Master Thesis

Cryptanalysis of MD4

Martin Schläffer

Cryptanalysis of MD4

Master Thesis

at

Graz University of Technology

submitted by

Martin Schläffer

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology
A-8010 Graz, Austria

February 1, 2006

Assessor:	Univ.-Prof. Dr. Vincent Rijmen
Advisor:	Dipl.-Ing. Dr. techn. Elisabeth Oswald

Graz University of Technology

Kryptoanalyse von MD4

Magisterarbeit

an der

Technischen Universität Graz

vorgelegt von

Martin Schläffer

Institut für Angewandte Informationverarbeitung und Kommunikation (IAIK),
Technische Universität Graz
A-8010 Graz

1. Februar 2006

Diese Arbeit ist in englischer Sprache verfasst.

Gutachter:	Univ.-Prof. Dr. Vincent Rijmen
Betreuer:	Dipl.-Ing. Dr. techn. Elisabeth Oswald

Graz University of Technology

Abstract

Hash functions play an important role in many cryptographic applications and protocols. In the last few years many cryptographic hash functions have been broken. Especially, the differential attacks by Wang *et al.* on the collision resistance of the MD-family have attracted a lot of attention. However, in all articles published by Wang *et al.*, few details about their methods are given. In particular, they do not describe how their differential paths can be found.

In this thesis the MD4 hash function is attacked using the approach of Wang *et al.* At the beginning the technical methods for the differential cryptanalysis of MD4 are defined. The main part of this thesis describes the development of an algorithm to find differential paths in an automated way. Many differential paths have been found so far and some are slightly better than the path published by Wang *et al.* Finally, the message modification techniques used by Wang *et al.* are analyzed and improved.

Keywords: hash function, cryptanalysis, MD4, collision resistance, differential path, message modification

Kurzfassung

Hashfunktionen spielen eine bedeutende Rolle in vielen kryptographischen Anwendungen und Protokollen. In den letzten Jahren wurden jedoch bereits mehrere kryptographische Hashfunktionen gebrochen. Besonders die differentiellen Attacken von Wang *et al.* auf die Kollisionsresistenz der MD-Familie haben viel Aufmerksamkeit auf sich gezogen. In ihren Veröffentlichungen wurden jedoch nur wenige Details über ihre Methoden bekannt gegeben. Im Speziellen wurde nie beschrieben, wie ein differentieller Pfad gefunden werden kann.

In dieser Magisterarbeit wird die Hashfunktion MD4 mit Hilfe der Methode von Wang *et al.* attackiert. Zu Beginn werden die technischen Methoden definiert, welche für die differentielle Kryptoanalyse benötigt werden. Der Hauptteil der Arbeit besteht darin, einen Algorithmus zu entwickeln der automatisiert differentielle Pfade findet. Bei der Suche wurden viele verschiedene differentielle Pfade gefunden, wobei einige zum Teil besser sind als jener von Wang *et al.* Abschließend wird auf die ,,Message-Modification" Techniken von Wang *et al.* eingegangen, welche analysiert und verbessert wurden.

Schlüsselwörter: Hashfunktion, Kryptoanalyse, MD4, kollisionsresistent, differentieller Pfad, Message-Modification

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Ich versichere hiermit, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Acknowledgements

I am indebted to my colleagues at the IAIK and the members of the Krypto group who have never been tired in answering my innumerable questions. Especially, I would like to thank my advisor, Elisabeth Oswald, for her support, my admission to the institute and the endless evening hours in correcting early draft versions of this document.

Special thanks goes to Mario Ivkovic and Mario Lamberger for their insightful comments and for reviewing many chapters of this thesis.

Last but not least, I would like to thank my girlfriend and my family for their support during my studies.

Martin Schläffer
Graz, Austria, February 2006

Contents

Chapter 1

Introduction

Cryptographic hash functions play an important role in modern cryptography. A hash function computes a short fingerprint (hash value, message digest) of a message and is used in many cryptographic applications and protocols (cf. [MvOV97]). The cryptanalysis of hash functions is of major interest, because their security is crucial for most applications.

1.1 Cryptanalysis of Hash Functions

During the last few years many important cryptographic hash functions have been broken. The main target is the collision resistance of hash functions. Especially the results of Wang *et al.* have drawn significant attention to the security problems of currently used hash functions.

The main target of these Chinese researchers has been the MD-family of (unkeyed) hash functions because they share a common structure and are used in many standards. In the last few years, Wang *et al.* have broken many members of this family. First, they have analyzed MD4 and RIPEMD in [WLF$^+$05], and shortly after their attack strategy was successfully applied to MD5 [WY05]. Finally, their techniques were adapted and extended to attack SHA-0 and SHA-1 in [WYY05d] and [WYY05c]. SHA-1 is currently the most used hash algorithm and during the "NIST Cryptographic Hash Workshop 2005" [WYY05a], the attack complexity was claimed to be 2^{63}. This complexity is close to being computable by supercomputers. The first real collision of SHA-1 will be found soon.

However, in all articles published by Wang *et al.* so far only little details about their methods are given. In particular no details on how a differential path can actually be found was provided. It is assumed that their differential paths have been found by hand with much intuition. Many details about message modification are omitted too. Some articles have already discussed the techniques used by Wang *et al.* Magnus Daum [Dau05] and an ECRYPT deliverable [ABB$^+$05] have provided some high level discussion without going into detail and Hawkes *et al.* [HPR04] have analyzed the differential path used for MD5. The different message modification techniques used by Wang *et al.* have been analyzed by Naito *et al.* [NSKO05], and Viktor Klima [Kli05].

The goal of this thesis is to understand the methods used by Wang *et al.* It analyzes the hash function MD4 because it is the simplest algorithm of the MD-family and the basis for all other variants. New methods how to find differential paths in MD4 are developed in this thesis. Using these methods, many paths can be found within minutes in an automated way. Among them are paths which are slightly better than the path found by Wang *et al.* In addition, the message modification techniques have been improved. This thesis is not intended to hide information and thus, all methods are explained in detail.

1.2 Thesis Outline

This thesis is organized as follows. In Chapter 2 the basic properties and the iterated design principle of unkeyed hash functions are presented. Then the most important general attacks are discussed.

Chapter 3 describes the design of the MD-family of hash functions. Because MD4 is the main target of this thesis, it is explained in more detail. In the last section of this chapter an overview of the most important attacks and their complexity is given.

In Chapter 4 the attack approach by Wang *et al.* is considered. An overview and the interconnection between the three main parts, deriving the message difference, searching for a differential path, and the message modification techniques are provided.

Chapter 5 provides the different technical methods, which are used in the differential analysis of MD4. Signed differences and differential operations on these differences are defined first. Then the propagation of differences by carry expansion and through the Boolean functions are discussed.

In Chapter 6 the first part of the attack approach by Wang *et al.* is analyzed. This part explains the special differential behavior of the message difference in their attack. Other message differences with a similar behavior, and the message differences used by Dobbertin and Yu *et al.*, are discussed as well.

The main part of this thesis is to find a differential path for the given message difference. Chapter 7 presents an algorithm which successfully searches for differential paths in MD4. Subsequently, the experiments and results provided by this algorithm are discussed.

Finally, in Chapter 8 the message modification techniques used by Wang *et al.* are analyzed. Furthermore, new advanced message modification techniques are presented which are used to fulfill all conditions in the first two rounds of the path of Wang *et al.*

The best differential path found is presented in Appendix A. The complete differential behavior of the path and its sufficient conditions are given.

Chapter 2

Cryptographic Hash Functions

There are two general types of cryptographic hash functions. Message detection codes (MDCs) are unkeyed hash functions and message authentication codes (MAC) are hash functions that use a secret key. This thesis discusses only unkeyed hash functions. In this chapter, their basic properties are defined and the design principle of iterated hash functions is shown. Then, the most important general attacks, which can be applied to all hash functions, are discussed.

2.1 Basic Properties

Definition 2.1. (Hash Function). A **hash function** is a function $h : X \to Y$, where $X = \{0,1\}^*$ and $Y = \{0,1\}^n$ for some $n \geq 1$. The image $y = h(x)$ of some message (preimage) $x \in X$ is called the **hash value** of x.

Hash functions are lossy compression functions, which compress messages of (almost) arbitrary length to values with a fixed length. An important property of hash functions is their efficient computation: $h(x)$ can be efficiently computed for any preimage x.

For **cryptographic hash functions** more properties than just compression and efficient computation are needed. The output of an ideal cryptographic hash function should behave randomly and independent of the input message. This cannot be reached in practice. Therefore, depending on the application a cryptographic hash function should fulfill some or all of the following properties (cf. [MvOV97]):

Definition 2.2. (Preimage Resistance). A hash function h is preimage resistant, if for a value y, it is computationally infeasible to find any message x with $h(x) = y$.

Definition 2.3. (2nd-Preimage Resistance). A hash function h is 2nd-preimage resistant, if for a message x, it is computationally infeasible to find any second message $x' \neq x$ (2nd-preimage) with $h(x') = h(x)$.

Definition 2.4. (Collision Resistance). A hash function h is collision resistant, if it is computationally infeasible to find any pair of two different messages x and x' (collision), which hash to the same value $h(x) = h(x')$.

Because of the compression from arbitrary to fixed length, the *existence* of collisions and preimages is unavoidable. The important fact about cryptographic hash functions is, that it is difficult to construct a collision or to find a preimage.

2.2 Design of Iterated Hash Functions

To compress the input message, most hash functions use some sort of iteration. The input message M is divided into k input blocks with equal block size and each block is processed by a compression function f. If the message length is not a multiple of the block size, some padding is used. Thus, almost every hash function can be described as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(M_i, H_{i-1}) \qquad \text{for } i = 1, 2, ..., k \\ h(M) &= g(H_k). \end{aligned}$$

IV is the abbreviation for *initial value*, which is a predefined starting value, and the H_i's are called *chaining variables*. The final hash value is computed by the output transformation g, which is often the identity function. An overview of an iterated hash function is given in Figure 2.1.

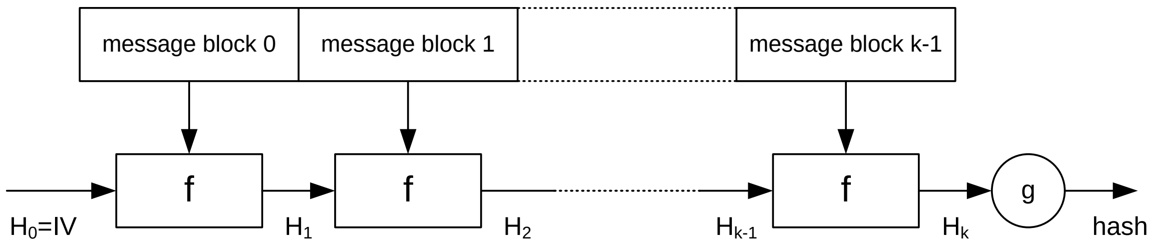


Figure 2.1: Overview of an iterated hash function.

The choice of the IV and the used padding scheme have an important influence on the security of a hash function. Hence, the IV should be part of the description of a hash function and the padding scheme should be unambiguous. A common padding scheme is to include the bitlength of the original message in some message block, which is referred to as *MD-strengthening*. Merkle [Mer90] and Damgård [Dam89] have proved, that if a compression function f is collision resistant, then the resulting hash function is collision resistant as well:

Theorem 2.1. ***(Merkle-Damgård).*** *Let f be a collision resistant function mapping l to n bits (with $l - n > 1$). If an unambiguous padding rule is used, the following construction yields a collision-resistant hash function:*

$$\begin{aligned} H_1 &= f(0^{n+1}||m_1) \\ H_i &= f(H_{i-1}||1||m_i) \qquad \textit{for } i = 2, 3, ..., k \end{aligned}$$

2.3 General Attacks

This section describes general attacks on hash functions. These attacks are independent of the hash algorithm. It is assumed that the hash value is a uniformly distributed and independent random variable. Otherwise, the following attacks would be even more successful (cf. [Pre93]).

2.3.1 Brute-Force Attack

Because of the compression property of a hash function, preimages, 2nd-preimages and collisions always exist. Thus, an attacker can always try to select random preimages and 2nd-preimages by brute-force until she succeeds. The success probability is $1/2^n$ and so the attack complexity is $O(2^n)$, where n is the bit-length of the hash value. To make random brute-force attacks computationally infeasible a hash value with at least $n = 80$ should be used nowadays.

2.3.2 Birthday Attack

The birthday attack is a brute-force attack on the *collision resistance* of a hash function. The goal of a collision attack is to find a pair of messages which hash to the same hash value. The birthday attack is based on the *birthday paradoxon*, which is not a paradoxon but rather a counter-intuitive fact of statistics [Sti02]: In a group of 23 randomly chosen people, the probability that at least two will share a common birthday is greater than $1/2$.

Proposition 2.1. ***(Generalized Birthday Paradox).*** *Given a set of t pair-wise distinct elements ($t > 10$), the probability that, a sample of size $k > 1.2\sqrt{t}$ (drawn with repetition) contains two equal elements, is greater than $1/2$ [Dau05].*

Thus, the complexity of finding a collision in a hash function, with a probability of $1/2$, is $O(2^{\frac{n}{2}})$. Note that this is the worst case for an attacker. The success probability will increase, if some hash values are more likely than others. Because birthday attacks on the collision resistance of hash functions always succeed, the hash size of a collision resistant hash function should be at least $n = 160$ nowadays.

Remark 2.1 (Academically Broken). A hash function with a hash value of length n is considered to be academically broken, if collisions can be found in less than $2^{\frac{n}{2}}$, or (second) preimages can be found in less than 2^n computations of the hash function.

Chapter 3

Hash Functions of the MD-Family

This chapter describes the MD-family of hash functions. MD is the abbreviation for Message Digest. Some hash function of this family have been the most reliable hash algorithms in the last decade and are used in many standards. Therefore, the next sections give an overview of the general properties, which are common to all hash functions of this family. Because this thesis concentrates on attacking MD4, this algorithm is described in more detail. Then the remaining hash functions of the MD-family and their main differences regarding MD4 are mentioned. The last section provides an overview of attacks on the MD-family.

3.1 General Properties

Hash functions of the MD-family are iterated hash functions and follow the MD-design principle (see Section 2.2). Further, these hash functions share a common structure of the compression function, which is shown in Figure 3.1. The compression function consists of two major parts which are the message expansion and the consecutive evaluation of a number of similar operations, called steps. These steps are usually grouped together into 3-5 rounds. After the last step of the compression function, the input chaining variables are added to the output, which complicates the inversion of the compression function.

The **Message Expansion** ensures, that each message block is used more than once during one iteration of the compression function. There are two different types of message expansions: the roundwise permutation and the recursive message expansion. In roundwise permutation the message words are not changed, but rather used in a different order in each round. The recursive message expansion was designed to increase the diffusion of the message words. Nearly all inputs of each step depend on all message words. Thus, a small change in one message word immediately effects many steps. See [Dau05] for more details on the message expansions.

In each **Step** of the compression function, a number of registers (4-8, depending on the

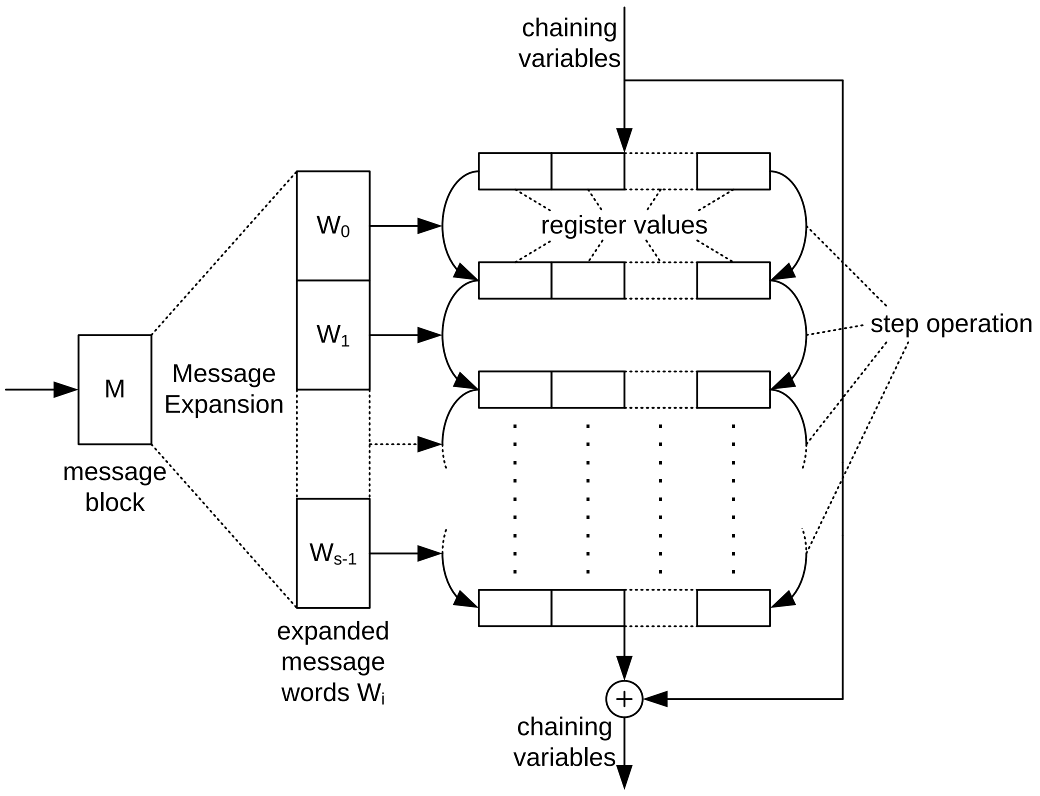


Figure 3.1: The common structure of the MD compression function.

algorithm) are updated by compressing one word of the expanded message. The operations of one step are very similar in every specific hash function and consist of the following basic operations:

- Bitwise Boolean functions
- Integer addition modulo 2^w
- Bit shifts and rotations

These operations have been chosen, because they can be efficiently evaluated and it is assumed that their combination is cryptographically strong. Each step of the compression function differs only in the use of different parameters or a different Boolean function. The hash functions of the MD-family use the following Boolean functions in their step:

Definition 3.1. (Bitwise Boolean Functions).

$$\begin{aligned}
XOR(x,y,z) &:= x \oplus y \oplus z \\
MAJ(x,y,z) &:= xy \oplus xz \oplus yz \\
IF(x,y,z) &:= xy \oplus \bar{x}z = xy \oplus xz \oplus z \\
ONX(x,y,z) &:= (x \vee \bar{y}) \oplus z = xy \oplus y \oplus z \oplus 1
\end{aligned}$$

These functions are considered to have strong cryptographic properties. See [Pre93] and [Dau05] for a detailed analysis of these Boolean functions. A differential analysis of the Boolean functions used for the attack in this theses is given in section 5.4.

