4 for more details.

9.4 Summary

In this chapter, we reviewed the recent collision attacks on the hash function SHA-1. First Wang et al. [WYY05b] presented a collision for 53 out of 80 steps. They used a long L-characteristic and a complex NL-characteristic to connect it to the chaining input. De Cannière and Rechberger [DR06] proposed a new algorithm to search automatically for such NL-characteristics and were able to present a collision for 64 steps. They later improved their result up to 70 steps.

In Section 9.2, we showed how L-characteristics with high probability can be found. The technique is based on coding theory, since finding differential characteristics for a linearised model of hash function is related to the problem of finding codewords with low Hamming weight in a linear code. We showed how a linear code is constructed from a linearised model and presented three probabilistic algorithms searching for low Hamming weight codewords. We published an open-source toolbox [Nad10] which searches for L-characteristics for any linearised model of hash function automatically. The toolbox implements a search algorithm, interfaces, data structures and several other useful methods. The abstraction level is high, so that a cryptanalyst does not have to care about complicated implementation details. Furthermore, the library was used in attacks on different hash functions, presented in the subsequent chapters.

In Section 9.3, we investigated the approach by De Cannière and Rechberger [DR06]. We presented a more general description of their algorithm and gave details which are missing in [DR06]. Furthermore, we extended their technique in several ways. First of all, we generalized their search strategy and presented a more efficient strategy. Secondly, we included two-bit conditions in the search process which is essential for SHA-2 and showed different ways to perform consistency checks. Finally, we show how the propagation of new conditions can be done efficiently.

10
Application to SIMD

SIMD is a round 2 candidate of the SHA-3 competition [Nat09] designed by Leurent *et al.* [LBF09a]. It is an iterative hash function based on the Merkle-Damgård design principle (see Section 8.1). It is a wide-pipe design producing a hash value up to 512 bits, denoted by SIMD-n, where n is the output length. For the remainder of this chapter wherever we mention SIMD we refer to SIMD-512. The design of the compression function is similar to the MD4 family. Furthermore, the mode of operation used in SIMD has been proven to be secure [CN08, MT07]. The designers additionally provide bounds for a large class of differential attacks. Most of the security is based on the message expansion.

Although, SIMD is a new hash function, with significantly increased complexity compared to the MD4 family, we show in two attacks that well-known techniques as described in Section 9.2 are applicable. However, due to the increased complexity automated tools are necessary. In our first attack we construct an L-characteristic that holds with high probability for the compression function of SIMD-512. The L-characteristic is used to build a differential q-multicollision which serves as a distinguisher for the compression function. The attack has a complexity of $2^{427.60}$ compression function calls. Including the output transformation we can distinguish the output of SIMD-512 from random with a complexity of about $2^{429.74}$ compression function calls.

Due to our first attack, the designers tweaked SIMD [LBF09b] to prevent it. However, we present a distinguisher for the tweaked version as well. We show how one can use the boomerang attack on a hash function to construct a distinguisher with high probability. For the boomerang attack we again construct L-characteristics that hold with high probability. The first result is a distinguishing attack for the full permutation of SIMD-512 with complexity $\approx 2^{226.52}$. Next we show how this distinguisher can be extended to the full compression

function of SIMD-512. with complexity $\approx 2^{200.6}$. The strategy to construct such second order differentials is based on the recently proposed cryptanalysis of reduced SHA-2 [LM11] and Blake [BNR11].

Although, we do not attack the whole hash function, we show non-random properties of the SIMD-512 compression function. Our attacks do not invalidate the security claims of the designers, since most of the security comes from the message expansion. However, we want to point out that the non-randomness of the compression function of SIMD effects the applicability of the proofs for the mode of operation build upon it. The results of this chapter have been published in [MN09, MN11].

10.1 Related Work

The amount of available cryptanalysis of SIMD is low compared to other SHA-3 candidates. A round reduced version of tweaked SIMD was attacked by Nikolić et al. [NPSS10]. They presented distinguishers for the compression function of SIMD-512 reduced to 24 round with a linearised message expansion and reduced to 12 rounds with unmodified message expansion. Both attacks work for the tweaked version and are based on rotational properties of the compression function. The success probabilities for the distinguishers are 2^{-497} and 2^{-236}, respectively.

Later Yu and Wang [YW11] presented a free-start near-collision attack for SIMD-256 reduced to 20 rounds and for SIMD-512 reduced to 24 rounds. The attack complexities are 2^{107} and 2^{208}, respectively. Furthermore, they showed a distinguisher for the full compression function with complexity 2^{398}. All attacks are for the tweaked version.

Finally, the designers [BFL10] published a free-start distinguisher for the compression function exploiting the existence of symmetric states. Furthermore, they showed that distinguishers without differences in the message have only a minimal impact on the security of the hash function.

Higher-order differentials have been introduced by Lai in [Lai92] and first applied to block ciphers by Knudsen in [Knu94]. The application to stream ciphers was proposed by Dinur and Shamir in [DS09] and Vielhaber in [Vie07]. Recently, Lamberger and Mendel [LM11] showed how higher-order differentials can be used to attack SHA-256 and presented a distinguisher for 46 steps. This result was improved to 47 steps in [BLMN11]. The attack is similar to the *boomerang attack* and the *inside-out* attack by Wagner [Wag99] or the *rectangle attack* by Biham et al. [BDK01], all three used in the cryptanalysis of block ciphers. A previous application of the boomerang attack to hash functions is due to Joux and Peyrin [JP07], who used the boomerang attack as a neutral bits tool to speed-up existing collision attacks. Furthermore, Biryukov et al. [BNR11] presented a boomerang attack on the SHA-3 finalist Blake resulting in a distinguisher for 7 rounds of the Blake-32 compression function with a complexity of 2^{232}.

10.2 Description of SIMD

SIMD is an iterative hash function that follows the Merkle-Damgård design principle (see Section 8.1). The main component of a Merkle-Damgård hash function is the compression function. In the case of SIMD-512 to compute the hash of a message M, it is first divided into k chunks of 1024 bits. By the use of a message expansion one block is expanded to 8192 bits. Then the compression function is used to compress the message chunks and the internal state. The padding rule to fill the last blocks is known as the Merkle-Damgård strengthening. The initial value of the internal state is called IV and is fixed in the specification of the hash function. The output transformation is a truncation in SIMD. The internal state of SIMD contains 32 32-bit words and is therefore twice as large as the output. SIMD consist of 4 rounds where each round consist of 8 steps. The feed-forward consists of four additional steps with the IV as message input. Since we apply a compression function attack independent from the message expansion, we omit the description of the message expansion. For a detailed description of the hash function we refer to [LBF09a].

Due to our first attack on SIMD [MN09], the designers changed the permutations and rotation constants in the step function to prevent the attack [LBF09b]. In the following, SIMD 1.0 refers to the unchanged specification and SIMD 1.1 to the new version. We first describe SIMD 1.0 and afterwards the changes made for SIMD 1.1.

Notation. For the specification of SIMD we use the notation presented in Table 10.1.

Table 10.1: Notation

Notation	Description
$\neg X$	inversion of X
$X \oplus Y$	bit-wise XOR of X and Y
$X \boxplus Y$	addition of X and Y modulo 2^{32}
$X \lll n$	bit-rotation of X by n positions to the left
$X \ggg n$	bit-rotation of X by n positions to the right
$X \ll n$	bit-shift of X by n positions to the left
$X \gg n$	bit-shift of X by n positions to the right

10.2.1 SIMD step function

The core part of SIMD is the step function of the state update. Figure 10.1 illustrates the step function at step t. The state update consists of eight step functions in parallel. To make the step function dependent from each other, $(A^{t-1}_{p^t(i)} \lll r^t)$ is included in a modular addition, where $p^t(i)$ is a permutation, which is different for each step. Equation (10.1) is the formal definition of the

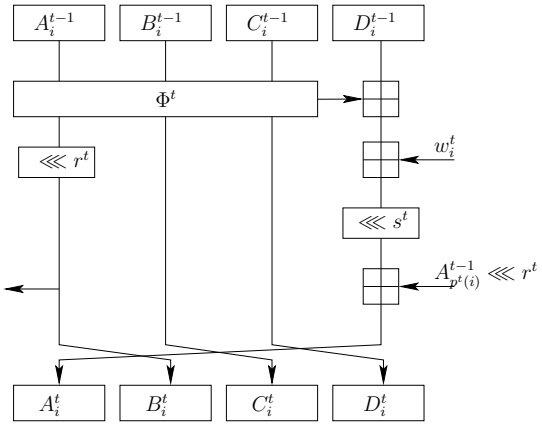


Figure 10.1: Update function of SIMD at step t. $i = 0, \cdots, 7$.

step function.

$$\begin{aligned} A_i^t &= (D_i^{t-1} \boxplus w_i^t \boxplus \Phi(A_i^{t-1}.B_i^{t-1}, C_i^{t-1})) \lll s^t \boxplus (A_{p^t(i)}^{t-1} \lll r^t) \\ B_i^t &= A_i^{t-1} \lll r^t \\ C_i^t &= B_i^{t-1} \\ D_i^t &= C_i^{t-1} \end{aligned} \tag{10.1}$$

The permutation p is separated in 4 different permutations:

$$p^0(x) = \begin{cases} x+1 \pmod 8, & \text{if } x = 0 \pmod 2 \\ x-1 \pmod 8, & \text{otherwise} \end{cases}$$

$$p^1(x) = \begin{cases} x+2 \pmod 8, & \text{if } x = 0 \pmod 4 \text{ or } x = 1 \pmod 4 \\ x-2 \pmod 8, & \text{otherwise} \end{cases}$$

$$p^2(x) = 7 - x \pmod 8$$
$$p^3(x) = x + 4 \pmod 8$$

The permutation used at step t is $p^{t \bmod 4}$. As mentioned before, the 32 steps of SIMD are divided into 4 rounds, each consisting of 8 steps. The boolean function Φ and the rotation constants (s and r) for a round are given in Table 10.2. The Boolean functions IF and MAJ are defined as follows:

$$f_{IF}(x, y, z) = (x \wedge y) | (\neg x \wedge z)$$
$$f_{MAJ}(x, y, z) = (x \wedge y) | (x \wedge z) | (y \wedge z).$$

In Table 10.3 the rotation constants for each round are given. The feed-forward consist of four steps using the same step function. Table 10.4 lists the used Boolean function and the rotation constants for the feed-forward.

Table 10.2: Φ and rotation constants for a round.

Step	$\boldsymbol{\Phi}$	$\boldsymbol{r}$	$\boldsymbol{s}$
0	IF	π_0	π_1
1	IF	π_1	π_2
2	IF	π_2	π_3
3	IF	π_3	π_0
4	MAJ	π_0	π_1
5	MAJ	π_1	π_2
6	MAJ	π_2	π_3
7	MAJ	π_3	π_0

Table 10.3: Rotation constants for SIMD 1.0.

Round	$\boldsymbol{\pi_0}$	$\boldsymbol{\pi_1}$	$\boldsymbol{\pi_2}$	$\boldsymbol{\pi_3}$
0	3	20	14	27
1	26	4	23	11
2	19	28	7	22
3	15	5	29	9

Permutations and Constants for SIMD 1.1.

Due to our first attack on SIMD [MN09], the designers changed the permutations and constants to prevent the finding of L-characteristics that hold with high

Table 10.4: Φ and rotation constants for the feed-forward of SIMD 1.0.

Step	$\boldsymbol{\Phi}$	$\boldsymbol{r}$	$\boldsymbol{s}$
0	IF	15	5
1	IF	5	29
2	IF	29	9
3	IF	9	15

probability. The permutation p is given by:

$$\begin{aligned} p^0(x) &= x \oplus 1 \\ p^1(x) &= x \oplus 6 \\ p^2(x) &= x \oplus 2 \\ p^3(x) &= x \oplus 3 \\ p^4(x) &= x \oplus 5 \\ p^5(x) &= x \oplus 7 \\ p^6(x) &= x \oplus 4 \end{aligned}$$

The permutation used at step t is $p^{t \mod 7}$. In Table 10.5 the rotation constants for tweaked SIMD are given. Table 10.6 lists the used Boolean function and the

Table 10.5: Rotation constants for SIMD 1.1.

Round	π_0	π_1	π_2	π_3
0	3	23	17	27
1	28	19	22	7
2	29	9	15	5
3	4	13	10	25

rotation constants for the feed-forward.

Table 10.6: Φ and rotation constants for the feed-forward of SIMD 1.1

Step	Φ	r	s
0	IF	4	13
1	IF	13	10
2	IF	10	25
3	IF	25	4

10.3 Finding L-Characteristics for SIMD

In both attacks we construct L-characteristics for the compression function using the approach described in Section 9.2. Therefore, we describe how the compression function is linearised in order to be able to construct a linear code. Afterwards, we construct a generator matrix for this code and reduce the code length significantly. Note that this step is equal for both versions of SIMD.

10.3.1 Linearising the SIMD Compression Function

The step function (10.1) is the only part of SIMD which has to be linearised. The non-linear parts of this function are the modular additions and the Boolean function Φ. In the attack, we replace all modular addition by XORs. The function Φ depends on the current step and is either the IF function or the MAJ function. To choose a good approximation for those, we have to take a closer look on the differential behaviour of them.

Differential behaviour of IF and MAJ

The differential behaviour of IF and MAJ is already discussed in [Dau05]. IF and MAJ have three inputs. Table 10.7 shows the differential propagation of the Boolean functions regarding XOR-differences.

Table 10.7: Differential propagation of IF and MAJ.

Δx	Δy	Δz	ΔIF	ΔMAJ
0	0	0	0	0
0	0	1	$x \oplus 1$	$x \oplus y$
0	1	0	x	$x \oplus z$
0	1	1	1	$y \oplus z \oplus 1$
1	0	0	$y \oplus z$	$y \oplus z$
1	0	1	$x \oplus y \oplus z$	$x \oplus z \oplus 1$
1	1	0	$x \oplus y \oplus z \oplus 1$	$x \oplus y \oplus 1$
1	1	1	$y \oplus z \oplus 1$	1

Since we aim for low weight characteristics, we replace the Boolean function Φ with the 0-function, with respect to its differential behaviour, i.e. an input difference in Φ results in no output difference, no matter if IF or MAJ is used. This has probability $1/2$ in most cases. One can see that there is exactly one input difference for IF and one for MAJ where the output difference is always one. We discard characteristics with such properties. Finally, the linearised step function looks as follows:

$$\begin{aligned} A_i^t &= (D_i^{t-1} \oplus w_i^t \oplus 0) \lll s^t \oplus (A_{p^t(i)}^{t-1} \lll r^t) \\ B_i^t &= A_i^{t-1} \lll r^t \\ C_i^t &= B_i^{t-1} \\ D_i^t &= C_i^{t-1} \end{aligned} \tag{10.2}$$

Note that for the feed-forward w_i^t is equal to one word of the IV.

10.3.2 Construction of a Generator Matrix

To construct the generator matrix for the linearised compression function of SIMD, we proceed as explained in Section 9.2.2. We first need to add the input of the linearised model to the codeword. Since we target the compression function and introduce only differences in the IV, the chaining input is the only input. Furthermore, we have to include each part where differences could decrease the success probability. Let the vector

$$\Delta cv^t := (\Delta A_i^t | \Delta B_i^t | \Delta C_i^t | \Delta D_i^t), \tag{10.3}$$

for $i = 0, \cdots, 7$ and $cv^t \in \{0,1\}^{1024}$ be the bit-wise concatenation of all differences in the chaining values at step t. Then the vector

$$\Delta dc := (\Delta IV, \Delta cv^1, \cdots, \Delta cv^{36}),$$

where $\Delta dc \in \{0,1\}^{N \cdot 1024}$, represents the differences in the IV, chaining values after each step and the output of the SIMD compression function for N steps including the feed-forward. Δdc is one codeword of the linear code and therefore a differential characteristic. To construct the generator matrix for the linear code, we proceed as follows:

1. Compute Δdc_j with the input difference $\Delta IV_j = e_j$, where $e_j \in \{0,1\}^{1024}$ is the j-th unit vector.

2. Repeat the computation for $j = 1, \ldots, 1024$.

The resulting systematic generator matrix of the linear code for the linearised SIMD compression function is defined in the following way:

$$G_{1024 \times N \cdot 1024} := [I_{1024 \times 1024} | CV], \tag{10.4}$$

where CV is defined by

$$\begin{pmatrix} \Delta dc_1 \\ \vdots \\ \Delta dc_{1024} \end{pmatrix}.$$

10.3.3 Reducing the Code Length

The linear code for all steps is large and therefore the search space. Hence, it is important to keep the code as small as possible. If we take a closer look on the dependencies of each chaining value, one can see that only the A_i's are updated at each step and the other values only depend on them. Therefore, we can reduce the code size by only considering the A_i's at each step function. The definition of Δcv^t in Equation (10.3) changes to

$$\Delta cv^t := (\Delta A_i^t), \tag{10.5}$$

Following the same procedure above, the resulting generator matrix is much smaller, namely

$$G_{1024\times(1024+N\cdot 256)} := [I_{1024\times 1024}|CV]. \tag{10.6}$$

Therefore, the performance of the search for low Hamming weight codewords is increased.

10.3.4 Finding Codewords with Low Hamming Weight

We used the probabilistic algorithm by Canteaut and Chabaud implemented in the CodingTool Library (see Section 9.2.3). Additionally, we check for each codeword if differences at the input of the Boolean function result in a guaranteed output difference. If this is the case we discard the codeword.

10.3.5 Estimating the Probability for a L-Characteristic

To compute the probability of the found differential characteristic, we have to consider the differences entering the Boolean function Φ and the modular additions.

The Boolean Function Φ

The probability for blocking a difference in one bit at the input of Φ is $1/2$ or 0 for some cases, but then the characteristic is discarded (see Section 10.3.1). Hence, the total probability is determined by the sum of all differences at the input. Note, that differences at the same bit positions are counted only once. The overall probability for step t is defined by 2^{-x}, where x is given by

$$\sum_{i=0}^{7} hw(\Delta A_i^{t-1} \vee \Delta B_i^{t-1} \vee \Delta C_i^{t-1})$$

and $hw(\cdot)$ is the bit-wise Hamming weight of a 32-bit word.

The Modular Additions

Consider the additions (10.7) from the step function (10.1).

$$(\Delta D_i^{t-1} \boxplus \Delta w_i^t) \lll s^t \boxplus (\Delta A_{p^t(i)}^{t-1} \lll r^t) \tag{10.7}$$

We could consider each modular addition separately and prevent a carry for each bit difference, but this would result in a rather conservative estimation. By allowing carries in the first addition, we can compensate them at the second addition. However, this is not that easy, because of the rotation after the first modular addition.

First we take a look at the following addition:

$$\Delta D_i^{t-1} \boxplus \Delta w_i^t.$$

If we have a difference at the same bit position, we can cancel them out with probability 1/2. The overall probability to cancel out such differences for step t is 2^{-y}, where y is defined by

$$y := \sum_{i=0}^{7} hw(\Delta D_i^{t-1} \wedge \Delta w_i^t).$$

Note that $\Delta w_i^t \neq 0$ only for the feed-forward. If there is only a difference in one input of the modular addition (bit-wise), we allow carries. However, we do not want that the carry expansion is destroyed, due to the rotation to left by s^t bits, since we cannot compensate this in the second addition. To take care of this problem we have to consider two cases.

Let be l_j the bit position of the j-th difference in ΔD_i^{t-1} before the rotation, l'_j after the rotation and $d_{\mathtt{MSB}}(l_j)$ ($d_{\mathtt{MSB}}(l'_j)$) the distance of l_j (l'_j) to the most significant bit (MSB). The first case is $d_{\mathtt{MSB}}(l_j) < s^t$, *i. e.* the difference is rotated over the MSB. Therefore, we have to ensure that the carry expands at most to the MSB from the position of the difference *before* the rotation. The probability for that is

$$1 - 2^{-d_{\mathtt{MSB}}(l_j)}.$$

The second case considers $d_{\mathtt{MSB}}(l_j) \geq s^t$, *i. e.* the difference is not rotated over the MSB. In this case we have to ensure that the carry expands at most to the MSB from the position of the difference *after* the rotation. The probability for that is

$$1 - 2^{-d_{\mathtt{MSB}}(l'_j)}.$$

This differentiation has to be done for each difference in ΔD_i^{t-1}. The overall probability is given by the product of all single probabilities.

In the last modular addition

$$(\Delta D_i^{t-1} \lll s^t) \boxplus (\Delta A_{p^t(i)}^{t-1} \lll r^t),$$

we first cancel out differences at the same bit positions of both variables with probability 1/2 for each such difference. In the last step we compensate the carries from the first addition with the same probability. Finally, the overall success probability for the second modular addition is 2^{-z}, where z is defined as follows:

$$z := \sum_{i=0}^{7} hw(\Delta D_i^{t-1} \lll s^t \vee \Delta A_{p^t(i)}^{t-1} \lll r^t).$$

Note, that we ignore differences in the MSB for these calculations, which results in a small improvement.

10.4 Distinguishers

In this section, we describe two types of distinguisher which are used in our subsequent attacks (see Section 10.5 and Section 10.6).

10.4.1 Differential q-multicollision

In our first attack, we use an L-characteristic to construct a q-multicollision as distinguisher (see Section 10.5). The notion of differential q-multicollision was introduced by Biryukov et al. [BKN09] in the cryptanalysis of AES-256. They show that differential q-multicollisions can be found for AES-256 with a complexity of $q \cdot 2^{67}$, while for an ideal cipher an adversary needs at least

$$O(q \cdot 2^{\frac{q-2}{q+2} \cdot n}) \tag{10.8}$$

time. Note that in [BKN09] the attack is described for a block cipher. However, it can be easily adapted for a random function. Below we repeat the basic definition and lemma, we need for the distinguishing attack for the compression function of SIMD 1.0.

Definition 10.1. *A set of two differences and q pairs*

$$\{\Delta IV, \Delta M; (IV_1, M_1), (IV_2, M_2), \cdots, (IV_q, M_q)\}$$

is called a differential q-multicollision for $f_{IV}(\cdot)$ *if*

$$\begin{aligned} f_{IV_1}(M_1) \oplus f_{IV_1 \oplus \Delta IV}(M_1 \oplus \Delta M) &= f_{IV_2}(M_2) \oplus f_{IV_2 \oplus \Delta IV}(M_2 \oplus \Delta M) \\ &= \cdots = f_{IV_q}(M_q) \oplus f_{IV_q \oplus \Delta IV}(M_q \oplus \Delta M). \end{aligned}$$

In the case of SIMD, f is the compression function and ΔM is equal 0.

Lemma 10.1. *To construct a differential q-multicollision for an ideal function with an n-bit output an adversary needs at least* $O(q \cdot 2^{\frac{q-2}{q+2} \cdot n})$ *queries on the average.*

The proof for Lemma 10.1 works similar as in [BKN09] for an ideal cipher. In Section 10.5.2, we show how to find a differential q-multicollision for the SIMD 1.0 compression function with a complexity of about $q \cdot 2^{425.28}$ instead of the expected

$$q \cdot 2^{\frac{q-2}{q+2} \cdot 1024}.$$

10.4.2 Higher-Order Differentials and Hash Functions

In Section 10.6 our attack is based on the boomerang attack. In order to find a distinguishing property we construct a second order differential collision for the compression function. In this section we recall the basic definitions and give a high level description of the attack strategy.

