Stefan SCHERZ

Cryptanalysis of RIPEMD-160

DIPLOMA THESIS

written to obtain the academic degree of a Master of Science (MSc)

Diploma programme Technical Mathematics

Graz University of Technology

Advisors:
Univ.-Prof. Dr. Vincent RIJMEN
and
Dipl.-Ing. Dr.techn. Martin SCHLÄFFER

Institute of Applied Information Processing and Communications

Graz, March 2012

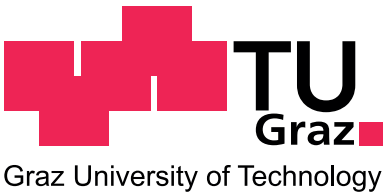
Stefan SCHERZ

Kryptoanalyse von RIPEMD-160

DIPLOMARBEIT

zur Erlangung des akademischen Grades eines Diplom-Ingenieur

Diplomstudium Technische Mathematik



Technische Universität Graz

Betreuer:
Univ.-Prof. Dr. Vincent RIJMEN
und
Dipl.-Ing. Dr.techn. Martin SCHLÄFFER

Institut für Angewandte Informationsverarbeitung und Kommunikationstechnologie

Graz, im März 2012

Diese Arbeit ist in englischer Sprache verfasst.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....
(Unterschrift)

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....
date (signature)

Acknowledgements

First of all, I would like to thank my advisor Martin Schläffer for his support. He not only lent a sympathetic ear to me and my concerns at any time, but also pointed out right directions. I am very thankful for the numerous hours and the effort he spent on discussions and corrections of drafts of this thesis, not to forget his endless patience in answering my numerous questions.

Further I would like to thank Florian Mendel for valuable comments on problems that arose and Vincent Rijmen for being the assessor.

Last but not least, I would like to express my sincere thanks to my family for their support throughout the whole years of studying.

Abstract

Hash functions play an important role in cryptography as they are used in various cryptographic applications and protocols. The most popular hash functions belong to the MD-family of cryptographic hash functions. Hence their security is of special interest. In 2004 the differential attacks by Wang *et al.* on the collision resistance of the MD-family attracted a lot of attention. Since then the members of the MD-family have been the target in many cryptographic attacks. Several commonly used members, *e.g.* MD5 and SHA-1, have been broken and cannot be considered secure anymore.
In this thesis the security of the ISO-standard hash function RIPEMD-160, also a MD-family member, is analyzed against collision attacks. Contrary to MD5 and SHA-1, RIPEMD-160 is a dual stream hash function and thus, considered to be much more secure. First methods for the differential cryptanalysis of RIPEMD-160 are discussed. Then, the different parts of the collision attack on step reduced but otherwise unmodified RIPEMD-160 are described in detail. For the attacks an automatic search tool is adapted, which is able to find complex non-linear differential characteristics as well as confirming message pairs.
We were able to construct differential characteristics for different approaches. Also a differential characteristic for both streams between round 1 and 2 has been found. Unfortunately it was not possible to also find a confirming message pair for this characteristic for RIPEMD-160. Using a modified variant of RIPEMD-320 we were at least able to verify this characteristic by constructing a semi-free start collision.

Kurzfassung

Hashfunktionen spielen eine bedeutende Rolle in der Kryptographie, da sie in verschiedenen kryptographischen Anwendungen und Protokollen verwendet werden. Die verbreitetsten Hashfunktionen gehören zur MD-Familie kryptographischer Hashfunktionen. Daher ist deren Sicherheit von besonderem Interesse. 2004 zogen die differentiellen Attacken von Wang *et al.* auf die Kollisionsresistenz der MD-Familie viel Aufmerksamkeit auf sich. Seit damals sind die Mitglieder der MD-Familie das Ziel vieler kryptographischer Attacken. Verschiedene weit verbreitete Mitglieder, z.B. MD5 und SHA-1, wurden gebrochen und können nicht mehr als sicher betrachtet werden.
In dieser Diplomarbeit wird die Sicherheit der ISO-Standard Hashfunktion RIPEMD-160, auch ein MD-Familienmitglied, gegen Kollisionsattacken analysiert. Im Gegensatz zu MD5 und SHA-1 ist RIPEMD-160 eine Hashfunktion mit zwei parallelen Berechnungslinien und wird daher als viel sicherer erachtet. Zunächst werden Methoden der differentiellen Kryptoanalyse von RIPEMD-160 besprochen. Dann werden die verschiedenen Teile der Kollisionsattacke auf schrittreduziertes, aber sonst unverändertes RIPEMD-160 im Detail beschrieben. Für die Attacken wird ein automatisches Suchwerkzeug angepasst, welches fähig ist komplexe nichtlineare differentielle Charakteristiken sowie der Charakteristik folgende Nachrichtenpaare zu finden.
Wir konnten differentielle Charakteristiken für unterschiedliche Ansätze konstruieren. Auch eine differentielle Charakteristik für beide Berechnungslinien zwischen Runde 1 und 2 wurde gefunden. Unglücklicherweise war es nicht möglich auch ein dieser Charakteristik folgendes Nachrichtenpaar für RIPEMD-160 zu finden. Unter Verwendung einer modifizierten Variante von RIPEMD-320 konnten wir zumindest diese Charakteristik durch Konstruktion einer sogenannten "semi-free start" Kollision verifizieren.