3.2 MD4

MD4 was designed by Ron Rivest in 1990 and is described in [Riv90b] and [Riv90a]. It is the basis of all other hash functions of the MD-family. Because it is the main target of this thesis, this hash algorithm is explained in more detail.

3.2.1 Notation

The following notation is used to describe MD4 in this thesis:

Input Messages: One 512-bit input message block is denoted by $M = (m_0, m_1, ..., m_{15})$. Each message word m_k consists of 32 bits, which are denoted by $m_{k,j}$, where $0 \leq j \leq 31$.

Register Words: The 32-bit register words (or state variables) are denoted by a_i where i is the number of the compression step with $0 \leq i \leq 47$ and the register bits are indexed with $a_{i,j}$, $0 \leq j \leq 31$. Each register word a_i is computed according to the update rule of Equation 3.1. The four output registers after each step i are grouped to $(a_{i-3}, a_i, a_{i-1}, a_{i-2})$.

Boolean Functions: The bitwise Boolean functions used in step i is denoted by $f_i(x,y,z)$ (or short f_i). The variables x, y, z are 32-bit words and the Boolean functions IF, MAJ and XOR are applied bitwise to these words.

3.2.2 Description

The MD4 algorithm compresses an input with a maximum length of 2^{64} to a 128-bit hash value. The size of one message block in MD4 is 512 bit. The input message is padded to fit this message block size (see [Riv90a]). First, the padding scheme always appends a single "1" bit to the end of the message. Then, "0" bits are appended until the message length is congruent 448 modulo 512. Finally, the 64-bit representation of the message length before the padding was applied is appended.

Each 512-bit message block of the padded message is compressed by the compression function which consists of three rounds having 16 steps each. In each round a different Boolean function f_i is used. The IF function is used for the first, the MAJ function for the second and the XOR function for the third round (Table 3.1). Each message word m_k with $0 \leq k \leq 15$ of a message block M is added exactly once in each round. Figure 3.2 gives an overview of MD4 and shows one step in detail.

Table 3.1: The Boolean functions f_i and the constants c_i in each step of MD4.

i	f_i	c_i
$0 \ldots 15$	$IF(x,y,z)$	$0x00000000$
$16 \ldots 31$	$MAJ(x,y,z)$	$0x5a827999$
$21 \ldots 47$	$XOR(x,y,z)$	$0x6ed9eba1$

In every step of MD4, a 32-bit register value (or often called state variable) a_i is computed according to the following recursive update rule:

$$a_i = (a_{i-4} + f_i(a_{i-1}, a_{i-2}, a_{i-3}) + m_{w_i} + c_i) \lll s_i \qquad 0 \leq i \leq 47. \tag{3.1}$$

The operator $+$ denotes the addition modulo 2^{32} and the operator $\lll s_i$ denotes a circular left shift (rotation) by s_i positions. The variable m_{w_i} specifies the message word to compress. The variable c_i defines a round constant, which is the same for each step in one round (see Table 3.1). The permutation of the message words is determined by the index w_i (see Table 3.2).

Table 3.2: The permutation of the message block words in MD4.

i	w_i
$0 \ldots 15$	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
$16 \ldots 31$	0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15
$21 \ldots 47$	0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15

The number of bit positions s_i in a rotation is changed for each step but repeated four times in every round (see Table 3.3). For each message block, the update rule of the compression function is initialized by the chaining variables $(A, B, C, D) = (a_{-1}, a_{-4}, a_{-3}, a_{-2})$. The initial values for MD4 are:

$$(A, B, C, D) = (\text{0x67452301}, \text{0xefcdab89}, \text{0x98badcfe}, \text{0x10325476})$$

Table 3.3: The size of the rotation s_i in each step of MD4.

i	s_i
$0 \ldots 15$	$3,\ 7, 11, 19,\ 3,\ 7, 11, 19,\ 3,\ 7, 11, 19,\ 3,\ 7, 11, 19$
$16 \ldots 31$	$3,\ 5,\ 9, 13,\ 3,\ 5,\ 9, 13,\ 3,\ 5,\ 9, 13,\ 3,\ 5,\ 9, 13$
$21 \ldots 47$	$3,\ 9, 11, 15,\ 3,\ 9, 11, 15,\ 3,\ 9, 11, 15,\ 3,\ 9, 11, 15$

After the processing of all 48 steps, the last four register values are added to the chaining variables: $(A, B, C, D) = (A, B, C, D) + (a_{47}, a_{44}, a_{45}, a_{46})$. If no message block is processed anymore, the resulting hash value is the concatenation of the four chaining variables (A, B, C, D).

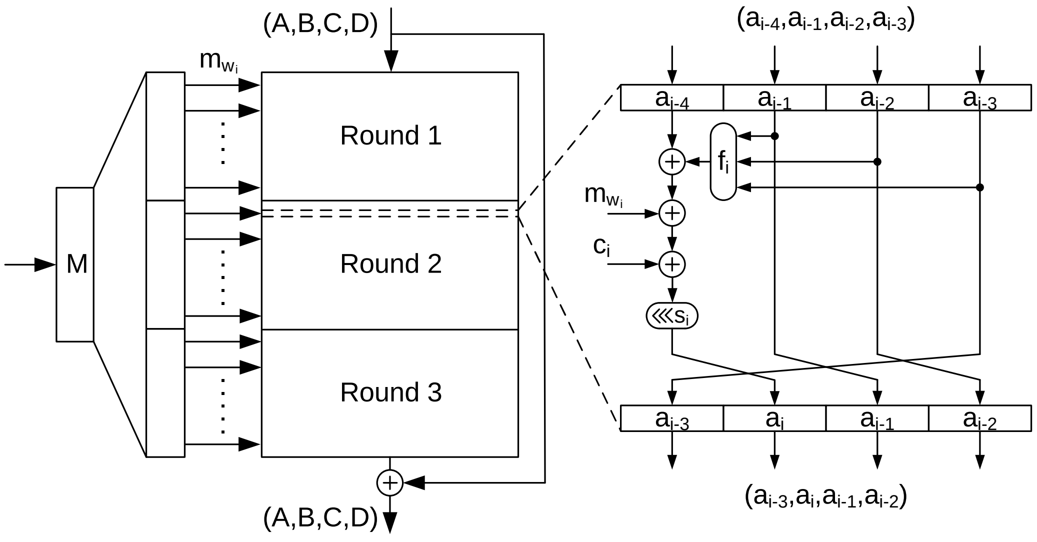


Figure 3.2: The structure of the MD4 hash function.

3.3 Other Hash Functions of the MD-Family

This section gives a short overview of the most important remaining hash functions of the MD-family.

3.3.1 MD5

MD5 is the successor of MD4 and was also designed by Ron Rivest in 1992. It is an improved version of MD4. MD5 uses the same message block (512 bits) and hash value size (128 bits) as MD4. The improvements in MD5 are:

- An additional round with the Boolean function $ONX(x, z, y)$ is used. Thus, each message word is used four times as an input for a step.
- To reduce the symmetry of the Boolean function used in the second round $MAJ(x, y, z)$ it is replaced by $IF(z, x, y)$ with swapped parameters.
- A unique additive constant is used for each step.
- The permutations of the message words were changed.
- The rotation values for each round are unique and have been optimized to maximize the avalanche effect.
- After the rotation, the register value of the previous step is added, which provides a faster avalanche effect.

3.3.2 RIPEMD and RIPEMD-160

The most important difference between MD4 and the RIPEMD variants is, that in the compression function the message words are processed in two parallel lines. RIPEMD uses four 32-bit register values and the result is a 128-bit hash value, RIPEMD-160 uses five 32-bit register values to generate a 160-bit hash value. The steps of RIPEMD are the same as in MD4 (with different parameters) and RIPEMD-160 uses a similar update rule.

3.3.3 The SHA-Family

The SHA-family (SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512), designed by the National Security Agency (NSA) descends from the MD-family of hash functions. All these hash functions use a recursive message expansion. SHA-0 and SHA-1 use the same Boolean functions in their step as the MD-functions. The difference is that their hash value is 160 bit and thus five registers are used in each step. They have further constant rotation values for each step and the four rounds have 20 steps each. SHA-1 has an improved message expansion compared to SHA-0. SHA-1 is currently used in many standards.

SHA-224, SHA-256, SHA-384 and SHA-512 use even more register values in each step. To increase the complexity of the step more than one register value is updated in each step and additional auxiliary functions are used to provide stronger inter-bit diffusion.

3.4 Attacks

This section gives a short overview of the main attacks on the MD-family of hash functions. A more detailed analysis of these attacks is given in [Dau05]. The first attempts to break the

MD-family have targeted the round functions of MD4 and MD5 and were done by Merkle (unpublished), den Boer and Bosselaers [dBB91, dBB94] and by Vaudenay [Vau94].

An overview of collision attacks on members of the MD-family is given in Table 3.4 and a short describtion of the three most important attack approaches is given in the following sections.

Table 3.4: In this table the three most important attack approaches on the MD-family are shown.

Name	hash function	complexity
Dobbertin [Dob96a, Dob98]	MD4	2^{20}
Dobbertin [Dob96b, Dob96c]	MD5	$\sim$10 hours
Dobbertin [Dob97]	RIPEMD	?
Chabaud, Joux [CJ98]	SHA-0	2^{61}
Biham, Chen [BC04]	SHA-0	2^{56}
Rijmen, Oswald [RO05]	53-round SHA-1	2^{71}
Biham/Joux *et al.* [BCJ+05]	SHA-0/58-round SHA-1	$2^{51}/2^{75}$
Wang *et al.* [WLF+05]	MD4/RIPEMD	$2^{6}/2^{16}$
Wang,Yu [WY05]	MD5	2^{39}
Wang,Yin,Yu [WYY05d]	SHA-0	2^{39}
Wang,Yin,Yu [WYY05c]	SHA-1	2^{69}
Wang,Yao,Yao [WYY05b]	SHA-1	2^{63}

3.4.1 Dobbertin

Dobbertin has developed a technique to attack the collision resistance of hash functions using differential cryptanalysis. Using a small difference in two messages, he constructs and efficiently solves an under-determined non-linear system of equations. He has applied this technique to the hash functions MD4 [Dob96a, Dob98], RIPEMD [Dob97] and MD5 [Dob96b, Dob96c].

3.4.2 Chabaud and Joux

Chabaud and Joux have replaced the non-linear operations of SHA-0 to find a *linear approximation* of the hash function. Following this approach, they have been able to show, that SHA-0 can be broken in theory [CJ98]. This approach has been extended by Biham and Chen. Using the *neutral bits technique*, they have produced near-collisions of SHA-0 [BC04]. Rijmen and Oswald have found a way to attack a reduced version of SHA-1 with 53 rounds in theory. They skip the first 20 rounds, where the IF function is used [RO05]. Biham *et al.* and Joux *et al.* have applied the neutral-bit technique to produce real collisions of SHA-0 [BCJ+05].

3.4.3 Wang et al.

Inspired by the MD5 collisions of Dobbertin, Wang *et al.* have developed another successful technique to attack the MD-family using differential cryptanalysis. Their approach is to first search for a differential path with a low probability, but then greatly increase this probability by message modification. They have first attacked MD4 [WLF$^+$05] and have extended their attack to efficiently find collisions in RIPEMD and MD5 [WY05]. Within one year more advanced attacks on SHA-0 and SHA-1 have been published by Wang, Yin and Yu in [WYY05d] and [WYY05c]. Yu *et al.* have improved the ideas of Wang *et al.* to construct a 2nd-preimage attack for chosen messages [YWZW05].

In all articles published by Wang *et al.* so far, only little insight in the details of their attack is given. Except for some small improvements and musings on their attacks, it is not possible to reproduce the full attacks yet. Therefore, the remainder of this thesis concentrates on reconstructing and improving the attack approach by Wang *et al.*, based on the MD4 hash function.

Chapter 4

The Approach by Wang et al.

This chapter outlines the approach of the attacks by Wang *et al.* They have first applied their approach to attack MD4 and have then extended it to the whole MD-family of hash functions. Because MD4 is the main target of this thesis, their approach is outlined using MD4 in this chapter as well. Their attack is a differential attack on the collision resistance of a hash function, which can be extended to chosen message 2nd-preimage attacks too. In a collision attack, two different messages M and M' need to be found, which hash to the same value. A differential attack does not use these two messages M and M' directly but is rather applied on their difference $\Delta M = M' - M$. This difference in the message words first propagates through the hash function and then results in state variable differences for each step (denoted with $\Delta a_i = a_i - a'_i$). A (global) collision is found, if the difference in the state variables after the last step is zero. Thus, the two messages M and M' will hash to the same value.

In differential cryptanalysis, the terms zero difference, differential path (or differential characteristic) and inner collision are used. The meaning of these terms vary slightly in the literature, this thesis makes use of the following definitions:

Definition 4.1 (Zero Difference)**.** A zero difference in step i contains zero differences in all four registers of the step i. Thus, the two different messages M and M' produce the same register values in this step:

$$(\Delta a_{i-3}, \Delta a_i, \Delta a_{i-1}, \Delta a_{i-2}) = (0, 0, 0, 0)$$

Definition 4.2 (Inner Collision)**.** An inner collision is a range of steps $(i_1, ..., i_n)$, with a zero difference in the step prior to the first $(i_1 - 1)$ and a zero difference in the last (i_n) step, but with nonzero differences in the intermediate steps.

Definition 4.3 (Differential Path)**.** A differential path is a specific sequence of differences in the state variables, over a given number of steps. In the remainder of this thesis a differential path is supposed to have a zero difference in its first and last step, unless otherwise stated.

The differential attack of Wang *et al.* consists of the following three major parts:

1. Find a message difference, which result in a collision with a high probability.
2. Derive a differential path for the given message difference. The results of this step are conditions on the state variables.
3. Reduce the complexity of finding messages, which fulfill all state variable conditions, by message modification techniques.

The following sections provide an overview of these three parts, while the remaining chapters of this thesis give a more detailed insight.

4.1 Deriving the Message Difference

In the first step the message difference $\Delta M = M' - M$ between two arbitrary input messages M and M' is determined. To produce collisions with high probability, message differences with a nice differential behavior in some parts of the hash function need to be found. There are several possibilities to find a suitable message difference.

In [WLF$^+$05] the whole message difference has been found by producing an inner collision with high probability in the third and last round of MD4. This message difference makes a second inner collision over the first two rounds (Figure 4.1) necessary. The message difference is fully determined by the third round. Therefore, no freedom in choosing a message difference is left to find a differential path for the second inner collision. Thus, the probability for this inner collision in round one and two is generally very low (Wang *et al.*: 2^{-122}). As message modification techniques can improve the probability for low rounds more easily (see Section 4.3), the overall message search complexity can be greatly reduced (Wang *et al.*: 2^{-2} to 2^{-6}). Note that message modification is only possible for collision attacks because the original messages is changed too.

To find inner collisions with high probability the number of steps and the Hamming weight of the differences should be kept small in general (see Chapter 5 for details). However, to find short inner collisions in the third round, Wang *et al.* uses message differences in more than one message word. Hence, it is possible to find message differences which lead to an inner collision in the third round with a high probability (2^{-2}). The advantage of the method by Wang *et al.* is, that their inner collisions are found directly for the actual steps and not for a linear approximation. The message difference presented by Wang *et al.* is:

$$\Delta m_1 = 2^{31},\ \Delta m_2 = 2^{31} - 2^{28},\ \Delta m_{12} = -2^{16}$$

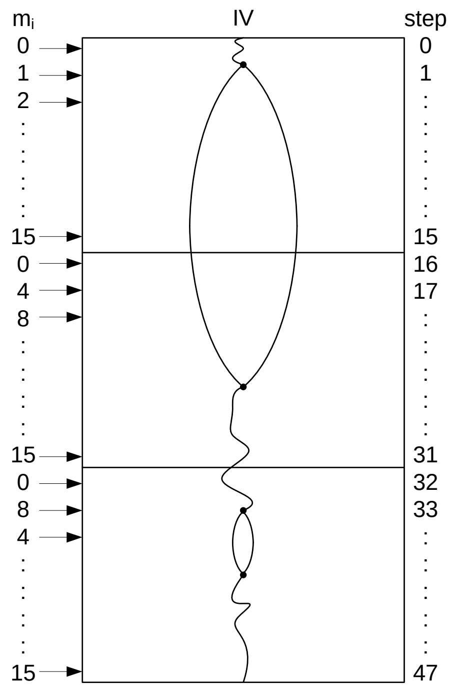


Figure 4.1: The message difference, determined by the inner collision in round three. Note, that this message difference causes a second inner collision over round one and two.

Chapter 6 shows how to derive this and other message differences in detail.

4.2 Finding a Differential Path

After the message difference has been determined, a differential path for this message difference needs to be found. This is the most difficult part and Wang *et al.* do not provide much details on how to find such a path. It is assumed that their path has been found by an ad hoc approach. Finding a differential path is the main task of this thesis and an algorithm is given in chapter 7.

The message difference used by Wang *et al.* has differences in the message words m_1, m_2, and m_{12}. These message words are used in steps 1, 2, and 12 of round one and in steps 19, 20, and 24 of round two. Thus, the differences introduced by the message words in these steps have to cancel out mutually, which then results in a differential path between step 1 and 24.

The complexity of a brute-force search through all possible paths is too high. Therefore, the approach of Wang *et al.* is to prevent uncontrolled propagation of differences and to allow the controlled propagation of differences in certain steps. The difficulty is to "know" where to allow these additional propagations.

4.2.1 Avoid Uncontrolled Propagation of Differences

To reduce the avalanche effect and thus, the search space, any uncontrolled propagation of differences through the f-function (see Section 5.4) or by carry propagation (see Section 5.3) is avoided.

Since the message difference is determined by round three, the message differences cannot be used to control the differential behavior of the f-function or the carry propagation anymore. But their differential behavior can be influenced by imposing certain conditions on the state variables. Then their differential behavior holds for all messages that fulfill these conditions. The resulting number of conditions determines the complexity of finding a suitable message. In order to reduce the resulting message search complexity low weight differences are used by default (see Chapter 7).

To find a differential path, the message difference propagation through the first two rounds of MD4 is controlled. The *modular* message differences used by Wang *et al.* suit the modular addition. Nevertheless, for the bitwise defined step operations of the MD hash functions, bitwise defined differences are more accurate. Therefore, Wang *et al.* mainly use signed bitwise defined differences but always keeps the modular differences in contrast. A detailed introduction of signed differences is given in Chapter 5.