While a standard differential attack exploits the propagation of the difference between a pair of inputs to the corresponding outputs, a higher-order differential attack exploits the propagation of the difference between differences. Higher-order differential cryptanalysis was introduced by Lai in [Lai94] and subsequently applied to block ciphers by Knudsen in [Knu94]. We recall the basic definitions that we will need in Section 10.6.

Definition 10.2. *Let $(S,+)$ and $(T,+)$ be Abelian groups. For a function $F : S \to T$, the derivative at a point $a \in S$ is defined as*

$$\Delta_a F(x) = F(x+a) - F(x)\,. \tag{10.9}$$

The i-th derivative of F at $(a_1, a_2, \ldots, a_i)$ is then recursively defined as

$$\Delta^{(i)}_{a_1,\ldots,a_i} F(x) = \Delta_{a_i}(\Delta^{(i-1)}_{a_1,\ldots,a_{i-1}} F(x))\,. \tag{10.10}$$

When applying differential cryptanalysis to a hash function, a collision for the hash function corresponds to a pair of inputs with output difference zero. Similarly, when using higher-order differentials we define a higher-order differential collision for a function F as follows.

Definition 10.3. *An i-th order differential collision for a function F is an i-tuple $(a_1, a_2, \ldots, a_i)$ together with a value x_0 such that*

$$\Delta^{(i)}_{a_1,\ldots,a_i} F(x_0) = 0\,. \tag{10.11}$$

Note that the common definition of a collision for hash functions corresponds to a higher-order differential collision of order $i = 1$.

From (10.11) we see that we can freely choose $i+1$ of the input parameters, i.e. x_0 and $a_1, \ldots, a_i$, which then fix the remaining input. Hence, the expected number of solutions to (10.11) is one after choosing $2^{n/(i+1)}$ values for the inputs and the query complexity is:

$$\approx 2^{n/(i+1)} \tag{10.12}$$

In the following, we will only consider the case $i = 2$ for which the query complexity of the attack is $2^{n/3}$.

In order to construct a second-order differential collision for the function F, we use a strategy recently proposed in cryptanalysis of reduced SHA-2 in [LM11] and [BLMN11]. The idea of the attack is quite simple. Assume we are given two differentials for F_0 and F_1 with $F = F_1 \circ F_0$, where one holds in the forward direction and one in the backward direction. To be more precise, we have

$$F_0^{-1}(y+\beta) - F_0^{-1}(y) = \alpha$$

and

$$F_1(y+\gamma) - F_1(y) = \delta$$

where the differential in F_0^{-1} holds with probability p_0 and in F_1 holds with probability p_1. Using these two differentials, we can now construct a second order differential collision for F. This can be summarized as follows (see also Figure 10.2).

1. Choose a random value for X and compute $X^* = X + \beta$, $Y = X + \gamma$, and $Y^* = X^* + \gamma$.
2. Compute backward from X, X^*, Y, Y^* using F_0^{-1} to obtain P, P^*, Q, Q^*.

3. Compute forward from X, X^*, Y, Y^* using F_1 to obtain R, R^*, S, S^*.
4. Check if $P^* - P = Q^* - Q$ and $S - R = S^* - R^*$ is fulfilled.

Since

$$P^* - P = Q^* - Q = \alpha, \text{ resp. } S - R = S^* - R^* = \delta, \tag{10.13}$$

will hold with probability at least p_0^2 in the backward direction, resp. p_1^2 in the forward direction and assuming that the differentials are independent the attack succeeds with a probability of $p_0^2 \cdot p_1^2$. Hence, the expected number of solutions to (10.13) is 1, if we repeat the attack about $1/(p_0^2 \cdot p_1^2)$ times.

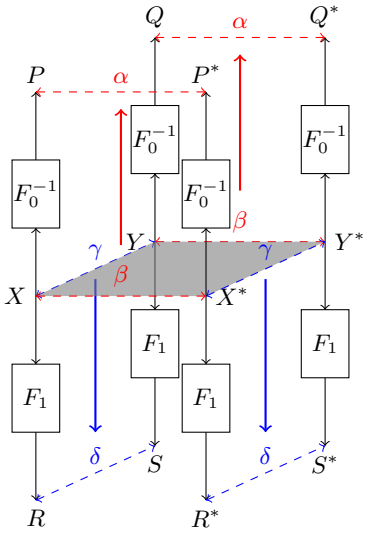


Figure 10.2: Schematic view of the attack.

10.5 Application to SIMD 1.0

In this section, we present a distinguisher attack on the compression function of SIMD 1.0 with a complexity of $2^{427.60}$ compression function calls. Including the output transformation we can distinguish the output of SIMD from random with a complexity of about $2^{429.74}$ compression function calls. The distinguisher is based on a L-characteristic with differences only in the IV. A characteristic with high success probability is found as described in Section 10.3.

We define a linear code for the whole compression function of SIMD 1.0, i.e. 36 steps including the feed-forward. Hence, the generator matrix is

$$G_{1024\times 10240} := [I_{1024\times 1024}|CV].$$

As described in Section 10.3.1, we block each input difference in the Boolean function Φ, no matter if IF or MAJ is used. This has probability 1/2 in most

cases. There is exactly one input difference for IF and one for MAJ where the output difference is always one. We discard characteristics with such properties, except in the feed-forward. There we manually correct the characteristic, resulting in a slightly higher Hamming weight. Furthermore, we use the nonlinearity of the IF function in the feed-forward to decrease the Hamming weight significantly (see Section 10.5.1).

10.5.1 The Differential Characteristic

We have found several characteristics with low Hamming weight. The best ones have a weight of 504. We can further reduce the weight by using the nonlinearity of the IF function in the feed-forward. If we do not block all input differences in the Boolean function, we can cancel out additional differences, which results in a lower Hamming weight for the subsequent steps. Thus, the overall success probability of the characteristic is increased. In that way we can improve the characteristics to a weight of 486. By a detailed analysis (see Section 10.3.5) we determine the success probability of the characteristics which is $\approx 2^{-507.34}$ without message modification. If we use additionally message modification, we increase the probability to $\approx 2^{-425.28}$. Table 10.8 presents one of the differential characteristics with weight 486. Due to space restriction we do not show the complete characteristic but the differences in the IV, which is enough to reconstruct the whole differential characteristic.

Table 10.8: Differences in the IV.

i	A_i^0	B_i^0	C_i^0	D_i^0
0	00000000	00000000	00000000	00000000
1	00000000	00000000	00000000	00000000
2	00000000	00000000	00000000	00000000
3	00000000	104804a0	00000000	00000000
4	00000000	00000000	050e0010	00000000
5	00000000	00000000	00000000	00000000
6	00000000	00000000	00000000	68801201
7	04004400	00000000	00000000	00000000

In Table 10.9 the characteristic in the steps of the feed-forward, including the above modifications, is given. The characteristic leads to a guaranteed difference in one bit at the output of Φ in the third step of the feed-forward. By correcting this manually, the success probability is slightly decreased.

Table 10.10 splits the probability estimation into rounds and steps (the probabilities are given in $\log_2$).

Message Modification

To improve the success probability of the differential characteristic we use message modification. We have the freedom choosing the actual values of the IV and the message words. Regarding the message words, we assume that we can

Table 10.9: Differences in the chaining values in the feed-forward.

(t,i)	A_i^t	B_i^t	C_i^t	D_i^t
$(33,0)$	00000000	00000000	00000000	00000000
$(33,1)$	00000000	00000000	00000000	00000000
$(33,2)$	00000000	00000000	00000000	00000000
$(33,3)$	00000000	00000000	83801001	00000000
$(33,4)$	00000000	00000000	00000000	0000c008
$(33,5)$	00000000	00000000	00000000	00000000
$(33,6)$	84d0c901	00000000	00000000	00000000
$(33,7)$	80088000	8410c1c0	00000000	00000000
$(34,0)$	00000000	00000000	00000000	00000000
$(34,1)$	00000000	00000000	00000000	00000000
$(34,2)$	00000000	00000000	00000000	00000000
$(34,3)$	02090094	00000000	00000000	83801001
$(34,4)$	9a193831	00000000	00000000	00000000
$(34,5)$	01100010	00000000	00000000	00000000
$(34,6)$	00000000	9a192030	00000000	00000000
$(34,7)$	00000000	01100010	8410c1c0	00000000
$(35,0)$	00000000	00000000	00000000	00000000
$(35,1)$	00000000	00000000	00000000	00000000
$(35,2)$	00220002	00000000	00000000	00000000
$(35,3)$	21620401	80412012	00000000	00000000
$(35,4)$	8c010008	33432706	00000000	00000000
$(35,5)$	00000000	00220002	00000000	00000000
$(35,6)$	00000000	00000000	9a192030	00000000
$(35,7)$	20000000	00000000	01100010	8410c1c0
$(36,0)$	02001118	00000000	00000000	00000000
$(36,1)$	00000000	00000000	00000000	00000000
$(36,2)$	00000000	44000400	00000000	00000000
$(36,3)$	00000040	c4080242	80412012	00000000
$(36,4)$	00000000	02001118	33432706	00000000
$(36,5)$	00000000	00000000	00220002	00000000
$(36,6)$	4d00b040	00000000	00000000	9a192030
$(36,7)$	a4e04042	00000040	00000000	01100010

Table 10.10: Probabilities in $\log_2$ for each round and step.

Round \ Step	0	1	2	3	4	5	6	7
0	-23.85	-23.03	-19.09	-16.19	-15.12	-12.09	-9	-8.03
1	-7.09	-5	-4	-4	-3	-2	-2	-2
2	-1	-1	-1	-2	-3	-4	-4	-3
3	-4.19	-6	-9	-12	-16	-19.42	-19	-23.30
feed-forward	-31.05	-46.09	-69.46	-77.34				

increase the success probability in the first 4 steps to 1. Since one message block in SIMD has 1024 bit and is expanded to 8192, we can at least choose the first 32 expanded message words w, but not completely arbitrary. The message modification for the first 4 steps results in a significant improvement of the overall success probability, since this probability is low in these steps. However, the message expansion needs to be studied in more detail to get a good view on the security of SIMD. It might be possible to improve the attack by using more sophisticated message modification techniques.

10.5.2 The Distinguisher

In this section, we present a distinguisher for the full (32 steps and feed-forward) compression function of SIMD 1.0. It is based on the differential multicollision distinguisher described in Section 10.4.1 and the high probability differential characteristic given in Section 10.5.1 for the compression function of SIMD 1.0.

The differential characteristic described in the Section 10.5.1 can be used to construct a distinguisher for the compression function of SIMD. It is easy to see that by using the differential characteristic q times one can find a differential q-multicollision with a complexity of about $q \cdot 2^{507.34}$ compression function evaluations. Furthermore, by using message modification (see Section 10.5.1) in the first 4 steps the complexity of the attack can be significantly reduced, resulting in a complexity of about $q \cdot 2^{425.28}$. Note that the generic attack has a complexity of about

$$q \cdot 2^{\frac{q-2}{q+2} \cdot 1024}$$

compression function evaluations. Hence, one can distinguish the compression function of SIMD from a random function with a complexity of about $q \cdot 2^{507.34}$ and $q \cdot 2^{425.28}$ for $q = 6$ and $q = 5$, respectively.

In a similar way as we can distinguish the compression function of SIMD from random, we can also distinguish the output transformation (last iteration of SIMD) from random. While the complexity for constructing a differential q-multicollision for the output transformation using the differential characteristic described in the previous section is the same as before, the complexity of the generic attack has changed, since the output is only 512 instead of 1024 bits in the last iteration due to the truncation at the end. Hence, the complexity of the generic attack is

$$q \cdot 2^{\frac{q-2}{q+2} \cdot 512}.$$

However, by setting $q = 438$ and $q = 22$ for the case with message modification in the first 4 rounds, we can distinguish the output transformation of SIMD from random with a complexity of about $438 \cdot 2^{507.34}$ and $22 \cdot 2^{425.28}$, respectively. Table 10.11 provides a summary of the complexities for our distinguisher and the generic complexities.

Table 10.11: Summary of the attack complexities.

	Compression function		*Output transformation*	
Message modification	**Generic**	**Our attack**	**Generic**	**Our attack**
no	$6 \cdot 2^{\frac{4}{8} \cdot 1024}$	$6 \cdot 2^{507.34}$	$438 \cdot 2^{\frac{436}{440} \cdot 512}$	$438 \cdot 2^{507.34}$
yes	$5 \cdot 2^{\frac{3}{7} \cdot 1024}$	$5 \cdot 2^{425.28}$	$22 \cdot 2^{\frac{20}{24} \cdot 512}$	$22 \cdot 2^{425.28}$

10.6 Application to SIMD 1.1

In this section, we will show how to construct a second order differential collision which is used as a distinguishing property for the full permutation (compression function without feed-forward) of SIMD 1.1. For the permutation, the attack strategy described in Section 10.4.2 can be directly applied using an L-characteristic that holds with high probability for the forward and backward direction. In contrast to the attack on SHA-256 [BLMN11], where the second-order collision for the internal block cipher immediately transfers to the compression function, we need to overcome the feed-forward which performs 4 additional steps with the chaining value as message input. In Section 10.6.2 we show how the attack can be extended to the compression function using a weaker attack scenario.

The characteristics for both directions are found as described in Section 10.3. We define for each direction a separate linear code. In order to find the best place to split the steps, we used linear code with different lengths. The generator matrix for the backward direction is given by

$$G^{bw}_{1024\times(1024+k\cdot 256)} := [I_{1024\times 1024}|CV]$$

and the generator matrix for the forward direction is given by

$$G^{fw}_{1024\times(1024+(32-k)\cdot 256)} := [I_{1024\times 1024}|CV]$$

k denote the number of steps in the backward direction. Note that, the permutation has 32 steps.

10.6.1 The Differential Characteristics

We have found several characteristics with low Hamming weight. The best ones are for 18 steps in the backward direction with Hamming weight 72 and 14 step in the forward direction with Hamming weight 52. The complete L-characteristic for the backward direction is given in Table 10.12 and for the forward direction in Table 10.13. To describe the differential characteristics we used signed-bit differences introduced by Wang *et al.* [WY05] in the cryptanalysis of MD5. The advantage of using signed-bit differences is that there exists a unique mapping to both XOR and modular differences.

Table 10.12: Backward characteristic. The state at step -1 is the chaining value.

Step	State	Probability
-1	$\Delta D_0 : -28, \Delta D_3 : -7, \Delta B_5 : -12, \Delta C_5 : +4, \Delta A_6 : +3, \Delta C_6 : +25, \Delta C_7 : +5, \Delta D_7 : -15$	
0	$\Delta A_0 : -19, \Delta A_3 : -30, \Delta C_5 : -12, \Delta D_5 : +4, \Delta B_6 : +6, \Delta D_6 : +25, \Delta D_7 : +5$	$2^{-8.24}$
1	$\Delta B_0 : -10, \Delta B_3 : -21, \Delta D_5 : -12, \Delta C_6 : +6, \Delta A_7 : +22$	2^{-7}
2	$\Delta C_0 : -10, \Delta C_3 : -21, \Delta D_6 : +6, \Delta B_7 : +7$	2^{-5}
3	$\Delta D_0 : -10, \Delta D_3 : -21, \Delta A_6 : +9, \Delta C_7 : +7$	2^{-4}
4	$\Delta A_0 : -1, \Delta B_6 : +12, \Delta D_7 : +7$	2^{-4}
5	$\Delta B_0 : -24, \Delta C_6 : +12$	2^{-3}
6	$\Delta C_0 : -24, \Delta D_6 : +12$	2^{-2}
7	$\Delta D_0 : -24, \Delta A_6 : +15$	2^{-2}
8	$\Delta B_6 : +11$	2^{-2}
9	$\Delta C_6 : +11$	2^{-1}
10	$\Delta D_6 : +11$	2^{-1}
11	$\Delta A_6 : +7$	2^{-1}
12	$\Delta A_1 : +3, \Delta B_6 : +3$	2^{-2}
13	$\Delta B_1 : +22, \Delta A_5 : +22, \Delta C_6 : +3$	2^{-3}
14	$\Delta C_1 : +22, \Delta A_4 : +12, \Delta B_5 : +12, \Delta D_6 : +3$	2^{-4}
15	$\Delta D_1 : +22, \Delta A_2 : +19, \Delta B_4 : +19, \Delta C_5 : +12, \Delta A_6 : +31$	$2^{-5.4}$
16	$\Delta A_0 : +16, \Delta A_1 : +31, \Delta B_2 : +16, \Delta A_4 : +28, \Delta C_4 : +19, \Delta D_5 : +12, \Delta B_6 : +28$	$2^{-7.4}$
17	$\Delta B_0 : +25, \Delta B_1 : +8, \Delta A_2 : +8, \Delta C_2 : +16, \Delta A_3 : +25, \Delta B_4 : +5, \Delta D_4 : +19, \Delta A_5 : +27, \Delta C_6 : +28, \Delta A_7 : +5$	2^{-10}

Table 10.13: Forward characteristic.

Step	State	Probability
17	$\Delta B_0 : +24, \Delta D_0 : +32, \Delta C_2 : -29, \Delta D_3 : +1, \Delta C_4 : -7, \Delta A_6 : -23, \Delta B_6 : +14, \Delta B_7 : +3, \Delta C_7 : -13$	
18	$\Delta A_0 : +5, \Delta C_0 : +24, \Delta D_2 : -29, \Delta D_4 : -7, \Delta B_6 : -6, \Delta C_6 : +14, \Delta C_7 : +3, \Delta D_7 : -13$	2^{-9}
19	$\Delta B_0 : +10, \Delta D_0 : +24, \Delta A_2 : -26, \Delta A_4 : -4, \Delta C_6 : -6, \Delta D_6 : +14, \Delta D_7 : +3$	2^{-8}
20	$\Delta C_0 : +10, \Delta B_2 : -23, \Delta B_4 : -1, \Delta D_6 : -6, \Delta A_7 : +12$	2^{-7}
21	$\Delta D_0 : +10, \Delta C_2 : -23, \Delta C_4 : -1, \Delta B_7 : +21$	2^{-5}
22	$\Delta A_0 : +15, \Delta D_2 : -23, \Delta D_4 : -1, \Delta C_7 : +21$	2^{-4}
23	$\Delta B_0 : +20, \Delta A_4 : -30, \Delta D_7 : +21$	2^{-4}
24	$\Delta C_0 : +20, \Delta B_4 : -2$	2^{-3}
25	$\Delta D_0 : +20, \Delta C_4 : -2$	2^{-2}
26	$\Delta A_0 : +13, \Delta D_4 : -2$	2^{-2}
27	$\Delta B_0 : +6$	2^{-2}
28	$\Delta C_0 : +6$	2^{-1}
29	$\Delta D_0 : +6$	2^{-1}
30	$\Delta A_0 : +31$	$2^{-1.4}$
31	$\Delta B_0 : +24, \Delta A_3 : +24$	2^{-2}

To estimate the success probability we proceed as described in Section 10.3.5. Table 10.14 summarizes the overall probability of each characteristic.

Table 10.14: Summary of the success probabilities.

Characteristic	Hamming weight	Probability
backward	72	$2^{-72.04}$
forward	52	$2^{-51.4}$

Independency of the Characteristics

The assumption of independent characteristics is quite strong (cf. [Mur11]). Nevertheless, one can check this property easily for few steps in both directions, which was done for the presented characteristics. Furthermore, the used characteristics have a low Hamming weight, which makes it very unlikely that they interfere with each other.

10.6.2 Extending the Attack to the Compression Function

In contrast to SHA-2 it is not easy to extend the second-order differential collision to the compression function since the feed-forward of SIMD is non-linear. However, the first step of the feed-forward is almost linear and therefore we can show non-random properties in the output of the state variables D_i for $i = 0, \ldots, 7$.

In the feed-forward 4 additional steps with the initial value as message input are performed. This destroys the distinguishing property at the output of the permutation. However, the values of D_i^{36} for $i = 0, \ldots, 7$ (output of the feed-forward) are determined already in the first step of the feed-forward and not modified in the other three steps. By considering only D_i^{36} for $i = 0, \ldots, 7$ and accordingly only A_i^0 for $i = 0, \ldots, 7$ of the initial value the attack complexity is only slightly increased. Consequently, the dimension of the input and output space for the distinguisher is reduced to 256 bits ($8 \cdot 32$). However, by fixing the differences in the rectangle in the middle of the second-order differential characteristic one can construct a distinguisher for the compression function.

Distinguisher for the Compression Function

For the feed-forward of SIMD we extend the scheme shown in Figure 10.2 to the one shown in Figure 10.3. The function F_2 takes two inputs, namely the state of the last step and the chaining value. As mentioned before we consider only A_i^0 in the initial value and D_i^{36} at the output which is denoted by the quartets $\{P_{A_i}, P^*_{A_i}, Q_{A_i}, Q^*_{A_i}\}$ and $\{\tilde{R}_{D_i}, \tilde{R^*}_{D_i}, \tilde{S}_{D_i}, \tilde{S^*}_{D_i}\}$, respectively.

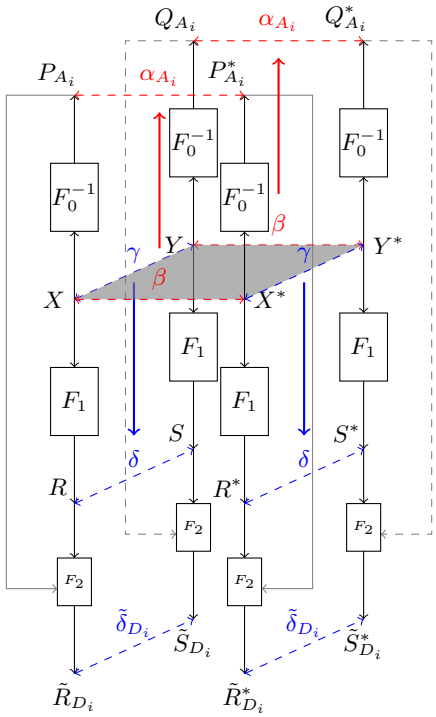


Figure 10.3: Extending the attack to the compression function.

So far the inputs X, β and γ were unrelated. Due to the way we build the second-order collisions, we can see that they are the inputs to a rectangle, hence they are related in the middle of the rectangle (gray layer in Figure 10.3). Therefore, we can extend the attacks by fixing β and γ, since the complexity of the generic case for this type of attacks is 2^n (or 2^t) [BNR11]. Since we show non-randomness only in part of the output, namely D_i for $i = 0, \ldots, 7$, the generic complexity of the attack becomes $2^t = 2^{8 \cdot 32} = 2^{256}$. Hence, by using the second-order differential characteristic from Section 10.6.1 one can construct a distinguisher for the compression function of SIMD 1.1. Note that the distinguisher becomes even more powerful if the attacker can find several of the above quartets with the same difference.

To summarize, the algorithm works as follows:

1. Use the differential from Section 10.6.1
2. Choose a random value for X and compute $X^* = X + \beta$, $Y = X + \gamma$, and $Y^* = X^* + \gamma$.
3. Compute backward from X, X^*, Y, Y^* using F_0^{-1} to obtain $P_{A_i}, P^*_{A_i}, Q_{A_i}, Q^*_{A_i}$.
4. Compute forward from X, X^*, Y, Y^* using F_1 and F_2 to obtain

$\tilde{R}_{D_i}, \tilde{R^*}_{D_i}, \tilde{S}_{D_i}, \tilde{S^*}_{D_i}$.