Contents

1 Introduction

In modern cryptography, cryptographic hash functions play an important role in many applications. Generally speaking, a cryptographic hash function maps a message of arbitrary finite length to a hash value of fixed length. The idea is that the hash value can serve as a short representation of the message and can be used as if it were uniquely identifiable with the message. The advantage of cryptographic hash functions is that instead of the long messages, the short hash value can be used to provide data integrity and message authentication, which is in general more efficient. Hence, cryptographic hash functions are used in various cryptographic applications and protocols. As their security is very important for those applications and protocols, also the cryptanalysis of hash functions plays an important role in cryptography [MOV97].

1.1 Cryptanalysis of Hash Functions

Cryptanalysis can be described as the science of analyzing cryptographic algorithms. In the process of analyzing these methods often new cryptographic attacks are developed or existing attacks are extended and improved.

Members of the MD family of hash functions are the most popular hash functions nowadays. Hence, their security and therefore their cryptanalysis is of special interest. MD4 [Riv91] can be seen as the origin of the whole MD family and was proposed in 1990 by Ron Rivest. As early cryptanalysis of MD4 already indicated weaknesses, Rivest proposed MD5 in 1991, a strengthened version of MD4. This nicely illustrates the interaction between the design and construction of cryptographic algorithms and their cryptanalysis. After weaknesses through cryptanalysis can be found new cryptographic algorithms are proposed to eliminate the discovered weaknesses. Also through cryptanalysis invented cryptographic attacks are taken into account in the design of new algorithms in order to resist the known attacks.
Although the successors of MD4 are more complex, they all follow the same design principles and have similar structures. Hence attack strategies for one member of the family are often adapted and improved to attack another member of the family.
In 2004 Wang *et al.* surprised the cryptographic community by announcing practical

collisions for MD4, RIPEMD [Wan+05], MD5 [WY05] and SHA-0 [WYY05a]. They introduced several new techniques in their attacks. Since then, many hash functions have been the target of cryptographic attacks improving and extending these powerful techniques.

Also RIPEMD-160 belongs to the MD family of hash functions and was proposed by Dobbertin, Bosselaers and Preneel in 1996 [DBP96]. But since then only few results for RIPEMD-160 have been published. One regarding the preimage resistance can be found in [OSS10]. Despite the breakthrough results of Wang *et al.* the only work concerning the collision resistance of RIPEMD-160 has been published by Mendel *et al.* in 2006 [Men+06]. In this work, the authors analyze the possibility to apply the attacks/ideas by Wang *et al.* [Wan+05] and Dobbertin [Dob97] to the dual-stream variants RIPEMD-128 and RIPEMD-160. They concluded that those methods cannot be used to attack full RIPEMD-160 respectively that they are not suitable for RIPEMD-160 reduced to three rounds due to a too high complexity. But it was not stated, if attacks reduced to two rounds are feasible.
The recent attacks on the collision resistance of RIPEMD-128 [MNS12] using a powerful automatic search tool, which was also used to successfully attack step-reduced SHA-2 [MNS11], motivated to use and modify this tool to attack two rounds of RIPEMD-160.

The goal of this thesis is to analyze the security of the ISO standard hash function RIPEMD-160 against collision attacks. Hence the theory behind differential attacks is examined as well as its relevance to practical application. Some of the steps in the analysis are done using an automatic search tool as doing them by hand seems almost impossible.
Although different difficulties arise because of the design of RIPEMD-160, it is possible to achieve some results and intermediate results for step-reduced RIPEMD-160. *E.g.* differential characteristics for certain areas can be found as well as a differential characteristic for both streams between round 1 and 2. Unfortunately no collision for RIPEMD-160 using this characteristic can be produced, but at least we are able to verify the characteristic by constructing a semi-free start collision for a modified variant of RIPEMD-320.

1.2 Thesis Outline

In Chapter 2 the definition and main security requirements of unkeyed hash functions as well as some basic properties are given. Furthermore the principles of iterated hash functions are described.
In Chapter 3 the general design properties of the MD-family of hash functions are

presented. As this thesis focuses on RIPEMD-160, it is described in more detail. Also an overview of the remaining hash functions of the MD-family is given.
Chapter 4 provides the basic terminology used in differential cryptanalysis. Further the propagation of differences through the functions used in RIPEMD-160 are considered. Finally an overview of differential attacks is given.
Chapter 5 describes the first part of the attack, the search for a good starting point, where different approaches are discussed.
In Chapter 6 the second part of the attack, the search for a differential characteristic is described in detail. Hence some general considerations on RIPEMD-160 specific properties as well as the used search strategies in combination with an automatic search tool are provided. Also some results are presented to illustrate the different strategies.
In Chapter 7 the last part of the attack, the search for a confirming message pair is considered. The difficulties of finding a confirming message pair in RIPEMD-160 are discussed and an alternative approach is presented.
Chapter 8 concludes and the obtained results and intermediate results are presented in the Appendix.

2 Cryptographic Hash Functions

Hash functions can be divided at a high level into two classes: unkeyed and keyed hash functions. This thesis only deals with unkeyed hash functions or more precisely with the subclass modification detection codes (MDCs). Therefore, in this chapter their definition as well as their main security requirements are discussed. Also the iterated hash function construction is described.

2.1 Definition and Properties

A hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ maps bit strings of arbitrary finite length into bit strings of fixed length n. The input is called message M and the output is the hash value $h = H(M)$. The size n of the hash value is often denoted as hash length. Further a hash function has to be efficiently computable and each hash-value should be a unique and randomly looking representation of the input message.
Mathematically a hash functions is a function that maps x elements to y elements with $x > y$. Therefore a unique association between those elements is impossible. The consequence for a hash function is that collisions are guaranteed. In practice a hash value should be uniquely identifiable with a single input message. Hence it should be computationally infeasible to find collisions [MOV97].