4.2.2 Control Differences to Cancel Mutually

After avoiding uncontrolled propagation of differences, the next task is to control the remaining differences to cancel out mutually. Therefore, additional differences are introduced, which in turn will cancel unwanted differences. These additional differences are introduced by controlling the differential behavior of the f-function and the carry expansion (see Chapter 5). The main difficulty is, to determine in which step these additional differences should be introduced to result in a differential path with a high probability.

However, Wang *et al.* have never provided any details about where to introduce these additional differences. Therefore, Chapter 7 gives a successful method for introducing and controlling differences to find many differential paths.

4.3 Message Modification

The result of the second part is a differential path with conditions on the state variables, which are needed for the path to hold. If a randomly chosen message fulfills all conditions, this message leads to a collision. One condition is fulfilled with probability of $1/2$. Therefore, the overall message search complexity is $2^{\#conditions}$, under the assumption that the conditions are independent. The path published by Wang *et al.* has 125 conditions and thus the complexity is 2^{125}, which is worse than the birthday attack (2^{64}). But Wang *et al.* have provided some methods to reduce this complexity. Instead of trying one message after the other, a randomly chosen messages is adjusted to fulfill as many conditions as possible. This technique is called message modification. There are different types of message modification techniques, depending in which round and step the message is changed.

The single-step message modification technique allows to fulfill all conditions that occur in the first round. For the second round, multi-step message modification techniques are needed, because the message words of round one have to be adjusted again. The important difference between single- and multi-step message modification is, that the latter may not always succeed because contradictions can occur. Wang *et al.* have further used advanced multi-step message modification techniques to bypass some contradictions. Thus, most conditions of the second round can be fulfilled and the overall complexity reduced to 2^{-6}. [NSKO05] have extended this idea and have been able to almost fulfill all conditions. They claimed, that a random message can be modified to produce a collision with probability $^{7}/_{8}$. A detailed treatment of message modification techniques is given by Chapter 8.

Note that in general it gets harder to fulfill conditions for higher steps. Therefore, it is desirable for a path search algorithm to find differential paths, having the majority of conditions in the first round or at the beginning of the second round.

Chapter 5

Technical Methods

This chapter provides some technical methods, which are used in the following chapters to attack MD4. Because signed differences are used in the attack, these differences are defined first. Their behavior in the addition and rotation is considered in the following section. Then the propagation of signed differences by carry expansion, and the differential behavior of the Boolean function f_i is analyzed.

5.1 Signed Differences

In differential cryptanalysis of hash functions, differences in the message and register words are used. There are many types of differences, the most common are modular differences and bitwise (XOR) differences. In the differential attack of Wang *et al.*, modular differences are used for the message words and signed differences for the register values. As stated in the previous chapter, modular differences cannot be used easily to control the behavior of the (bitwise defined) Boolean functions. Therefore, in the following chapters signed differences that are defined bitwise are used. Because mainly signed differences are used in this thesis, they are often referred to as differences:

Definition 5.1 (Signed Difference)**.** The signed difference Δx between two 32-bit words x and x' is defined bitwise by

$$\Delta x = x' - x = (\delta x_{31}, \ldots, \delta x_0) \text{ with } \delta x_j = x'_j - x_j \in \{\text{-}1, 0, 1\},\ 0 \leq j \leq 31,$$

where δx_j is the value of the difference at position j and called *single bit difference.*

Definition 5.2 (Hamming Weight)**.** The Hamming weight $w_H(\Delta x) = \sum_{j=0}^{31} |\delta x_j|$ is the number of non-zero elements in the signed difference Δx.

In many cases the Hamming weight $w_H(\Delta x)$ of a signed difference Δx is small and thus, most of its elements δx_j are zero. Therefore, a short notation is defined which contains only non-zero single bit differences:

Definition 5.3 (Short Notation)**.** To abbreviate the representation of Δx with Hamming weight $w = w_H(\Delta x)$ we use the following short notation:

$$\Delta x = \Delta[d_1, d_2, ..., d_w] \quad \text{where} \quad d_i = \begin{cases} \text{-}j & \text{if } \delta x_j = \text{-}1 \\ j & \text{if } \delta x_j = 1 \end{cases}$$
$$\text{and} \quad \delta x_j = 0 \text{ if } j \notin \{|d_1|, |d_2|, ..., |d_w|\}$$

where $|d_i|$ is the bit position j and $sign(d_i) \in \{\text{-}1, 0, 1\}$ the sign of the difference δx_j.

In this notation the zero difference $\Delta x = 0$ can be represented as $\Delta[]$ and has the Hamming weight $w_H(\Delta[]) = 0$. A single bit difference at the least significant bit x_0 is denoted by $\Delta[\text{-}0]$ if $\delta x_0 = -1$ or $\Delta[0]$ if $\delta x_0 = 1$.

Note, that the representation of a modular difference with these signed differences is redundant because

$$2^i = \Delta[i] = \Delta[i+1, \text{-}i] = 2^{i+1} - 2^i,$$

but every signed difference Δx can be converted into a unique modular difference by the following equation (see [Dau05]):

$$\Delta x = \Delta[d_1, d_2, ..., d_w] = \sum_{i=1}^{w} sign(d_i) \cdot 2^{|d_i|} \bmod 2^{32} \tag{5.1}$$

An important fact about signed differences is, that non-zero differences δx_j of $\Delta x = x' - x$ already determine some values of the corresponding bits in x and x':

$$\Delta x = \Delta[d_1, d_2, ..., d_w] \quad \Rightarrow \quad x_{|d_i|} = \begin{cases} 0 & \text{if } sign(d_i) = 1 \\ 1 & \text{if } sign(d_i) = \text{-}1. \end{cases}$$
$$\text{and}$$
$$x'_{|d_i|} = \begin{cases} 1 & \text{if } sign(d_i) = 1 \\ 0 & \text{if } sign(d_i) = \text{-}1. \end{cases}$$

Thus it is impossible to have a single bit difference of $\delta x_j = 1$ if $x_j = 1$ (or to have $\delta x_j = \text{-}1$ if $x_j = 0$). Hence, a signed difference Δx imposes conditions on the value x. In some cases not these conditions, but the probability for a signed difference Δx to occur is needed:

Lemma 5.1 (Probability of a Signed Difference)**.** *We assume that the conditions for a signed difference Δx with Hamming weight $w = w_H(\Delta x)$, are independent. Then, the probability for*

Δx to occur is (see [Dau05]):

$$Pr(\Delta x) = Pr(\Delta[d_1, d_2, ..., d_w]) = 2^{-w}$$

Example 5.1. The following difference Δx can be represented in different ways:

$$\begin{aligned}\Delta x = x' - x = -2^{27} + 2^{15} - 2^{3} = \Delta[\text{-}27, 15, \text{-}3] = \\ = (0,0,0,0,\text{-}1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,\text{-}1,0,0,0)\end{aligned}$$

This difference implies that $x_{27} = 1$, $x_{15} = 0$, $x_3 = 1$ and $x'_{27} = 0$, $x'_{15} = 1$, $x'_3 = 0$. The probability of this difference is therefore $Pr(\Delta[\text{-}27, 15, \text{-}3]) = 2^{-3}$.

5.2 Basic Operations on Signed Differences

Signed differences are used throughout the whole differential analysis of this thesis. Thus, a detailed analysis of the basic step operations (addition and rotation) for signed differences is given in this section. [Dau05] gives a definition and analysis of these step operations for some other types of differences too.

5.2.1 Addition

Every step of the MD4 hash function consists of three modular additions. In a signed differential analysis, the inputs for each addition are signed differences. As the representation of these differences is redundant the result of an addition is redundant as well. The following algorithm provides a straight forward addition of signed differences, whereas other representations can be found by carry expansion, which will be discussed in detail in Section 5.3.

Algorithm 5.1 (Addition of Signed Differences). The addition of the two signed differences Δx and Δy is defined signed and bitwise for each bit $j = 0, ..., 31$, where s_j denotes the sum and c_j the carry after each bit addition:

$$\Delta z = \Delta x + \Delta y = (\delta x_{31}, ..., \delta x_0) + (\delta y_{31}, ..., \delta y_0) = (s_{31}, ..., s_0)$$

$$\text{where } s_j = \begin{cases} \text{-}1 & \text{if } \delta x_j + \delta y_j + c_j \in \{\text{-}3, \text{-}1\} \\ 0 & \text{if } \delta x_j + \delta y_j + c_j \in \{\text{-}2, 0, 2\} \\ 1 & \text{if } \delta x_j + \delta y_j + c_j \in \{1, 3\} \end{cases}$$

$$\text{where } c_0 = 0, c_{j+1} = \begin{cases} \text{-}1 & \text{if } \delta x_j + \delta y_j + c_j \in \{\text{-}3, \text{-}2\} \\ 0 & \text{if } \delta x_j + \delta y_j + c_j \in \{\text{-}1, 0, 1\} \\ 1 & \text{if } \delta x_j + \delta y_j + c_j \in \{2, 3\} \end{cases}$$

Throughout the remainder of this thesis mainly the abbreviated notation (see Definition 5.3) will be used. The addition can be performed using this short notation as well. In the following an algorithm for adding two signed differences with Hamming weight $w = 1$ is presented.

Algorithm 5.2 (Addition of Signed Differences in Short Notation). When adding two signed differences $\Delta x = \Delta[dx_1]$ and $\Delta y = \Delta[dy_1]$ with Hamming weight $w_H(\Delta x) = w_H(\Delta y) = 1$ one of the following four cases occurs:

$$\Delta[dx_1] + \Delta[dy_1] = \begin{cases} \Delta[] & \text{if} \quad dx_1 = \text{-}dy_1 \\ \Delta[dx_1 + 1] & \text{if} \quad dx_1 = dy_1 \text{ and } sign(dx_1) = 1 \\ \Delta[\text{-}(|dx_1| + 1)] & \text{if} \quad dx_1 = dy_1 \text{ and } sign(dx_1) = \text{-}1 \\ \Delta[dx_1, dy_1] & \text{otherwise,} \end{cases}$$

A signed bit difference at position 32 is always discarded because the addition is performed modulo 2^{32}.

Example 5.2. The following example illustrates the four different cases in the addition of two signed differences with Hamming weight $w = 1$:

$$\begin{aligned} &\Delta[24] + \Delta[\text{-}24] = \Delta[] \\ &\Delta[24] + \Delta[24] = \Delta[25] \\ &\Delta[\text{-}24] + \Delta[\text{-}24] = \Delta[\text{-}25] \\ &\Delta[\text{-}24] + \Delta[25] = \Delta[25, \text{-}24] \end{aligned}$$

Example 5.3. In this example two differences with Hamming weight $w > 1$ are added in the short notation of signed differences. In the addition of

$$\Delta[31, \text{-}27, 15, \text{-}0] + \Delta[31, 27, 16, 15, 4] = \Delta[17, 4, \text{-}0]$$

carries at position $15, 16$ and 31 occur. The carry at position 31 is discarded. The addition performed in power notation yields:

$$\begin{array}{ccccccccc} & & 2^{31} & -2^{27} & & & +2^{15} & & -2^{0} \\ + & & 2^{31} & +2^{27} & & +2^{16} & +2^{15} & +2^{4} & \\ \hline \text{carry} & 2^{32} & & & +2^{17} & +2^{16} & & & \\ \hline\hline = & & & & +2^{17} & & & +2^{4} & -2^{0} \end{array}$$

5.2.2 Rotation

Because each step contains a rotation, signed differences need to be rotated as well. The result after an addition is redundant, whereas this is not true for rotating a signed difference.

Algorithm 5.3 (Rotation of Signed Differences). To rotate the 32-bit signed difference Δx by s bits, each element δx_j is rotated as:

$$\delta x_j \lll s = \delta x_{(j+s) mod\, 32}$$

A rotation can be performed in the short notation using the following algorithm:

Algorithm 5.4 (Rotation of Signed Differences in Short Notation). The signed difference $\Delta x = \Delta[d_1, d_2, ..., d_w]$ is rotated in the short notation element-wise by s bits with:

$$\Delta[\tilde{d_i}] \lll s = \Delta[d_i] \text{ where } d_i = \begin{cases} d_i + s \bmod 32 & \text{if } sign(d_i) = 1 \\ \text{-}(|d_i| + s \bmod 32) & \text{if } sign(d_i) = \text{-}1 \end{cases}$$

Example 5.4. The signed difference $\Delta x = \Delta[31, \text{-}27, \text{-}3]$ is rotated by 9 bits. The result is again sorted by descending bit position:

$$\begin{aligned} \Delta x \lll s =& \Delta[31, \text{-}27, \text{-}3] \lll 9 \\ =& \Delta[(31 + 9 \bmod 32), \text{-}(27 + 9 \bmod 32), \text{-}(3 + 9 \bmod 32)] = \\ =& \Delta[9, \text{-}4, \text{-}12] = \Delta[\text{-}12, 9, \text{-}4] \end{aligned}$$

5.3 Carry Expansion

Because the representation of signed differences is redundant, every (non-zero) element d_i of a signed difference can be expanded as described in the following. Note that differences at position 32 are discarded.

$$\Delta[d_1, ..., d_i, ..., d_w] = \begin{cases} \Delta[d_1, ..., \text{-}d_i, ..., d_w] & \text{if } |d_i| = 31 \\ \Delta[d_1, ..., \text{-}d_i, ..., d_w] + \Delta[d_i] \lll 1 & \text{if } |d_i| \neq 31 \end{cases} \quad (5.2)$$

To get all possible representations for a signed difference Δx, Equation 5.2 needs to be applied recursively on each element d_i. The maximum number of expansion steps to allow and thus the recursion depth is called *additional carries.*

Example 5.5. In this example the difference $\Delta x = \Delta[\text{-}11, 9]$ is expanded. The maximum number of expansion steps performed in every recursion branch, and thus the number of *additional carries*, is 2. The expanded element is marked with $\overleftarrow{d_i}$:

$$\begin{aligned}\Delta x \to \Delta[\text{-}11, \overleftarrow{9}\,] &\to \Delta[\text{-}11, \overleftarrow{10}, \text{-}9] \to \Delta[\text{-}10, \text{-}9] \\ &\to \Delta[\text{-}\overleftarrow{11}, 10, \text{-}9] \to \Delta[\text{-}12, 11, 10, \text{-}9] \\ \to \Delta[\text{-}\overleftarrow{11}, 9] &\to \Delta[\text{-}\overleftarrow{12}, 11, 9] \to \Delta[\text{-}13, 12, 11, 9] \\ &\to \Delta[\text{-}12, \overleftarrow{11}, 9] \to \Delta[\text{-}12, 11, 10, \text{-}9]\end{aligned}$$

Hence, all representations for Δx with a maximum carry expansion of two, sorted by their hamming weight, are:

$$\begin{aligned}\Delta x &= \Delta[\text{-}11, 9] = \Delta[\text{-}10, \text{-}9] \\ &= \Delta[\text{-}11, 10, \text{-}9] = \Delta[\text{-}12, 11, 9] \\ &= \Delta[\text{-}12, 11, 10, \text{-}9] = \Delta[\text{-}13, 12, 11, 9]\end{aligned}$$

An expanded signed difference can be reduced to an equivalent difference with less weight again. However, usually a *unique* reduced signed difference with minimum weight does not exist because:

$$\Delta[j+1, j] = \Delta[j+2, \text{-}j]$$

Remark 5.1. (Expand-Rotate-Reduce) If a signed difference Δx with $w = w_H(\Delta x)$ is rotated after the expansion, it cannot always be reduced to a difference with weight w again. In general, this reduction is not possible, if some expanded part of a single bit difference $|d_i|$ is rotated over position 31. For a detailed analysis of carry expansion with rotations see [Dau05].

Example 5.6. This example shows that the difference $\Delta[12]$ cannot be reduced to a difference with weight $w = 1$, if the expanded part is rotated over position 31:

$$\Delta[12] \lll 19 = \Delta[13, \text{-}12] \lll 19 = \Delta[\text{-}31, 0] \neq \Delta[31] = \Delta[12] \lll 19.$$

5.4 f-Propagation

In this section the propagation of signed differences through the Boolean functions IF, MAJ and XOR is discussed. Wang *et al.* give only some basic signed properties about these functions in [WLF$^+$05] and Magnus Daum provides an analysis based on probabilities for xor and signed difference in his thesis [Dau05]. However, for the attack in this thesis, the complete differential behavior which can be influenced by conditions is needed.

The propagation of differences through the Boolean functions can be controlled by imposing certain conditions on the input values. In order to completely control this propagation, the signed differential behavior for *all* cases is provided by Table 5.1. In this table, all possible combinations of input and output differences with their respective conditions are shown. In the following, some useful properties of the Boolean functions are listed as well.

5.4.1 IF

The output difference of the IF function can be manipulated for the majority of all input cases. However, if a zero output difference is desired, consecutive ones in the input differences should be avoided. Note, that IF is the only Boolean function, which can flip a *single* input difference at the first input δx of $IF(x, y, z)$. The disadvantage of this behavior is, that *two* conditions are needed to control the propagation of the single input difference δx.

5.4.2 MAJ

The output difference of the MAJ function can only be influenced by imposing conditions, if the number of input differences is exactly one (cf. Table 5.1). However, another trick to influence the differential behavior of MAJ, is to control the number of input differences and their sign. Thus, to achieve a zero difference at the output, exactly two differences with opposite sign can be used as the input:

$$\delta MAJ(\delta x, \text{-}\delta x, 0) = \delta MAJ(0, \delta x, \text{-}\delta x) = \delta MAJ(\delta x, 0, \text{-}\delta x) = 0$$

5.4.3 XOR

The XOR function has a differential behavior similar to that of the MAJ function. The output can only be influenced by imposing conditions, if exactly one input difference is not zero. Note, that it is never possible to achieve a zero output difference by imposing conditions. A zero difference at the output of XOR occurs if and only if exactly two (or none) of the input differences are not zero:

$$\delta XOR(|\delta x|, |\delta x|, 0) = \delta XOR(|\delta x|, 0, |\delta x|) = \delta XOR(0, |\delta x|, |\delta x|) = 0$$

Table 5.1: Signed differential behavior of f_i with necessary conditions, probability 1 or "-" for an impossible output.