5. Check if $P^*_{A_i} - P_{A_i} = Q^*_{A_i} - Q_{A_i}$ and $\tilde{S}_{D_i} - \tilde{R}_{D_i} = \tilde{S^*}_{D_i} - \tilde{R^*}_{D_i}$ and therefore $P^*_{A_i} - P_{A_i} - Q^*_{A_i} + Q_{A_i} + \tilde{S}_{D_i} - \tilde{R}_{D_i} - \tilde{S^*}_{D_i} + \tilde{R^*}_{D_i} = 0$ is fulfilled.

10.6.3 Complexity of the Attacks

Distinguisher for the Permuation

As described in Section 10.4.2 the generic complexity for the attack is $2^{n/3}$. For the SIMD 1.1 compression function n is 1024 bits. Hence, the generic complexity is $\approx 2^{342}$. The total complexity of the attack based on the presented characteristic is $(2^{72.04} \cdot 2^{51.4})^2 \approx 2^{247}$ which can be improved by ignoring conditions at the end. As was already observed by Wang *et al.* [WYY05b] in the cryptanalysis of SHA-1 conditions resulting from the modular addition in the last steps of the differential characteristic can be ignored, due to the fact that carries can be ignored since the modular difference at the output stays the same. This reduces the complexity by a factor $2^{8.24}$ in the backward direction and 2^2 in the forward direction which improves the overall complexity by a factor of $2^{2 \cdot 10.24}$ resulting in $2^{226.52}$.

Hence, one can distinguish the permutation of SIMD 1.1 from a random function with a complexity of about $2^{226.52}$ compared to the generic complexity of 2^{342}.

Distinguisher for the Compression Function

As mentioned before the attack complexity is increased slightly by the feed-forward. In fact using the backward and forward characteristics from Table 10.12 and Table 10.13 the additional costs are negligible. In backward direction we have at the end only a difference in ΔA_6^{-1} which needs to be considered. This difference is rotated to the left by s bits. In the forward direction we have differences in ΔB_0^{31} and ΔA_3^{31}. Both are input to the Boolean IF function. Blocking each difference at the input of the IF function costs 2^2 for both differences. Additionally, ΔA_3^{31} is used to compute ΔA_6^{32} in the following way:

$$\begin{aligned} \Delta A_6^{32} =& (\Delta D_6^{31} + \Delta A_6^{-1} + IF(\Delta A_6^{31}, \Delta B_6^{31}, \Delta C_6^{31})) \lll s^{32} \\ &+ (\Delta A_3^{31} \lll r^{32}) \end{aligned} \tag{10.14}$$

In Equation (10.14) only ΔA_6^{-1} and ΔA_3^{31} have differences. Only the rotation to the left by s^{32} bits adds a complexity about 2^1. Finally, we can ignore the costs of the last three steps in the backward ($2^{8.24+7+5}$) and forward ($2^{1+1.4+2}$) direction since we only consider the state variables A_i and D_i for $i = 0, \ldots, 7$ respectively. The differences in these variables do not change in the last three steps. Therefore, the total complexity is $(2^{72.04-20.24} \cdot 2^{51.4-4.4})^2 \cdot 2^1 \cdot 2^2 \approx 2^{200.6}$.

Hence, one can distinguish the compression function of SIMD 1.1 from a random function with a complexity of about $2^{200.6}$. Note that the generic complexity for this attack is 2^{256}.

10.7 Summary

We presented two distinguishers for the compression function of SIMD-512. In our first attack (see Section 10.5), we constructed a L-characteristic that hold with high probability for the compression function of SIMD 1.0. We have found several such L-characteristics with weight 486. Our attack strategy for the distinguisher is similar to the multicollision distinguisher introduced by Biryukov et al.. By using the characteristic with the highest success probability, we showed how to construct a distinguisher, which complexity is below the generic bound. Even with a still conservative probability estimation, we are able to distinguish the compression function from random with a complexity of $2^{427.60}$ compression function calls. Including the output transformation the complexities are still below the generic bound, i.e. we can distinguish the output transformation of SIMD from random with a complexity of about $2^{429.74}$ compression function calls.

Due to this attack, the designers tweaked SIMD. However, in Section 10.6 we presented a distinguisher for the full permutation of SIMD 1.1 by an application of the boomerang attack on hash functions. Starting from the middle of the compression function we used the technique from Section 9.2 to find two differential characteristics, one for the backward direction and one for the forward direction, which hold with high probability. Then we constructed a second-order differential and define a distinguishing property such that we can distinguish the permutation from a random permutation with a complexity of $2^{226.52}$. Furthermore, we extended the attack to the full compression function of SIMD 1.1. By fixing the differences in the rectangle we can distinguish the output of the compression function from a random function with a complexity of $2^{200.6}$ compression function evaluations. This is a significant improvement to the current best known distinguisher with complexity 2^{398} [YW11].

Even if we do not attack the whole hash function, we showed unexpected properties for the SIMD-512 compression function. However, our attacks do not invalidate the security claims of the designers, since most of the security comes from the message expansion. In [BFL10] the designers presented a more detailed analysis of SIMD regarding differential characteristics without differences in the message and are claiming that such characteristics do not affect the security of the SIMD hash function. Nevertheless, non-randomness of the compression function of SIMD affects the applicability of the proofs for the mode of operation build upon it.

Beside the results on SIMD, we showed how boomerang attacks can be effectively used on compression functions, even with a more complex feed-forward. Furthermore, with the successful application of the technique described in Section 9.2 on SIMD, we showed that even new designs can be vulnerable to such

well-known techniques. Though, the increased complexity in the design makes the need for automated tools apparent.

11 Application to HAS-160

In this chapter, we focus on the hash function HAS-160. It is standardized by the Korean government (TTAS.KO-12.0011/R1) [Tel08] and hence widely used in Korea. It is an iterated cryptographic hash function that produces a 160-bit hash value. The design of HAS-160 is similar to SHA-1 and MD5. We combine the techniques presented in Section 9.2 and Section 9.3 to construct a semi-free start collision for 65 (out of 80) steps of HAS-160 with practical complexity. The basic idea of our attack is similar to the attack on a DES based hash function by Rijmen and Preneel [RP94] and to the recent attack on the SHA-3 candidate Skein by Yu et al. [YCKW11]. The idea is to construct a long differential characteristic by connecting two short ones with a complex third characteristic. In our attack, we construct two L-characteristics using the coding theory approach from Section 9.2 and connect them by a NL-characteristics using the technique and algorithm from Section 9.3. The construction of these characteristics is done automated using our set of tools. Furthermore, we present an actual colliding message pair and IV fulfilling all conditions of the differential characteristics. This is so far the best attack in terms of number of steps on HAS-160 with practical complexity. The results of this chapter have been published in [MNS11a].

11.1 Related Work

In [YSP+05], Yun et al. applied the techniques invented by Wang et al. in the cryptanalysis of MD5 and SHA-1 to the HAS-160 hash function. They show that a collision can be found for HAS-160 reduced to 45 steps with a complexity of about 2^{12}. This attack was later extended by Cho et al. [CPSY06] to HAS-160 reduced to 53 steps. The attack has a complexity of about 2^{55} 53-step HAS-160

computations. Mendel and Rijmen [MR07] improved the attack and reduced the complexity to 2^{35} and presented an actual colliding message pair for HAS-160 reduced to 53 steps. Furthermore, they presented a theoretical attack on 59 steps. Finally, preimage attacks on 52 steps by Sasaki and Aoki [SA08] and on 68 steps by Hong et al. [HKS09] have been presented. Both attacks have only theoretical complexity and are only slightly faster than the generic attack which has complexity 2^{160}.

11.2 Description of HAS-160

HAS-160 is an iterative hash function that processes 512-bit input message blocks, operates on 32-bit words and produces a 160-bit hash value. The design of HAS-160 is similar to the design principles of MD5 and SHA-1. In the following, we briefly describe the hash function. It basically consists of two parts: message expansion and state update transformation. A detailed description of the HAS-160 hash function is given in [Tel08].

Message Expansion

The message expansion of HAS-160 is a permutation of 20 expanded message words W_i in each round. The 20 expanded message words W_i used in each round are constructed from the 16 input message words m_i as shown in Table 11.1.

Table 11.1: Message expansion of HAS-160.

	Round 1	Round 2	Round 3	Round 4
W_0	m_0	m_0	m_0	m_0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
W_{15}	m_{15}	m_{15}	m_{15}	m_{15}
W_{16}	$W_0 \oplus W_1 \oplus W_2 \oplus W_3$	$W_3 \oplus W_6 \oplus W_9 \oplus W_{12}$	$W_{12} \oplus W_5 \oplus W_{14} \oplus W_7$	$W_7 \oplus W_2 \oplus W_{13} \oplus W_8$
W_{17}	$W_4 \oplus W_5 \oplus W_6 \oplus W_7$	$W_{15} \oplus W_2 \oplus W_5 \oplus W_8$	$W_0 \oplus W_9 \oplus W_2 \oplus W_{11}$	$W_3 \oplus W_{14} \oplus W_9 \oplus W_4$
W_{18}	$W_8 \oplus W_9 \oplus W_{10} \oplus W_{11}$	$W_{11} \oplus W_{14} \oplus W_1 \oplus W_4$	$W_4 \oplus W_{13} \oplus W_6 \oplus W_{15}$	$W_{15} \oplus W_{10} \oplus W_5 \oplus W_0$
W_{19}	$W_{12} \oplus W_{13} \oplus W_{14} \oplus W_{15}$	$W_7 \oplus W_{10} \oplus W_{13} \oplus W_0$	$W_8 \oplus W_1 \oplus W_{10} \oplus W_3$	$W_{11} \oplus W_6 \oplus W_1 \oplus W_{12}$

For the ordering of the expanded message words W_i the permutation in Table 11.2 is used.

Table 11.2: Permutation of the message words.

step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Round 1	18	0	1	2	3	19	4	5	6	7	16	8	9	10	11	17	12	13	14	15
Round 2	18	3	6	9	12	19	15	2	5	8	16	11	14	1	4	17	7	10	13	0
Round 3	18	12	5	14	7	19	0	9	2	11	16	4	13	6	15	17	8	1	10	3
Round 4	18	7	2	13	8	19	3	14	9	4	16	15	10	5	0	17	11	6	1	12

State Update Transformation

The state update transformation of HAS-160 starts from a (fixed) initial value IV of five 32-bit registers and updates them in 4 rounds of 20 steps each. Figure 11.1

shows one step of the state update transformation of the hash function.

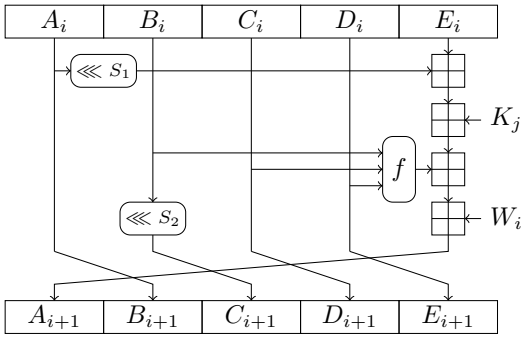


Figure 11.1: The step function of HAS-160.

Note that the function f is different in each round: f_0 is used in the first round, f_1 is used in round 2 and round 4, and f_2 is used in round 3.

$$\begin{aligned} f_0(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ f_1(x,y,z) &= x \oplus y \oplus z \\ f_2(x,y,z) &= (x \vee \neg z) \oplus y \end{aligned}$$

A step constant $K_j \in \{0, \texttt{5a827999}, \texttt{6ed9eba1}, \texttt{8f1bbcdc}\}$ is added in every step and is different for each round. While rotation value $s_2 \in \{10, 17, 25, 30\}$ is different in each round of the hash function, the rotation value s_1 is different in each step of a round. The rotation value s_1 for each step of a round is given in Table 11.3.

Table 11.3: Permutation of the message words.

step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s_1	5	11	7	15	6	13	8	14	7	12	9	11	8	15	6	12	9	14	5	13

After the last step of the state update transformation, the initial value and the output values of the last step are combined, resulting in the final value of one iteration known as Davies-Meyer hash construction (feed-forward). The feed-forward is a word-wise modular addition of the IV and the output of the state update transformation. The result is the final hash value or the initial value for the next message block.

11.2.1 Alternative Description of HAS-160

As one can see in the description of the step update transformation (see Figure 11.1) only the state variable A_i is updated in each step. The values of the other state variables are defined by A_i. Therefore, we can redefine the state update such that only one state variable is used.

$$\begin{aligned} A_{i+1} = & A_{i-4} \ggg s_2 + A_i \lll s_1 + \\ & f(A_{i-1}, A_{i-2} \ggg s_2, A_{i-3} \ggg s_2) + \\ & K_j + W_i \end{aligned} \tag{11.1}$$

Note that s_2 need to be adapted accordingly if the update uses A's between two rounds. The chaining values are represented by $A_0, A_{-1}, A_{-2}, A_{-3}, A_{-4}$. The reduction of the state size has an significant impact on both search algorithms (for finding L- and NL-characteristics), since it reduces the search space.

11.3 Basic Attack Strategy

In this section, we briefly describe the attack strategy to construct a semi-free-start collision for 65 steps of HAS-160. A similar attack was done on a DES based hash function by Rijmen and Preneel [RP94] and recently on Skein by Yu et al. [YCKW11]. The main idea is to construct a long differential characteristic by connecting two short ones. First, proper differences in the expanded message words need to be chosen, such that they result in two short L-characteristics with low Hamming weight and hence hold with high probability. Second, we connect the two short differential characteristics by a NL-characteristic. This one can have low probability, since we can use message modification to fulfil the conditions. Figure 11.2 illustrates the strategy.

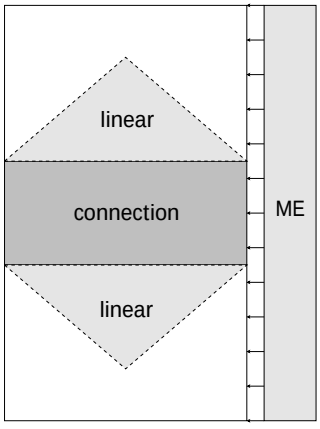


Figure 11.2: Basic attack strategy. Differences occur only in the parts with grey background.

The attack can be summarized as follows:

1. Choose an optimal position for the connection and find two L-characteristics, which hold with high probability.
2. Find a connecting NL-characteristic.
3. Find inputs fulfilling the conditions and use message modification to improve the attack complexity.

To find two good L-characteristics and to determine an optimal position, we use the technique from Section 9.2. Therefore, we define a linearised model of the hash function defining a linear code, construct a generator matrix and use a probabilistic algorithm to find codewords with low Hamming weight.

We are constructing different linear codes for different positions and lengths of the connecting part to determine the optimal choice. Afterwards, we use the search technique from Section 9.3 to find a connecting differential characteristic. Finally, we use message modification, introduced by Wang et al. in [WY05], to find inputs fulfilling all conditions.

11.4 Finding Two Short L-Characteristics

In this section, we show how HAS-160 is linearised in order to be able to construct a linear code. Afterwards, we construct a generator matrix for this code.

11.4.1 Linearisation of HAS-160

Since the message expansion is already linear, only the step update transformation has to be linearised. The non-linear parts of this function are the modular additions and the Boolean functions f_0 and f_2 (f_1 is linear). In the attack, we replace all modular addition by XORs. For the Boolean functions we tried several different approximations. However, the following variant turned out to be the best and is similar as in the attacks on SIMD (see Section 10.3.1). The function f_0 (IF) is replaced by the 0-function, i.e. we block each input difference in f_0. As already shown in Section 10.3.1, this has probability $1/2$ in most cases. One can see that there is exactly one input difference for f_0 where the output difference is always one. In that case we discard the characteristic. f_2 is approximated by its second input which holds with probability higher than $1/2$. In summary we get the following approximation for the Boolean functions:

$$\begin{aligned} f_0'(x,y,z) &= 0 \\ f_2'(x,y,z) &= y \end{aligned}$$

11.4.2 Construction of a Generator Matrix

To construct a generator matrix for the linearised model, we proceed as described in Section 9.2.2. Usually, we first need to add the input of the linearised

model to the codeword. In this case we introduce only differences in the message. Hence, the message input is the only input. However, since the message expansion is linear in HAS-160, the characteristic for the message expansion will always hold with probability one. Therefore, we do not need to include the message in the codeword, reducing the code length even more. Our goal is to find codewords with low Hamming weight, i.e. characteristics with high probability. Therefore, we have to include all intermediate chaining values where differences could decrease the success probability in the linear code. Based on the alternative description of HAS-160 (see Section 11.2.1) we include only A_i in the linear code, since the other state variables do not add any additional information to the code. This decreases the length of the code significantly and therefore also the running time of the search algorithm.

Let $\Delta A_i \in \{0,1\}^{32}$ be the difference vector of the chaining value A_i in bit representation at step i. Then the vector

$$cw := (\Delta A_1, \cdots, \Delta A_n), \tag{11.2}$$

where $cw \in \{0,1\}^{n \cdot 32}$, represents the differences in the chaining value A_i after each step of n steps of HAS-160. cw is one codeword of the linear code and therefore a differential characteristic. To construct the generator matrix for the linear code, we proceed as follows:

1. Compute cw_j with the input difference $\Delta M = e_j$, where $e_j \in \{0,1\}^{512}$ is the j-th unit vector and ΔM the difference of the message block in bit representation.

2. Repeat the computation for $j = 1, \ldots, 512$.

The resulting generator matrix of the linear code representing linearised HAS-160 is defined in the following way:

$$G_{512 \times n \cdot 32} := \begin{pmatrix} cw_1 \\ \vdots \\ cw_{512} \end{pmatrix}. \tag{11.3}$$

Since we are aiming for a collision in the last step, we need to apply code shortening on the last 160 bits, i.e. ensuring that all code words are zero in the last 160 bits (see Section 9.2.2). This reduces the dimension and length of the code to 352 and $(n \cdot 32 - 160)$, respectively.

Using this matrix one can search for low Hamming weight codewords over all n steps. As explained in Section 11.3 we are looking for two short characteristics, which will be connected later. Therefore, we need to modify the linear code to include this requirement.

Modification

The easiest way to define a linear code for both characteristics simultaneously and ensuring that both use the same expanded message, is the following. Firstly,

ignore t steps in the middle. Hence, we change the vector (11.2) to:

$$cw := (\Delta A_1, \cdots, \Delta A_l, \Delta A_{l+t+1}, \cdots, \Delta A_n). \tag{11.4}$$

At the beginning of the second characteristic (after step $l+t$), the state variables can have any difference, since the differences in the steps before are yet undefined. Therefore, we need to add the information to the code that after step $l+t$ all differences are possible. Hence, we add the chaining variables at step $l+t+1$ to the linear code. The construction of the generator matrix changes to:

1. Compute cw_j with the input difference $\Delta M = e_j$, where $e_j \in \{0,1\}^{512}$ is the j-th unit vector and ΔM the difference of the message block in bit representation.
2. Repeat the computation for $j = 1, \ldots, 512$.
3. Compute cw_{512+k} as follows:
   (a) Set $\Delta M = 0$ and $cw_s = e_k$, where $e_k \in \{0,1\}^{160}$ is the k-th unit vector and
   $$cw_s = (\Delta A_{l+t-3}, \Delta A_{l+t-2}, \Delta A_{l+t-1}, \Delta A_{l+t}, \Delta A_{l+t+1}).$$
   (b) Compute ΔA_i for $(l+t+1) < i \leq n$ with cw_s and ΔM as input. Hence, we get following codeword:
   $$cw_{512+k} := (\Delta A_1 = 0, \cdots, \Delta A_l = 0, cw_s, \Delta A_{l+t+2}, \cdots, \Delta A_n).$$
4. Repeat the computation for $k = 1, \ldots, 160$.

Note that $\Delta B_{l+t+1} = \Delta A_{l+t}, \Delta C_{l+t+1} = \Delta A_{l+t-1}, \Delta D_{l+t+1} = \Delta A_{l+t-2}$ and $\Delta E_{l+t+1} = \Delta A_{l+t-3}$ and therefore all possible chaining values after step $l+t$ are included in the code. The resulting generator matrix is

$$G_{672\times(n-t+4)\cdot 32} := \begin{pmatrix} cw_1 \\ \vdots \\ cw_{672} \end{pmatrix}. \tag{11.5}$$

Again code shorting is applied to ensure that all codewords result in a collision after n steps.

Determining l, t and n.

There exist several possible choices for the parameters l, t and n of the linear code. First of all we limit $t \leq 21$. The reason for this is simple. We have 21 words (16 message words and 5 IV words) which can be chosen freely and hence can be used for message modification to fulfil all conditions in the connecting part which is usually the most expensive part of the attack. However, we aimed for a smaller t to reduce the search space for the connecting part as well.

For the search we constructed generator matrices for $21 \leq l \leq (n-21)$ and $t = 21$. If we have found two characteristics with high probability we reduce t.

11.4.3 Searching for Low Hamming Weight Codewords

We used the CodingTool Library (see Section 9.2.4) which contains all tools needed to search for codewords with low Hamming weight. In Table 11.4 we present the best (lowest Hamming weight) characteristics we have found for different parameters. As one can see after 65 steps the Hamming weight is getting too high such that we cannot find a characteristic and conforming inputs with practical complexity.

Table 11.4: Results for the low weight search.

n	l	t	Hamming weight
53	18	21	3
60	18	21	3
65	18	21	3
66	19	21	25
67	18	21	25
68	18	21	72
69	18	21	72
70	18	21	119
75	19	21	123
80	19	21	247

Note that decreasing t always increases the Hamming weight, since more state variables with differences are included in the linear code. Furthermore, the Hamming weights in Table 11.4 include only differences in A. To estimate the probability one has to take the differences in all state variables into account. Therefore, the probability for the linear characteristic can be roughly estimated by four times the Hamming weight of A.

Using this general approach we can cover the whole (linear) search space and allow arbitrary differences in the message words. However, it turned out that the best characteristics we have found are indeed the trivial ones which have only few differences in the message words and only a one bit difference per message word. To describe the differential characteristics we use generalized conditions which are explained in Section 8.3.2. We have found several different characteristics, depending on the choice of l and t. In Table A.1 in Appendix A we present two short characteristics, where t is kept small. To improve readability, we used the alternative description of HAS-160 (see Section 11.2.1)

11.5 Finding Connecting NL-Characteristics

In this section, we apply the technique from Section 9.3 to find a connecting differential characteristic. Note that, this is the most expensive part in our attack.

11.5.1 Determining a Starting Point

In this approach, we first need to determine a starting point for the search algorithm. Therefore, we use the two L-characteristics constructed in the previous section. Table 11.5 shows the starting point of the search algorithm using the notation of generalized conditions leaving only five words unrestricted.