2.1.1 Main Security Requirements

A cryptographic hash function has to satisfy additional properties since they are used in many diverse applications. Hence the following three main security requirements evolved:

- *preimage resistance:* it is computationally infeasible to find any input message M which results in any pre-specified hash value h, *i.e.* for any given $h = H(M)$ find the unknown M.

- *second preimage resistance:* it is computationally infeasible to find any second input message M' which has the same hash value h as any specified input message M, *i.e.* for any given M find a second $M' \neq M$ such that $H(M') = H(M) = h$.
- *collision resistance:* it is computationally infeasible to find any two distinct input messages M, M' which result in the same hash value h, *i.e.* find M and $M' \neq M$ such that $H(M) = H(M') = h$.

With these properties the subclass MDC of unkeyed hash functions can be further classified:

- A *one-way hash function* has to satisfy the properties preimage resistance and second preimage resistance. These two properties are therefore also denoted as one-way property.
- A *collision resistant hash function* is usually a one-way hash function that also satisfies the collision resistance property.

Remark: In the strict definition of collision resistant hash functions they only satisfy the properties second preimage resistance and collision resistance, as collision resistance only implies second preimage resistance but does not guarantee preimage resistance. In practice they almost always have the preimage resistance property [MOV97].

2.1.2 Generic Attacks

The three main security requirements are often set in relation to the bit length n of the hash value in order to observe some mathematical results about cryptographic hash functions that are independent of the hash algorithm.

A naive method is the brute-force attack, where a random input message M (of bounded bitlength) is picked and $H(M)$ is computed. Assuming the hash function approximates a uniform random variable, the probability of a match is 2^{-n}. So for any hash function a preimage or a second preimage can be found by guessing approximately 2^n random input messages M, computing $H(M)$ and checking if $H(M)$ matches the given respectively calculated h.

The complexity of the generic attack to find collisions differs due to the birthday paradox. A birthday attack allows to find collisions for any function f with output size n with an optimal complexity of $2^{n/2}$.
The two most well-known birthday attack implementations are Yuval's birthday attack and Floyd's cycle-finding algorithm. Both attacks require $\mathcal{O}(2^{n/2})$ time, but while Yuval's birthday attack also requires $\mathcal{O}(2^{n/2})$ storage, Floyd's cycle-finding algorithm

can be considered memoryless. Further details about the two algorithms can be found in [MOV97].

As both obtained results hold for any function f with an output size n, a cryptographic hash function that meets these bounds is called ideal hash function.
A hash function is considered to be broken if the ideal requirements are not met, *i.e.* if preimages or second preimages can be found in less than 2^n respectively collisions in less than $2^{n/2}$ computations.

The hash length n is usually chosen large enough to make generic attacks computationally infeasible. Hence common hash lengths range between $n = 128$ and $n = 512$ bits, because then it is impossible to find collisions or preimages by these generic attacks.

2.2 Iterated Hash Function Construction

Most commonly used hash functions use some kind of iteration to map the arbitrary long input message to the hash value of fixed length n. An iterated hash function makes use of some compression function F which maps an input of size w to an output of size m with $w \geq m$. In each iteration a part of the message M_i is used as input to the compression function which then outputs the chaining value H_i. Iterated hash functions may also use a final output transformation G which maps the last compression function output of size m to the final hash value of size n. The output transformation is often the identity function or a truncation. Figure 2.1 illustrates an iterated hash function which uses the same compression function in each iteration.

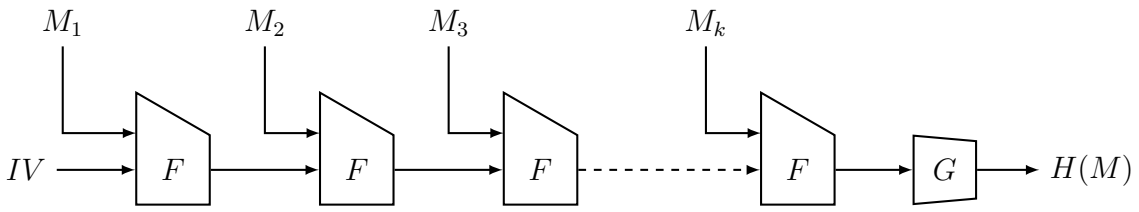


Figure 2.1: Iterated hash function construction

Formally let $H : \{0,1\}^* \to \{0,1\}^n$ be an iterated hash function, which uses a compression function $F : \{0,1\}^w \to \{0,1\}^m$ and a output transformation $G : \{0,1\}^m \to \{0,1\}^n$. The input message is split into k message blocks $M_1 \parallel \ldots \parallel M_k$ of size w. To

ensure that the input message length is a multiple of the block size w some padding is used. Then the hash value $h = H(M)$ is computed as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= F(H_{i-1}, M_i) \text{ for } 1 \leq i \leq k \\ h &= G(H_k) \end{aligned}$$

IV denotes the predefined initial value, which initializes the m-bit intermediate variable H_i called chaining value.

Remark: Sometimes other additional inputs to the compression function are used in iterated constructions, *e.g.* a salt or a counter.

The security of an iterated hash function depends on the security of the compression function F, on the output transformation G and on the bitsize m of the intermediate chaining values.
The most commonly used strategy is the Merkle-Damgård design principle [Dam89; Mer89]. There the bitsize m of the intermediate chaining values can be as small as the bitsize n of the final hash value, but the compression function F has to be secure. The Merkle-Damgård Theorem states that an iterated hash function is collision resistant if the used compression function is collision resistant. Further the Merkle-Damgård strengthening requires that the padding includes the length of the message and that the initial value is fixed to some predefined constant.