$\delta x \delta y \delta z$	δIF=0	δIF=1	δIF=-1	δMAJ=0	δMAJ=1	δMAJ=-1	δXOR=0	δXOR=1	δXOR=-1
0 0 0	1	-	-	1	-	-	1	-	-
1 0 0	$y = z$	$y = 1, z = 0$	$y = 0, z = 1$	$y = z$	$y \neq z$	-	-	$y = z$	$y \neq z$
-1 0 0	$y = z$	$y = 0, z = 1$	$y = 1, z = 0$	$y = z$	-	$y \neq z$	-	$x \neq y$	$y = z$
0 1 0	$x = 0$	$x = 1$	-	$x = z$	$x \neq z$	-	-	$x = z$	$x \neq z$
0 -1 0	$x = 0$	-	$x = 1$	$x = z$	-	$x \neq z$	-	$x \neq y$	$x = z$
0 0 1	$x = 1$	$x = 0$	-	$x = y$	$x \neq y$	-	-	$x = y$	$x \neq y$
0 0 -1	$x = 1$	-	$x = 0$	$x = y$	-	$x \neq y$	-	$x \neq y$	$x = y$
1 0 -1	$y = 1$	-	$y = 0$	1	-	-	1	-	-
-1 0 1	$y = 1$	$y = 0$	-	1	-	-	1	-	-
1 0 1	$y = 0$	$y = 1$	-	-	1	-	1	-	-
-1 0 -1	$y = 0$	-	$y = 1$	-	-	1	1	-	-
1 -1 0	$z = 0$	-	$z = 1$	1	-	-	1	-	-
-1 1 0	$z = 0$	$z = 1$	-	1	-	-	1	-	-
1 1 0	$z = 1$	$z = 0$	-	-	1	-	1	-	-
-1 -1 0	$z = 1$	-	$z = 0$	-	-	1	1	-	-
0 1 -1	-	$x = 1$	$x = 0$	1	-	-	1	-	-
0 -1 1	-	$x = 0$	$x = 1$	1	-	-	1	-	-
0 1 1	-	1	-	-	1	-	1	-	-
0 -1 -1	-	-	1	-	-	1	1	-	-
1 1 1	-	1	-	-	1	-	-	1	-
-1 1 -1	1	-	-	-	-	1	-	1	-
-1 -1 1	1	-	-	-	-	1	-	1	-
1 -1 -1	-	-	1	-	-	1	-	1	-
-1 1 1	-	1	-	-	1	-	-	-	1
1 1 -1	1	-	-	-	1	-	-	-	1
1 -1 1	1	-	-	-	1	-	-	-	1
-1 -1 -1	-	-	1	-	-	1	-	-	1

Chapter 6

Finding a Message Difference

The differential attacks that are discussed in this thesis start by defining the message difference first. Therefore, the choice of this difference is crucial for the success of the attack. The message difference determines, how difficult it is to find a differential path for the hash function. It further influences the probability, that a random message leads to a collision using this message difference.

To increase the success probability of an attack the overall number of conditions and thus, the number of conditions on each state variable, should be kept small. This can be achieved by using low weight differences for each state variable. Because message differences become state variable differences, message differences with a low weight are used in many attacks too.

Prior to choosing a message difference its impact on the register values is analyzed. The following lemma is an immediate consequence of the update rule of Equation 3.1 ($W_i = m_{w_i}$):

Lemma 6.1 (Message Difference and Step Difference)**.** *If the step difference in some step i $(\Delta a_{i-3}, \Delta a_i, \Delta a_{i-1}, \Delta a_{i-2}) = (0, 0, 0, 0)$, then the following two statements hold (cf. [Dau05]):*

$$\Delta a_{i+1} = 0 \Longleftrightarrow \Delta W_{i+1} = 0 \textit{ and}$$
$$\Delta a_{i-4} = 0 \Longleftrightarrow \Delta W_i = 0.$$

Hence, an inner collision (see Definition 4.2) can only begin or end in a step, where a message difference is introduced:

Proposition 6.1 (Message Difference and Inner Collision)**.** *For every inner collision over the successive range of steps $(i_1, ..., i_n)$, $\Delta W_{i_1} \neq 0$ and $\Delta W_{i_n} \neq 0$ holds.*

Note, that an inner collision cannot end in the same step i, where another inner collision begins. Otherwise, a zero difference would occur in step $i-1$ and in step i. Because a message difference is introduced in step i, this contradicts Lemma 6.1.

Finally, to find a collision in the MD4-compression function, the differences in the registers before the first and after the last step must be zero:

$$(\Delta a_{\text{-}4}, \Delta a_{\text{-}1}, \Delta a_{\text{-}2}, \Delta a_{\text{-}3}) = (\Delta a_{44}, \Delta a_{47}, \Delta a_{46}, \Delta a_{45}) = (0, 0, 0, 0)$$

6.1 Single Bit Message Difference

Dobbertin [Dob96a] and Yu *et al.* [YWZW05] use a one bit message difference in their attack. Because this is the lowest possible message difference, it is assumed that only a few disturbances are caused.

In MD4 a one bit message difference in message word m_k is introduced exactly three times. Exactly one inner collision occurs between the first and the last introduction of m_k. Hence, a differential path for this inner collision can be found (see Figure 6.1-b). Note, that the search cannot be divided by producing two separate inner collisions (Figure 6.1-a). The reason is that the first inner collision would have to end where the second inner collision begins.

Dobbertin uses the message difference $\Delta m_{12} = \Delta[0]$ to find an inner collision. The idea of Dobbertin is to subdivide the work further, by first finding an *inner almost collision* in step 19, at the second introduction of m_{12} (Figure 6.1-c). An inner almost collision, is a step difference with low Hamming weight and the state variable differences of this inner almost collision are carefully chosen to be:

$$(\Delta a_{16}, \Delta a_{19}, \Delta a_{18}, \Delta a_{17}) = (\Delta[], \Delta[25], \Delta[\text{-}5], \Delta[])$$

This difference propagates to step 35 and cancels the message difference Δm_{12} in this step with high probability. Then, Dobbertin constructs and solves a nonlinear system of equations for the first part of the inner collision (steps $12 - 19$). He finds messages that lead to a collision of the compression function with a complexity of about 2^{20} evaluations of MD4.

Yu *et al.* have found a one bit message difference and an according differential path for this inner collision too. (see Figure 6.1-b). They do not provide any details on how the path was found. Yu *et al.* have selected the one bit message difference $\Delta m_4 = \Delta[\pm d]$ with $0 \leq d \leq 31$. Note, that their differential path can be rotated and holds for most values of d because very short carry expansions are used (cf. Remark 5.1). For all remaining values of d, Yu *et al.* stated that they found a similar path. The complexity of finding a message which suits that path is stated to be 2^{56}.

The advantage of a single bit message difference is, that a differential path with low complexity can be found more easily because less differences need to be considered. The disadvantage is, that the conditions for this differential path are spread over all three rounds and message modification techniques cannot easily fulfill conditions of higher rounds.

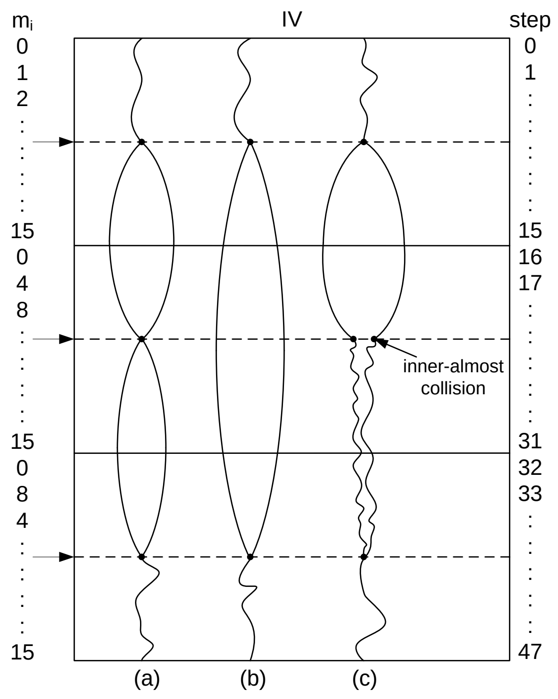


Figure 6.1: All inner collisions with a message difference in exactly one message bit. The first case (a) with two inner collisions is not possible, case (b) was used by Yu *et al.* and Dobbertin divided the work by producing an inner almost collision (c).

6.2 Multiple Message Difference

To improve the impact of the message modification, Wang *et al.* have derived a different message difference. The resulting differential path has most of its conditions at the beginning and only a few conditions at the end of MD4. The idea is to use differences in more than one message word. Hence, it is possible to produce more than one inner collision.

6.2.1 Possible Inner Collisions

Figure 6.2 shows all types of inner collisions in MD4 with a message difference in exactly two message words. The best choice is probably to find differential paths for three inner collisions (Figure 6.2-a) because these paths are short and may contain less conditions. Because different Boolean functions, different rotation values and a different order of message words are used in each round, it is difficult to find a message difference that behaves nicely in every round.

Therefore, Wang *et al.* concentrated on the third case (Figure 6.2-c). The advantage of this case is that most conditions are expected in the first two rounds and only a few in the last. This is suited best for message modification, because conditions in lower rounds can be eliminated more easily. To increase the probability of the inner collision in the third

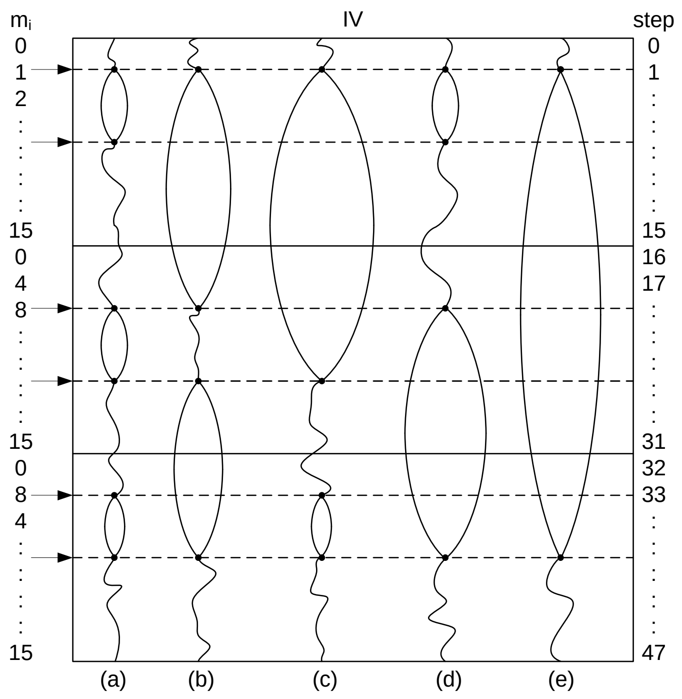


Figure 6.2: All possible inner collisions with a difference in exactly two message words.

round, Wang *et al.* have used a third difference in another message word. Further, they have utilized some special differential behavior of the XOR-function and the modular addition. The message difference is completely determined by the inner collision of the third round. Thus, the difference suits the parameters of this round best. However, a differential path for the larger inner collision of round one and two is harder to find (see Chapter 7).

6.2.2 An Inner Collision with High Probability in the third Round

Two simple observations are exploited in order, to produce an inner collision with a high probability in the third round. The first observation is that the unsigned differential behavior of the XOR function cannot be influenced. In other words: no conditions are needed to control the XOR function. However, no output difference occurs at the XOR function if and only if two (or zero) input values of the XOR function have a difference. An output difference always occurs, if and only if one or three input differences are nonzero (cf. Section 5.4.3). Second, when adding two differences at position 31 the result is always the zero difference, independent of the differences' sign because the addition is performed modulo 2^{32}:

$$\Delta[31] + \Delta[31] = \Delta[\text{-}31] + \Delta[\text{-}31] = \Delta[\text{-}31] + \Delta[31] = \Delta[]$$

Table 6.1: Propagation of the differences in the third round of MD4 according to the update rule $a_i = (a_{i\text{-}4} + XOR(a_{i\text{-}3}, a_{i\text{-}2}, a_{i\text{-}1}) + m_{w_i} + c_i) \lll s_i$:

Step	Δa_i	$\sum$	Δm_{w_i}	ΔXOR	$\Delta a_{i\text{-}1}$	$\Delta a_{i\text{-}2}$	$\Delta a_{i\text{-}3}$	$\Delta a_{i\text{-}4}$
i−1	0	0	0	0	0	0	0	0
i	$\Delta[31]$	$\Delta[31\text{-}s_i]$	$\Delta[31\text{-}s_i]$	0	0	0	0	0
i+1	$\Delta[31]$	$\Delta[31\text{-}s_{i+1}]$	$\Delta[31, 31\text{-}s_{i+1}]$	$\Delta[31]$	$\Delta[31]$	0	0	0
i+2	0	0	0	0	$\Delta[31]$	$\Delta[31]$	0	0
i+3	0	0	0	0	0	$\Delta[31]$	$\Delta[31]$	0
i+4	0	0	0	0	0	0	$\Delta[31]$	$\Delta[31]$
i+5	0	0	$\Delta[31]$	$\Delta[31]$	0	0	0	$\Delta[31]$
i+6	0	0	0	0	0	0	0	0

Therefore, in the following analysis only absolute values need to be considered, because each sign can be chosen arbitrarily.

In the following, the differential propagation in each step is analyzed. Suppose, that the first difference of round three is introduced in step i by the message word m_{w_i}. Thus, there is a zero difference in step $i-1$. The difference Δm_{w_i} is chosen, such that after the rotation of s_i steps, the state variable difference is $\Delta a_i = \Delta[31]$. Therefore, the message difference should be $\Delta m_{w_i} = \Delta[31\text{-}s_i]$. The probability that a carry occurs during the addition is $1/2$. However, by predetermining the sign of the message difference and setting an according condition on the resulting state variable, a carry can be avoided.

In the following step $(i+1)$, the difference of $\Delta a_i = \Delta[31]$ propagates through the XOR function. In order to cancel it, the difference of $m_{w_{i+1}}$ is chosen to be $\Delta[31]$. To exploit the first observation of the previous paragraph, another difference at position $31\text{-}s_{i+1}$ in the message word $m_{w_{i+1}}$ is used. Thus, a second input difference of $\Delta[31]$ for the XOR function is obtained after the rotation by s_{i+1} positions. The probability of a carry to occur is again $1/2$. By imposing a condition on a_{i+1} the sign of the message difference at position $31\text{-}s_{i+1}$ is fixed. Note, that no carry occurs at position 31 and thus, a further condition is avoided.

In the $(i+2)$-nd step, two input differences of the XOR function are set at position 31 which cancel each other. Hence, the difference in a_{i+2} is zero. The same holds for step $i+3$. In step $i+4$, only one input difference of the XOR function ($\Delta a_{i+1} = \Delta[31]$) is not zero and the difference propagates through XOR, but gets cancelled by $\Delta a_i = \Delta[31]$.

Consequently, in the $(i+5)$-th step, the difference $\Delta[31]$ of the register a_{i+1} needs to be cancelled. This is done by inserting a difference at the same position using the message word $m_{w_{i+5}}$. The differential behavior of this inner collision is summarized in Table 6.1. No condition is needed for the XOR function and only two conditions are needed to avoid the carry propagations in step i and in step $i+1$. Thus the probability for this differential path to hold is $1/4$. The signs of all message differences can be chosen independently and the conditions must be set accordingly.

The first step i of this inner collision can be chosen freely, as long as the collision remains in the third round. The choice of i determines in which message words the differences are introduced. The used message words influence the length and the position of the inner collision over the first two rounds.

Choosing for instance $i = 32$ determines that the differences are introduced by the message words m_0, m_8 and m_{10}. Message word m_0 is used in the 0-th and the 16-th step, m_8 is used in the 8-th and the 18-th step and m_{10} is used in step 10 and 26. The number of steps between the first and the last introduction of a message difference, and thus the length of this inner collision would be 27. The choice $i = 35$ leads to an inner collision between step 1 and 24. This message difference is used by Wang *et al.* because it ends in the lowest possible step. Thus, most conditions are supposed to occur in low steps too. This message difference is used in this thesis as well and the signs of the differences are:

$$\Delta m_1 = 2^{31},\ \Delta m_2 = 2^{31} - 2^{28},\ \Delta m_{12} = -2^{16}$$

Chapter 7

Searching for Differential Paths

The main part of this thesis has been to determine a differential path for the message difference of Wang *et al.* (see Section 4.1). This is the most difficult part because no insight in their methods was provided. However, it has been possible to find an algorithm, that finds many differential paths within a couple of minutes on an ordinary PC. The methods in this algorithm can be extended to search for differential paths for many other message differences as well. Further, the algorithm is able to search for paths with nonzero differences at the beginning and at the end too.

In the first section of this chapter, the algorithm is described in detail. Then some experiments and results are discussed and finally, the best differential path which has been found is presented.

7.1 An Algorithm for Searching Differential Paths

In Section 6.2.2, the message difference ΔM has been selected as:

$$\Delta m_1 = 2^{31},\ \Delta m_2 = 2^{31} - 2^{28},\ \Delta m_{12} = -2^{16}$$

These message differences are introduced in steps 1, 2, and 12 of round one and in steps 19, 20, and 24 of round two. Thus, in order to derive a differential path for MD4, the differences between step 0 and 24 have to cancel mutually. The complexity of a brute-force search through all possible paths is extremely high and the number of paths is estimated by:

$$25^{2^{19} \cdot 2^{32}},$$

because any combination of carry expansions and f_i propagations is possible in each of the 25 steps. The f_i-function can block or propagate each bit position independently, which yields

2^{32} possibilities, and the average number of possible representations of a signed difference Δa_i is estimated as:

$$\frac{\#\text{signed differences}}{\#\text{modular differences}} = \frac{3^{32}}{2^{32}} \sim 2^{19}.$$

To reduce the search space of an algorithm, any uncontrolled propagation of differences through the f_i function and by carry propagation is avoided (see section 5.4). Further, low-weight signed differences are used by default which reduces the resulting number of conditions and thus, the final message search complexity (see Chapter 8).

The algorithm consists of three major parts which are called *deriving the target differences* (see Section 7.1.1), *cancellation search* (see Section 7.1.2), and *correction step* (see Section 7.1.3). An overview is given by Figure 7.1.