Table 11.5: Steps free of conditions at the beginning of the search algorithm.

step	∇A	∇W
⋮	⋮	⋮
20	x----------x--x---------	--------------------------------
21	????????????????????????????????	--------------------------------
22	????????????????????????????????	--------------------------------
23	????????????????????????????????	--------------------------------
24	????????????????????????????????	--------------------------------
25	????????????????????????????????	x-------------------------------
26	-x-x------x---x-xxx--x-------x--	--------------------------------
⋮	⋮	⋮

Using a small number of unrestricted words reduces the search space and running time of the algorithm significantly. Therefore, we reduced this number by extending the two short L-characteristics linearly. Since there are only few differences at the end of the first L-characteristic and at the beginning of the second L-characteristic, we can extend them forward and backward respectively, without increasing the Hamming weight too much. In fact for the characteristic in Table A.1 in Appendix A, we extended the L-characteristics linearly forward by two and backwards by ten steps.

11.5.2 Search Strategy

Due to the similarities of HAS-160 to SHA-1 using the first search strategy from Section 9.3.2 is already sufficient. However, using our extensions and advanced search strategy, the running time of the algorithm is significantly improved. Using the presented starting point, the algorithm converges already after an hour (on a standard PC) to a complete characteristic for 65 steps, compared to several days using the original search strategy. The chosen parameters are given in Table 11.6.

Table 11.6: Search parameters for HAS-160.

Parameter	**Value**
size limit for $\|C\|$	10
number of two-bit conditions for a bit in U'	3
number of contradictions before a restart from scratch	20.000

Determining the complexity of the probabilistic algorithm in general is still an open problem. Among others it depends on the hash function, search strategy, start characteristic and implementation. The complete characteristic is given

in Table A.1 in Appendix A. Note that with this approach we can find several different characteristics.

Efficient Condition Propagation

For an efficient propagation of new conditions, we use the alternative description of HAS-160 (see Section 11.2.1) and split up one HAS-160 step (including the message expansion) into 3 less complex sub-steps. This way, the propagation of differences can be implemented much more efficiently while losing only a small amount of information. One HAS-160 step is split up in the following way:

$$\begin{aligned} W_i &= ME(M), \\ T_{i+1} &= f(A_{i-1}, A_{i-2} \ggg s_2, A_{i-3} \ggg s_2), \\ A_{i+1} &= A_{i-4} \ggg s_2 + A_i \lll s_1 + T_{i+1} + K_j + W_i. \end{aligned}$$

Note that, for the propagation of differences in T_{i+1} and $f(A_{i-1}, A_{i-2} \ggg s_2, A_{i-3} \ggg s_2)$, a table of all possibilities can be pre-computed (see Section 9.3.4).

11.5.3 Finding a Message Pair

Almost all of the differences in the characteristic of Table A.1 in Appendix A are within 21 steps. Since we can choose up to 21 words (16 message and 5 IV) freely we can use message modification to find efficiently inputs which fulfil all the conditions of the characteristic. The conditions for the characteristic are listed in Table A.2 in Appendix A. The resulting colliding message pair and IV is given in Table 11.7.

Table 11.7: A colliding message pair and IV for HAS-160.

IV	ed3c8ca6 38127dc3 bcf7b374 264eeb2b 73be1247
M	467d7948 3c433177 981f570c 6bf43c12 3dc04b7c cb85a46d 3356206e bff3ea04 9603f6ca 252c37eb 3a1d6197 479ca8d1 badbe3d9 4e23c48c c52a6189 53f1ea06
M'	467d7948 3c433177 981f570c 6bf43c12 3dc04b7c cb85a46d 3356206e bff3ea04 9603f6ca 252c37eb 3a1d6197 479ca8d1 3adbe3d9 4e23c48c 452a6189 53f1ea06
ΔM	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 80000000 00000000 80000000 00000000
h	4b0a28ae bc82dbb1 a4805bfd cd226435 7cb7eb52
h'	4b0a28ae bc82dbb1 a4805bfd cd226435 7cb7eb52

11.6 Summary

The progress in the cryptanalysis of hash functions in the last years shows that the security of existing standards needs to be re-evaluated. Therefore, we analysed in this chapter the Korean hash function standard (TTAS.KO-12.0011/R1) HAS-160. The main idea of our attack is to construct two short

L-characteristics which hold with high probability and connect them by a complex NL-characteristic by using the non-linearity of the state update function. We used techniques from coding theory described in Section 9.2 to search efficiently for the short characteristics and simultaneously determine an optimal position and length of the connecting characteristic. In a second step we used an automatic search algorithm presented in Section 9.3 to find a connecting characteristic taking the non-linearity of the state update into account.

We presented a semi-free-start collision for 65 (out of 80) steps HAS-160 with practical complexity. Extending the attack to more rounds seems to be difficult. One can always extend the size of the connecting part, but this also increases the complexity of finding the connecting characteristic, which running time is hard to estimate. If we limit the length of the connecting part to 21 steps, then the best short characteristics we can find with probability below the generic complexity of a collision attack, are for up to 65 steps.

Although, we only presented a semi-free-start collision, it is a step forward in the analysis of HAS-160. This is so far the best known attack with practical complexity in terms of attacked steps for HAS-160.

12
Application to SHA-256

Since the breakthrough results of Wang et al. [WYY05b, WY05], hash functions have been the target in many cryptanalytic attacks. These attacks have especially shown that several well-known and commonly used algorithms such as MD5 and SHA-1 can no longer be considered to be secure. In fact, practical collisions have been shown for MD5 and collisions for SHA-1 can be constructed with a complexity of about 2^{63} [WYY05a]. For this reason, NIST has proposed the transition from SHA-1 to the SHA-2 family as a first solution. As a consequence, more and more companies and organizations are migrating to SHA-2. Hence, a detailed analysis of this hash function family is needed to get a good view on its security.

Although, the design principles of SHA-2 are very similar to SHA-1, it is still unknown whether or how the attacks on MD5 and SHA-1 can be extended to SHA-2. Since 2008, no collision attacks have been published on SHA-2. One reason might be that the SHA-3 competition [Nat07] initiated by NIST has attracted more attention by the cryptographic community. However, a more likely reason is the increased difficulty of extending previous collision attacks to more steps of SHA-2. In this chapter, we show that apart from a good attack strategy, advanced automated tools are essential to construct differential characteristics and to find conforming message pairs.

Currently, all collision attacks on SHA-2 are of practical complexity and based on the same basic idea: extending a local collision over 9 steps to more steps. As already mentioned in [IMPR08], this kind of attack is unlikely to be extended beyond 24 steps. In this work, we investigate new ideas to progress in the cryptanalysis of SHA-2. First, we extend the idea of finding local collisions to more than 9 steps by exploiting the non-linearity of both the state update and message expansion.

To find such local collisions an automated tool to search for complex differential characteristics is needed. Therefore, we apply the technique described in Section 9.3 to SHA-256. Unfortunately, the approach of De Cannière and Rechberger on SHA-1 cannot directly be applied to SHA-2. We have observed several problems in finding valid differential characteristics for SHA-2. We have identified these problems and show how to solve them efficiently. Most importantly, a very high number of contradicting conditions occurs which render most differential characteristics impossible.

Applying our tool to SHA-256 results in practical examples of semi-free-start collisions for 32 and collisions for 27 out of 64 steps of SHA-256. The best semi-free-start collision and collision attack so far was on 24 steps of SHA-256. The results of this chapter have been published in [MNS11b].

12.1 Related Work

In the past, several attempts have been made to apply the techniques known from the analysis of SHA-1 to SHA-2. The first known cryptanalysis of the SHA-2 family was published by Gilbert and Handschuh [GH03]. They have shown 9-step local collisions which hold with a probability of 2^{-66}. Hawkes et al. [HPR04] have improved these results to get local collisions with a probability of 2^{-39} by considering modular differences.

In [MPRR06a], Mendel et al. have analysed how collision attacks can be applied to step reduced SHA-256. They have shown that the properties of the message expansion of SHA-256 prevent an efficient extension of the techniques of Chabaud and Joux [CJ98] and Wang et al. [WYY05b]. Nevertheless, they presented a collision for 18 steps of SHA-256. In [SS07], Sanadhya and Sarkar have revisited the problem of obtaining a local collision for the SHA-2 family, and in [SS08a] they have shown how to use one of these local collisions to construct another 18-step collision for SHA-256.

Finally, Nikolić and Biryukov [NB08] found a 9-step differential using modular differences which can be used to construct a practical collision for 21 steps and a semi-free-start collision for 23 steps of SHA-256. This was later extended to 22, 23 and 24 steps by Sanadhya and Sarkar in a series of papers [SS08d, SS08b, SS08c]. The best known collision attack on SHA-256 so far was for 24 steps and has been found by Indesteege et al. [IMPR08], and Sanadhya and Sarkar [SS08c].

12.2 Description of SHA-256

SHA-256 is one of four hash functions defined in the Federal Information Processing Standard (FIPS-180-3) [Nat08]. All four hash functions were designed by the National Security Agency (NSA) and issued by NIST in 2002. SHA-256 is an iterated cryptographic hash function with a hash output size of 256 bits, a message block size of 512 bits and using a word size of 32 bits. In the compres-

sion function of SHA-2, a state of eight chaining variables $A,\ldots,H$ is updated using 16 message words $M_0,\ldots,M_{15}$.

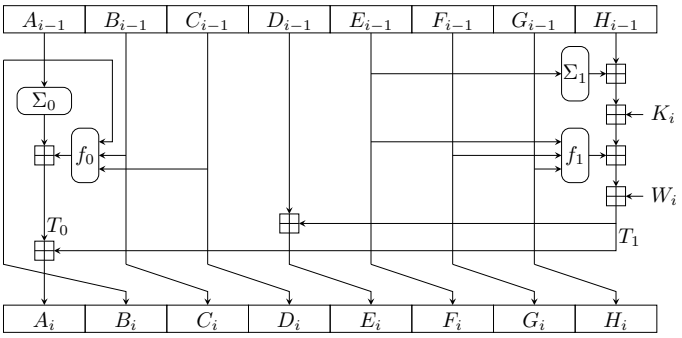


Figure 12.1: The SHA-2 step update function.

The compression function of SHA-256 consists of 64 identical step update functions which are illustrated in Figure 12.1 and given as follows:

$$\begin{array}{lcl} T_0 & = & \Sigma_0(A_{i-1}) + f_0(A_{i-1}, B_{i-1}, C_{i-1}) \\ T_1 & = & \Sigma_1(E_{i-1}) + f_1(E_{i-1}, F_{i-1}, G_{i-1}) + H_{i-1} + K_i + W_i \\ A_i & = & T_0 + T_1\,, \;\; B_i = A_{i-1}\,, \;\; C_i = B_{i-1}\,, \;\; D_i = C_{i-1} \\ E_i & = & D_{i-1} + T_1 \;\; F_i = E_{i-1}\,, \;\; G_i = F_{i-1}\,, \;\; H_i = G_{i-1} \end{array} \tag{12.1}$$

The Boolean functions f_0 (MAJ) and f_1 (IF) are given by

$$\begin{aligned} f_0(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)\,, \\ f_1(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z)\,. \end{aligned}$$

The two $GF(2)$-linear functions Σ_0 and Σ_1 are defined as follows:

$$\begin{aligned} \Sigma_0(x) &= x \ggg 2 \oplus x \ggg 13 \oplus x \ggg 22\,, \\ \Sigma_1(x) &= x \ggg 6 \oplus x \ggg 11 \oplus x \ggg 25\,. \end{aligned}$$

In the i-th step of the update function, a fixed constant K_i and the i-th word W_i of the expanded message are added to the state. The message expansion takes the 16 message words M_i as input and outputs 64 expanded message words W_i as follows:

$$W_i = \begin{cases} M_i & \text{for } 0 \le i < 16 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & \text{for } 16 \le i < 64 \end{cases}$$

where the functions $\sigma_0(x)$ and $\sigma_1(x)$ are defined as follows:

$$\begin{aligned} \sigma_0(x) &= x \ggg 7 \oplus x \ggg 18 \oplus x \gg 3\,, \\ \sigma_1(x) &= x \ggg 17 \oplus x \ggg 19 \oplus x \gg 10\,. \end{aligned}$$

12.3 Basic Attack Strategy

In this section, we give a brief overview of our attack strategy. We first generalize the approach of Nikolić and Biryukov [NB08] to find semi-free-start collisions on a higher number of steps. Due to this extension, differential characteristics cannot be constructed manually or semi-automatic anymore. Hence, we use our fully automated tool to construct complex differential characteristics in SHA-2. Finding valid differential characteristics for SHA-2 is extremely difficult. In fact, we were not able to find a valid differential characteristic without including the search for a confirming message pair in the process. Therefore, the approach of first finding a valid differential characteristic and then, independently search for a conforming message pair does not apply very well to SHA-2. Hence, our attack strategy can be summarized as follows:

1. Determine a starting point for the search which results in an attack on a large number of steps. The resulting start characteristic should span over few steps and only some message words should contain differences.

2. Use the automated search tool described in Section 9.3 with the advanced search strategy to find a differential characteristic and a conforming message pair.

Basically, the advanced search strategy (see Section 9.3.2) searches first for a differential characteristic. After one is found, it continues the search to find a conforming message pair. If no message pair can be found, the differential characteristic is adjusted accordingly.

12.4 Finding NL-Characteristics

We used our automated search tool described in Section 9.3 to construct NL-characteristics and to find conforming message pairs for SHA-256.

12.4.1 Determining a Starting Point

By exploiting the non-linearity of the step update function, Nikolić and Biryukov [NB08] found a 9-step differential characteristic for which it is not necessary to apply corrections (differences in the message words) in each step of the differential characteristic. The fact that not all (only 5 out of 9) message words contain differences helped to overcome several steps of the message expansion resulting in a collision and semi-free-start collision attack for 21 and 23 steps, respectively. Later this approach was extended to a collision attack on 24 steps [IMPR08, SS08c]. However, as pointed out in [IMPR08] it is unlikely that this approach can be extended beyond 24 steps.

In our attack, we construct differential characteristics which span over more than 9 steps. This allows us to attack more steps of SHA-256. As in the attack of Nikolić and Biryukov we are interested in differential characteristics with

differences in only a few message words. Then, large parts of the expanded message have no difference which in turn, results in an attack on more than 24 steps. Already by using a differential characteristic spanning over 10 steps (with differences in only 3 message words) we can construct a semi-free-start collision for 27 steps of SHA-256. This can be extended to 32 steps using a differential characteristic spanning over 16 steps with differences in 8 message words.

To construct these starting points, we first fix the number of steps with differences and consider only differential characteristics which may result in collisions on more than 24 steps. Then, we identify those message words which need to have differences such that the differential characteristic holds for the whole message expansion. Table A.3 and Table A.6 in Appendix A show the used starting points for the attack on 27 and 32 steps. Note that, we have further optimized the message difference slightly to keep it sparse, which reduces the search space for the automated tool.

12.4.2 Search Strategy

We applied the first strategy described in Section 9.3.2 but could not find a valid differential characteristics. In any case at least one of the checks described in Section 9.3.3 failed. The reason for this is that conditions which are not covered by generalized or two-bit conditions appear much more often in SHA-2 than in SHA-1. Since more advanced checks are too expensive, we have developed a more sophisticated search strategy to find valid differential characteristics for SHA-2 which is described in Section 9.3.2. The chosen parameters are given in Table 12.1.

Table 12.1: Search parameters for SHA-256.

Parameter	Value
size limit for $\|C\|$	10
number of two-bit conditions for a bit in U'	3 for bits of A_i 5 for bits of E_i
number of contradictions before a restart from scratch	15.000

This way complex hidden conditions are resolved at an earlier stage in the search. Furthermore, we correct impossible characteristics once they are detected by jumping back to the first phase.

12.4.3 Two-Bit Conditions for SHA-256

The main problem in SHA-2 is that it is difficult to determine whether a differential characteristic is valid, i.e. whether a conforming message pair exists. As mentioned in Section 9.3.2, apart from generalized conditions, additional conditions on more than a single bit are present in a differential characteristic. Especially, conditions on two bits are needed such that a differential characteristic is valid since they may lead to additional inconsistencies. Note that, two-bit

conditions occur in SHA-2 much more often than in SHA-1, mostly caused by the Boolean functions f_i, σ_i and Σ_i but also by the modular additions. Since such contradicting conditions occur only rarely in SHA-1, simple additional checks are sufficient to verify whether a given differential characteristic is valid at the end of the search.

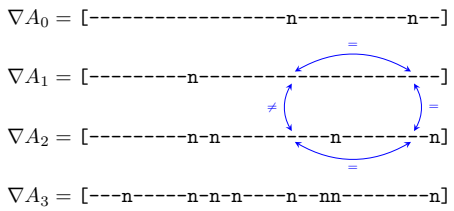


Figure 12.2: Example of four cyclic and contradicting two-bit conditions.

This is not the case in SHA-2 where combinations of the Boolean functions f_0 and f_1 with Σ_0 and Σ_1 can cause even more complex cyclic two-bit conditions. A simple example is given in Figure 12.2. In this case, 4 bits form a cyclic contradicting condition due to Σ_0 and f_0. For the two Σ_0 functions (XOR) we have twice $\Sigma_0(\mathtt{n}, \mathtt{-}, \mathtt{-}) = \mathtt{n}$ which results in the two equalities $A_{1,2} = A_{1,13}$ and $A_{2,2} = A_{2,13}$. For the f_0 function (MAJ) at bit position 2 we get $f_0(\mathtt{-}, \mathtt{-}, \mathtt{n}) = \mathtt{n}$ if and only if $A_{2,2} = A_{1,2}$, while for bit position 13 we get $f_0(\mathtt{-}, \mathtt{-}, \mathtt{n}) = \mathtt{-}$ if and only if $A_{2,13} \neq A_{1,13}$. Note that in this example, more two-bit conditions occur which are not shown. We observed that for a given differential characteristic even more complex relations with cycle lengths larger than 10 commonly occur.

Additionally, more complex conditions on more bits occur. One reason for these additional conditions is that two state variables (A_i, E_i) are updated using a single message word (W_i). Unfortunately, it is not possible to determine all these conditions in general. However, we used different tests to efficiently check for many contradictions (see Section 9.3.3).

12.5 Efficient Condition Propagation in SHA-2

As described in Section 9.3.4, we propagate conditions on bit-level. The larger state size, the combined update of two state variables, and the higher diffusion due to the Σ_i functions increases the complexity of the propagation significantly. To limit the increase in complexity, we take several measures.

12.5.1 Alternative Description of SHA-2

In the case of the SHA-2, one bit of A and E is updated using 15 input bits. Hence, to simplify the update of conditions during the search, we use an alternative description of SHA-2. In the state update transformation of SHA-2, only two state variables are updated in each step, namely A_i and E_i. Therefore, we can redefine the state update such that only these two variables are involved.

In this case, we get the following mapping between the original and new state variables:

A_i	B_i	C_i	D_i	E_i	F_i	G_i	H_i
A_i	A_{i-1}	A_{i-2}	A_{i-3}	E_i	E_{i-1}	E_{i-2}	E_{i-3}

Note that, A_i is updated using an intermediate result of the step update of E_i (see Equation 12.1). Since this complicates the efficient bit representation of the SHA-2 step update transformation we propose the following alternative description:

$$\begin{aligned} E_i &= \quad E_{i-4} + \Sigma_1(E_{i-1}) + f_1(E_{i-1}, E_{i-2}, E_{i-3}) + A_{i-4} + K_i + W_i \\ A_i &= -A_{i-4} + \Sigma_0(A_{i-1}) + f_0(A_{i-1}, A_{i-2}, A_{i-3}) + E_i \end{aligned} \tag{12.2}$$

In this case we get two SHA-1 like state update transformations, one for the left (A_i) and one for the right (E_i) side of the SHA-2 state update transformation. Note that in this description, the state variables $A_{-4}, \ldots, A_{-1}$ and $E_{-4}, \ldots, E_{-1}$ represent the chaining input or initial value of the compression function. The alternative description is also illustrated in Figure 12.3.

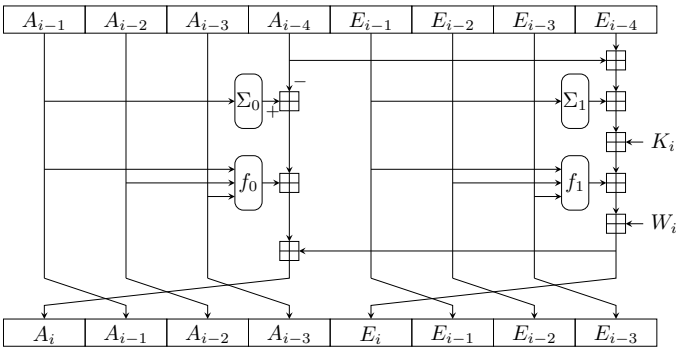


Figure 12.3: Alternative description of the SHA-2 state update transformation.

12.5.2 Split-Up

The complexity of propagating generalized conditions increases exponentially with the number of input bits and additions. While there are only 6 input bits in the case of SHA-1 (excluding the carry), we have 9 input bits in the update of E_i and 8 input bits in the update of each of A_i and W_i in SHA-2.

As already mentioned in Section 9.3.4 to reduce the computational complexity of the propagation, we further split the update of W_i, E_i and A_i into

sub-steps. The split is done in the following way:

$$\begin{aligned}
s_i &= \sigma_0(W_{i-15}),\\
s'_i &= \sigma_1(W_{i-2}),\\
W_i &= s'_i + W_{i-7} + s_i + W_{i-16},\\
F_i &= f_1(E_{i-1}, E_{i-2}, E_{i-3}),\\
S_i &= \Sigma_1(E_{i-1}),\\
E_i &= E_{i-4} + S_i + F_i + A_{i-4} + K_i + W_i,\\
F'_i &= f_0(A_{i-1}, A_{i-2}, A_{i-3}),\\
S'_i &= \Sigma_0(A_{i-1}),\\
A_i &= -A_{i-4} + S'_i + F'_i + E_i.
\end{aligned}$$

We independently compute each output bit of the Boolean functions and afterwards we compute the modular additions. This way, the number of input bits is reduced to 3 for the Boolean functions and we get at most 5 input bits for the modular additions. This split of functions reduces the computation complexity by a factor of about 100.

12.5.3 Using the Cache

For the sub-steps without modular addition we have precomputed the propagation of all generalized input conditions. For the modular additions we use the regular cache (see Section 9.3.4). Our experiments have shown a speed-up of another factor 100 by caching already computed results. Note that, due to memory restrictions we are not able to precompute or keep all possibilities for the modular additions.

12.5.4 Propagation of Conditions for Linear Functions

As described in Section 9.3.4 the propagation of conditions through linear functions is treated differently in our approach. In the following we show why this is important for SHA-256. Let us consider the linear function σ_0 which is defined as follows:

$$\sigma_0(x) = x \ggg 7 \oplus x \ggg 18 \oplus x \gg 3 = y.$$

Updating the conditions for each bit in ∇x and ∇y is done by updating each bitslice $\{\nabla x_{i+7}, \nabla x_{i+18}, \nabla x_{i+3}, \nabla y_i\}$ for $i = 0, \ldots, 28$ and $\{\nabla x_{i+7}, \nabla x_{i+18}, \nabla y_i\}$ for $i = 29, \ldots, 31$ (see Section 9.3.4). Note that, the additions in the indices are modulo 32.

Now consider the following two cases for σ_0.