3 RIPEMD and Other Hash Functions of the MD-Family

In this chapter the MD-family of hash functions is described. First an overview over the common properties of the members of this family is given. As this thesis focuses on RIPEMD-160, we then describe this hash function in more detail. Finally the remaining members of the MD-family are briefly covered by dividing the whole family into three subfamilies, where the main focus lies on the RIPEMD-subfamily. Also some collision attacks on the hash functions of those subfamilies are mentioned.

3.1 General Design

Members of the MD-family are iterative hash functions based on the Merkle-Damgård design principle (see Section 2.2). Further the hash functions are word-oriented, which means that all data is divided into words of a certain length (w bits). The compression function uses operations on words of this length. The word length w of the MD-family hash functions is 32 or 64 bits [Rom04]. Furthermore they have several other similarities regarding the construction of the compression function.
The compression function can be split into two major parts, the message expansion and the state update transformation. The state update transformation processes the expanded message while following a sequential structure. This means that a certain number of similar operations called step operations are executed sequentially. Apart from the dependence on the (expanded) message words, these single steps only vary little. After the last step the output of the compression function is computed from the final state variables and the input chaining variables. This prevents simple inversion of the compression function and therefore simple meet-in-the-middle attacks. This kind of output computation is also very similar to the Davies-Meyer mode of constructing a compression function from a block cipher. Figure 3.1 illustrates the common structure of the MD-family compression function.

The purpose of message expansion is that each message block is used more than once in the compression function. Therefore the message expansion describes how the message

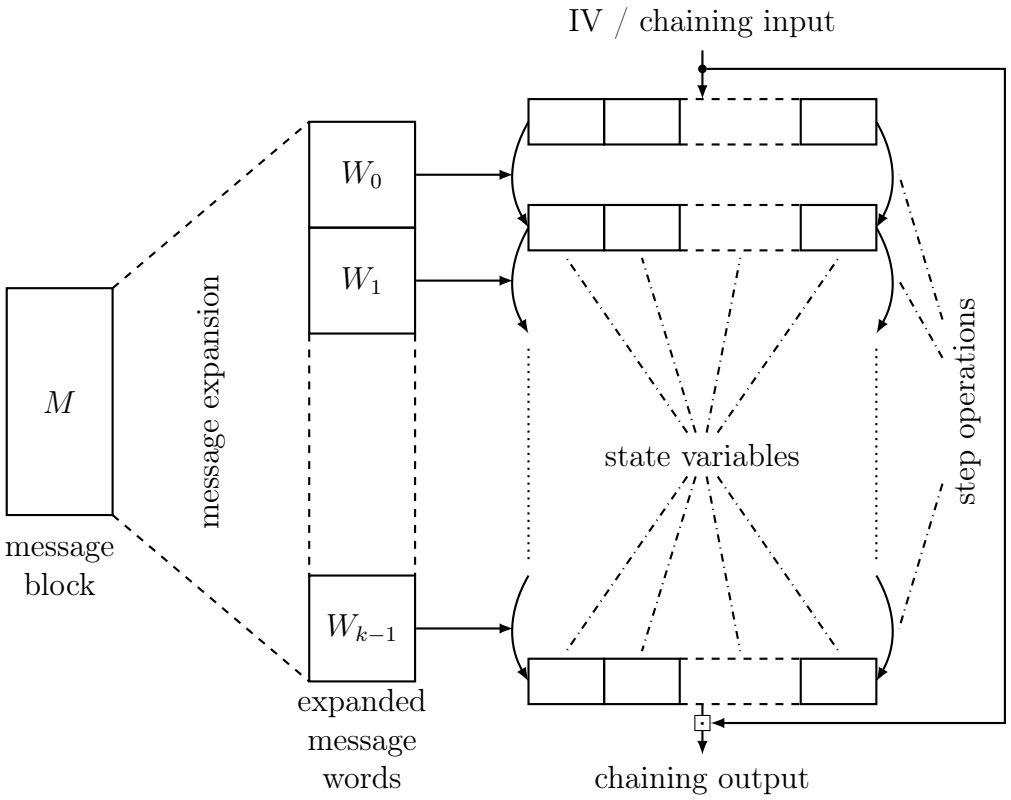


Figure 3.1: Common structure of the compression function of the MD-family

words, which serve as input for the single steps of the state update transformation, are derived from the message block. For the MD-family there exist two different methods of message expansion, the round-wise permutations and the recursive message expansion. In round-wise permutation the steps are grouped together to rounds consisting of n steps each and the message block is split into n message words of w bits each. Then each of the message words is used in one step as input in every round, but in each round the message words are used in a different order. In the recursive message expansion the message block is split into n message words of w bits each, which serve as starting values. The following message words are computed recursively from the previous message words. That has the advantage of an increased diffusion as nearly all message words depend on nearly all starting values. Hence a small change in the message block and thus in one starting value affects many steps. More details about the MD-family message expansions can be found in [Dau05].
In each step of the compression function one expanded message word is used to update a certain (small) number of state variables by the state update transformation. As already mentioned the operations in each step of the state update transformation are

very similar in every hash function of the MD-family. The requirements concerning these step operations are not only to be very efficient, as they are used in every step, but also to be hard to analyze and to provide good diffusion. In order to be efficient the members of the MD-family only change one or at most two state variables in each step by using a combination of very efficient and simple basic operations. This combination of operations follows the ARX design principle, so it consists of the following basic operations on words:

A Integer addition modulo 2^w (modular additions)

R Bit shifts and rotations

X Bitwise Boolean operations

These specific efficient basic operations are chosen, because the mixing of Boolean functions and addition is believed to be cryptographically strong and hence they fulfill the second requirement concerning step operations. The operations in each step of the state update transformation only vary in some specific parameters (e.g. number of bits to rotate or used constants) and/or in the used Boolean function (especially in round-based structures).
The hash functions of the MD-family use the following Boolean functions:

$$\begin{aligned}
\mathbf{XOR}(X,Y,Z) &:= X \oplus Y \oplus Z \\
\mathbf{MAJ}(X,Y,Z) &:= (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z) \\
\mathbf{IF}(X,Y,Z) &:= (X \wedge Y) \oplus (\neg X \wedge Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus Z \\
\mathbf{ONX}(X,Y,Z) &:= (X \vee \neg Y) \oplus Z = (X \wedge Y) \oplus Y \oplus Z \oplus 1
\end{aligned}$$

The non-symmetric functions IF and ONX are sometimes also applied with swapped parameters. These functions also have been chosen, because they are supporting a strong avalanche effect, which means that small differences in the state variables are mapped to large differences in only very few steps. A detailed analysis of these functions can be found in [Dau05].

3.2 RIPEMD-160

RIPEMD-160 was designed by Dobbertin, Bosselaers and Preneel in 1996 as a replacement for RIPEMD and is described in detail in [DBP96]. It belongs to the MD-family of hash functions and builds together with other RIPEMD variants and Extended MD4 the RIPEMD-subfamily (see Subsection 3.3.2). RIPEMD-160 is also part of the ISO/IEC international standard ISO/IEC 10118-3:2004 on dedicated hash functions, together with six other hash functions (RIPEMD-128, SHA-1, SHA-2 and WHIRLPOOL). As this thesis is focused on RIPEMD-160, it is described in more detail.

3.2.1 Standard Description

RIPEMD-160 produces a 160-bit hash value of an input message with a maximum length of 2^{64} bits. It processes message blocks of 512 bits. To guarantee that the input message length is an exact multiple of 512 bits the input message is padded using an unambiguous padding method, which is identical to the one of MD4. First, the padding method appends a single '1' bit to the message, then '0' bits such that the length in bits of the padded message becomes congruent to 448 modulo 512. Finally a 64-bit representation of the length of the message before the padding bits were added is appended to the padded message. This way the padded message is an exact multiple of 512 bits [Riv92a; Riv91].
Like in the other members of the RIPEMD subfamily the compression function of RIPEMD-160 consists of two parallel streams of computations. In each stream the corresponding message block is used to update the state variables. After the computations the results of both streams are combined with the chaining input, which is illustrated in Figure 3.2. The two streams of RIPEMD-160 are designed more differently than those of RIPEMD or Extended MD4, which are basically identical parallel streams (for details see Section 3.3.2). Hence, in the following description of the compression function the differences between the left and the right stream are pointed out.

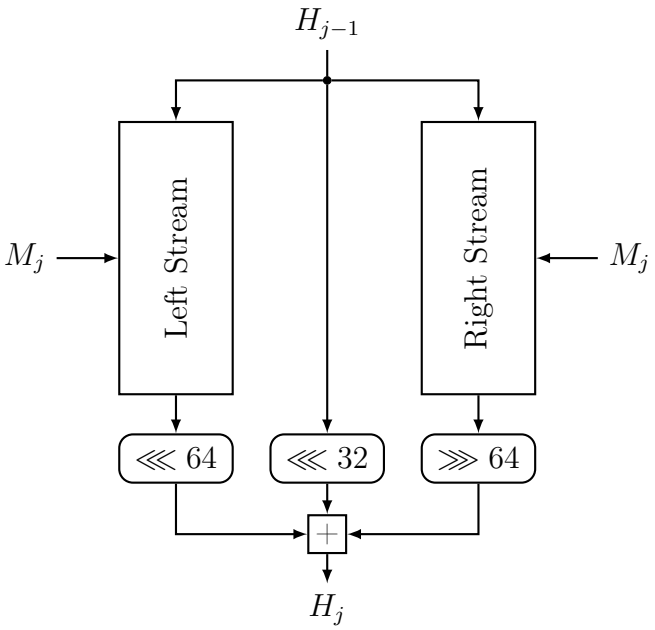


Figure 3.2: Structure of the RIPEMD-160 compression function

Both streams are initialized with the five 32-bit initial values respectively the five 32-bit chaining input values $h_0, \ldots, h_4$, so $A = h_0 = A', \ldots, E = h_4 = E'$. Each stream consists of a message expansion and a state update transformation.

The *state update transformation* updates five state variables $A, \ldots, E$ of 32 bits each in five rounds of 16 steps each using one expanded message word W_i in each step. The whole update process for one step in one stream is given by the following equations:

$$\begin{aligned} T &= ((A + f(B, C, D) + W_i + K_i) \lll s) + E \\ A &= E,\ E = D,\ D = (C \lll 10),\ C = B,\ B = T \end{aligned} \tag{3.1}$$

Figure 3.3 shows one step of the state update transformation of each stream.