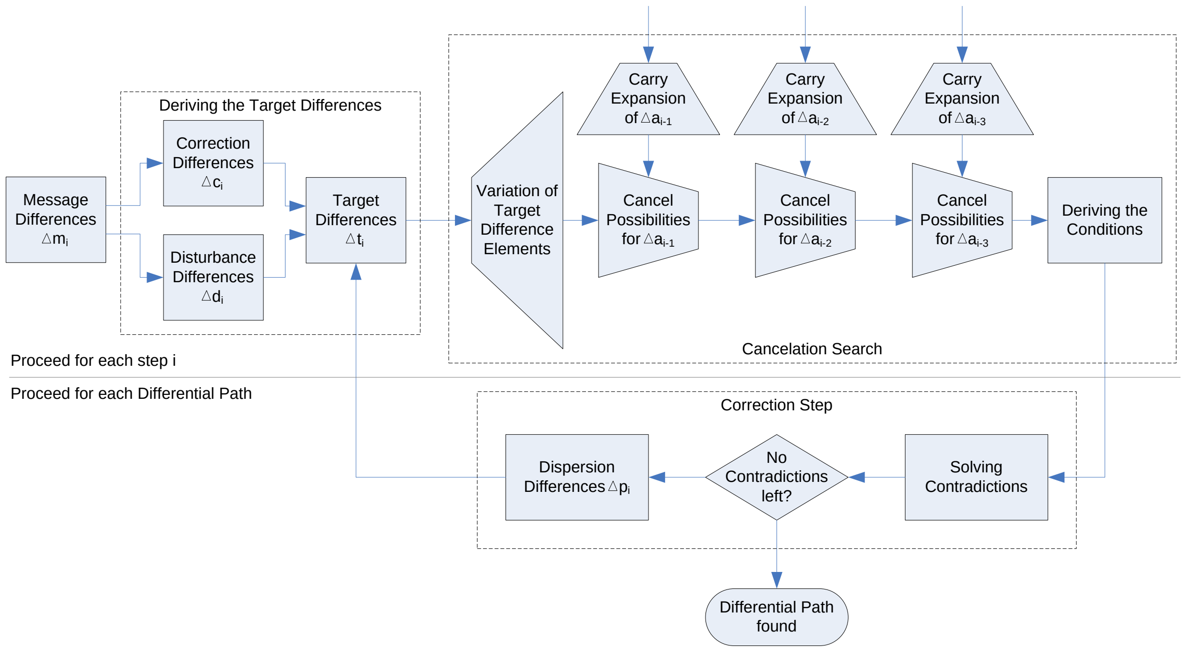

Deriving the Target Differences
Message Differences △m_i
Correction Differences △c_i
Disturbance Differences △d_i
Target Differences △t_i
Carry Expansion of △a_{i-1}
Carry Expansion of △a_{i-2}
Carry Expansion of △a_{i-3}
Variation of Target Difference Elements
Cancel Possibilities for △a_{i-1}
Cancel Possibilities for △a_{i-2}
Cancel Possibilities for △a_{i-3}
Deriving the Conditions
Cancelation Search
Proceed for each step i
Proceed for each Differential Path
Correction Step
Dispersion Differences △p_i
No Contradictions left?
Solving Contradictions
Differential Path found

Figure 7.1: An overview of the differential path search algorithm. Trapezoids represent expansions and restrictions of the search space.

In the first part the target differences are derived, which are target output differences of the f_i-function. The message differences are computed backward and forward to derive the so-called correction and disturbance differences, which are then combined to build the target differences.

During the cancellation search of each step i, all variations of the elements of the target difference are considered. The variations of the target differences need to be cancelled by the function f_i. To achieve an output difference of f_i at a specific bit position, the input differences Δa_{i-1}, Δa_{i-2} and Δa_{i-3} need to be expanded. To reduce the complexity, only possible cancellations for each input difference are considered. Finally, the conditions for each step are derived.

In the correction step, conditions which contradict are resolved without searching for a new differential path. If some contradictions cannot be corrected, so-called dispersion differences are added to the target differences. These dispersion differences distribute the conditions, such that a new differential path with no contradicting conditions can be found.

7.1.1 Deriving the Target Differences

The goal of an algorithm for finding a differential path is to cancel out all differences that are introduced by the message words. Differences in MD4 can only be cancelled or introduced by a message word difference or by a difference of the f_i-function. Because the message difference has already been determined, the only possibility to cancel a difference is by using the f_i-function. One state variable is updated in each step (cf. Equation 3.1). Thus, a message difference can only be cancelled every fourth step. A message difference introduced in step i can be cancelled by introducing an opposite difference prior or after step i, in particular in all steps $(i \pm 4k)$. To distinguish between differences which should be cancelled and differences that are introduced to cancel a subsequent difference, the terms *disturbance differences* (Δd_i) and *correction differences* (Δc_i) are used in the remainder of this chapter.

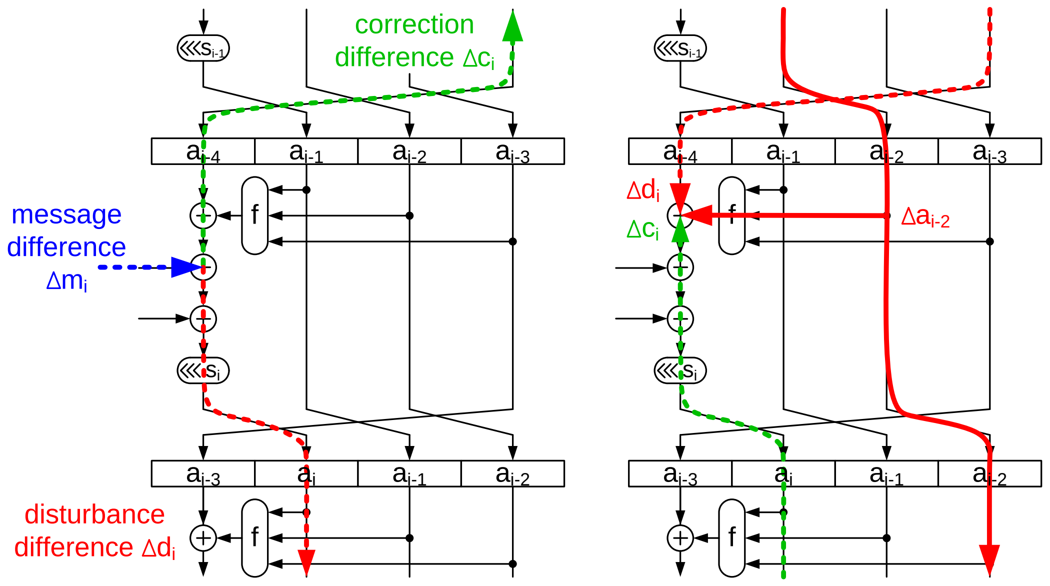


Figure 7.2: Left: Forward and backward computation of the message differences to get a target difference for each step. Right: Fulfilling the target difference using the input differences of the f-function.

7.1.1.1 Disturbance Differences

Disturbance differences Δd_i are simply derived by forward propagating the message differences. The output difference of the f_i-function is required to be zero. This is done for all steps $0 \leq i \leq 24$ and the differences in the initial values are assumed to be zero ($\Delta d_{\text{-}4} = \Delta d_{\text{-}3} = \Delta d_{\text{-}2} = \Delta d_{\text{-}1} = 0$):

$$\begin{aligned}\Delta d_i &= \Delta d_{i-4} \lll s_{i-4} + \Delta f_i(\Delta d_{i-1}, \Delta d_{i-2}, \Delta d_{i-3}) + \Delta m_{w_i} \\ &= \Delta d_{i-4} \lll s_{i-4} + 0 + \Delta m_{w_i}.\end{aligned}$$

If the differences Δd_{21}, Δd_{22}, Δd_{23} and Δd_{24} are zero, an inner collision for the first 25 steps of MD4 is found.

7.1.1.2 Correction Differences

The correction differences Δc_i need to be backward propagated and the f_i-function output difference should be zero in each step. Using the correction differences it can be determined where to introduce a difference, which in turn can cancel a message difference in a subsequent step:

$$\begin{aligned}\Delta c_{i-4} &= (\Delta c_i + \Delta f_i(\Delta c_{i-1}, \Delta c_{i-2}, \Delta c_{i-3}) + \Delta m_{w_i}) \ggg s_i \\ &= (\Delta c_i + 0 + \Delta m_{w_i}) \ggg s_i \\ &\text{with } i := \{24, 23, ..., 4\} \text{ and } \Delta c_{24} = \Delta c_{23} = \Delta c_{22} = \Delta c_{21} = 0\end{aligned}$$

7.1.1.3 Target Differences

The correction and disturbance differences are merged and called target differences. Thus, a target difference Δt_i in step i is the target output difference of the f_i-function. This difference is known to cancel a message difference in a previous or later step. This can be considered as if the *target differences* Δt_i has to be cancelled by the function f_i in step i. The target differences are built by the sum of the disturbance and correction differences (Equation 7.1). The target differences of steps $0 - 24$ are shown in Table 7.1.

$$\Delta t_i = -(\Delta a_i + \Delta b_i) \tag{7.1}$$

Example 7.1. Forward propagation of the message and disturbance differences in step $i = 3$:

$$\begin{aligned}\Delta d_3 &= \Delta d_{-1} \lll s_i + \Delta f_3(\Delta d_2, \Delta d_1, \Delta d_0) + \Delta m_i \\ &= 0 \lll 11 + \Delta IF(\Delta[6], 0, 0) + \Delta[\text{-}31, 28] = \Delta[\text{-}31, 28]\end{aligned}$$

Table 7.1: Deriving the target differences Δt_i by forward propagation of the disturbance Δd_i and backward propagation of the correction differences Δc_i for all message differences of step $0-24$.

step	Δm_{w_i}	s_i	Δd_i	Δc_i	Δt_i
0		3		Δ[16, 13, -10, -7]	Δ[-16, -13, 10, 7]
1	Δ[31]	7	Δ[31]		Δ[-31]
2	Δ[31, -28]	11	Δ[31, -28]		Δ[-31, 28]
3		19		Δ[-4]	Δ[4]
4		3		Δ[19, 16, -13, -10]	Δ[-19, -16, 13, 10]
5		7	Δ[6]		Δ[-6]
6		11	Δ[10, -7]		Δ[-10, 7]
7		19		Δ[-23]	Δ[23]
8		3		Δ[22, 19, -16, -13]	Δ[-22, -19, 16, 13]
9		7	Δ[13]		Δ[-13]
10		11	Δ[21, -18]		Δ[-21, 18]
11		19		Δ[-10]	Δ[10]
12	Δ[-16]	3	Δ[-16]	Δ[25, 22, -19]	Δ[-25, -22, 19, 16]
13		7	Δ[20]		Δ[-20]
14		11	Δ[-29, 0]		Δ[29, -0]
15		19		Δ[-29]	Δ[29]
16		3	Δ[-19]	Δ[28, 25, -22]	Δ[-28, -25, 22, 19]
17		5	Δ[27]		Δ[-27]
18		9	Δ[11, -8]		Δ[-11, 8]
19	Δ[-16]	13	Δ[-16]		Δ[16]
20	Δ[31]	3	Δ[31, -22]	Δ[28, -25]	Δ[-31, -28, 25, 22]
21		5	Δ[0]		Δ[-0]
22		9	Δ[20, -17]		Δ[-20, 17]
23		11	Δ[-29]		Δ[29]
24	Δ[31, -28]	3	Δ[31, -28, -25, 2]		Δ[-31, 28, 25, -2]

7.1.2 Cancellation Search

The cancellation search tries to cancel the target differences in each step. It is not known in advance, which target difference should be cancelled in which step. Therefore, the search is performed recursively for all steps $i = \{0, ..., 24\}$. If no difference is left after the last step, a differential path with a zero difference in step 24 has been found.

7.1.2.1 Variation of Target Difference Elements

To find a differential path with most conditions at the beginning, the first idea is to fulfill all target differences as early as possible. Unfortunately, in this case, less differences will be left to achieve the target differences in later steps. Indeed, it is hard to find *any* differential path at all. Thus, all possibilities to cancel the target differences are considered. If the signed difference Δt_i has Hamming weight w then there are 2^w possible variations to cancel its elements (see Example 7.2 and Table 7.2). All possible variations of the elements of Δt_i need to be examined.

7.1.2.2 Carry Expansion

The elements of the target difference are cancelled by the f_i-function (see Figure 7.2-right). A specific output difference of the f_i-function is only possible if an input difference at the same bit position is present. This is usually not the case. Therefore, the input differences of the f_i-function often need to be expanded. Note, that there are again many possibilities to achieve a specific bit position. Each element of the target can be cancelled by any carry expansion of any input difference of f_i. To limit the complexity of the search algorithm, a predefined maximum length for each of the carry expansions (usually 3) is used. Beside limiting the search space, the low weight differences reduce the number of conditions too.

Example 7.2. Table 7.2 shows all variations of the elements of the target difference $\Delta t_i = \Delta[\text{-}29, 20, \text{-}17]$ in the first column. Each target variation may be cancelled by any carry expansion of the inputs of $f_i(a_{i-1}, a_{i-2}, a_{i-3})$. The carry expansions with two expansion steps of $\Delta a_{i-1} = \Delta[19, \text{-}17]$, $\Delta a_{i-2} = \Delta[16]$ and $\Delta a_{i-3} = \Delta[14, \text{-}7]$ are shown in columns 2-4 respectively.

Table 7.2: All target variations and all carry expansions of Example 7.2.

Δt_i	Δa_{i-1}	Δa_{i-2}	Δa_{i-3}
Δ[-29, 20, -17]	Δ[19, -17]	Δ[16]	Δ[14, -7]
Δ[-29, 20]	Δ[18, 17]	Δ[17, -16]	Δ[15, -14, -7]
Δ[-29, -17]	Δ[20, -19, -17]	Δ[18, -17, -16]	Δ[14, -8, 7]
Δ[-29]	Δ[19, -18, 17]		Δ[15, -14, -8, 7]
Δ[20, -17]	Δ[20, -19, -18, 17]		Δ[16, -15, -14, -7]
Δ[20]	Δ[21, -20, -19, -17]		Δ[14, -9, 8, 7]
Δ[-17]			
Δ[ ]			

7.1.2.3 Cancel Possibilities

To achieve one specific target variation, all combinations of the inputs Δa_{i-1}, Δa_{i-2} and Δa_{i-3} of f_i could be tried. However, most inputs of f_i cannot fulfill a specific target variation anyway. In Table 7.2 the difference at position 29 of Δt_i cannot be achieved by any input difference listed. Thus, the search space can be significantly reduced by considering only combinations of target and input differences, which share the same bit positions.

Example 7.3. This example shows all cancel possibilities for all variations of the target difference $\Delta t_i = \Delta[\text{-}29, 20, \text{-}17]$. In this example the expansions of the input difference a_{i-1} are considered:

$$\begin{aligned}\Delta a_{i-1} &= \Delta[19,\text{-}17] = \Delta[18,17] = \Delta[20,\text{-}19,\text{-}17] = \Delta[19,\text{-}18,17] \\ &= \Delta[21,\text{-}20,\text{-}19,\text{-}17] = \Delta[20,\text{-}19,\text{-}18,17]\end{aligned}$$

$$\begin{array}{lcl}
\Delta t_i = \Delta[\text{-}29, 20, \text{-}17] & \Longrightarrow & \text{not possible} \\
\Delta t_i = \Delta[\text{-}29, 20 \quad\;\;] & \Longrightarrow & \text{not possible} \\
\Delta t_i = \Delta[\text{-}29, \quad\;\; \text{-}17] & \Longrightarrow & \text{not possible} \\
\Delta t_i = \Delta[\text{-}29 \qquad\quad\;] & \Longrightarrow & \text{not possible} \\
\Delta t_i = \Delta[\qquad 20, \text{-}17] & \Longrightarrow & \Delta[20, \text{-}19, \text{-}17] \\
\Delta t_i = \Delta[\qquad 20 \quad\;\;] & \Longrightarrow & \Delta[20, \text{-}19, \text{-}17], \Delta[20, \text{-}19, \text{-}18, 17] \\
\Delta t_i = \Delta[\qquad\qquad \text{-}17] & \Longrightarrow & \Delta[19, \text{-}17], \Delta[20, \text{-}19, \text{-}17], \Delta[21, \text{-}20, \text{-}19, \text{-}17] \\
\Delta t_i = \Delta[\qquad\qquad\quad\;] & \Longrightarrow & \text{all expansions of } \Delta a_{i-1}
\end{array}$$

7.1.2.4 Cancellation Step

In every step i, the cancellation search starts with the first difference Δa_{i-1} of f_i and tries to meet all target combinations by carry expanding Δa_{i-1}. Some targets will not be met at all, whereas others can be met with several expansions of Δa_{i-1}. The expanded difference of Δa_{i-1} with the lowest Hamming weight is used first. All target variations that cannot be fulfilled are discarded. Differences, which have been cancelled by a_{i-1} are removed from the target difference.

For the remaining target differences the same procedure is applied with Δa_{i-2} and Δa_{i-3}. All *possible* target difference variations are examined recursively. Thus, each element of the target difference is tested if it can be cancelled by any of the three input differences during the recursion. Note, that carry expansions of Δa_{i-2} and Δa_{i-3} can only be used, if they do not contradict with an expandion of a previous step. This means, that if the difference at position 18 of $\Delta a_{i-1} = \Delta[18, 17]$ has been used to cancel a target difference, then the carry expansion $\Delta a_{i-1} = \Delta[20, \text{-}19, \text{-}17]$ is not allowed anymore in this branch of the recursion.

Already cancelled target differences are removed from Δt_i in each cancellation step. Thus, two carry expansions at the same bit positions of two or more inputs of f_i are avoided as this would potentially lead to contradicting conditions (see section 7.1.3). Furthermore, shorter carry expansions keep the number of conditions small as well.

7.1.2.5 Deriving the Conditions

Only after all three input differences of the f_i function are fixed, it is possible to determine the conditions. If a certain output difference of the f_i function is not possible, this path is marked as an *impossible path*, even if it has a zero difference in its last step. Further, in many cases where a certain output difference of f_i is possible, a previously set condition can contradict a newly set condition. Then, this path is also marked as an impossible path. However, Example 7.3 shows that there can be several possibilities for a difference to cancel a target variation. Consequently, it can be checked whether a contradicting condition can be resolved with a different carry expansion.

The result of the cancellation search are paths from step 0 to step 24 that have a zero difference after step 24. In addition, the conditions for this differential path to hold are derived. However, quite often all found paths have at least one contradicting condition and are thus, impossible paths.

7.1.3 Correction Step

In the correction step, contradicting conditions or impossible output differences of impossible paths are tried to be corrected. In such impossible paths a specific target difference cannot be met in some step or a zero output difference of the f_i function cannot be achieved. As a consequence, these additional (disturbance) differences induced by the contradicting conditions need to be cancelled in some other step.

7.1.3.1 Correction by Solving Contradictions

To cancel the additional disturbances, they can be computed forward and backward through the already determined differential path. As there are only a few disturbances (caused by contradictions) to correct, longer carry expansions can be allowed. This does not increase the search space because no recursion is used. However, in many cases this does not work because further contradictions may occur. The reason is, that the conditions and differences stick together throughout the differential path (see Figure 7.3).