Case 1: *Forward Propagation*

$$\begin{aligned}
\nabla x &= [\texttt{----------}\ldots\texttt{----------}],\\
\nabla y &= [\texttt{??????????}\ldots\texttt{??????????}].
\end{aligned}$$

All bit-slices are equal to {-,-,-,?} and {-,-,?} respectively. According to the definition of σ_0 the update results in {-,-,-,-} and {-,-,-} respectively. Hence, after updating all conditions, we receive

$$\nabla x = [\texttt{----------------------------------}],$$
$$\nabla y = [\texttt{----------------------------------}].$$

Case 2: *Backward Propagation*

$$\nabla x = [\texttt{??????????????????????????????????}],$$
$$\nabla y = [\texttt{----------------------------------}].$$

All bit-slices are equal to {?,?,?,-} and {?,?,-} respectively. According to the definition of σ_0 the update results in {?,?,?,-} and {?,?,-} respectively. In that case there are no new conditions and we receive

$$\nabla x = [\texttt{??????????????????????????????????}],$$
$$\nabla y = [\texttt{----------------------------------}].$$

The result of the first case is perfectly fine. In the second case no new conditions are detected. Although, we know from the first case that every generalized condition of ∇x should be -. Hence, no information propagates using the bit-slice update approach described in Section 9.3.4. Therefore, we update the conditions for every linear function of SHA-256 (σ_0, σ_1, Σ_0 and Σ_1) according to the method described in Section 9.3.4. Using this method the efficient propagation of linear generalized conditions works in both direction. However, the efficient propagation of non-linear generalized conditions is still an open problem.

12.6 Results

Semi-Free-Start Collision for 32 Steps

Using the start characteristic given in Table A.3 in Appendix A and the advanced search strategy, we can find a valid characteristic and conforming inputs which result in semi-free-start collisions for 32 out of 64 steps of SHA-256. An example of a semi-free-start collision for 32 steps is shown in Table 12.2.
The corresponding differential characteristic and the set of conditions is given in Table A.4 and Table A.5 in Appendix A. To find such an example for 32 steps, our tool needs few days on a cluster with 32 nodes.

Collision for 27 Steps

So far we have only considered semi-free-start collision attacks in which an attacker is allowed to choose the chaining value. However, in a collision attack on the hash function the chaining value is fixed, which makes an attack much more

Table 12.2: Semi-free-start collision for 32 steps of SHA-256.

h_0	764d264f	268a3366	285fecb1	4c389b22	75cd568d	f5c8f99b	6e7a3cc3	1b4ea134
h_0^*	764d264f	268a3366	285fecb1	4c389b22	75cd568d	f5c8f99b	6e7a3cc3	1b4ea134
Δh_0	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
m	52a600a8	2c3b8434	ea92dfcf	d4eaf9ad	b77fe08d	7c50e542	69c783a6	86a14e10
	baf88b0b	12665efb	ce7c3a31	3030f09d	9bd52eb8	7549997e	fa976e0d	86ebacbc
m^*	52a600a8	2c3b8434	ea92dfcb	0cdba38b	f514e39d	7a5bb4cb	ee6bcba6	c58f6a0f
	b2f78b0b	12665efb	ce7c3a31	3030f09d	9bd52eb8	7549997e	fa976e0d	86ebacbc
Δm	00000000	00000000	00000004	d8315a26	426b0310	060b5189	87ac4800	432e241f
	080f0000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
h_1	d0b41ffa	e1f519a2	e3cad2ed	a19d5795	906ac05f	c995f6c8	cf309f95	9fb9ca57
h_1^*	d0b41ffa	e1f519a2	e3cad2ed	a19d5795	906ac05f	c995f6c8	cf309f95	9fb9ca57
Δh_1	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000

difficult. In order to construct a collision for step-reduced SHA-256, we are interested in differential characteristics with no differences in the first few message words. Then, the additional freedom in the first message words can be used to transform a semi-free-start collision into a real collision. Similar characteristics have also been used in the collision attacks on 24 steps of SHA-256 in [IMPR08].

In order to find a characteristics under these requirements, we used a starting point where the characteristic is spanning over 11 steps with differences in only 5 expanded message words and with no differences in the first 7 message words. Such a starting point is presented in Table A.6 in Appendix A. Using this starting point and our tool we are able to construct collisions for 27 steps of SHA-256. A colliding message pair is shown in Table 12.3. The differential characteristic

Table 12.3: Collision for 27 steps of SHA-256.

h_0	6a09e667	bb67ae85	3c6ef372	a54ff53a	510e527f	9b05688c	1f83d9ab	5be0cd19
m	725a0370	0daa9f1b	071d92df	ec8282c1	7913134a	bc2eb291	02d33a84	278dfd29
	0c40f8ea	d8bd68a0	0ce670c5	5ec7155d	9f6407a8	729fbfe8	aa7c7c08	607ae76d
m^*	725a0370	0daa9f1b	071d92df	ec8282c1	7913134a	bc2eb291	02d33a84	27460e6d
	08c8fbea	d8bd68a0	0ce670c5	5ec7155d	9f4425fb	729fbfe8	aa7c7c08	2d32d129
Δm	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00cbf344
	04880300	00000000	00000000	00000000	00202253	00000000	00000000	4d483644
h_1	5864015f	133494fa	fa42bb35	94bc44f9	29eabb36	9e461e33	2eab27f8	106467c9

and the set of conditions is given in Table A.7 and Table A.8 in Appendix A.

12.7 Summary

In this chapter, we applied the technique described in Section 9.3 to SHA-256. Compared to SHA-1 or HAS-160, SHA-256 has a far more complex structure. A direct application of the original technique by De Cannière and Rechberger, is not possible, since several problems occur. Most importantly, found NL-

characteristics are not valid, due to many contradicting conditions in SHA-2. We identified these problems and showed how to overcome them. Important for the successful application of our tool to SHA-2, is the detection of two-bit conditions and the check for contradictions in these conditions. Furthermore, only with our advanced search strategy, we were able to find valid NL-characteristics for SHA-256. This strategy combines the search for differential characteristics with the computation of conforming message pairs.

Finally, we presented a collision for 27 and a semi-free-start collision for 32 steps of SHA-256 with practical complexity. This significantly improves upon the best previously published (semi-free-start) collision attacks on SHA-256 for up to 24 steps.

13 Conclusions

In the second part of this thesis, we focused on different tools used in the cryptanalysis of ARX based hash functions. We reviewed two distinct techniques which are based on differential cryptanalysis, the most common tool in the cryptanalysis of hash functions.

The first technique is based on coding theory, since finding differential characteristics for a linearised model of a hash function is related to the problem of finding codewords with low Hamming weight in a linear code. We showed how a linear code is constructed from a linearised model and presented three probabilistic algorithms searching for low Hamming weight codewords. Furthermore, we presented an open-source toolbox which implements a search algorithm, interfaces, data structures and several other useful methods. Using this library, one can find differential characteristics fully automated.

The second technique is based on the recent attacks on SHA-1. It consists of a search algorithm and the concept of generalized conditions. Generalized conditions reflect that both the differences and the actual values of bits are important for an attack. The search algorithm searches automated for complex differential characteristics. We extended this technique in several aspects. Most importantly, we presented a more efficient search strategy and included two-bit conditions in the search process, hence combining the search for a differential characteristic with the search for conforming message pairs. Furthermore, we showed how the propagation of generalized conditions can be done efficiently. The resulting tools were used to attack the hash functions SIMD-512, HAS-160 and SHA-256.

For the compression function of SIMD-512 we presented two distinguishers. In the first attack, we constructed differential characteristics that hold with high probability for the full compression function of SIMD-512 using the first tech-

nique. We have found several such characteristics with a low weight. Our attack strategy for the distinguisher is similar to the multicollision distinguisher introduced by Biryukov et al.. By using the characteristic with the highest success probability, we showed how to construct a distinguisher, which complexity is below the generic bound. Even with a conservative probability estimation, we are able to distinguish the compression function from random with a complexity of $2^{427.60}$ compression function calls. Furthermore, we can distinguish the compression function with the output transformation of SIMD from random with a complexity of about $2^{429.74}$ compression function calls.

Due to this attack, the designers tweaked SIMD. However, we presented a distinguisher for the full permutation of tweaked SIMD-512 by an application of the boomerang attack on hash functions. Starting from the middle of the compression function we used the same technique to find two differential characteristics which are used in the boomerang attack. Then we constructed a second-order differential and defined a distinguishing property such that we can distinguish the permutation from a random permutation with a complexity of $2^{226.52}$. Furthermore, we extended the attack to the full compression function of tweaked SIMD-512 such that we can distinguish the output of the compression function from a random function with a complexity of $2^{200.6}$ compression function evaluations.

Our attacks do not invalidate the security claims of the designers, since most of the security comes from the message expansion. Nevertheless, non-randomness of the compression function of SIMD effect the applicability of the proofs for the mode of operation build upon it. Beside the results on SIMD, we showed how boomerang like attacks can be effectively used on compression functions, even with a more complex feed-forward. Furthermore, with the successful application of the coding theoretic technique on SIMD, we showed that even new designs can be vulnerable to such well-known techniques. Though, the increased complexity in the design makes the need for automated tools apparent.

For HAS-160, we presented a semi-free-start collision for 65 (out of 80) steps with practical complexity. The main idea of our attack is to combine both of the above techniques by constructing two short characteristics which hold with high probability and connect them by a complex characteristic using the non-linearity of the state update function. Extending the attack to more rounds seems to be difficult. One can always extend the size of the connecting part, but this also increases the complexity of finding the connecting characteristic, which running time is hard to estimate. If we limit the length of the connecting part to 21 steps, then the best short characteristics we can find with probability below the generic complexity of a collision attack, are for up to 65 steps. Although, we only presented a semi-free-start collision, it is a step forward in the analysis of HAS-160 and shows how powerful our automated tools can be. This is so far the best known attack with practical complexity in terms of attacked steps for HAS-160.

In the last chapter, we applied the search algorithm for complex characteristics to SHA-256. Compared to SHA-1 or HAS-160, SHA-256 has a far more

complex structure. A direct application of the original technique is not possible, since several problems occur. We identified these problems and showed how to overcome them. Important for the successful application of our tool to SHA-2, is the detection of two-bit conditions and the check for contradictions in these conditions. Furthermore, only with our advanced search strategy, we were able to find valid complex differential characteristics for SHA-256. Finally, we presented a collision for 27 and a semi-free-start collision for 32 steps of SHA-256 with practical complexity. This significantly improves upon the best previously published (semi-free-start) collision attacks on SHA-256 for up to 24 steps. To summarize, the search for valid differential characteristics and conforming message pairs in SHA-2 is increasingly difficult and unpredictable, compared to more simple ARX-based designs like HAS-160 and SHA-1. Nevertheless, we were able to construct a powerful tool to find practical examples for (semi-free-start) collisions in SHA-256 which can also be applied to other ARX based hash functions.

A

Differential Characteristics and Conditions

Table A.1: Characteristic for 65 steps HAS-160 using generalized conditions. The rows with darkgray background represent the connecting part. The rows with lightgray background represent the two L-characteristics.

i	∇A	∇W
-4	----------	
-3	----------	
-2	----------	
-1	----------	
0	----------	
⋮	⋮	⋮
16	----------	----------
17	u---------	x---------
18	----------u	----------
19	n----------u	x---------
20	u----------u--u---------	----------
21	------n-uuuuuu--u----n----u----	----------
22	u--n---uu-nu---uu---nn--------uu	----------
23	--n-n-nnnu-n-u--nu------nu------	----------
24	uuun-nu--u-u----n-n-unnuuuuuuu-n	----------
25	--n----uu---uu-un-u-----nu-n-n--	x---------
26	-n-n------n---n-uun--u-------n--	----------
27	-unu------u-n---uu---u-n-u-u---n	----------
28	--n---u---u---u--u-n---u-----u-n	----------
29	--------n---u--------n-------u-n	----------
30	--u-----n-u----------u----------	----------
31	--n-------n----------------n-n--	----------
32	-------------n--------------n--	----------
33	----------n----------	x---------
34	----------n----------u--	----------
35	----------u--	----------
36	----------	----------
37	----------	----------
38	----------	----------
39	----------n------	----------
40	----------	----------
41	----------	----------
42	----------	x---------
43	----------	----------
44	----------	x---------
45	----------	----------
⋮	⋮	⋮
65	----------	----------

Table A.2: Set of conditions for the semi-free-start collision for 65 steps of HAS-160.

Step	Set of conditions	#
16	$A_{16,3} = 0$, $A_{16,21} = A_{15,21}$	2
17	$A_{17,3} = 1$, $A_{17,31} = 1$	2
18	$A_{18,9} = 1$, $A_{18,13} = 1$, $A_{18,8} \neq A_{17,8}$	3
19	$A_{19,18} = 1$, $A_{19,31} = 0$, $A_{19,23} \neq A_{17,13}$, $A_{19,27} \neq A_{18,2}$, $A_{19,9} \neq A_{18,31}$, $A_{19,24} = A_{18,31}$	6
20	$A_{20,9} = 1$, $A_{20,12} = 1$, $A_{20,31} = 1$, $A_{20,16} \neq A_{18,6}$, $A_{20,3} = A_{18,25}$, $A_{20,0} \neq A_{19,0}$, $A_{20,1} = A_{19,1}$, $A_{20,2} = A_{19,2}$, $A_{20,3} = A_{19,3}$, $A_{20,4} = A_{19,4}$, $A_{20,5} = A_{19,5}$, $A_{20,23} \neq A_{19,6}$, $A_{20,7} = A_{19,7}$, $A_{20,19} = A_{19,19}$, $A_{20,24} = A_{19,24}$, $A_{20,29} \neq A_{19,29}$	16
21	$A_{21,4} = 1$, $A_{21,9} = 0$, $A_{21,14} = 1$, $A_{21,17} = 1$, $A_{21,18} = 1$, $A_{21,19} = 1$, $A_{21,20} = 1$, $A_{21,21} = 1$, $A_{21,22} = 1$, $A_{21,24} = 0$, $A_{21,26} \neq A_{19,9}$, $A_{21,29} = A_{19,12}$, $A_{21,3} \neq A_{20,3}$, $A_{21,6} \neq A_{20,6}$, $A_{21,7} \neq A_{20,7}$, $A_{21,11} \neq A_{20,11}$, $A_{21,15} \neq A_{20,15}$, $A_{21,16} \neq A_{20,16}$, $A_{21,3} \neq A_{20,18}$, $A_{21,25} \neq A_{20,25}$, $A_{21,26} = A_{20,26}$, $A_{21,30} = A_{20,30}$	22
22	$A_{22,0} = 1$, $A_{22,1} = 1$, $A_{22,10} = 0$, $A_{22,11} = 0$, $A_{22,15} = 1$, $A_{22,16} = 1$, $A_{22,20} = 1$, $A_{22,21} = 0$, $A_{22,23} = 1$, $A_{22,24} = 1$, $A_{22,28} = 0$, $A_{22,31} = 1$, $A_{22,2} = A_{20,17}$, $A_{22,3} = A_{20,18}$, $A_{22,4} \neq A_{20,19}$, $A_{22,5} \neq A_{20,20}$, $A_{22,6} = A_{20,21}$, $A_{22,7} \neq A_{20,22}$, $A_{22,9} = A_{20,24}$, $A_{22,3} \neq A_{21,3}$, $A_{22,5} = A_{21,5}$, $A_{22,6} \neq A_{21,6}$, $A_{22,7} = A_{21,7}$, $A_{22,8} = A_{21,8}$, $A_{22,12} \neq A_{21,12}$, $A_{22,29} \neq A_{21,12}$, $A_{22,29} = A_{21,29}$, $A_{22,30} = A_{21,30}$	28
23	$A_{23,6} = 1$, $A_{23,7} = 0$, $A_{23,14} = 1$, $A_{23,15} = 0$, $A_{23,18} = 1$, $A_{23,20} = 0$, $A_{23,22} = 1$, $A_{23,23} = 0$, $A_{23,24} = 0$, $A_{23,25} = 0$, $A_{23,27} = 0$, $A_{23,29} = 0$, $A_{23,17} = A_{21,0}$, $A_{23,28} \neq A_{21,11}$, $A_{23,0} \neq A_{21,15}$, $A_{23,1} \neq A_{21,16}$, $A_{23,8} = A_{21,23}$, $A_{23,13} = A_{21,28}$, $A_{23,16} = A_{21,31}$, $A_{23,3} = A_{22,3}$, $A_{23,21} \neq A_{22,4}$, $A_{23,5} = A_{22,5}$, $A_{23,8} = A_{22,8}$, $A_{23,9} \neq A_{22,9}$, $A_{23,26} = A_{22,9}$, $A_{23,12} \neq A_{22,12}$, $A_{23,13} \neq A_{22,13}$, $A_{23,2} = A_{22,17}$, $A_{23,17} \neq A_{22,17}$, $A_{23,3} = A_{22,18}$, $A_{23,4} \neq A_{22,19}$, $A_{23,19} \neq A_{22,19}$, $A_{23,26} \neq A_{22,26}$, $A_{23,30} \neq A_{22,30}$	34
24	$A_{24,0} = 0$, $A_{24,2} = 1$, $A_{24,3} = 1$, $A_{24,4} = 1$, $A_{24,5} = 1$, $A_{24,6} = 1$, $A_{24,7} = 1$, $A_{24,8} = 1$, $A_{24,9} = 0$, $A_{24,10} = 0$, $A_{24,11} = 1$, $A_{24,13} = 0$, $A_{24,15} = 0$, $A_{24,20} = 1$, $A_{24,22} = 1$, $A_{24,25} = 1$, $A_{24,26} = 0$, $A_{24,28} = 0$, $A_{24,29} = 1$, $A_{24,30} = 1$, $A_{24,31} = 1$, $A_{24,23} = A_{22,6}$, $A_{24,24} \neq A_{22,7}$, $A_{24,12} = A_{22,27}$, $A_{24,14} = A_{22,29}$, $A_{24,17} \neq A_{23,0}$, $A_{24,1} \neq A_{23,1}$, $A_{24,18} \neq A_{23,1}$, $A_{24,27} = A_{23,10}$, $A_{24,12} \neq A_{23,12}$, $A_{24,1} = A_{23,16}$, $A_{24,17} \neq A_{23,17}$, $A_{24,19} = A_{23,19}$, $A_{24,21} = A_{23,21}$, $A_{24,16} \neq A_{23,31}$	35
25	$A_{25,2} = 0$, $A_{25,4} = 0$, $A_{25,6} = 1$, $A_{25,7} = 0$, $A_{25,13} = 1$, $A_{25,15} = 0$, $A_{25,16} = 1$, $A_{25,18} = 1$, $A_{25,19} = 1$, $A_{25,23} = 1$, $A_{25,24} = 1$, $A_{25,29} = 0$, $A_{25,17} \neq A_{23,0}$, $A_{25,20} = A_{23,3}$, $A_{25,21} = A_{23,4}$, $A_{25,22} \neq A_{23,5}$, $A_{25,25} \neq A_{23,8}$, $A_{25,26} \neq A_{23,9}$, $A_{25,27} = A_{23,10}$, $A_{25,28} = A_{23,11}$, $A_{25,30} = A_{23,13}$, $A_{25,11} = A_{23,26}$, $A_{25,17} = A_{24,17}$, $A_{25,3} = A_{24,18}$, $A_{25,8} = A_{24,23}$, $A_{25,9} = A_{24,24}$, $A_{25,12} = A_{24,27}$	27
26	$A_{26,2} = 0$, $A_{26,10} = 1$, $A_{26,13} = 0$, $A_{26,14} = 1$, $A_{26,15} = 1$, $A_{26,17} = 0$, $A_{26,21} = 0$, $A_{26,28} = 0$, $A_{26,30} = 0$, $A_{26,1} = A_{24,16}$, $A_{26,3} = A_{24,18}$, $A_{26,4} \neq A_{24,19}$, $A_{26,8} = A_{24,23}$, $A_{26,9} = A_{24,24}$, $A_{26,20} \neq A_{25,3}$, $A_{26,22} \neq A_{25,5}$, $A_{26,25} \neq A_{25,8}$, $A_{26,26} = A_{25,9}$, $A_{26,27} = A_{25,10}$, $A_{26,11} = A_{25,11}$, $A_{26,12} = A_{25,12}$, $A_{26,5} = A_{25,20}$, $A_{26,7} = A_{25,22}$, $A_{26,25} \neq A_{25,25}$, $A_{26,11} = A_{25,26}$, $A_{26,16} \neq A_{25,31}$	26
27	$A_{27,0} = 0$, $A_{27,4} = 1$, $A_{27,6} = 1$, $A_{27,8} = 0$, $A_{27,10} = 1$, $A_{27,14} = 1$, $A_{27,15} = 1$, $A_{27,19} = 0$, $A_{27,21} = 1$, $A_{27,28} = 1$, $A_{27,29} = 0$, $A_{27,30} = 1$, $A_{27,27} \neq A_{25,10}$, $A_{27,2} = A_{25,17}$, $A_{27,13} = A_{25,28}$, $A_{27,23} \neq A_{26,6}$, $A_{27,24} = A_{26,7}$, $A_{27,12} \neq A_{26,12}$, $A_{27,1} \neq A_{26,16}$, $A_{27,3} \neq A_{26,18}$, $A_{27,23} = A_{26,23}$, $A_{27,9} = A_{26,24}$, $A_{27,27} \neq A_{26,27}$	23
28	$A_{28,0} = 0$, $A_{28,2} = 1$, $A_{28,8} = 1$, $A_{28,12} = 0$, $A_{28,14} = 1$, $A_{28,17} = 1$, $A_{28,21} = 1$, $A_{28,25} = 1$, $A_{28,29} = 0$, $A_{28,23} = A_{26,6}$, $A_{28,4} = A_{26,19}$, $A_{28,19} \neq A_{27,2}$, $A_{28,30} \neq A_{27,13}$	13
29	$A_{29,0} = 0$, $A_{29,2} = 1$, $A_{29,10} = 0$, $A_{29,19} = 1$, $A_{29,23} = 0$, $A_{29,29} = A_{27,12}$, $A_{29,4} \neq A_{28,4}$, $A_{29,21} \neq A_{28,4}$, $A_{29,6} = A_{28,6}$, $A_{29,27} \neq A_{28,10}$, $A_{29,4} = A_{28,19}$, $A_{29,13} = A_{28,28}$, $A_{29,15} = A_{28,30}$	13
30	$A_{30,10} = 1$, $A_{30,21} = 1$, $A_{30,23} = 0$, $A_{30,29} = 1$, $A_{30,27} = A_{28,10}$, $A_{30,4} = A_{28,19}$, $A_{30,8} \neq A_{28,23}$, $A_{30,4} = A_{29,4}$, $A_{30,25} = A_{29,8}$, $A_{30,12} \neq A_{29,12}$, $A_{30,2} \neq A_{29,17}$, $A_{30,17} = A_{29,17}$, $A_{30,6} \neq A_{29,21}$, $A_{30,14} \neq A_{29,29}$	14
31	$A_{31,2} = 0$, $A_{31,4} = 0$, $A_{31,21} = 0$, $A_{31,29} = 0$, $A_{31,6} \neq A_{29,21}$, $A_{31,14} = A_{29,29}$, $A_{31,0} \neq A_{30,0}$, $A_{31,17} \neq A_{30,0}$, $A_{31,19} \neq A_{30,2}$, $A_{31,17} \neq A_{30,17}$	10
32	$A_{32,2} = 0$, $A_{32,17} = 0$, $A_{32,19} = A_{30,2}$, $A_{32,21} = A_{30,4}$, $A_{32,27} = A_{31,10}$, $A_{32,8} \neq A_{31,23}$	6
33	$A_{33,21} = 0$, $A_{33,2} \neq A_{31,17}$, $A_{33,4} \neq A_{32,4}$, $A_{33,6} = A_{32,21}$, $A_{33,14} = A_{32,29}$	5
34	$A_{34,2} = 1$, $A_{34,21} = 0$, $A_{34,6} \neq A_{32,21}$, $A_{34,19} \neq A_{33,2}$, $A_{34,17} = A_{33,17}$	5
35	$A_{35,2} = 1$, $A_{35,19} = A_{33,2}$	2
36	$A_{36,6} \neq A_{35,21}$	1
37	$A_{37,21} = 0$, $A_{37,19} \neq A_{36,2}$	2
39	$A_{39,6} = 0$	1
41	$A_{41,31} = 1$	1