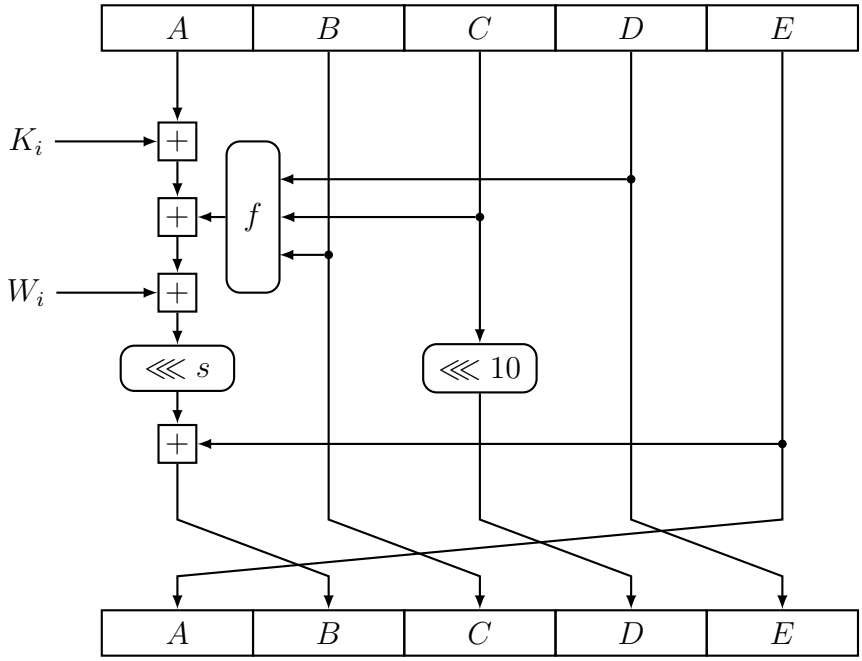


Figure 3.3: One step of the state update transformation of RIPEMD-160

The Boolean function f is different in each of the five rounds:

$$\begin{aligned} f_1(X, Y, Z) &= X \oplus Y \oplus Z &&= XOR(X, Y, Z) \\ f_2(X, Y, Z) &= (X \wedge Y) \vee (\neg X \wedge Z) &&= IF(X, Y, Z) \\ f_3(X, Y, Z) &= (X \vee \neg Y) \oplus Z &&= ONX(X, Y, Z) \\ f_4(X, Y, Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) &&= IF(Z, X, Y) \\ f_5(X, Y, Z) &= X \oplus (Y \vee \neg Z) &&= ONX(Y, Z, X) \end{aligned}$$

f_r is used for the r-th round in the left stream and f_{6-r} for the r-th round in the right stream with $1 \leq r \leq 5$.
A constant K_i is added in every step. This constant is different for each stream and for each round. The actual values can be found in [DBP96].
In each stream and each step different rotation values s are used. These values can be found in Table 3.1.

	Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Left Stream	Round 1	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
	Round 2	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12
	Round 3	11	13	6	7	14	9	13	15	14	8	13	6	5	12	7	5
	Round 4	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12
	Round 5	9	15	5	11	6	8	13	12	5	12	13	14	11	8	5	6
Right Stream	Round 1	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
	Round 2	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
	Round 3	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
	Round 4	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8
	Round 5	8	5	12	9	12	5	14	6	8	13	6	5	15	13	11	11

Table 3.1: Rotation values s for each step and each stream of RIPEMD-160

The *message expansion* of RIPEMD-160 is a round-wise permutation of the 16 message block words, but for the left and the right stream different permutations are used. The message block words are permuted according to Table 3.2.

	Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Left Stream	Round 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	Round 2	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8
	Round 3	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12
	Round 4	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2
	Round 5	4	0	5	9	7	12	2	10	14	1	3	8	11	6	15	13
Right Stream	Round 1	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
	Round 2	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
	Round 3	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
	Round 4	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14
	Round 5	12	15	10	4	1	5	8	7	6	2	13	14	0	3	9	11

Table 3.2: Index of the message block words, which are used as expanded message words W_i in each step and each stream of RIPEMD-160

After the last step of the state update transformation, the output values of the last

step of the left stream $A, \ldots, E$ and of the right stream $A', \ldots, E'$ are combined with the chaining input values respectively the initial values $h_0, \ldots, h_4$:

$$T = h_1 + C + D', \; h_1 = h_2 + D + E', \; h_2 = h_3 + E + A',$$
$$h_3 = h_4 + A + B', \; h_4 = h_0 + B + C', \; h_0 = T$$

The final values of one iteration $h_0, \ldots, h_4$ are either the final hash value or the chaining input for the next message block.

3.2.2 Alternative Description

In this thesis an alternative description of RIPEMD-160 is used. Like [MNS12] suggests, it is possible to describe the state update transformation by only considering a single state variable. Hence the following mapping for the input values $A, \ldots, E$ of one step of the left stream respectively $A', \ldots, E'$ of one step of the right stream is used:

$$A = (A_{i-5} \lll 10), \; B = A_{i-1}, \; C = A_{i-2}, \; D = (A_{i-3} \lll 10), \; E = (A_{i-4} \lll 10)$$
$$A' = (B_{i-5} \lll 10), \; B' = B_{i-1}, \; C' = B_{i-2}, \; D' = (B_{i-3} \lll 10), \; E' = (B_{i-4} \lll 10)$$

With these substitutions the update process for one step in one stream given in (3.1) can be rewritten as

$$A_i = (((A_{i-5} \lll 10) + f(A_{i-1}, A_{i-2}, (A_{i-3} \lll 10)) + W_i + K_i) \lll s) + (A_{i-4} \lll 10) \tag{3.2}$$

It can be easily seen that the temporary value T can be avoided as well as the whole handing over process of the remaining state variables. Figure 3.4 illustrates one step of the state update transformation of one stream in the alternative description.