7.1.3.2 Correction by Dispersion Differences

Typically, differences propagate from the least significant bit in the first few steps to the most significant bit in the last steps (see Figure 7.3). The reason for this propagation is, that the rotation values s_i are very similar for most differences. In order to spread the differences and thus the conditions, *dispersion differences* are introduced in steps with a high rotation value, *i.e.* $s_2 = 11$ or $s_3 = 19$. This high rotation allows the dispersion differences to spread within only a few steps. These dispersion differences are then used in the following steps to cancel

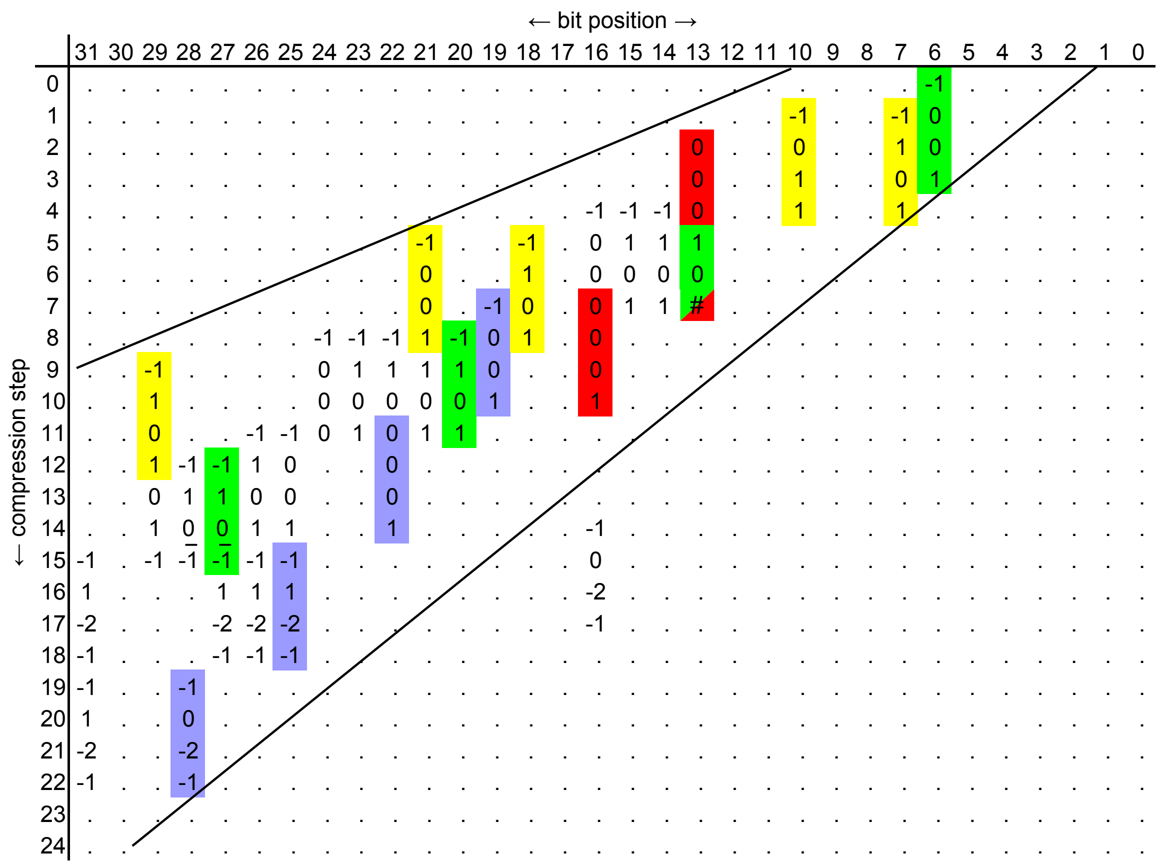


Figure 7.3: This figure shows that most differences are rotated with low values and thus differences and conditions do not spread. Hence, contradicting condition are more likely to occur although the number of conditions, which is 119, is small. A value of $c = 0$ or $c = 1$ requires $a_{i,j} = c$ for a specific bit j and step i. A negative value c or $\bar{c}$ requires the respective bit to be $a_{i,j} = a_{i-|c|,j}$ or $a_{i,j} \neq a_{i-|\bar{c}|,j}$. The entry marked by # denotes a contradicting condition.

differences in areas with a low condition density. The dispersion difference $\Delta d_3 = \Delta[6]$ leads to a differential path with spread conditions and no contradictions (see Figure 7.4). The same difference was used by Wang *et al.* too.

7.2 Experiments and Results

The search algorithm is written in C and it takes less than a minute to find many differential paths. During the development of this path-search algorithm different experiments have been made. The main difficulty was, that either the complexity of searching differential paths is too high, or no differential path could be found. A small adjustment in the cancellation search might lead to a huge change in the complexity. Therefore, the main task was not to reduce the overall complexity of the path search, but to consider the right differential paths, which

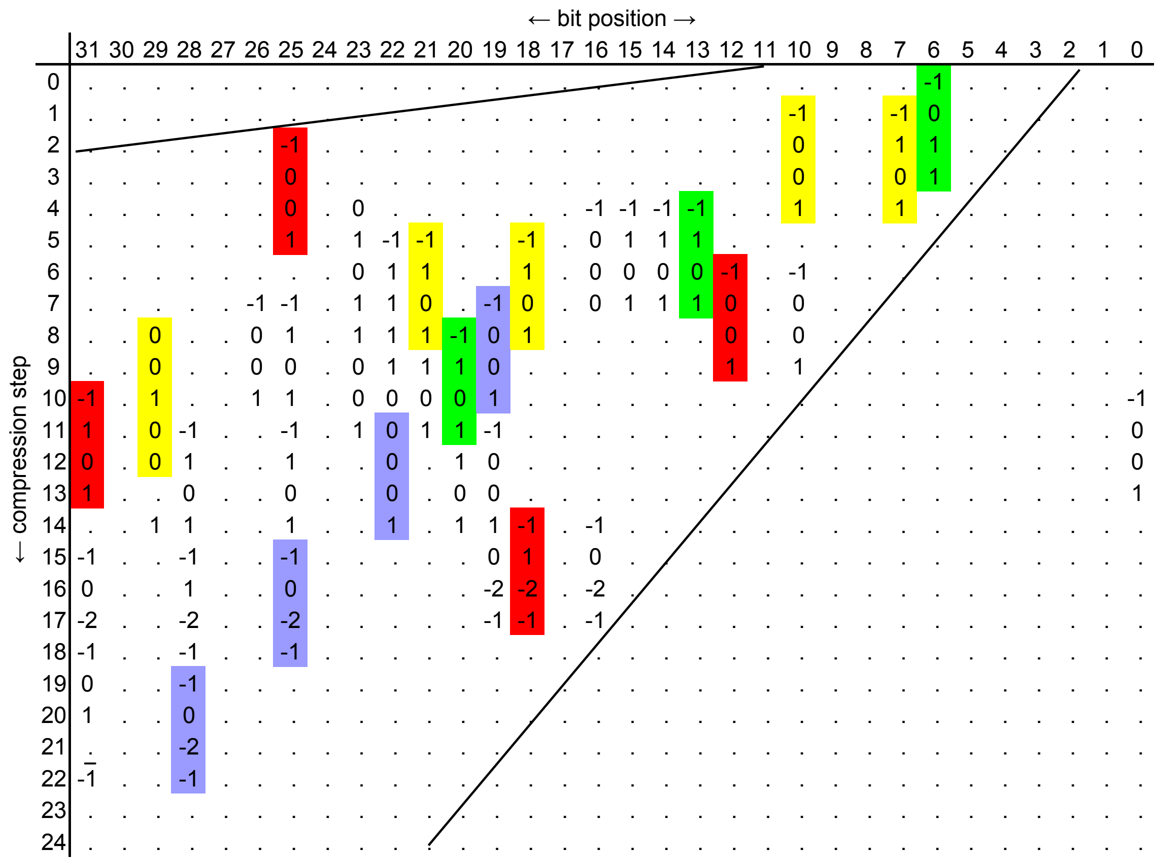


Figure 7.4: In this path, differences are spread by introducing a *dispersion difference* (blue) in a step ($i = 3$) with a high rotation value of $s_3 = 19$. Because of introducing a new difference, the number of conditions is higher (146) but nevertheless no contradicting conditions appear.

are likely to have a zero difference in their last step but contain no contradicting conditions.

In the experiments thousands of impossible paths with 1-3 unsolvable contradicting conditions have been produced. Many experiments with carry expansions of different lengths have been made as well. It turned out that at least in one step, a carry expansion of length three is needed. Otherwise, a zero difference is not possible in the last step of the differential path. Further, increasing the maximum length of the carry expansions to 10-15 does not lead to less contradicting conditions in the found paths. Moreover, the number of contradicting conditions increases when using long carry expansions, whereas no additional path with zero or one contradiction was found.

However, over 400 paths without contradicting conditions have been found after the correction by a dispersion difference. All of these paths can be found within only a couple of minutes. However, with respect to the message modification, the conditions of the best path is shown in Figure 7.4. It was chosen, because it has the smallest number of conditions in the second round. Remember that all conditions in the first round can be easily fulfilled by single-step message modification (see Section 4.3). Furthermore, most differences occur in the first few steps of the second round and are thus also easily fulfilled (see Section 8.2). The path has 146 conditions with only 22 conditions in round two and 2 conditions in round three. In contrast, the path of Wang *et al.* has 122+2 conditions where 25 conditions occur in round two and 2 conditions occur in round three. The two additional conditions were found by [NSKO05].

The complete differential behavior of the best path found in this thesis is given in Appendix A. Further, the list of conditions, as well as an overview of the conditions in matrix form is provided.

Some experiments with other message differences have been made as well. It turned out, that with other message differences it is harder to find a differential path. The reason is, that most differences do not have similar bit positions when propagating through the first two rounds (see Figure 7.5). Hence, it is not possible to cancel differences without short carry expansions. Therefore, it is suggested to introduce so called convergence differences, which bring the spread differences together.

← bit position →

← compression step	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.
5	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.
6	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	x	.	.
7	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	x	.	.	.	.	.	.	.	x	.	.
8	.	.	.	x	x	x	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	x	x	x	x	.	.	.	.	.
9	.	.	.	x	x	x	x	.	.	.	.	.	.	.	.	.	.	x	x	.	.	x	.	x	x	x	x	.	.	.	.	.
10	.	.	.	x	x	x	x	.	.	.	x	.	.	.	.	.	.	x	x	.	.	.	.	x	x	x	x	.	.	.	.	.
11	.	.	.	x	x	x	x	.	.	.	x	.	.	.	.	.	.	x	x	.	.	.	.	x	x	x	x	.	.	.	.	.
12	.	x	x	x	.	.	.	.	.	x	x	.	.	.	.	.	.	x	x	.	x	x	x	x	.	.	.	.	.	.	.	.
13	.	x	x	x	.	.	.	.	x	x	x	x	.	.	.	.	.	.	.	.	x	x	x	x	.	.	.	.	.	.	.	.
14	.	x	x	x	.	.	.	.	x	x	x	x	.	.	.	.	.	.	.	.	x	x	x	x	.	.	.	.	.	.	.	x
15	.	x	x	x	.	.	.	.	x	x	x	x	.	.	.	.	.	.	.	.	x	x	x	x	.	.	.	.	.	.	.	x
16	x	.	.	.	.	.	.	.	x	x	x	x	.	.	.	.	.	x	x	x	x	.	.	.	.	.	.	.	.	x	x	x
17	x	.	.	.	.	x	x	.	.	x	.	.	.	.	.	.	.	x	x	x	x	.	.	.	.	.	.	x	.	x	x	x
18	x	.	.	.	.	x	x	.	.	.	.	.	.	.	.	.	.	x	x	x	x	x	x	.	.	.	.	x	.	x	x	x
19	x	.	.	.	.	x	x	.	.	.	.	.	.	.	.	.	.	x	x	x	x	x	x	.	.	.	.	x	.	x	x	x
20	x	.	.	.	.	x	x	.	.	.	.	.	.	.	x	x	x	x	.	x	x	x	x	.	.	x	.	x	x	x	x	x
21	x	x	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x	.	x	x	x	x	.	.	x	.	x	x	x	x	x
22	x	x	.	.	.	.	.	.	.	.	.	.	.	.	x	x	x	x	.	x	x	x	x	.	.	x	.	x	x	x	x	x
23	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	.	x	x
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Figure 7.5: When starting the inner collision of the third round at $i = 33$, the message differences $\Delta m_4 = \Delta[31, 20]$, $\Delta m_6 = \Delta[31]$, $\Delta m_8 = \Delta[23]$ are used. Because the differences marked by "x" in the first two rounds do not meet (the maximum carry expansion is 3), convergence differences are needed.

Chapter 8

Message Modification Techniques

In the previous chapter, a differential path together with conditions on the state variables has been found. A differential path does only lead to a collision if all conditions are fulfilled. To find a sequence of state variables which fulfill all conditions, random messages are examined. One condition is fulfilled with probability of $1/2$ under the assumption that the conditions are independent. Thus, a random message fulfills all conditions of the differential path with a probability of $2^{\#conditions}$. If only one of the conditions in the state variables is not fulfilled, a zero difference does not occur at the output of the hash function.

In this chapter, the conditions of the path published by Wang *et al.* are considered. This path has 125 conditions and thus, the complexity of randomly searching adequate messages is 2^{125}. This is worse than the birthday attack (2^{64}). Therefore, Wang *et al.* introduced so-called the message modification techniques to reduce this complexity. The message modification techniques modify a random message to fulfill as many conditions as possible. The single-step message modification is used to correct the conditions in the first round. To correct conditions in the second round, more sophisticated multi-step message modification and advanced multi-step message modification are needed.

8.1 Single-Step Message Modification

The single-step message modification can modify any message, such that all conditions which occur in the first round are fulfilled. These conditions can be fulfilled by inverting each step of the first round or by flipping single message bits.

8.1.1 Inverting the Step Operation

First a random message is chosen. Then, the conditions in each step $i = 0, ..., 15$ of the first round are corrected by applying the following procedure:

1. Compute the register value a_i using the update rule of Equation 3.1:

$$a_i = (a_{i-4} + f_i(a_{i-1}, a_{i-2}, a_{i-3}) + m_{w_i} + c_i) \lll s_i$$

2. For every condition on a_i at bit position j which is not fulfilled, correct it by flipping the j-th bit. The new state variable is called a_i^*.

3. Get the new message word m_i^* by inverting the update rule:

$$m_i^* = (a_i^* \ggg s_i) - a_{i-4} - f(a_{i-1}, a_{i-2}, a_{i-3})$$

Every changed state variable a_i and thus, every changed message word m_i, produces different values in the following steps and thus, a different hash value. Using this simple method, all conditions in the first round are fulfilled by computing new message words.

8.1.2 Flipping the Message Bits

If conditions in the second round have to be corrected later on by a multi-step message modification, a slightly changed single-step message modification, which changes less message bits should be applied (see Section 8.2):

1. Compute the register value a_i using the update rule of Equation 3.1:

$$a_i = (a_{i-4} + f_i(a_{i-1}, a_{i-2}, a_{i-3}) + m_{w_i} + c_i) \lll s_i$$

2. For every bit position $j = 0, ..., 31$ flip the message bit $m_{i,j}$, if the corresponding condition of a_i at position $(j - s_i)$ is not fulfilled.

Note, that the message has to be changed starting at the least significant bit (LSB) of the message word (see Figure 8.1) and not at the LSB of the state variable, because the addition is performed prior to the rotation. Otherwise, a carry in the addition may occur and change a previously corrected condition.

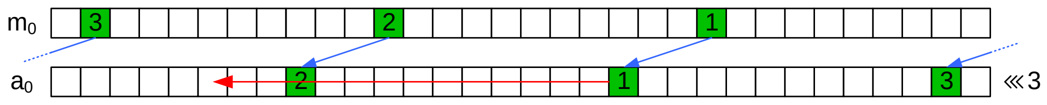


Figure 8.1: To correct the conditions of a_0 flip the corresponding bits of m_0. Start at the (LSB) of m_0 because a carry may occur in the addition (horizontal arrays).

8.2 Multi-Step Message Modification

To fulfill the conditions in the second round the message words, which have been used to correct conditions of the first round, are used again. Therefore, every modification of these message words influences the state variables in both rounds, e.g. for message word m_0:

$$a_0 = (a_{\text{-}4} + IF(a_{\text{-}1}, a_{\text{-}2}, a_{\text{-}3}) + m_0 + c_0) \lll s_0$$
$$a_{16} = (a_{12} + MAJ(a_{15}, a_{14}, a_{13}) + m_0 + c_{16}) \lll s_{16}$$

Hence, all modifications of the message word in the second round have to be corrected in the first round.

8.2.1 Flipping the Message Bits Interleavingly

To correct the conditions of a_{16}, the corresponding bits of m_0 are flipped. For every changed bit in m_0 a carry in the addition of step 16 and step 0 may occur. The conditions have to be corrected interleavingly, starting at the LSB of m_0. If a condition in a_0 is not fulfilled anymore because of a carry, its corresponding bit in m_0 has to be flipped again (see Figure 8.2).

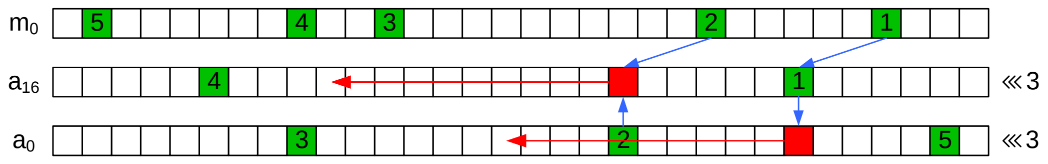


Figure 8.2: The conditions of a_0 and a_{16} are corrected interleavingly starting at the LSB of m_0.

The state variable a_0 might be referenced by a condition of a following step, *i.e.* $a_{1,6} = a_{0,6}$. These condition can become false if a_0 is changed. To correct these conditions, the corresponding bits in m_0 need to be interleavingly flipped too (see Figure 8.3).

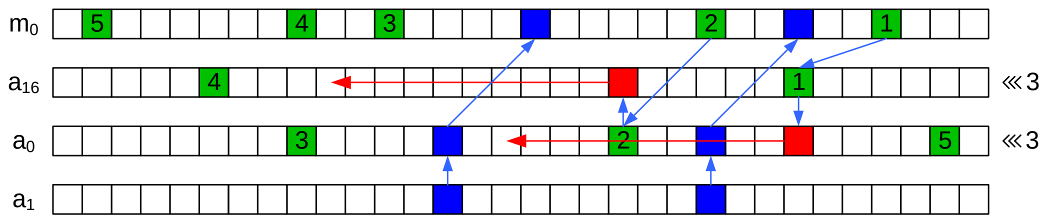


Figure 8.3: The conditions of a_0, a_{16} and the bits in a_0 which are referenced by a_1 are corrected interleavingly starting at the LSB of m_0.