Table A.3: Starting point for a semi-free-start collision for 32 steps. Using the alternative description of SHA-2 (Section 12.5.1) and the notion of generalized conditions (Section 8.3.2).

i	∇A_i	∇E_i	∇W_i
-4	----------	----------	
-3	----------	----------	
-2	----------	----------	
-1	----------	----------	
0	----------	----------	----------
1	----------	----------	----------
2	----------x--	----------x--	----------x--
3	??????????????????????????????	??????????????????????????????	??????????????????????????????
4	??????????????????????????????	??????????????????????????????	??????????????????????????????
5	??????????????????????????????	??????????????????????????????	??????????????????????????????
6	??????????????????????????????	??????????????????????????????	??????????????????????????????
7	????????????????----------	??????????????????????????????	??????????????????????????????
8	????????????????----------	??????????????????????????????	????????????????x----------
9	????????????????x----------	??????????????????????????????	----------
10	----------	??????????????????????????????	----------
11	----------	????????????????----------	----------
12	----------	????????????????----------	----------
13	----------	????????????????x----------	----------
14	----------	----------	----------
15	----------	----------	----------
16	----------	----------	----------
17	----------	----------	----x-------x----------
18	----------	----------	----------
19	----------	----------	----------
	...	...	...
30	----------	----------	----------
31	----------	----------	----------

Table A.4: Characteristic for a semi-free-start collision for 32 steps of SHA-256.

i	∇A_i	∇E_i	∇W_i
-4	----------	----------	
-3	----------	----------	
-2	----------	----------	
-1	----------	----------	
0	----------	----------0--	----------
1	----------1-----	--0-0---1--1----1-0-0-------011-	----------
2	----------0--u--	--1-1-1000-1--11101101---1--1u0-	----------u--
3	10n10nnn1n0n-11n1u01u11000uu0n0n	-1n1n10un0un101-n1n1n0110un0u0n0	uu-un-----un---n-u-uu-n---u--un-
4	-----n----------0---------0-1---	-0n0n1nuuun0-1u1unnnuu011n000nn1	1n---1u--uu1u-uu------nn--0n----
5	----------n---------1-----	0u1nn1n-1010-00001u0101-11101110	01-1-un0-1-1n-nn1u1n0-0un-0-n--n
6	----------n--u--------u---n--	00u01un0000000n111u00100101uu11u	n----nnuu-n-nu---n--n----------
7	----------	-n10u000u1un0101nn10n00001n000u1	1n0001un10u0nnn-01n01u10000unnnn
8	----------	-10-1n0-0--1-01-0-1-0----n011-10	----u-------unnn----------0-----
9	----u----------u----------	-0--u00-1-01-1--1---1----n1---0-	----------
10	----------	---nunn------n--n--------u-u-u--	----------
11	----------	---0-10----100--0--------1-0-0--	----------
12	----------	---0011----011--1--------0-1-1--	----------
13	----------	---un------unnnn----------------	----------
14	----------	---00------00000----------------	----------
15	----------	---11------11111----------------	----------
16	----------	----------	----------
17	----------	----------	----n----------n----------
18	----------	----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	----------	----------	----------
23	----------	----------	----------
24	----------	----------	----------
25	----------	----------	----------
26	----------	----------	----------
27	----------	----------	----------
28	----------	----------	----------
29	----------	----------	----------
30	----------	----------	----------
31	----------	----------	----------

Table A.5: Set of conditions for the semi-free-start collision for 32 steps of SHA-256.

Step	Set of conditions	#
0	$E_{0,2} = 0$	1
1	$A_{1,5} = 1$, $A_{1,2} \neq A_{0,2}$, $A_{1,0} = E_{1,0}$, $E_{1,1} = 1$, $E_{1,2} = 1$, $E_{1,3} = 0$, $E_{1,11} = 0$, $E_{1,13} = 0$, $E_{1,15} = 1$, $E_{1,20} = 1$, $E_{1,23} = 1$, $E_{1,27} = 0$, $E_{1,29} = 0$	13
2	$A_{2,2} = 1$, $A_{2,5} = 0$, $A_{2,0} = A_{1,0}$, $A_{2,4} \neq A_{1,4}$, $A_{2,11} = A_{1,11}$, $A_{2,14} \neq A_{1,14}$, $A_{2,16} = A_{1,16}$, $A_{2,20} = A_{1,20}$, $A_{2,22} = A_{1,22}$, $A_{2,24} \neq A_{1,24}$, $A_{2,25} \neq A_{1,25}$, $A_{2,26} \neq A_{1,26}$, $A_{2,29} \neq A_{1,29}$, $A_{2,23} = A_{2,11}$, $A_{2,22} = A_{2,13}$, $A_{2,25} \neq A_{2,14}$, $E_{2,1} = 0$, $E_{2,2} = 1$, $E_{2,3} = 1$, $E_{2,6} = 1$, $E_{2,10} = 1$, $E_{2,11} = 0$, $E_{2,12} = 1$, $E_{2,13} = 1$, $E_{2,14} = 0$, $E_{2,15} = 1$, $E_{2,16} = 1$, $E_{2,17} = 1$, $E_{2,20} = 1$, $E_{2,22} = 0$, $E_{2,23} = 0$, $E_{2,24} = 0$, $E_{2,25} = 1$, $E_{2,27} = 1$, $E_{2,29} = 1$, $E_{2,5} \neq E_{1,5}$, $E_{2,21} = E_{2,7}$	40
	$W_{2,2} = 1$, $W_{2,30} \neq W_{2,13}$, $W_{2,23} \neq W_{2,19}$	
3	$A_{3,0} = 0$, $A_{3,1} = 0$, $A_{3,2} = 0$, $A_{3,3} = 0$, $A_{3,4} = 1$, $A_{3,5} = 1$, $A_{3,6} = 0$, $A_{3,7} = 0$, $A_{3,8} = 0$, $A_{3,9} = 1$, $A_{3,10} = 1$, $A_{3,11} = 1$, $A_{3,12} = 1$, $A_{3,13} = 0$, $A_{3,14} = 1$, $A_{3,15} = 1$, $A_{3,16} = 0$, $A_{3,17} = 1$, $A_{3,18} = 1$, $A_{3,20} = 0$, $A_{3,21} = 0$, $A_{3,22} = 0$, $A_{3,23} = 1$, $A_{3,24} = 0$, $A_{3,25} = 0$, $A_{3,26} = 0$, $A_{3,27} = 0$, $A_{3,28} = 1$, $A_{3,29} = 0$, $A_{3,30} = 0$, $A_{3,31} = 1$, $E_{3,0} = 0$, $E_{3,1} = 0$, $E_{3,2} = 0$, $E_{3,3} = 1$, $E_{3,4} = 0$, $E_{3,5} = 0$, $E_{3,6} = 1$, $E_{3,7} = 0$, $E_{3,8} = 1$, $E_{3,9} = 1$, $E_{3,10} = 0$, $E_{3,11} = 0$, $E_{3,12} = 1$, $E_{3,13} = 0$, $E_{3,14} = 1$, $E_{3,15} = 0$, $E_{3,17} = 1$, $E_{3,18} = 0$, $E_{3,19} = 1$, $E_{3,20} = 0$, $E_{3,21} = 1$, $E_{3,22} = 0$, $E_{3,23} = 0$, $E_{3,24} = 1$, $E_{3,25} = 0$, $E_{3,26} = 1$, $E_{3,27} = 0$, $E_{3,28} = 1$, $E_{3,29} = 0$, $E_{3,30} = 1$	92
	$W_{3,1} = 0$, $W_{3,2} = 1$, $W_{3,5} = 1$, $W_{3,9} = 0$, $W_{3,11} = 1$, $W_{3,12} = 1$, $W_{3,14} = 1$, $W_{3,16} = 0$, $W_{3,20} = 0$, $W_{3,21} = 1$, $W_{3,27} = 0$, $W_{3,28} = 1$, $W_{3,30} = 1$, $W_{3,31} = 1$, $W_{3,17} = W_{3,0}$, $W_{3,24} \neq W_{3,3}$, $W_{3,25} = W_{3,4}$, $W_{3,10} = W_{3,6}$, $W_{3,23} \neq W_{3,6}$, $W_{3,22} = W_{3,7}$, $W_{3,24} \neq W_{3,7}$, $W_{3,23} = W_{3,8}$, $W_{3,25} = W_{3,10}$, $W_{3,17} = W_{3,13}$, $W_{3,24} \neq W_{3,13}$, $W_{3,19} = W_{3,15}$, $W_{3,26} = W_{3,15}$, $W_{3,22} \neq W_{3,18}$, $W_{3,29} = W_{3,18}$, $W_{3,23} = W_{3,19}$, $W_{3,26} = W_{3,22}$	
4	$A_{4,3} = 1$, $A_{4,5} = 0$, $A_{4,15} = 0$, $A_{4,26} = 0$, $A_{4,0} = A_{2,0}$, $A_{4,4} \neq A_{2,4}$, $A_{4,11} = A_{2,11}$, $A_{4,14} \neq A_{2,14}$, $A_{4,16} \neq A_{2,16}$, $A_{4,20} = A_{2,20}$, $A_{4,22} = A_{2,22}$, $A_{4,24} \neq A_{2,24}$, $A_{4,29} \neq A_{2,29}$, $A_{4,17} = A_{4,6}$, $E_{4,0} = 1$, $E_{4,1} = 0$, $E_{4,2} = 0$, $E_{4,3} = 0$, $E_{4,4} = 0$, $E_{4,5} = 0$, $E_{4,6} = 0$, $E_{4,7} = 1$, $E_{4,8} = 1$, $E_{4,9} = 0$, $E_{4,10} = 1$, $E_{4,11} = 1$, $E_{4,12} = 0$, $E_{4,13} = 0$, $E_{4,14} = 0$, $E_{4,15} = 1$, $E_{4,16} = 1$, $E_{4,17} = 1$, $E_{4,18} = 1$, $E_{4,20} = 0$, $E_{4,21} = 0$, $E_{4,22} = 1$, $E_{4,23} = 1$, $E_{4,24} = 1$, $E_{4,25} = 0$, $E_{4,26} = 1$, $E_{4,27} = 0$, $E_{4,28} = 0$, $E_{4,29} = 0$, $E_{4,30} = 0$, $E_{4,19} \neq A_{3,19}$	67
	$W_{4,4} = 0$, $W_{4,5} = 0$, $W_{4,8} = 0$, $W_{4,9} = 0$, $W_{4,16} = 1$, $W_{4,17} = 1$, $W_{4,19} = 1$, $W_{4,20} = 1$, $W_{4,21} = 1$, $W_{4,22} = 1$, $W_{4,25} = 1$, $W_{4,26} = 1$, $W_{4,30} = 0$, $W_{4,31} = 1$, $W_{4,18} \neq W_{4,1}$, $W_{4,13} = W_{4,2}$, $W_{4,23} \neq W_{4,2}$, $W_{4,10} = W_{4,6}$, $W_{4,11} \neq W_{4,7}$, $W_{4,23} = W_{4,12}$, $W_{4,27} = W_{4,12}$, $W_{4,28} = W_{4,13}$	
5	$A_{5,5} = 1$, $A_{5,15} = 0$, $A_{5,0} \neq A_{4,0}$, $A_{5,2} \neq A_{4,2}$, $A_{5,4} \neq A_{4,4}$, $A_{5,6} \neq A_{4,6}$, $A_{5,11} = A_{4,11}$, $A_{5,14} = A_{4,14}$, $A_{5,16} \neq A_{4,16}$, $A_{5,18} = A_{4,18}$, $A_{5,20} = A_{4,20}$, $A_{5,22} = A_{4,22}$, $A_{5,24} = A_{4,24}$, $A_{5,25} = A_{4,25}$, $A_{5,29} \neq A_{4,29}$, $A_{5,26} = A_{5,3}$, $A_{5,24} = A_{5,4}$, $A_{5,27} \neq A_{5,6}$, $E_{5,0} = 0$, $E_{5,1} = 1$, $E_{5,2} = 1$, $E_{5,3} = 1$, $E_{5,4} = 0$, $E_{5,5} = 1$, $E_{5,6} = 1$, $E_{5,7} = 1$, $E_{5,9} = 1$, $E_{5,10} = 0$, $E_{5,11} = 1$, $E_{5,12} = 0$, $E_{5,13} = 1$, $E_{5,14} = 1$, $E_{5,15} = 0$, $E_{5,16} = 0$, $E_{5,17} = 0$, $E_{5,18} = 0$, $E_{5,20} = 0$, $E_{5,21} = 1$, $E_{5,22} = 0$, $E_{5,23} = 1$, $E_{5,25} = 0$, $E_{5,26} = 1$, $E_{5,27} = 0$, $E_{5,28} = 0$, $E_{5,29} = 1$, $E_{5,30} = 1$, $E_{5,31} = 0$, $E_{1,0} = A_{1,0}$	74
	$W_{5,0} = 0$, $W_{5,3} = 0$, $W_{5,5} = 0$, $W_{5,7} = 0$, $W_{5,8} = 1$, $W_{5,9} = 0$, $W_{5,11} = 0$, $W_{5,12} = 0$, $W_{5,13} = 1$, $W_{5,14} = 1$, $W_{5,15} = 1$, $W_{5,16} = 0$, $W_{5,17} = 0$, $W_{5,19} = 0$, $W_{5,20} = 1$, $W_{5,22} = 1$, $W_{5,24} = 0$, $W_{5,25} = 0$, $W_{5,26} = 1$, $W_{5,28} = 1$, $W_{5,30} = 1$, $W_{5,31} = 0$, $W_{5,29} = W_{5,1}$, $W_{5,23} = W_{5,2}$, $W_{5,21} = W_{5,4}$, $W_{5,29} \neq W_{5,18}$	
6	$A_{6,2} = 0$, $A_{6,6} = 1$, $A_{6,15} = 1$, $A_{6,18} = 0$, $A_{6,26} = A_{5,26}$, $A_{6,26} = A_{6,3}$, $A_{6,24} \neq A_{6,4}$, $A_{6,27} \neq A_{6,7}$, $A_{6,30} \neq A_{6,9}$, $A_{6,23} \neq A_{6,11}$, $A_{6,22} \neq A_{6,13}$, $A_{6,25} = A_{6,14}$, $A_{6,26} \neq A_{6,17}$, $E_{6,0} = 1$, $E_{6,1} = 1$, $E_{6,2} = 1$, $E_{6,3} = 1$, $E_{6,4} = 1$, $E_{6,5} = 1$, $E_{6,6} = 0$, $E_{6,7} = 1$, $E_{6,8} = 0$, $E_{6,9} = 0$, $E_{6,10} = 1$, $E_{6,11} = 0$, $E_{6,12} = 0$, $E_{6,13} = 1$, $E_{6,14} = 1$, $E_{6,15} = 1$, $E_{6,16} = 1$, $E_{6,17} = 0$, $E_{6,18} = 0$, $E_{6,19} = 0$, $E_{6,20} = 0$, $E_{6,21} = 0$, $E_{6,22} = 0$, $E_{6,23} = 0$, $E_{6,24} = 0$, $E_{6,25} = 0$, $E_{6,26} = 1$, $E_{6,27} = 1$, $E_{6,28} = 0$, $E_{6,29} = 1$, $E_{6,30} = 0$, $E_{6,31} = 0$,	73
	$W_{6,11} = 0$, $W_{6,14} = 0$, $W_{6,18} = 1$, $W_{6,19} = 0$, $W_{6,21} = 0$, $W_{6,23} = 1$, $W_{6,24} = 1$, $W_{6,25} = 0$, $W_{6,26} = 0$, $W_{6,31} = 0$, $W_{6,17} \neq W_{6,0}$, $W_{6,28} = W_{6,0}$, $W_{6,22} = W_{6,1}$, $W_{6,7} \neq W_{6,3}$, $W_{6,20} = W_{6,3}$, $W_{6,8} \neq W_{6,4}$, $W_{6,22} = W_{6,5}$, $W_{6,10} = W_{6,6}$, $W_{6,27} \neq W_{6,6}$, $W_{6,22} = W_{6,7}$, $W_{6,28} \neq W_{6,7}$, $W_{6,12} \neq W_{6,8}$, $W_{6,29} = W_{6,8}$, $W_{6,13} \neq W_{6,9}$, $W_{6,30} = W_{6,9}$, $W_{6,27} \neq W_{6,10}$, $W_{6,30} = W_{6,15}$, $W_{6,20} \neq W_{6,16}$	
7	$A_{7,2} = A_{5,2}$, $A_{7,6} \neq A_{5,6}$, $A_{7,18} = A_{5,18}$, $E_{7,0} = 1$, $E_{7,1} = 1$, $E_{7,2} = 0$, $E_{7,3} = 0$, $E_{7,4} = 0$, $E_{7,5} = 0$, $E_{7,6} = 1$, $E_{7,7} = 0$, $E_{7,8} = 0$, $E_{7,9} = 0$, $E_{7,10} = 0$, $E_{7,11} = 0$, $E_{7,12} = 0$, $E_{7,13} = 1$, $E_{7,14} = 0$, $E_{7,15} = 0$, $E_{7,16} = 1$, $E_{7,17} = 0$, $E_{7,18} = 1$, $E_{7,19} = 0$, $E_{7,20} = 0$, $E_{7,21} = 1$, $E_{7,22} = 1$, $E_{7,23} = 1$, $E_{7,24} = 0$, $E_{7,25} = 0$, $E_{7,26} = 0$, $E_{7,27} = 1$, $E_{7,28} = 0$, $E_{7,29} = 1$, $E_{7,30} = 0$	66
	$W_{7,0} = 0$, $W_{7,1} = 0$, $W_{7,2} = 0$, $W_{7,3} = 0$, $W_{7,4} = 1$, $W_{7,5} = 0$, $W_{7,6} = 0$, $W_{7,7} = 0$, $W_{7,8} = 0$, $W_{7,9} = 1$, $W_{7,10} = 1$, $W_{7,11} = 1$, $W_{7,12} = 0$, $W_{7,13} = 0$, $W_{7,14} = 1$, $W_{7,15} = 0$, $W_{7,17} = 0$, $W_{7,18} = 0$, $W_{7,19} = 0$, $W_{7,20} = 0$, $W_{7,21} = 1$, $W_{7,22} = 0$, $W_{7,23} = 1$, $W_{7,24} = 0$, $W_{7,25} = 1$, $W_{7,26} = 1$, $W_{7,27} = 0$, $W_{7,28} = 0$, $W_{7,29} = 0$, $W_{7,30} = 0$, $W_{7,31} = 1$, $W_{7,16} \neq E_{3,16}$	
8	$A_{8,2} = A_{7,2}$, $A_{8,6} \neq A_{7,6}$, $A_{8,15} \neq A_{7,15}$, $A_{8,16} \neq A_{7,16}$, $A_{8,18} = A_{7,18}$, $A_{8,27} \neq A_{7,27}$, $E_{8,0} = 0$, $E_{8,1} = 1$, $E_{8,3} = 1$, $E_{8,4} = 1$, $E_{8,5} = 0$, $E_{8,6} = 0$, $E_{8,11} = 0$, $E_{8,13} = 1$, $E_{8,15} = 0$, $E_{8,17} = 1$, $E_{8,18} = 0$, $E_{8,20} = 1$, $E_{8,23} = 0$, $E_{8,25} = 0$, $E_{8,26} = 0$, $E_{8,27} = 1$, $E_{8,29} = 0$, $E_{8,30} = 1$, $E_{8,12} = E_{8,7}$, $E_{8,21} \neq E_{8,8}$,	46
	$W_{8,5} = 0$, $W_{8,16} = 0$, $W_{8,17} = 0$, $W_{8,18} = 0$, $W_{8,19} = 1$, $W_{8,27} = 1$, $W_{8,0} \neq A_{4,0}$, $W_{8,1} = A_{4,1}$, $W_{8,4} \neq A_{4,4}$, $W_{8,21} = W_{8,0}$, $W_{8,22} = W_{8,1}$, $W_{8,23} \neq W_{8,2}$, $W_{8,7} \neq W_{8,3}$, $W_{8,8} \neq W_{8,4}$, $W_{8,23} \neq W_{8,6}$, $W_{8,31} \neq W_{8,10}$, $W_{8,28} \neq W_{8,13}$, $W_{8,29} \neq W_{8,14}$, $W_{8,30} \neq W_{8,15}$, $W_{8,31} = W_{8,20}$	
9	$A_{9,16} = 1$, $A_{9,27} = 1$, $A_{9,25} = A_{9,5}$, $A_{9,15} = A_{9,6}$, $A_{9,18} \neq A_{9,7}$, $A_{9,28} = A_{9,7}$, $E_{9,1} = 0$, $E_{9,5} = 1$, $E_{9,6} = 0$, $E_{9,11} = 1$, $E_{9,15} = 1$, $E_{9,18} = 1$, $E_{9,20} = 1$, $E_{9,21} = 0$, $E_{9,23} = 1$, $E_{9,25} = 0$, $E_{9,26} = 0$, $E_{9,27} = 1$, $E_{9,30} = 0$, $E_{9,14} \neq E_{9,0}$, $E_{9,13} \neq E_{9,8}$, $E_{9,22} = E_{9,9}$	22
10	$A_{10,16} = A_{8,16}$, $A_{10,27} = A_{8,27}$, $E_{10,2} = 1$, $E_{10,4} = 1$, $E_{10,6} = 1$, $E_{10,15} = 0$, $E_{10,18} = 0$, $E_{10,25} = 0$, $E_{10,26} = 0$, $E_{10,27} = 1$, $E_{10,28} = 0$, $E_{10,13} \neq E_{10,0}$, $E_{10,14} \neq E_{10,0}$, $E_{10,23} = E_{10,5}$, $E_{10,20} = E_{10,7}$, $E_{10,21} \neq E_{10,8}$, $E_{10,22} \neq E_{10,9}$, $E_{10,23} = E_{10,9}$, $E_{10,23} = E_{10,10}$, $E_{10,29} = E_{10,10}$, $E_{10,30} \neq E_{10,12}$, $E_{10,31} \neq E_{10,13}$, $E_{10,29} \neq E_{10,16}$, $E_{10,22} \neq E_{10,17}$, $E_{10,24} = E_{10,19}$	25
11	$A_{11,16} = A_{10,16}$, $A_{11,27} = A_{10,27}$, $E_{11,2} = 0$, $E_{11,4} = 0$, $E_{11,6} = 1$, $E_{11,15} = 0$, $E_{11,18} = 0$, $E_{11,19} = 0$, $E_{11,20} = 1$, $E_{11,25} = 0$, $E_{11,26} = 1$, $E_{11,28} = 0$	12
12	$E_{12,2} = 1$, $E_{12,4} = 1$, $E_{12,6} = 0$, $E_{12,15} = 1$, $E_{12,18} = 1$, $E_{12,19} = 1$, $E_{12,20} = 0$, $E_{12,25} = 1$, $E_{12,26} = 1$, $E_{12,27} = 0$, $E_{12,28} = 0$, $E_{12,16} \neq E_{11,16}$	12
13	$E_{13,16} = 0$, $E_{13,17} = 0$, $E_{13,18} = 0$, $E_{13,19} = 0$, $E_{13,20} = 1$, $E_{13,27} = 0$, $E_{13,28} = 1$, $E_{13,13} \neq E_{13,0}$, $E_{13,14} = E_{13,0}$, $E_{13,6} = E_{13,1}$, $E_{13,14} \neq E_{13,1}$, $E_{13,15} \neq E_{13,1}$, $E_{13,15} = E_{13,2}$, $E_{13,29} = E_{13,2}$, $E_{13,21} \neq E_{13,3}$, $E_{13,30} \neq E_{13,3}$, $E_{13,22} \neq E_{13,4}$, $E_{13,31} \neq E_{13,4}$, $E_{13,23} \neq E_{13,5}$, $E_{13,24} \neq E_{13,6}$, $E_{13,25} = E_{13,7}$, $E_{13,13} \neq E_{13,8}$, $E_{13,14} = E_{13,9}$, $E_{13,30} = E_{13,11}$, $E_{13,31} \neq E_{13,12}$	25
14	$E_{14,16} = 0$, $E_{14,17} = 0$, $E_{14,18} = 0$, $E_{14,19} = 0$, $E_{14,20} = 0$, $E_{14,27} = 0$, $E_{14,28} = 0$	7
15	$E_{15,16} = 1$, $E_{15,17} = 1$, $E_{15,18} = 1$, $E_{15,19} = 1$, $E_{15,20} = 1$, $E_{15,27} = 1$, $E_{15,28} = 1$	7
17	$W_{17,16} = 0$, $W_{17,27} = 0$, $W_{17,14} \neq W_{4,15}$, $W_{17,4} = W_{17,2}$, $W_{17,29} \neq W_{17,20}$	5