Both streams are initialized by $A_{-5} = h_0 = B_{-5}, A_{-1} = h_1 = B_{-1}, \ldots, A_{-4} = h_4 = B_{-4}$. It is worth noting that the initial values A_{-5}, A_{-4}, A_{-3} and B_{-5}, B_{-4}, B_{-3} are not rotated 10 bits to left in (3.2). So the update process in the first three steps of each stream is slightly different.
The equations for the final values of one iteration $h_0, \ldots, h_4$ can be rewritten as follows:

$$\begin{aligned}
h_0 &= A_{-1} + A_{78} &&+ (B_{77} \lll 10) \\
h_1 &= A_{-2} + (A_{77} \lll 10) &&+ (B_{76} \lll 10) \\
h_2 &= A_{-3} + (A_{76} \lll 10) &&+ (B_{75} \lll 10) \\
h_3 &= A_{-4} + (A_{75} \lll 10) &&+ B_{79} \\
h_4 &= A_{-5} + A_{79} &&+ B_{78}
\end{aligned}$$

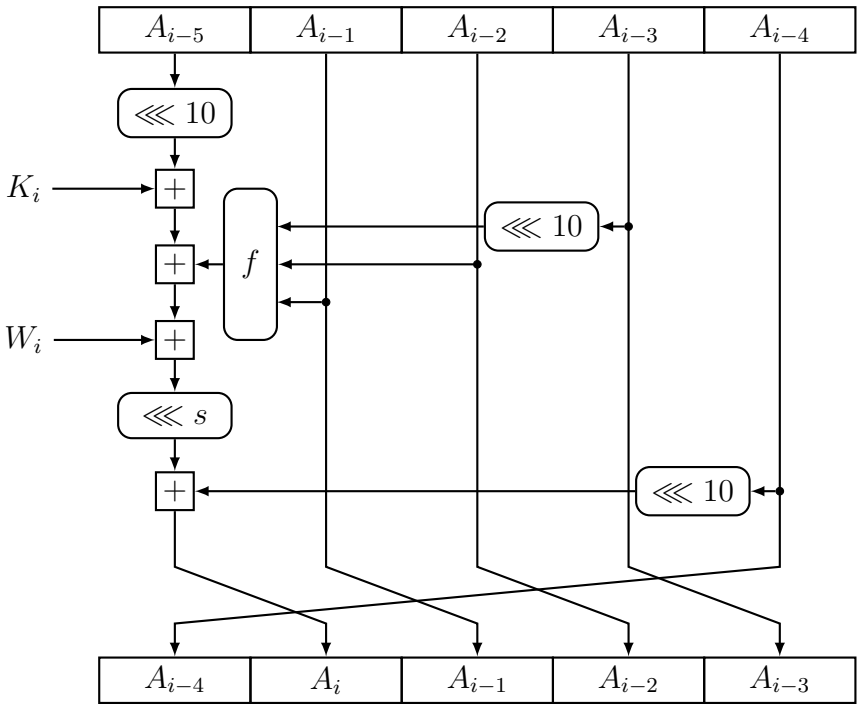


Figure 3.4: One step of the state update transformation of RIPEMD-160 in the alternative description

Remark: Instead of adapting the update process in the first three steps, it is also possible to rotate the initial values and adapt the equations for the final values of one iteration.

3.3 Other Hash Functions of the MD-Family

In this section a short overview of the remaining hash functions of the MD-family is given. They are categorized into three subfamilies according to two main distinctions. First the message expansion is either a round-wise permutation or a recursive message expansion. The second criterion is the number of parallel lines of computation.

3.3.1 MD-Subfamily

MD4 and MD5 form this subfamily, as both use one line of computation and round-wise permutations as message expansion method. Both of them were designed by Ronald Rivest and produce a 128-bit hash value by processing message blocks of 512 bits.
The compression function of MD4 consists of three rounds of 16 steps each. In each

round a different boolean function ($IF(X,Y,Z)$, $MAJ(X,Y,Z)$ and $XOR(X,Y,Z)$) as well as a different constant are used. In each step operation a rotation is used as well, but only a few different rotation values are used that follow a certain pattern throughout the whole compression function. The exact description of the step operation and further details can be found in [Riv92a; Riv91].
MD5 is the successor of MD4 and hence quite similar. It was designed to fix certain vulnerabilities of MD4. The compression function of MD5 has four rounds of 16 steps each. In addition to the application of a Boolean function in each step also a further state variable is added after the rotation. Further in MD5 each step has an unique constant and the rotation values of each round are unique. Details can be found in [Riv92b].

Both MD4 and MD5 are considered broken. The most important recent attack on MD4 was published by Wang *et al.* in 2004 [Wan+05]. There they presented an attack, which can find a collision with a complexity of less than 2^8 MD4 hash operations. Also further optimizations to this collision attack have been proposed, *e.g.* by Schläffer and Oswald [SO06] or Sasaki *et al.* [Sas+08].
Wang and Yu also presented a collision attack on MD5 in the same year with complexity 2^{39} [WY05]. Since then several improvements to those attacks have been published *e.g.* by Klíma [Klí06] or Stevens [Ste06].