8.2.2 Correcting the Modifications in the First Round

The changes made to a_0 need to be corrected. Otherwise, all state variables in the first round and finally a_{16} will change too. The changes of a_0 are corrected by inverting all steps in which a_0 is used as an input (steps $1-4$). This results in new message words m_1, m_2, m_3 and m_4. Thus, a_1, a_2, a_3 and a_4 and all following state variables remain the same:

$$\begin{aligned} m_1 &= (a_1 \ggg 7) - a_{\text{-}3} - f(a_0^*, a_{\text{-}1}, a_{\text{-}2}) \\ m_2 &= (a_2 \ggg 11) - a_{\text{-}2} - f(a_1, a_0^*, a_{\text{-}1}) \\ m_3 &= (a_3 \ggg 19) - a_{\text{-}1} - f(a_2, a_1, a_0^*) \\ m_4 &= (a_4 \ggg 3) - a_0^* - f(a_3, a_2, a_1) \end{aligned} \tag{8.1}$$

This multi-step message modification can be used in step 16 and 17. In step 17 the message word m_4 is introduced. Therefore, the conditions of the state variable a_4 have to be interleavingly corrected with the conditions of a_{17} and a_5, starting at the LSB of m_4.

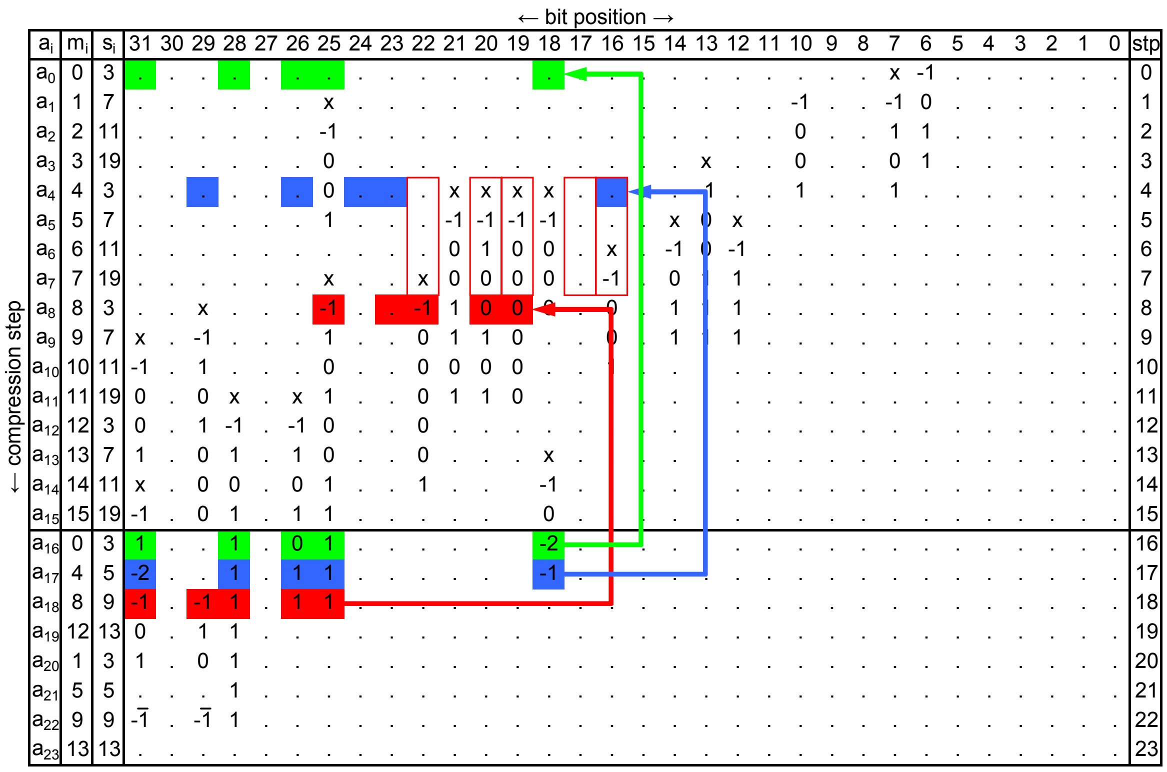
However, it is not possible to correct all conditions of step 18 using the same technique, because there are already conditions at the corresponding bit positions of m_8 (see Figure 8.4). Another technique called advanced multi-step message modification technique needs to be applied instead.

8.3 Advanced Multi-Step Message Modification

In step 18 the conditions of a_{18} at the bit positions $j = \{25, 26, 28, 29, 31\}$ need to be met. This can be achieved by flipping the corresponding message word bits of m_8 $\{16, 17, 19, 20, 22\}$ ($s_{18} = 9$). The changed bits of m_8 influence the state variable a_8 at the bit positions $\{19, 20, 22, 23, 25\}$ ($s_8 = 3$). However, there are conditions on $a_{8,19}$, $a_{8,20}$, $a_{8,22}$ and $a_{8,25}$ which become false, if the corresponding message bit is flipped. The message bit $m_{8,j}$ can only correct one of the conditions of $a_{18,j+9}$ and $a_{8,j+3}$ but not both. Figure 8.4 shows the corresponding conditions of step 16, 17 and 18. Only the condition $a_{18,29} =$ -1 can be corrected without tampering a condition of a_8.

8.3.1 Correcting using a Previous Step

To preserve the conditions on the state variable a_8, the corresponding bits should not be changed. Note, that a flip of the message word bit $m_{i,j}$ is the same as a difference in m_i at position j thus, $\Delta m_i = \Delta[\pm j]$. To prevent the message word difference $\Delta[\pm j]$ from propagating to a_8, an additional difference with opposite sign ($\Delta[\mp j]$) is used to cancel it. This difference needs to be introduced in a previous step using the state variable a_4, a_5, a_6



← bit position →

a_i	m_i	s_i	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	stp
a_0	0	3	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	x	-1	.	.	.	.	.	.	0
a_1	1	7	.	.	.	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1	.	.	-1	0	.	.	.	.	.	.	1
a_2	2	11	.	.	.	.	.	.	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0	.	.	1	1	.	.	.	.	.	.	2
a_3	3	19	.	.	.	.	.	.	0	.	.	.	.	.	.	.	.	.	.	.	x	.	.	0	.	.	0	1	.	.	.	.	.	.	3
a_4	4	3	.	.	.	.	.	.	0	.	.	.	x	x	x	x	.	.	.	.	1	.	.	1	.	.	1	.	.	.	.	.	.	.	4
a_5	5	7	.	.	.	.	.	.	1	.	.	.	-1	-1	-1	-1	.	.	.	x	0	x	.	.	.	.	.	.	.	.	.	.	.	.	5
a_6	6	11	.	.	.	.	.	.	.	.	.	.	0	1	0	0	.	x	.	-1	0	-1	.	.	.	.	.	.	.	.	.	.	.	.	6
a_7	7	19	.	.	.	.	.	.	x	.	.	x	0	0	0	0	.	-1	.	0	1	1	.	.	.	.	.	.	.	.	.	.	.	.	7
a_8	8	3	.	.	x	.	.	.	-1	.	.	-1	1	0	0	0	.	0	.	1	1	1	.	.	.	.	.	.	.	.	.	.	.	.	8
a_9	9	7	x	.	-1	.	.	.	1	.	.	0	1	1	0	.	.	0	.	1	1	1	.	.	.	.	.	.	.	.	.	.	.	.	9
a_{10}	10	11	-1	.	1	.	.	.	0	.	.	0	0	0	0	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	10
a_{11}	11	19	0	.	0	x	.	x	1	.	.	0	1	1	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	11
a_{12}	12	3	0	.	1	-1	.	-1	0	.	.	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	12
a_{13}	13	7	1	.	0	1	.	1	0	.	.	0	.	.	.	x	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	13
a_{14}	14	11	x	.	0	0	.	0	1	.	.	1	.	.		-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	14
a_{15}	15	19	-1	.	0	1	.	1	1	.	.	.	.	.		0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	15
a_{16}	0	3	1	.	.	1	.	0	1	.	.	.	.	.	.	-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	16
a_{17}	4	5	-2	.	.	1	.	1	1	.	.	.	.	.	.	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	17
a_{18}	8	9	-1	.	-1	1	.	1	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	18
a_{19}	12	13	0	.	1	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	19
a_{20}	1	3	1	.	0	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	20
a_{21}	5	5	.	.	.	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	21
a_{22}	9	9	$-\bar{1}$	.	$-\bar{1}$	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	22
a_{23}	13	13	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	23

Figure 8.4: This figure highlights the corresponding conditions of the first and the second round. The arrows show the rotation values s_i and referenced bits are marked by "x". The conditions at position 25, 26, 28 and 31 of a_{18} contradict with the conditions at 19, 20, 22 and 25 of a_8.

or a_7 (see Figure 8.5-left). In Figure 8.4 the steps and bit positions where these differences can be introduced are marked by a (red) box.

8.3.2 Controlling the Propagation

The propagation of these additional differences needs to be controlled. Assume, that the difference $\Delta m_k = \Delta[\pm j - s_k]$ is introduced in step $k = \{i\text{-}1, i\text{-}2, i\text{-}3, i\text{-}4\}$. To avoid the carry propagation, an additional condition on $a_{k,j}$ is needed in the first round (see Figure 8.5-right):

$$\Delta a_{k,j} = \Delta[j] \Longrightarrow a_{k,j} = 0$$
$$\Delta a_{k,j} = \Delta[\text{-}j] \Longrightarrow a_{k,j} = 1$$

To avoid the propagation of this difference in the following steps, it can be blocked by the IF-function. This requires further conditions in the first round. In the step i, where the difference should cancel the message difference m_i, the IF-function has to forward the difference a_k. See Tabel 5.1 for the required conditions of each case. Because there are usually already

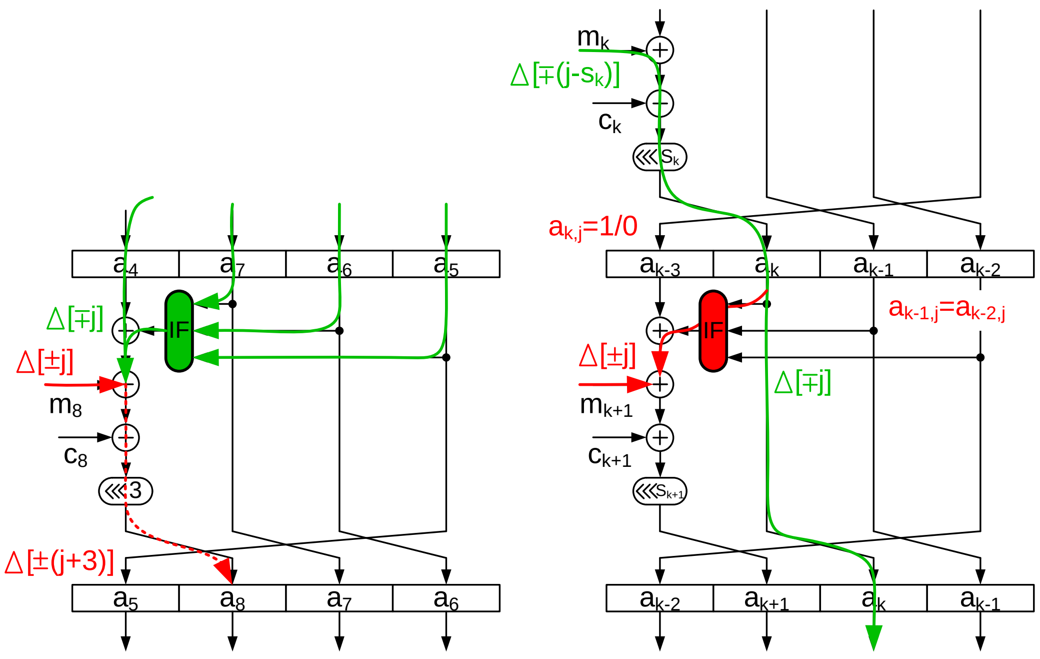


Figure 8.5: Left: To cancel the difference introduced by m_8, a difference in a_4, a_5, a_6 or a_7 can be used. Right: This difference needs to be introduced in a previous step k and its propagation accordingly controlled.

conditions in the first round, the newly set conditions may contradict. In this case, the message difference m_k can be introduced in a different step.

However, instead of conditions which block the IF-function, the difference can be cancelled by introducing another message difference too (see Figure 8.5-right). Note, that the propagation of the difference m_k needs to be stopped after four steps.

8.3.3 Correction Possibilities

The possible steps to introduce a difference and the possibilities to control the propagation are listed in Table 8.1. Every combination is allowed, as long as it does not contradict with already set conditions. Therefore, in areas with many conditions in the first round the propagation needs to be stopped mainly using the message difference. However, to prevent the carry propagation and to forward the difference through the IF-function a condition is always needed (see Table 8.1).

In Table 8.1, the message difference to be cancelled in step i is denoted by δm_j. In the first four cases, the negative message difference has to be inserted to cancel δm_j. Note, that the IF-function is able to forward the negated difference of its first input (see Section 5.4.1). Thus, it is possible to introduce a message difference with the same sign and flip the sign using the IF-function (case 5). The last case is the same as the regular multi-step message modification but without inverting the following steps to correct the changes of $a_{i,j}$ (see Equation 8.1). Instead, the propagation of the difference caused by $\delta m_{i,j-s_i}$ is controlled using additional conditions or by a message differences at position j. Note, that there are further possible cases which have not been used here, e.g. using intended carry propagations.

Table 8.1: The 6 possible cases for the advanced multi-step message modification. The message difference δm_j to cancel in step i is marked red. The green message differences at the beginning and the end are needed in every case. Always needed conditions for carry propagation are marked blue and conditions to forward the difference are marked green. In all other cases (black) it can be chosen for each step, whether to stop the propagation using a message difference or a condition.

	case 1		case 2		case 3	
step	Δm_i	conditions	Δm_i	conditions	Δm_i	conditions
$i-4$	$-\delta m_{j-s_i}$	$a_{i-4} = 1/0$				
$i-3$	$\pm\delta m_j$	$a_{i-5} = -1$	$-\delta m_{j-s_i}$	$a_{i-3} = 1/0$		
$i-2$	$\pm\delta m_j$	$a_{i-3} = 0$	$\pm\delta m_j$	$a_{i-4} = -1$	$-\delta m_{j-s_i}$	$a_{i-2} = 1/0$
$i-1$	$\pm\delta m_j$	$a_{i-2} = 1$	$\pm\delta m_j$	$a_{i-2} = 0$	$\pm\delta m_j$	$a_{i-3} = -1$
i	δm_j		δm_j	$a_{i-1} = 0$	δm_j	$a_{i-1} = 1$
$i+1$			δm_j		$\pm\delta m_j$	$a_i = 1$
$i+2$					δm_j	

	case 4		case 5		case 6	
$i-1$	$-\delta m_{j-s_i}$	$a_{i-1} = 1/0$	δm_{j-s_i}	$a_{i-1} = 0/1$		
i	δm_j	$a_{i-2} = 1, a_{i-3} = 0$	δm_j	$a_{i-2} = 0, a_{i-3} = 1$	δm_{j-s_i}	$a_i = 1/0$
$i+1$	$\pm\delta m_j$	$a_i = 0$	$\pm\delta m_j$	$a_i = 0$	$\pm\delta m_j$	$a_{i-1} = -1$
$i+2$	$\pm\delta m_j$	$a_{i+1} = 1$	$\pm\delta m_j$	$a_{i+1} = 1$	$\pm\delta m_j$	$a_{i+1} = 0$
$i+3$	δm_j		$-\delta m_j$		$\pm\delta m_j$	$a_{i+2} = 1$
$i+4$					$-\delta m_j$	

The steps to introduce the differences for the advanced multi-step message modification of step 18 are highlighted in Figure 8.6. However, the condition on $a_{18,29}$ cannot be fulfilled in the *first* round using any of these cases. Therefore, this condition is corrected using advanced multi-step message modification in the *second* round. This results in additional conditions of the second round which can be corrected by the multi-step message modification.

8.3.4 Avoid Message Word Permutation

For the correction of the conditions in step 19, the message word m_{12} needs to be modified. If the modifications in the first round are corrected by inverting the steps (see Section 8.2.2), the message word m_0 is modified again. As this message word is used in step 0 and 16, many changes would occur. Therefore, to correct the conditions in step 19, only the cases 1-5 can

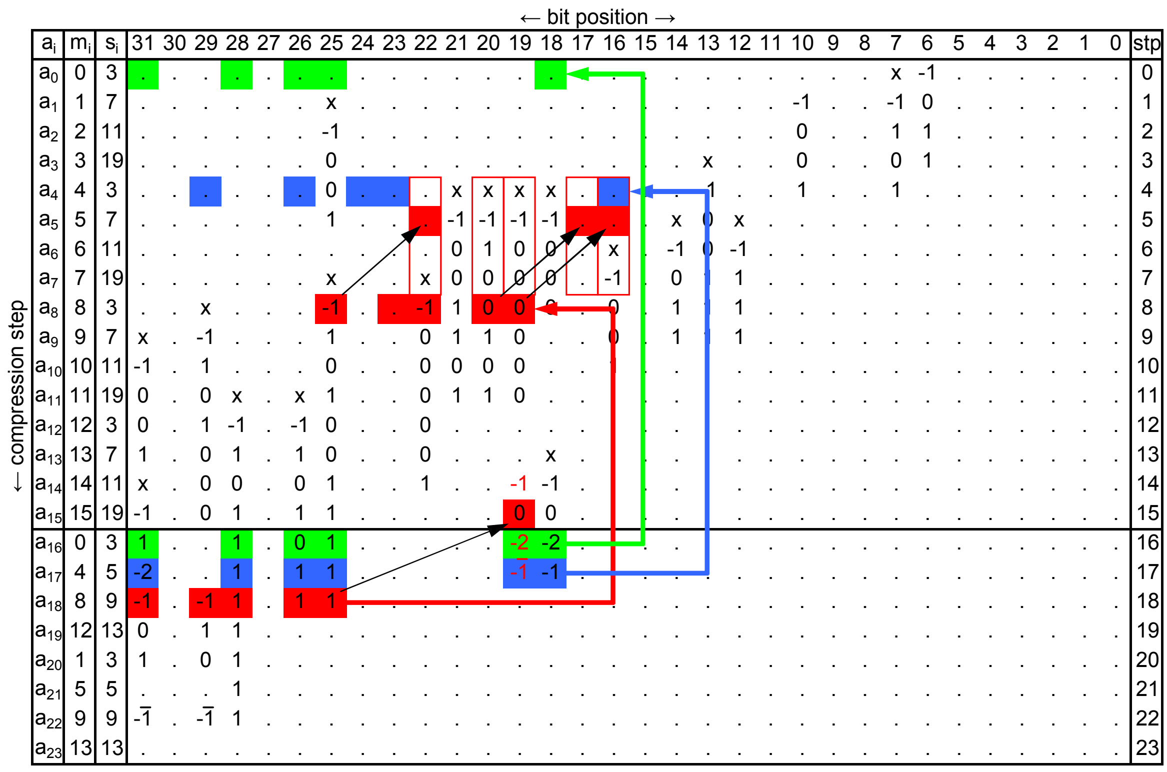


Figure 8.6: An additional difference is introduced in the highlighted bit positions of the red boxes. The condition $a_{18,28}$ is corrected by advanced multi-step message modification in the 2nd round.

be used. Similar problems occur for the following steps (20, 21, 22), because m_4, m_8 and m_{12} have already been used in the second round too. For these steps, the propagation of the message difference in the first round is controlled by the IF-function and thus, m_4, m_8, and m_{12} do not get changed (see Figure 8.7). By applying these advanced multi-step message modification cases, all conditions of the first two rounds can be fulfilled with probability 1.