Table A.6: Starting point for a collision for 27 steps of SHA-256.

i	∇A_i	∇E_i	∇W_i
-4	---------- ----------	----------	
-3	----------	----------	
-2	----------	----------	
-1	----------	----------	
0	----------	----------	----------
1	----------	----------	----------
2	----------	----------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	----------	----------
6	----------	----------	----------
7	????????????????????????????????	????????????????????????????????	????????????????????????????x??
8	????????????????????????????????	????????????????????????????????	????????????????????????????????
9	????????????????????????????????	????????????????????????????????	----------
10	----------	????????????????????????????????	----------
11	----------	????????????????????????????????	----------
12	----------	????????????????????????????????	????????????????????????????????
13	----------	????????????????????????????????	----------
14	----------	----------	----------
15	----------	----------	????????????????????????????????
16	----------	----------	----------
17	----------	----------	????????????????????????????????
18	----------	----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	----------	----------	----------
23	----------	----------	----------
24	----------	----------	----------
25	----------	----------	----------
26	----------	----------	----------

Table A.7: Characteristic for a collision for 27 steps of SHA-256.

i	∇A_i	∇E_i	∇W_i
-4	----------	----------	
-3	----------	----------	
-2	----------	----------	
-1	----------	----------	
0	----------	----------	----------
1	----------	----------	----------
2	----------	----------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	----------1--------1------	----------
6	----------	-1--------0--0-10-1----0-0------	----------
7	-------unn--u------n---nn-uuuu--	101-11---u10u1-0nuu-uuuu1n---n0-	00---1--un-0u-nuuuuu1-nu0n101n--
8	nnnnn-nnnn--------nuu----------	0n0n001001u-1u1n01un010n01n00110	-----u--n---n---------nn--------
9	----un--n--nu-------nu-u-------	-1n1n1011u011100nn100u10-10000u-	----------
10	----------	u00000nuuu10uun01u00n00n110-u-u1	----------
11	----------	0n000uuuuu01010111n-uun01n000n01	----------
12	----------	01---1010u01u----111-010-0--110-	------110-u-------n0--u--n-n--nn
13	----------	01-10u1nunuuu---1110-1nn11---01-	----------
14	----------	-----1-01011----------00--------	----------
15	----------	-----1-001000---------11--------	0u1-nn-n-u-1u---11un0uu10u101u0-
16	----------	----------	----------
17	----------	----------	---0-1nnn---u-1-----10uu0-------
18	----------	----------	----------
19	----------	----------	----------
20	----------	----------	----------
21	----------	----------	----------
22	----------	----------	----------
23	----------	----------	----------
24	----------	----------	----------
25	----------	----------	----------
26	----------	----------	----------

Table A.8: Set of conditions for the collision for 27 steps of SHA-256.

Step	Set of conditions	#
5	$E_{5,6} = 1$, $E_{5,15} = 1$	2
6	$A_{6,2} = A_{5,2}$, $A_{6,3} \neq A_{5,3}$, $A_{6,7} = A_{5,7}$, $A_{6,8} \neq A_{5,8}$, $A_{6,19} \neq A_{5,19}$, $A_{6,22} \neq A_{5,22}$, $A_{6,23} = A_{5,23}$, $E_{6,6} = 0$, $E_{6,8} = 0$, $E_{6,13} = 1$, $E_{6,15} = 0$, $E_{6,16} = 1$, $E_{6,18} = 0$, $E_{6,21} = 0$, $E_{6,30} = 1$, $E_{6,2} = E_{5,2}$, $E_{6,9} \neq E_{5,9}$, $E_{6,14} = E_{5,14}$	18
7	$A_{7,2} = 1$, $A_{7,3} = 1$, $A_{7,4} = 1$, $A_{7,5} = 1$, $A_{7,7} = 0$, $A_{7,8} = 0$, $A_{7,12} = 0$, $A_{7,19} = 1$, $A_{7,22} = 0$, $A_{7,23} = 0$, $A_{7,24} = 1$, $A_{7,10} \neq A_{7,1}$, $A_{7,11} \neq A_{6,11}$, $A_{7,25} \neq A_{6,25}$, $A_{7,31} \neq A_{7,10}$, $A_{7,31} \neq A_{7,11}$, $A_{7,25} = A_{7,14}$, $A_{7,26} = A_{7,15}$, $A_{7,27} = A_{7,16}$, $A_{7,28} = A_{7,16}$, $A_{7,28} = A_{7,17}$, $A_{7,29} = A_{7,17}$, $A_{7,31} \neq A_{7,20}$, $E_{7,1} = 0$, $E_{7,2} = 0$, $E_{7,6} = 0$, $E_{7,7} = 1$, $E_{7,8} = 1$, $E_{7,9} = 1$, $E_{7,10} = 1$, $E_{7,11} = 1$, $E_{7,13} = 1$, $E_{7,14} = 1$, $E_{7,15} = 0$, $E_{7,16} = 0$, $E_{7,18} = 1$, $E_{7,19} = 1$, $E_{7,20} = 0$, $E_{7,21} = 1$, $E_{7,22} = 1$, $E_{7,26} = 1$, $E_{7,27} = 1$, $E_{7,29} = 1$, $E_{7,30} = 0$, $E_{7,31} = 1$, $E_{7,5} = E_{6,5}$, $E_{7,12} = E_{6,12}$, $E_{7,28} = E_{6,28}$, $E_{7,5} = E_{7,0}$, $E_{7,23} = E_{7,4}$, $E_{7,28} \neq E_{7,23}$, $W_{7,2} = 0$, $W_{7,3} = 1$, $W_{7,4} = 0$, $W_{7,5} = 1$, $W_{7,6} = 0$, $W_{7,7} = 0$, $W_{7,8} = 1$, $W_{7,9} = 0$, $W_{7,11} = 1$, $W_{7,12} = 1$, $W_{7,13} = 1$, $W_{7,14} = 1$, $W_{7,15} = 1$, $W_{7,16} = 1$, $W_{7,17} = 0$, $W_{7,19} = 1$, $W_{7,20} = 0$, $W_{7,22} = 0$, $W_{7,23} = 1$, $W_{7,26} = 1$, $W_{7,30} = 0$, $W_{7,31} = 0$, $W_{7,21} \neq W_{7,0}$, $W_{7,18} \neq W_{7,1}$, $W_{7,29} \neq W_{7,1}$, $W_{7,21} \neq W_{7,10}$, $W_{7,25} = W_{7,10}$, $W_{7,29} = W_{7,18}$, $W_{7,29} = W_{7,25}$	80
8	$A_{8,11} = 1$, $A_{8,12} = 1$, $A_{8,13} = 0$, $A_{8,22} = 0$, $A_{8,23} = 0$, $A_{8,24} = 0$, $A_{8,25} = 0$, $A_{8,27} = 0$, $A_{8,28} = 0$, $A_{8,29} = 0$, $A_{8,30} = 0$, $A_{8,31} = 0$, $A_{8,10} \neq W_{12,10}$, $A_{8,14} \neq W_{12,14}$, $A_{8,26} \neq W_{12,26}$, $A_{8,19} \neq A_{6,19}$, $A_{8,10} \neq A_{7,10}$, $A_{8,20} = A_{7,20}$, $A_{8,26} \neq A_{7,26}$, $A_{8,10} = A_{8,1}$, $A_{8,16} \neq A_{8,4}$, $A_{8,17} = A_{8,5}$, $A_{8,15} \neq A_{8,6}$, $A_{8,18} = A_{8,6}$, $A_{8,18} \neq A_{8,7}$, $A_{8,19} = A_{8,7}$, $A_{8,20} \neq A_{8,8}$, $E_{8,0} = 0$, $E_{8,1} = 1$, $E_{8,2} = 1$, $E_{8,3} = 0$, $E_{8,4} = 0$, $E_{8,5} = 0$, $E_{8,6} = 1$, $E_{8,7} = 0$, $E_{8,8} = 0$, $E_{8,9} = 0$, $E_{8,10} = 1$, $E_{8,11} = 0$, $E_{8,12} = 0$, $E_{8,13} = 1$, $E_{8,14} = 1$, $E_{8,15} = 0$, $E_{8,16} = 0$, $E_{8,17} = 1$, $E_{8,18} = 1$, $E_{8,19} = 1$, $E_{8,21} = 1$, $E_{8,22} = 1$, $E_{8,23} = 0$, $E_{8,24} = 0$, $E_{8,25} = 1$, $E_{8,26} = 0$, $E_{8,27} = 0$, $E_{8,28} = 0$, $E_{8,29} = 0$, $E_{8,30} = 0$, $E_{8,31} = 0$, $W_{8,8} = 0$, $W_{8,9} = 0$, $W_{8,19} = 0$, $W_{8,23} = 0$, $W_{8,26} = 1$, $W_{8,20} \neq W_{8,5}$, $W_{8,22} = W_{8,5}$, $W_{8,27} = W_{8,6}$, $W_{8,15} = W_{8,11}$, $W_{8,24} \neq W_{8,13}$, $W_{8,30} \neq W_{8,15}$, $W_{8,29} = W_{8,25}$	70
9	$A_{9,8} = 1$, $A_{9,10} = 1$, $A_{9,11} = 0$, $A_{9,19} = 1$, $A_{9,20} = 0$, $A_{9,23} = 0$, $A_{9,26} = 0$, $A_{9,27} = 1$, $A_{9,13} \neq A_{7,13}$, $A_{9,4} = A_{8,4}$, $A_{9,7} \neq A_{8,7}$, $A_{9,12} \neq A_{9,0}$, $A_{9,22} \neq A_{9,1}$, $A_{9,15} \neq A_{9,3}$, $A_{9,16} \neq A_{9,4}$, $A_{9,15} \neq A_{9,6}$, $A_{9,18} \neq A_{9,7}$, $A_{9,30} \neq A_{9,7}$, $A_{9,29} \neq A_{9,9}$, $A_{9,30} \neq A_{9,21}$, $A_{9,31} \neq A_{9,22}$, $E_{9,1} = 1$, $E_{9,2} = 0$, $E_{9,3} = 0$, $E_{9,4} = 0$, $E_{9,5} = 0$, $E_{9,6} = 1$, $E_{9,8} = 0$, $E_{9,9} = 1$, $E_{9,10} = 1$, $E_{9,11} = 0$, $E_{9,12} = 0$, $E_{9,13} = 1$, $E_{9,14} = 0$, $E_{9,15} = 0$, $E_{9,16} = 0$, $E_{9,17} = 0$, $E_{9,18} = 1$, $E_{9,19} = 1$, $E_{9,20} = 1$, $E_{9,21} = 0$, $E_{9,22} = 1$, $E_{9,23} = 1$, $E_{9,24} = 1$, $E_{9,25} = 0$, $E_{9,26} = 1$, $E_{9,27} = 0$, $E_{9,28} = 1$, $E_{9,29} = 0$, $E_{9,30} = 1$	50
10	$A_{10,8} \neq A_{8,8}$, $A_{10,10} \neq A_{8,10}$, $A_{10,19} \neq A_{8,19}$, $A_{10,20} = A_{8,20}$, $A_{10,26} \neq A_{8,26}$, $A_{10,12} = A_{9,12}$, $A_{10,13} = A_{9,13}$, $A_{10,22} = A_{9,22}$, $A_{10,24} \neq A_{9,24}$, $A_{10,25} \neq A_{9,25}$, $A_{10,28} = A_{9,28}$, $A_{10,29} = A_{9,29}$, $A_{10,30} \neq A_{9,30}$, $A_{10,31} \neq A_{9,31}$, $E_{10,0} = 1$, $E_{10,1} = 1$, $E_{10,3} = 1$, $E_{10,5} = 0$, $E_{10,6} = 1$, $E_{10,7} = 1$, $E_{10,8} = 0$, $E_{10,9} = 0$, $E_{10,10} = 0$, $E_{10,11} = 0$, $E_{10,12} = 0$, $E_{10,13} = 0$, $E_{10,14} = 1$, $E_{10,15} = 1$, $E_{10,16} = 0$, $E_{10,17} = 0$, $E_{10,18} = 1$, $E_{10,19} = 1$, $E_{10,20} = 0$, $E_{10,21} = 1$, $E_{10,22} = 1$, $E_{10,23} = 1$, $E_{10,24} = 1$, $E_{10,25} = 0$, $E_{10,26} = 0$, $E_{10,27} = 0$, $E_{10,28} = 0$, $E_{10,29} = 0$, $E_{10,30} = 0$, $E_{10,31} = 1$	44
11	$A_{11,8} = A_{10,8}$, $A_{11,10} = A_{10,10}$, $A_{11,11} \neq A_{10,11}$, $A_{11,19} \neq A_{10,19}$, $A_{11,20} \neq A_{10,20}$, $A_{11,23} = A_{10,23}$, $A_{11,26} \neq A_{10,26}$, $A_{11,27} \neq A_{10,27}$, $E_{11,0} = 1$, $E_{11,1} = 0$, $E_{11,2} = 0$, $E_{11,3} = 0$, $E_{11,4} = 0$, $E_{11,5} = 0$, $E_{11,6} = 0$, $E_{11,7} = 1$, $E_{11,8} = 0$, $E_{11,9} = 0$, $E_{11,10} = 1$, $E_{11,11} = 1$, $E_{11,13} = 0$, $E_{11,14} = 1$, $E_{11,15} = 1$, $E_{11,16} = 1$, $E_{11,17} = 0$, $E_{11,18} = 1$, $E_{11,19} = 0$, $E_{11,20} = 1$, $E_{11,21} = 0$, $E_{11,22} = 1$, $E_{11,23} = 1$, $E_{11,24} = 1$, $E_{11,25} = 1$, $E_{11,26} = 1$, $E_{11,27} = 0$, $E_{11,28} = 0$, $E_{11,29} = 0$, $E_{11,30} = 0$, $E_{11,31} = 0$	39
12	$E_{12,1} = 0$, $E_{12,2} = 1$, $E_{12,3} = 1$, $E_{12,6} = 0$, $E_{12,8} = 0$, $E_{12,9} = 1$, $E_{12,10} = 0$, $E_{12,12} = 1$, $E_{12,13} = 1$, $E_{12,14} = 1$, $E_{12,19} = 1$, $E_{12,20} = 1$, $E_{12,21} = 0$, $E_{12,22} = 1$, $E_{12,23} = 0$, $E_{12,24} = 1$, $E_{12,25} = 0$, $E_{12,26} = 1$, $E_{12,30} = 1$, $E_{12,31} = 0$, $E_{12,11} = W_{12,11}$, $E_{12,27} \neq W_{12,27}$, $E_{12,0} \neq A_{8,0}$, $E_{12,5} = E_{12,0}$, $W_{12,0} = 0$, $W_{12,1} = 0$, $W_{12,4} = 0$, $W_{12,6} = 0$, $W_{12,9} = 1$, $W_{12,12} = 0$, $W_{12,13} = 0$, $W_{12,21} = 1$, $W_{12,23} = 0$, $W_{12,24} = 1$, $W_{12,25} = 1$	35
13	$E_{13,1} = 1$, $E_{13,2} = 0$, $E_{13,6} = 1$, $E_{13,7} = 1$, $E_{13,8} = 0$, $E_{13,9} = 0$, $E_{13,10} = 1$, $E_{13,12} = 0$, $E_{13,13} = 1$, $E_{13,14} = 1$, $E_{13,15} = 1$, $E_{13,19} = 1$, $E_{13,20} = 1$, $E_{13,21} = 1$, $E_{13,22} = 0$, $E_{13,23} = 1$, $E_{13,24} = 0$, $E_{13,25} = 1$, $E_{13,26} = 1$, $E_{13,27} = 0$, $E_{13,28} = 1$, $E_{13,30} = 1$, $E_{13,31} = 0$, $E_{13,5} = E_{13,0}$, $E_{13,17} = E_{13,4}$, $E_{13,18} = E_{13,5}$, $E_{13,29} = E_{13,11}$	27
14	$E_{14,8} = 0$, $E_{14,9} = 0$, $E_{14,20} = 1$, $E_{14,21} = 1$, $E_{14,22} = 0$, $E_{14,23} = 1$, $E_{14,24} = 0$, $E_{14,26} = 1$	8
15	$E_{15,8} = 1$, $E_{15,9} = 1$, $E_{15,19} = 0$, $E_{15,20} = 0$, $E_{15,21} = 0$, $E_{15,22} = 1$, $E_{15,23} = 0$, $E_{15,24} = 0$, $E_{15,26} = 1$, $W_{15,1} = 0$, $W_{15,2} = 1$, $W_{15,3} = 1$, $W_{15,4} = 0$, $W_{15,5} = 1$, $W_{15,6} = 1$, $W_{15,7} = 0$, $W_{15,8} = 1$, $W_{15,9} = 1$, $W_{15,10} = 1$, $W_{15,11} = 0$, $W_{15,12} = 0$, $W_{15,13} = 1$, $W_{15,14} = 1$, $W_{15,15} = 1$, $W_{15,19} = 1$, $W_{15,20} = 1$, $W_{15,22} = 1$, $W_{15,24} = 0$, $W_{15,26} = 0$, $W_{15,27} = 0$, $W_{15,29} = 1$, $W_{15,30} = 1$, $W_{15,31} = 0$, $W_{15,23} \neq W_{15,0}$, $W_{15,25} \neq W_{15,0}$, $W_{15,25} = W_{15,18}$, $W_{15,28} \neq W_{15,21}$	37
17	$W_{17,7} = 0$, $W_{17,8} = 1$, $W_{17,9} = 1$, $W_{17,10} = 0$, $W_{17,11} = 1$, $W_{17,17} = 1$, $W_{17,19} = 1$, $W_{17,23} = 0$, $W_{17,24} = 0$, $W_{17,25} = 0$, $W_{17,26} = 1$, $W_{17,28} = 0$, $W_{17,2} \neq W_{17,0}$, $W_{17,30} = W_{17,0}$, $W_{17,31} = W_{17,1}$, $W_{17,31} = W_{17,6}$, $W_{17,21} = W_{17,12}$, $W_{17,21} = W_{17,14}$, $W_{17,22} = W_{17,15}$, $W_{17,27} \neq W_{17,18}$	20

Bibliography

[AC09] Martin R. Albrecht and Carlos Cid. Algebraic Techniques in Differential Cryptanalysis. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2009.

[ACD+10] Martin R. Albrecht, Carlos Cid, Thomas Dullien, Jean-Charles Faugère, and Ludovic Perret. Algebraic Precomputations in Differential and Integral Cryptanalysis. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 387–403. Springer, 2010.

[AHMP11] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. SHA-3 Proposal BLAKE. Submission to NIST (Round 3), January 2011. Available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html`.

[Alb08] Martin R. Albrecht. Algebraic Attacks on the Courtois Toy Cipher. *Cryptologia*, 32(3):220–276, 2008.

[Atk89] Kendall E. Atkinson. *An Introduction to Numerical Analysis*. John Wiley & Sons Inc., New York, second edition, 1989.

[BB99] Richard Beigel and Anna Bernasconi. A Note on the Polynomial Representation of Boolean Functions over GF(2). *Internat. J. Found. Comput. Sci.*, 10(4):535–542, 1999. Irregular '99 (San Juan, PR).

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *EUROCRYPT*, pages 12–23, 1999.

[BC04] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.

[BCJ+05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 36–57. Springer, 2005.

[BCJ07] Gregory V. Bard, Nicolas Courtois, and Chris Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. *IACR Cryptology ePrint Archive*, 2007:24, 2007.

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. The Rectangle Attack - Rectangling the Serpent. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2001.

[Bei93] Richard Beigel. The Polynomial Method in Circuit Complexity. In *Structure in Complexity Theory Conference*, pages 82–95, 1993.

[Ber09] Daniel J. Bernstein. CubeHash specification (2.B.1). Submission to NIST (Round 2), September 2009. Available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html`.

[BFL10] Charles Bouillaguet, Pierre-Alain Fouque, and Gatan Leurent. Security Analysis of SIMD. Cryptology ePrint Archive, Report 2010/323, 2010. `http://eprint.iacr.org/`.

[Bjö96] Åke Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996.

[BKM10] Julia Borghoff, Lars R. Knudsen, and Krystian Matusiewicz. Hill Climbing Algorithms and Trivium. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 57–73. Springer, 2010.

[BKN09] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009.

[BKR97] Johan Borst, Lars R. Knudsen, and Vincent Rijmen. Two Attacks on Reduced IDEA. In *EUROCRYPT*, pages 1–13, 1997.

[BKS09] Julia Borghoff, Lars R. Knudsen, and Mathias Stolpe. Bivium as a Mixed-Integer Linear Programming Problem. In Matthew G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2009.

[BLMN11] Alex Biryukov, Mario Lamberger, Florian Mendel, and Ivica Nikolic. Second-Order Differential Collisions for Reduced SHA-256. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 270–287. Springer, 2011.

[BLP10] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller Decoding Exponents: Ball-Collision Decoding. Cryptology ePrint Archive, Report 2010/585, 2010.

[BMvT78] Elwyn R. Berlekamp, Robert McEliece, and Henk van Tilborg. On the Inherent Intractability of Certain Coding Problems (Corresp.). *Information Theory, IEEE Transactions on*, 24(3):384 – 386, may 1978.