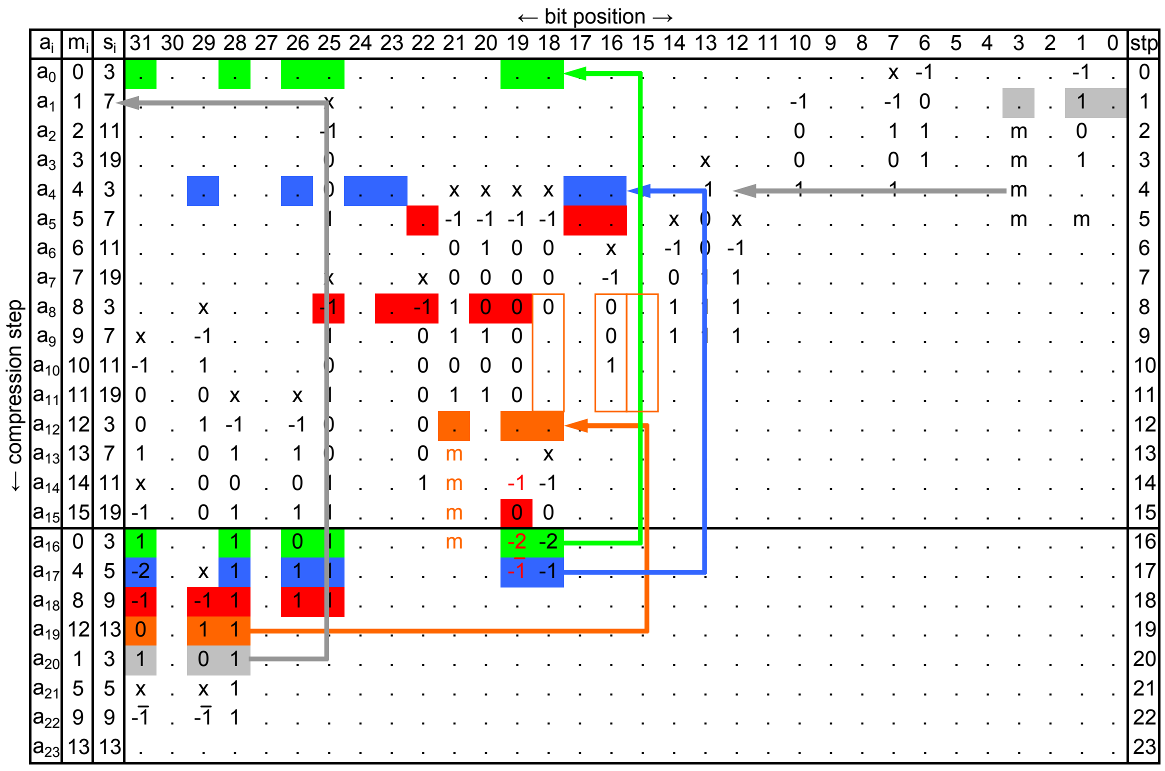


Figure 8.7: To correct the conditions in step 19 only the cases 1-5 can be used. Otherwise the message word m_0 in step 16 is changed. The changes of the message word m_1 cannot be corrected using m_4 because m_4 is already used in the second round too. m_1 is corrected using conditions (case 6).

Chapter 9

Conclusions

In the first chapters of this thesis, cryptographic hash functions and in particular hash functions of the MD-family have been defined and discussed. The most important attacks on the collision resistance have been mentioned and their complexity was given. This thesis has concentrated on the attack approach by Wang *et al.*, which is a differential attack on the collision resistance of a hash function. The idea of this strategy has been explained in detail.

To apply new ideas, methods for differential cryptanalysis of the MD-family of hash functions have been defined and used in the attack on MD4. The main part of this attack is to search for differential paths which lead to a collision of the compression function. An algorithm has been developed which is able to find many differential paths for the message difference chosen by Wang *et al.* More than 400 differential paths have been found so far. Some paths are slightly better than the path found by Wang *et al.* because less conditions are needed in the second round. This is an advantage regarding the message modification techniques. The different types of message modification techniques have been analyzed and improved too. The techniques applied in this thesis are not very specific for MD4 and can be extended to the use in other hash functions.

In future work, the algorithm for searching differential paths can be enhanced to find paths with less conditions. With the use of dispersion and convergence differences, other message differences can be tackled. The path search can be applied to different types of attacks as well. With minor modifications, the algorithm can search for differential paths of chosen message 2nd-preimage attacks, multi-collision attacks, multi-block collision attacks, herding attacks or collision attacks with different IVs. With more changes and improvements the differential path search presented in this thesis can be applied to MD5 as well.

Appendix A

The Differential Path

← compression step	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1	.	.	.	.	.	
1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1	.	.	-1	0	.	.	.	.	.	.
2	.	.	.	.	.	.	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0	.	.	1	1	.	.	.	.	.	.
3	.	.	.	.	.	.	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0	.	.	0	1	.	.	.	.	.	.
4	.	.	.	.	.	.	0	.	0	.	.	.	.	.	.	-1	-1	-1	-1	.	.	1	.	.	1	.	.	.	.	.	.	.
5	.	.	.	.	.	.	1	.	1	-1	-1	.	.	-1	.	0	1	1	1	.	.	.	.	.	.	.	.	.	.	.	.	.
6	.	.	.	.	.	.	.	.	0	1	1	.	.	1	.	0	0	0	0	-1	.	-1	.	.	.	.	.	.	.	.	.	.
7	.	.	.	.	.	-1	-1	.	1	1	0	.	-1	0	.	0	1	1	1	0	.	0	.	.	.	.	.	.	.	.	.	.
8	.	.	0	.	.	0	1	.	1	1	1	-1	0	1	.	.	.	.	.	0	.	0	.	.	.	.	.	.	.	.	.	.
9	.	.	0	.	.	0	0	.	0	1	1	1	0	.	.	.	.	.	.	1	.	1	.	.	.	.	.	.	.	.	.	.
10	-1	.	1	.	.	1	1	.	0	0	0	0	1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	-1
11	1	.	0	-1	.	.	-1	.	1	0	1	1	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0
12	0	.	0	1	.	.	1	.	.	0	.	1	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0
13	1	.	.	0	.	.	0	.	.	0	.	0	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1
14	.	.	1	1	.	.	1	.	.	1	.	1	1	-1	.	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
15	-1	.	.	-1	.	.	-1	.	.	.	.	.	0	1	.	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
16	0	.	.	1	.	.	0	.	.	.	.	.	-2	-2	.	-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
17	-2	.	.	-2	.	.	-2	.	.	.	.	.	-1	-1	.	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
18	-1	.	.	-1	.	.	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
19	0	.	.	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
20	1	.	.	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
21	.	.	.	-2	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
22	-$\overline{1}$	.	.	-1	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
23	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
24	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

← bit position →

Figure A.1: The 146 conditions of the differential path with no contradicting conditions.

Table A.1: Differential characteristic of the differential path with 146 conditions.

Step	a_i	s_i	m_{w_i}	Δm_{w_i}	Δf_i	Δa_i
0	a_0	3	m_0			
1	a_1	7	m_1	Δ[31]		Δ[6]
2	a_2	11	m_2	Δ[31,-28]		Δ[10,-7]
3	a_3	19	m_3		Δ[6]	Δ[25]
4	a_4	3	m_4			
5	a_5	7	m_5			Δ[16,-15,-14,-13]
6	a_6	11	m_6			Δ[23,-22,-21,-18]
7	a_7	19	m_7		Δ[23]	Δ[12,10]
8	a_8	3	m_8		Δ[23,-22,16]	Δ[26,-25,19]
9	a_9	7	m_9			Δ[23,-22,-21,-20]
10	a_{10}	11	m_{10}		Δ[-21]	Δ[-29]
11	a_{11}	19	m_{11}			Δ[-31,29,0]
12	a_{12}	3	m_{12}	Δ[-16]	Δ[-22]	Δ[29,-28,-25,22,-20,19]
13	a_{13}	7	m_{13}		Δ[-20]	
14	a_{14}	11	m_{14}		Δ[29]	
15	a_{15}	19	m_{15}			Δ[19,-18,16]
16	a_{16}	3	m_0		Δ[19]	Δ[31,-28,25]
17	a_{17}	5	m_4			
18	a_{18}	9	m_8			
19	a_{19}	13	m_{12}	Δ[-16]		Δ[31]
20	a_{20}	3	m_1	Δ[31]		Δ[-31,28]
21	a_{21}	5	m_5			
22	a_{22}	9	m_9			
23	a_{23}	13	m_{13}		Δ[-31]	
24	a_{24}	3	m_2	Δ[31,-28]		
...	...					
35	a_{35}	15	m_{12}	Δ[-16]		Δ[-31]
36	a_{36}	3	m_2	Δ[31,-28]	Δ[-31]	Δ[-31]
37	a_{37}	9	m_{10}			
38	a_{38}	11	m_6			
39	a_{39}	15	m_{14}			
40	a_{40}	3	m_1	Δ[31]		

Table A.2: The list of 146 conditions for the differential path with no contradiction.

Step	Conditions for a_i
0	$a_{0,6} = a_{-1,6}$
1	$a_{1,6} = 0, a_{1,7} = a_{0,7}, a_{1,10} = a_{0,10}$
2	$a_{2,6} = 1, a_{2,7} = 1, a_{2,10} = 0, a_{2,25} = a_{1,25}$
3	$a_{3,6} = 1, a_{3,7} = 0, a_{3,10} = 0, a_{3,25} = 0$
4	$a_{4,7} = 1, a_{4,10} = 1, a_{4,13} = a_{3,13}, a_{4,14} = a_{3,14}, a_{4,15} = a_{3,15}, a_{4,16} = a_{3,16}, a_{4,23} = 0, a_{4,25} = 0$
5	$a_{5,13} = 1, a_{5,14} = 1, a_{5,15} = 1, a_{5,16} = 0, a_{5,18} = a_{4,18}, a_{5,21} = a_{4,21}, a_{5,22} = a_{4,22}, a_{5,23} = 1, a_{5,25} = 1$
6	$a_{6,10} = a_{5,10}, a_{6,12} = a_{5,12}, a_{6,13} = 0, a_{6,14} = 0, a_{6,15} = 0, a_{6,16} = 0, a_{6,18} = 1, a_{6,21} = 1, a_{6,22} = 1, a_{6,23} = 0$
7	$a_{7,10} = 0, a_{7,12} = 0, a_{7,13} = 1, a_{7,14} = 1, a_{7,15} = 1, a_{7,16} = 0, a_{7,18} = 0, a_{7,19} = a_{6,19}, a_{7,21} = 0, a_{7,22} = 1, a_{7,23} = 1, a_{7,25} = a_{6,25}, a_{7,26} = a_{6,26}$
8	$a_{8,10} = 0, a_{8,12} = 0, a_{8,18} = 1, a_{8,19} = 0, a_{8,20} = a_{7,20}, a_{8,21} = 1, a_{8,22} = 1, a_{8,23} = 1, a_{8,25} = 1, a_{8,26} = 0, a_{8,29} = 0$
9	$a_{9,10} = 1, a_{9,12} = 1, a_{9,19} = 0, a_{9,20} = 1, a_{9,21} = 1, a_{9,22} = 1, a_{9,23} = 0, a_{9,25} = 0, a_{9,26} = 0, a_{9,29} = 0$
10	$a_{10,0} = a_{9,0}, a_{10,19} = 1, a_{10,20} = 0, a_{10,21} = 0, a_{10,22} = 0, a_{10,23} = 0, a_{10,25} = 1, a_{10,26} = 1, a_{10,29} = 1, a_{10,31} = a_{9,31}$
11	$a_{11,0} = 0, a_{11,19} = a_{10,19}, a_{11,20} = 1, a_{11,21} = 1, a_{11,22} = 0, a_{11,23} = 1, a_{11,25} = a_{10,25}, a_{11,28} = a_{10,28}, a_{11,29} = 0, a_{11,31} = 1$
12	$a_{12,0} = 0, a_{12,19} = 0, a_{12,20} = 1, a_{12,22} = 0, a_{12,25} = 1, a_{12,28} = 1, a_{12,29} = 0, a_{12,31} = 0$
13	$a_{13,0} = 1, a_{13,19} = 0, a_{13,20} = 0, a_{13,22} = 0, a_{13,25} = 0, a_{13,28} = 0, a_{13,31} = 1$
14	$a_{14,16} = a_{13,16}, a_{14,18} = a_{13,18}, a_{14,19} = 1, a_{14,20} = 1, a_{14,22} = 1, a_{14,25} = 1, a_{14,28} = 1, a_{14,29} = 1$
15	$a_{15,16} = 0, a_{15,18} = 1, a_{15,19} = 0, a_{15,25} = a_{14,25}, a_{15,28} = a_{14,28}, a_{15,31} = a_{14,31}$
16	$a_{16,16} = a_{14,16}, a_{16,18} = a_{14,18}, a_{16,19} = a_{14,19}, a_{16,25} = 0, a_{16,28} = 1, a_{16,31} = 0$
17	$a_{17,16} = a_{16,16}, a_{17,18} = a_{16,18}, a_{17,19} = a_{16,19}, a_{17,25} = a_{15,25}, a_{17,28} = a_{15,28}, a_{17,31} = a_{15,31}$
18	$a_{18,25} = a_{17,25}, a_{18,28} = a_{17,28}, a_{18,31} = a_{17,31}$
19	$a_{19,28} = a_{18,28}, a_{19,31} = 0$
20	$a_{20,28} = 0, a_{20,31} = 1$
21	$a_{21,28} = a_{19,28}$
22	$a_{22,28} = a_{21,28}, a_{22,31} = \overline{a_{18,31}}$

List of Figures

List of Tables

Bibliography

[ABB$^+$05] Daniel Augot, Alex Biryukov, An Braeken, Carlos Cid, Hans Dobbertin, Hakan Englund, Henri Gilbert, Louis Granboulan, Helena Handschuh, Martin Hell, Thomas Johansson, Alexander Maximov, Matthew Parkerand Thomas Pornin, Bart Preneel, Matt Robshaw, and Michael Ward. Ongoing Research Areas in Symmetric Cryptography, January 2005. 1

[BC04] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004: Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004. 13

[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, *Advances in Cryptology - EUROCRYPT 2005: Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 36–57. Springer, 2005. 13

[CJ98] Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98: Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462, pages 56–71. Springer, 1998. 13

[Dam89] Ivan Bjerre Damgård. A Design Principle for Hash Functions. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 416–427, New York, NY, USA, 1989. Springer-Verlag New York, Inc. 4

[Dau05] Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family.* PhD thesis, Ruhr Universität Bochum, 2005. Available at `http://http://www.cits.rub.de/imperia/md/content/magnus/dissmd4.pdf`. 1, 5, 7, 9, 12, 21, 22, 25, 28

[dBB91] Bert den Boer and Antoon Bosselaers. An Attack on the Last Two Rounds of MD4. *Lecture Notes in Computer Science*, 576:194–??, 1991. 13

[dBB94] Bert den Boer and Antoon Bosselaers. Collisions for the Compression Function of MD5. *Lecture Notes in Computer Science*, 765:293–??, 1994. 13

[Dob96a] Hans Dobbertin. Cryptanalysis of MD4. In Bart Preneel, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *LNCS*, pages 53–69. Springer, 1996. 13, 29

[Dob96b] Hans Dobbertin. Cryptanalysis of MD5 compress. In *Rump Session of EuroCrypt '96*, 1996. 13

[Dob96c] Hans Dobbertin. The status of MD5 after a recent attack. *Crypto-Bytes*, 2:1–6, 1996. 13

[Dob97] Hans Dobbertin. RIPEMD with two round compress is not collision-free. *Journal of Cryptology*, 10(1):51–70, 1997. 13

[Dob98] Hans Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, 1998. 13

[HPR04] Philip Hawkes, Michael Paddon, and Gregory G. Rose. Musings on the Wang et al. MD5 Collision. Cryptology ePrint Archive, Report 2004/264, 2004. 1

[Kli05] Vlastimil Klima. Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications, 2005. Preprint, available at `http://eprint.iacr.org/2005/102`. 2

[Mer90] Ralph C. Merkle. One Way Hash Functions and DES. In *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 428–446, London, UK, 1990. Springer-Verlag. 4

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. Available online at `http://www.cacr.math.uwaterloo.ca/hac/`. 1, 3

[NSKO05] Yusuke Naito, Yu Sasaki, Noboru Kunihiro, and Kazuo Ohta. Improved Collision Attack on MD4. Cryptology ePrint Archive, Report 2005/151, 2005. `http://eprint.iacr.org/`. 2, 19, 44

[Pre93] Bart Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993. 5, 9

[Riv90a] Ronald L. Rivest. RFC 1320: The MD4 Message-Digest Algorithm. RFC1320, April 1990. `http://www.ietf.org/rfc/rfc1320.txt`. 9

[Riv90b] Ronald L. Rivest. The MD4 Message Digest Algorithm. pages 303–311, 1990. 9

[RO05] Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *LNCS*, pages 58–71. Springer, 2005. 13

[Sti02] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 2nd edition, 2002. ISBN 1-58488-206-9. 5

[Vau94] Serge Vaudenay. On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In *Fast Software Encryption*, pages 286–297, 1994. 13

[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, *Advances in Cryptology – EUROCRYPT 2005: Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005. 1, 13, 14, 16, 25

[WY05] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005. 1, 13, 14

[WYY05a] Xiaoyun Wang, Andrew Yao, and Frances Yao. Cryptanalysis of SHA-1. Presented at the Cryptographic Hash Workshop hosted by NIST (Green Auditorium), Gaithersburg, Maryland, October 31 - November 1, 2005, October 2005. 1

[WYY05b] Xiaoyun Wang, Andrew Yao, and Frances Yao. New Collision Search for SHA-1. In *Presented at rump session of CRYPTO 2005*, 2005. 13

[WYY05c] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005. 1, 13, 14

[WYY05d] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005. 1, 13, 14

[YWZW05] Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The Second-Preimage Attack on MD4. In *Lecture Notes in Computer Science, Volume 3810, Nov 2005, Pages 1 - 12*, 2005. 14, 29