[BNR11] Alex Biryukov, Ivica Nikolic, and Arnab Roy. Boomerang Attacks on BLAKE-32. In Antoine Joux, editor, *FSE*, volume 6733 of *LNCS*, pages 218–237. Springer, 2011.

[BPW06] Johannes Buchmann, Andrei Pyshkin, and Ralf-Philipp Weinmann. Block Ciphers Sensitive to Gröbner Basis Attacks. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 313–331. Springer, 2006.

[BR93] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[BS92] Eli Biham and Adi Shamir. Differential Cryptanalysis of the Full 16-Round DES. In Ernest F. Brickell, editor, *CRYPTO*, volume 740 of *LNCS*, pages 487–496. Springer, 1992.

[Can06] Christophe De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[CB07] Nicolas Courtois and Gregory V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In Steven D. Galbraith, editor, *IMA Int. Conf.*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007.

[CC98] Anne Canteaut and Florent Chabaud. A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.

[Cha94] Florent Chabaud. On the Security of Some Cryptosystems Based on Error-Correcting Codes. In Alfredo De Santis, editor, *EUROCRYPT*, volume 950 of *LNCS*, pages 131–139. Springer, 1994.

[CJ98] Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *LNCS*, pages 56–71. Springer, 1998.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In Bart Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[CL96] Thomas F. Coleman and Yuying Li. An Interior Trust Region Approach for Nonlinear Minimization Subject to Bounds. *SIAM J. Optim.*, 6(2):418–445, 1996.

[CLO07] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms*. Undergraduate Texts in Mathematics. Springer, New York, third edition, 2007. An Introduction to Computational Algebraic Geometry and Commutative Algebra.

[CM03] Nicolas Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[CN08] Donghoon Chang and Mridul Nandi. Improved Indifferentiability Security Analysis of chopMD Hash Function. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 429–443. Springer, 2008.

[CNO08] Nicolas Courtois, Karsten Nohl, and Sean O'Neil. Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards. *IACR Cryptology ePrint Archive*, 2008:166, 2008.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[CPSY06] Hong-Su Cho, Sangwoo Park, Soo Hak Sung, and Aaram Yun. Collision Search Attack for 53-Step HAS-160. In Min Surp Rhee and Byoungcheon Lee, editors, *ICISC*, volume 4296 of *LNCS*, pages 286–295. Springer, 2006.

[Dam89] Ivan Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *LNCS*, pages 416–427. Springer, 1989.

[Dau05] Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family*. PhD thesis, Ruhr-Universität Bochum, May 2005. Available online: `http://www.cits.rub.de/imperia/md/content/magnus/dissmd4.pdf`.

[Deu04] Peter Deuflhard. *Newton Methods for Nonlinear Problems*, volume 35 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2004. Affine invariance and adaptive algorithms.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A Machine Program for Theorem-Proving. *Commun. ACM*, 5(7):394–397, 1962.

[DMR07] Christophe De Cannière, Florian Mendel, and Christian Rechberger. Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *LNCS*, pages 56–73. Springer, 2007.

[Dob97] Hans Dobbertin. RIPEMD with Two-Round Compress Function is Not Collision-Free. *J. Cryptology*, 10(1):51–70, 1997.

[Dob98] Hans Dobbertin. Cryptanalysis of MD4. *J. Cryptology*, 11(4):253–271, 1998.

[DP92] Peter Deuflhard and Florian A. Potra. Asymptotic Mesh Independence of Newton-Galerkin Methods via a Refined Mysovskiĭ Theorem. *SIAM J. Numer. Anal.*, 29(5):1395–1412, 1992.

[DR06] Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.

[ECR] ECRYPT. The eSTREAM Project. http://www.ecrypt.eu.org/stream/.

[EGG+12] Thomas Eisenbarth, Zheng Gong, Tim Güneysu, Stefan Heyse, Sebastiaan Indesteege, Stéphanie Kerckhof, François Koeune, Tomislav Nad, Thomas Plos, Francesco Regazzoni, François-Xavier Standaert, and Loïc van Oldeneel tot Oldenzeel. Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT*, volume 7374 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2012.

[EPV08] Tobias Eibach, Enrico Pilz, and Gunnar Völkel. Attacking Bivium Using SAT Solvers. In Hans Kleine Büning and Xishun Zhao, editors, *SAT*, volume 4996 of *Lecture Notes in Computer Science*, pages 63–76. Springer, 2008.

[Fau99] Jean-Charles Faugére. A New Efficient Algorithm for Computing Gröbner Bases (F_4). *J. Pure Appl. Algebra*, 139(1-3):61–88, 1999. Effective Methods in Algebraic Geometry (Saint-Malo, 1998).

[Fau02] Jean-Charles Faugère. A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 75–83 (electronic), New York, 2002. ACM.

[FKM08] Simon Fischer, Shahram Khazaei, and Willi Meier. Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. In Serge Vaudenay, editor, *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 236–245. Springer, 2008.

[FLS+11] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein Hash Function Family. Submission to NIST (Round 3), January 2011. Available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html`.

[FMM03] Claudia Fiorini, Enrico Martinelli, and Fabio Massacci. How to Fake a RSA Signature by Encoding Modular Root Finding as a SAT Problem. *Discrete Applied Mathematics*, 130(2):101–127, 2003.

[GA11] E.A. Grechnikov and A.V. Adinetz. Collision for 75-step SHA-1: Intensive Parallelization with GPU. Cryptology ePrint Archive, Report 2011/641, 2011. `http://eprint.iacr.org/`.

[GH03] Henri Gilbert and Helena Handschuh. Security Analysis of SHA-256 and Sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *LNCS*, pages 175–193. Springer, 2003.

[GPFW96] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 19–152. American Mathematical Society, 1996.

[HKS09] Deukjo Hong, Bonwook Koo, and Yu Sasaki. Improved Preimage Attack for 68-Step HAS-160. In Donghoon Lee and Seokhie Hong, editors, *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 332–348. Springer, 2009.

[HPR04] Philip Hawkes, Michael Paddon, and Gregory G. Rose. On Corrective Patterns for the SHA-2 Family. Cryptology ePrint Archive, Report 2004/207, 2004.

[IMPR08] Sebastiaan Indesteege, Florian Mendel, Bart Preneel, and Christian Rechberger. Collisions and Other Non-random Properties for Step-Reduced SHA-256. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 276–293. Springer, 2008.

[IP09] Sebastiaan Indesteege and Bart Preneel. Practical Collisions for EnRUPT. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 246–259. Springer, 2009.

[JJ05] Dejan Jovanovic and Predrag Janicic. Logical Analysis of Hash Functions. In Bernhard Gramlich, editor, *FroCos*, volume 3717 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2005.

[JP07] Antoine Joux and Thomas Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *LNCS*, pages 244–263. Springer, 2007.

[JPS93] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian Optimization Without the Lipschitz Constant. *J. Optim. Theory Appl.*, 79(1):157–181, 1993.

[KH05] Shahram Khazaei and Mehdi Hassanzadeh. Linear Sequential Circuit Approximation of the TRIVIUM Stream Cipher. eSTREAM submitted papers, 2005.

[KKL88] J. Kahn, G. Kalai, and N. Linial. The Influence of Variables on Boolean Functions. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, SFCS '88, pages 68–80, Washington, DC, USA, 1988. IEEE Computer Society.

[Knu94] Lars R. Knudsen. Truncated and Higher Order Differentials. In Bart Preneel, editor, *FSE*, volume 1008 of *LNCS*, pages 196–211. Springer, 1994.

[KR07] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 315–324. Springer, 2007.

[Lai92] Xuejia Lai. Higher Order Derivatives and Differential Cryptanalysis. In Richard Blahut, Daniel Costello Jr., Ueli Maurer, and Thomas Mittelholzer, editors, *Communications and Cryptography*, pages 227–233. Kluwer, 1992.

[Lai94] Xuejia Lai. Higher Order Derivatives and Differential Cryptanalysis. In Richard E. Blahut, Daniel J. Costello Jr., Ueli Maurer, and Thomas Mittelholzer, editors, *Communications and Cryptography: Two Sides of One Tapestry*, pages 227–233. Kluwer Academic Publishers, 1994.

[LBF09a] Gaëtan Leurent, Charles Bouillaguet, and Pierre-Alain Fouque. SIMD Is a Message Digest. Submission to NIST (Round 2), September 2009. Available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html`.

[LBF09b] Gaëtan Leurent, Charles Bouillaguet, and Pierre-Alain Fouque. Tweaking SIMD, 2009. `http://www.di.ens.fr/~leurent/files/SIMD-tweak.pdf`.

[Leo88] J.S. Leon. A Probabilistic Algorithm for Computing Minimum Weights of Large Error-Correcting Codes. *Information Theory, IEEE Transactions on*, 34(5):1354 –1359, sep 1988.

[LM92] Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In Rainer A. Rueppel, editor, *EUROCRYPT*, volume 658 of *LNCS*, pages 55–70. Springer, 1992.

[LM01] Helger Lipmaa and Shiho Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *LNCS*, pages 336–350. Springer, 2001.

[LM11] Mario Lamberger and Florian Mendel. Higher-Order Differential Attack on Reduced SHA-256. Cryptology ePrint Archive, Report 2011/037, 2011.

[LNR09a] Mario Lamberger, Tomislav Nad, and Vincent Rijmen. Numerical Solvers and Cryptanalysis. *Journal of Mathematical Cryptology*, 3(3):249–263, September 2009.

[LNR09b] Mario Lamberger, Tomislav Nad, and Vincent Rijmen. Numerical Solvers and Cryptanalysis. *J. Math. Cryptol.*, 3(3):249–263, 2009.

[MB07] Alexander Maximov and Alex Biryukov. Two Trivial Attacks on Trivium. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *LNCS*, pages 36–55. Springer, 2007.

[MCP07] Cameron McDonald, Chris Charnes, and Josef Pieprzyk. An Algebraic Analysis of Trivium Ciphers Based on the Boolean Satisfiability Problem. *IACR Cryptology ePrint Archive*, 2007:129, 2007.

[MCP08] Cameron McDonald, Chris Charnes, and Josef Pieprzyk. An Algebraic Analysis of Trivium Ciphers Based on the Boolean Satisfiability Problem. In Jean-Francis Michon, editor, *International*

Conference on Boolean Functions: Cryptography and Applications, BFCA. Springer, 2008.

[Mer89] Ralph C. Merkle. One Way Hash Functions and DES. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *LNCS*, pages 428–446. Springer, 1989.

[MM00] Fabio Massacci and Laura Marraro. Logical Cryptanalysis as a SAT Problem. *J. Autom. Reasoning*, 24(1/2):165–203, 2000.

[MN09] Florian Mendel and Tomislav Nad. A Distinguisher for the Compression Function of SIMD-512. In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT*, volume 5922 of *LNCS*, pages 219–232. Springer, 2009.

[MN11] Florian Mendel and Tomislav Nad. Boomerang Distinguisher for the SIMD-512 Compression Function. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *INDOCRYPT*, volume 7107 of *Lecture Notes in Computer Science*, pages 255–269. Springer, 2011.

[MNS09] Florian Mendel, Tomislav Nad, and Martin Schläffer. Collision Attack on Boole. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS*, volume 5536 of *LNCS*, pages 369–381, 2009.

[MNS11a] Florian Mendel, Tomislav Nad, and Martin Schläffer. Cryptanalysis of Round-Reduced HAS-160. In Howon Kim, editor, *Information Security and Cryptology - ICISC 2011*. Springer, 2011. in press.

[MNS11b] Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding SHA-2 Characteristics: Searching Through a Minefield of Contradictions. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 288–307. Springer, 2011.

[MNS12] Florian Mendel, Tomislav Nad, and Martin Schläffer. Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128. In *FSE*, 2012. In press.

[MNSS12] Florian Mendel, Tomislav Nad, Stefan Scherz, and Martin Schläffer. Differential Attacks on Reduced RIPEMD-160. In *ISC 2012 - Information Security Conference*, 2012. In press.

[Mon94] Yishay Monsour. Learning Boolean Functions via the Fourier Transform. In *Theoretical Advances in Neural Computation and Learning*, 1994.

[Mor78] Jorge J. Moré. The Levenberg-Marquardt Algorithm: Implementation and Theory. In *Numerical Analysis (Proc. 7th Biennial Conf., Univ. Dundee, Dundee, 1977)*, pages 105–116. Lecture Notes in Math., Vol. 630. Springer, Berlin, 1978.

[MPRR06a] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Analysis of Step-Reduced SHA-256. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2006.

[MPRR06b] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. The Impact of Carries on the Complexity of Collision Attacks on SHA-1. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 278–292. Springer, 2006.

[MR07] Florian Mendel and Vincent Rijmen. Colliding Message Pair for 53-Step HAS-160. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC*, volume 4817 of *LNCS*, pages 324–334. Springer, 2007.

[MT07] Ueli M. Maurer and Stefano Tessaro. Domain Extension of Public Random Functions: Beyond the Birthday Barrier. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *LNCS*, pages 187–204. Springer, 2007.

[Mur11] S. Murphy. The Return of the Cryptographic Boomerang. *Information Theory, IEEE Transactions on*, 57(4):2517 –2521, April 2011.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. Series on Discrete Mathematics and its Applications. CRC Press, 1997. ISBN 0-8493-8523-7, Available online at `http://www.cacr.math.uwaterloo.ca/hac/`.

[MY92] Mitsuru Matsui and Atsuhiro Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *EUROCRYPT*, pages 81–91, 1992.

[MZ06] Ilya Mironov and Lintao Zhang. Applications of SAT Solvers to Cryptanalysis of Hash Functions. In Armin Biere and Carla P. Gomes, editors, *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115. Springer, 2006.

[Nad10] Tomislav Nad. The CodingTool Library. Workshop on Tools for Cryptanalysis 2010, 2010. `http://www.iaik.tugraz.at/content/research/krypto/codingtool/`.

[Nat95] National Institute of Standards and Technology. FIPS PUB 180-1: Secure Hash Standard. Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce, April 1995. Available online: `http://www.itl.nist.gov/fipspubs`.

[Nat02] National Institute of Standards and Technology. FIPS PUB 180-2: Secure Hash Standard. Federal Information Processing Standards

Publication 180-2, U.S. Department of Commerce, August 2002. Available online: `http://www.itl.nist.gov/fipspubs`.

[Nat07] National Institute of Standards and Technology. Cryptographic Hash Algorithm Competition, November 2007. Available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`.

[Nat08] National Institute of Standards and Technology. FIPS PUB 180-3: Secure Hash Standard. Federal Information Processing Standards Publication 180-3, U.S. Department of Commerce, October 2008. Available online: `http://www.itl.nist.gov/fipspubs`.

[Nat09] National Institute of Standards and Technology. Second Round Candidates. Official notification from NIST, 2009. Available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/submissions_rnd2.html`.

[NB08] Ivica Nikolić and Alex Biryukov. Collisions for Step-Reduced SHA-256. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 1–15. Springer, 2008.

[NPSS10] Ivica Nikolić, Josef Pieprzyk, Przemyslaw Sokolowski, and Ron Steinfeld. Rotational Cryptanalysis of (Modified) Versions of BMW and SIMD. Available online, 2010.

[NS94] Noam Nisan and Mario Szegedy. On the Degree of Boolean Functions as Real Polynomials. *Computational Complexity*, 4:301–313, 1994.

[OK94] Luke O'Connor and Andrew Klapper. Algebraic Nonlinearity and its Applications to Cryptography. *J. Cryptology*, 7(4):213–227, 1994.

[OR00] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*, volume 30 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. Reprint of the 1970 original.

[PGV93] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *LNCS*, pages 368–378. Springer, 1993.

[PR02] Panos M. Pardalos and H. Edwin Romeijn, editors. *Handbook of Global Optimization. Vol. 2*, volume 62 of *Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht, 2002.

[PRR05] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In Nigel P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *LNCS*, pages 78–95. Springer, 2005.

[QSS02] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerische Mathematik 1*. Springer-Verlag, Berlin, 2002.

[Rad07] Havard Raddum. Cryptanalytic Results on Trivium. eSTREAM submitted papers, 2007. `http://www.ecrypt.eu.org/stream/papersdir/2006/039.ps`.

[Riv90] Ronald L. Rivest. The MD4 Message Digest Algorithm. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *LNCS*, pages 303–311. Springer, 1990.

[Riv92] Ronald L. Rivest. The MD5 Message-Digest Algorithm. IETF Request for Comments (RFC) 1321, 1992. Available online at: `http://www.faqs.org/rfcs/rfc1321.html`.

[RO05] Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In Alfred Menezes, editor, *CT-RSA*, volume 3376 of *LNCS*, pages 58–71. Springer, 2005.

[Rob08] Matthew Robshaw. *New Stream Cipher Designs: The eSTREAM Finalists*. Springer-Verlag, Berlin, Heidelberg, 2008.

[RP94] Vincent Rijmen and Bart Preneel. Improved Characteristics for Differential Cryptanalysis of Hash Functions Based on Block Ciphers. In Bart Preneel, editor, *FSE*, volume 1008 of *LNCS*, pages 242–248. Springer, 1994.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *LNCS*, pages 371–388. Springer, 2004.

[SA08] Yu Sasaki and Kazumaro Aoki. A Preimage Attack for 52-Step HAS-160. In Pil Joong Lee and Jung Hee Cheon, editors, *ICISC*, volume 5461 of *LNCS*, pages 302–317. Springer, 2008.

[Sch97] Hans Rudolf Schwarz. *Numerische Mathematik*. B. G. Teubner, Stuttgart, fourth edition, 1997. With a contribution by Jörg Waldvogel.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT Solvers to Cryptographic Problems. In Oliver Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[Spe93] P. Spellucci. *Numerische Verfahren der Nichtlinearen Optimierung*. Internationale Schriftenreihe zur Numerischen Mathematik. [International Series of Numerical Mathematics]. Birkhäuser Verlag, Basel, 1993.

[SS07] Somitra Kumar Sanadhya and Palash Sarkar. New Local Collisions for the SHA-2 Hash Family. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC*, volume 4817 of *LNCS*, pages 193–205. Springer, 2007.

[SS08a] Somitra Kumar Sanadhya and Palash Sarkar. Attacking Reduced Round SHA-256. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS*, volume 5037 of *LNCS*, pages 130–143, 2008.

[SS08b] Somitra Kumar Sanadhya and Palash Sarkar. Deterministic Constructions of 21-Step Collisions for the SHA-2 Hash Family. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC*, volume 5222 of *LNCS*, pages 244–259. Springer, 2008.

[SS08c] Somitra Kumar Sanadhya and Palash Sarkar. New Collision Attacks against Up to 24-Step SHA-2. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *LNCS*, pages 91–103. Springer, 2008.

[SS08d] Somitra Kumar Sanadhya and Palash Sarkar. Non-linear Reduced Round Attacks against SHA-2 Hash Family. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP*, volume 5107 of *LNCS*, pages 254–266. Springer, 2008.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.

[Tel08] Telecommunications Technology Association. Hash Function Standard Part 2: Hash Function Algorithm Standard (HAS-160), TTAS.KO-12.0011/R1, 2008.

[Tho94] Thomas F. Coleman and Yuying Li. On the Convergence of Interior-Reflective Newton Methods for Nonlinear Minimization Subject to Bounds. *Math. Program.*, 67:189–224, 1994.

[Tis11] Elmar Tischhauser. Nonsmooth Cryptanalysis, with an Application to the Stream Cipher MICKEY. *J. Math. Cryptol.*, 4(4):317–348, 2011.

[TK07] M. Sönmez Turan and O. Kara. Linear Approximations for 2-round Trivium. In *Proc. First International Conference on Security of Information and Networks (SIN 2007)*, pages 96–105. Trafford Publishing, 2007.

[Vie07] Michael Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. Cryptology ePrint Archive, Report 2007/413, 2007. `http://eprint.iacr.org/`.

[Wag99] David Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *LNCS*, pages 156–170. Springer, 1999.

[WLF+05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.

[WY05] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

[WYY05a] Xiaoyun Wang, Andrew Yao, and Frances Yao. New Collision Search for SHA-1. Presented at rump session of CRYPTO, 2005.

[WYY05b] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.

[WYY05c] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 1–16. Springer, 2005.

[YCC04] Bo-Yin Yang, Jiun-Ming Chen, and Nicolas Courtois. On Asymptotic Security Estimates in XL and Gröbner Bases-Related Algebraic Cryptanalysis. In Javier Lopez, Sihan Qing, and Eiji Okamoto, editors, *ICICS*, volume 3269 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 2004.

[YCKW11] Hongbo Yu, Jiazhe Chen, Ketingjia, and Xiaoyun Wang. Near-Collision Attack on the Step-Reduced Compression Function of Skein-256. Cryptology ePrint Archive, Report 2011/148, 2011.

[YSP+05] Aaram Yun, Soo Hak Sung, Sangwoo Park, Donghoon Chang, Seokhie Hong, and Hong-Su Cho. Finding Collision on 45-Step HAS-160. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *LNCS*, pages 146–155. Springer, 2005.

[YW11] Hongbo Yu and Xiaoyun Wang. Cryptanalysis of the Compression Function of SIMD. In Udaya Parampalli and Philip Hawkes, editors, *ACISP*, volume 6812 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2011.

List of Publications

International Journals

1. Mario Lamberger, Tomislav Nad, and Vincent Rijmen. Numerical Solvers and Cryptanalysis. *J. Math. Cryptol.*, 3(3):249-263, 2009.

Refereed Conference Proceedings

1. Florian Mendel, Tomislav Nad, Stefan Scherz, and Martin Schläffer. Differential Attacks on Reduced RIPEMD-160. In *ISC 2012 - Information Security Conference*, 2012. In press

2. Florian Mendel, Tomislav Nad, and Martin Schläffer. Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128. In *FSE*, 2012. In press.

3. Thomas Eisenbarth, Zheng Gong, Tim Güneysu, Stefan Heyse, Sebastiaan Indesteege, Stéphanie Kerckhof, François Koeune, Tomislav Nad, Thomas Plos, Francesco Regazzoni, François-Xavier Standaert, and Loïc van Oldeneel tot Oldenzeel. Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *AFRICACRYPT*, volume 7374 of *LNCS*, pages 172-187. Springer, 2012.

4. Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding SHA-2 Characteristics: Searching Through a Minefield of Contradictions. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *LNCS*, pages 288-307. Springer, 2011

5. Florian Mendel, Tomislav Nad, and Martin Schläffer. Cryptanalysis of Round-Reduced HAS-160. In Howon Kim, editor, *Information Security and Cryptology - ICISC 2011*. Springer, 2011. In press.

6. Florian Mendel and Tomislav Nad. Boomerang Distinguisher for the SIMD-512 Compression Function. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *INDOCRYPT*, volume 7107 of *LNCS*, pages 255-269. Springer, 2011.

7. Florian Mendel and Tomislav Nad. A Distinguisher for the Compression Function of SIMD-512. In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT*, volume 5922 of *LNCS*, pages 219-232. Springer, 2009.

8. Florian Mendel, Tomislav Nad, and Martin Martin Schläffer. Collision Attack on Boole. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS*, volume 5536 of *LNCS*, pages 369-381, 2009.

Code Publications

1. Tomislav Nad. The CodingTool Library. *Workshop on Tools for Cryptanalysis 2010*, 2010. `http://www.iaik.tugraz.at/content/research/krypto/codingtool/`.

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)