3.3.2 RIPEMD-Subfamily

The RIPEMD-subfamily consists of RIPEMD, the strengthened versions RIPEMD-128, RIPEMD-160, RIPEMD-256 and RIPEMD-320, but also Extended MD4. All of them use round-wise permutations for the message expansion and process 512-bit message blocks. The crucial difference to other MD-family members is that two parallel lines of computations are used.
RIPEMD, sometimes also denoted RIPEMD-0, was proposed as a strengthened version of MD4. The compression function of RIPEMD consists of two parallel lines of the MD4 compression function with different improved parameters. So compared to MD4 the round constants, the rotation values and the used permutation of the message words are different. Both lines only differ in the round constants, so message expansion and rotation values are the same for both lines. At the end of the compression function the words of the left line, the right line and the chaining input are combined to produce a 128-bit hash value respectively chaining output. Details can be found in [BP95].
RIPEMD-128 was designed as an instant replacement of the vulnerable RIPEMD, as it also produces a 128-bit hash value. The number of rounds is increased from three to four and the two lines are made more different. Not only the constants, but also the Boolean functions and permutation of message words are different in the two lines.

The message expansion is the same as the one of RIPEMD-160 up to round four and also the same rotation values are used (Tables 3.2 and 3.1). The step operation of RIPEMD-128 is still the same as the one of MD4, but also different Boolean functions are used and not only different parameters like in RIPEMD. The actual values of the parameters and other details can be found in [DBP96].
RIPEMD-256 and RIPEMD-320 are optional extensions of RIPEMD-128 and RIPEMD-160. The compression function is almost the same, the only two differences are that after each round there is an interaction between the two lines (swapping of two state variables) and that the combination of the two lines at the end of the compression function is omitted to achieve the desired double length of the hash values.
Extended MD4 was proposed as an extension of MD4 by Ronald Rivest to produce a 256-bit hash value. Its compression function consists of two lines of MD4, which only differ in the initial values and the round constants of the second and third round. There is no interaction between both lines in the compression function, the only interaction happens between the iterations of the compression function. More details can be found in [Riv91].

RIPEMD and Extended MD4 are broken. The collision attack on RIPEMD with complexity 2^{16} was also published by Wang *et al.* in 2004 [Wan+05]. Last year Wang presented a collision attack on Extended MD4 with complexity 2^{37}.
The most recent result is the collision attack on 38 steps of RIPEMD-128 by Mendel, Nad and Schläffer [MNS12] with a complexity of 2^{14}. It also seems to be the first published collision attack on RIPEMD-128.

3.3.3 SHA-Subfamily

The SHA-family is formed by the hash functions SHA, SHA-1 and SHA-2. The compression functions use only one line of computations, but the message expansions are recursive message expansions, so the expanded message words are computed by recursively defined functions.
SHA, now also denoted SHA-0, and SHA-1 produce 160-bit hash values and process 512-bit message blocks. The compression function consists of four rounds of 20 steps and in each round a different Boolean function is used. Unlike the other MD-family members the rotation values are constant and state variables instead of intermediate results are rotated in each step.
The remaining members form the SHA-2 subfamily. They not only differ in the hash value size (224, 256, 384 and 512 bits), but also in the complexity of the step operations, as more than one state variable is updated in each step. Further not only bitwise Boolean functions, but also other auxiliary functions are used in each step. It is worth to note that the same functions are used in all steps of the compression function, so the steps only differ in the constants and expanded message words.

In contrast to the other SHA variants, SHA-384 and SHA-512 use 64-bit words and process message blocks of 1024 bits each. Further details on all SHA-subfamily members can be found in [ST93; ST08].

The hash functions SHA-0 and SHA-1 are considered broken. Wang, Yin and Yu presented collision attacks on SHA-0 with complexity 2^{39} [WYY05a] and for SHA-1 with complexity 2^{69} [WYY05b]. For SHA-1 an improvement of the complexity to 2^{63} was presented by Wang, Yao and Yao in 2005. Also a practical collision for SHA-1 was found for 75 steps by Grechnikov [GA11].
For SHA-2 the currently best collision attack is a 27-step collision published by Mendel, Nad and Schläffer last year [MNS11].

4 Differential Cryptanalysis & Attacks

Differential cryptanalysis was first published by Biham and Shamir in 1990 for the block cipher DES [BS90]. Den Boer and Bosselaers presented the first results on hash functions, namely MD5 [BB93].
An obvious target for differential attacks is the collision resistance of a hash function. In a collision attack two distinct input messages M and M' which result in the same hash value need to be found. From a differential point of view a non-zero input difference should result in a zero output difference (Section 4.1). So the main idea of differential attacks is to consider the propagation of differences between two distinct input messages without considering the actual values of the input messages. The propagation of differences is predicted and results in a sequence of differences. This sequence should produce a zero output difference, *i.e.* a collision, with high probability. Finally a message pair needs to be found, that follows the sequence of differences throughout the hash function.

4.1 Definitions and Terminology

In differential cryptanalysis most commonly two different types of differences are considered, the bitwise XOR differences and wordwise modular differences.
The bitwise XOR difference is defined as follows:

Definition 4.1. *[Sch11] Let X and X^* be two n-bit vectors. The n-bit XOR difference is defined by*

$$\Delta^{\oplus} X := \Delta^{\oplus}(X, X^*) = X \oplus X^*$$

The modular difference is defined as follows:

Definition 4.2. *Let X and X^* be two n-bit values. The n-bit modular difference is defined by*

$$\Delta^{+} X := \Delta^{+}(X, X^*) = X - X^* \mod 2^n$$

In differential attacks, not only differences but sometimes also values need to be considered. On one hand XOR differences do not keep track of the specific value of the involved bit, on the other hand modular differences cannot be used in an easy way in combination with bitwise defined Boolean functions. Hence Wang *et al.* introduced signed differences in their attacks on MD4 and RIPEMD [Wan+05]. A signed difference can be defined as follows:

Definition 4.3. *[SO06] Let X and X^* be two n-bit vectors, then the bitwise n-bit signed difference is defined by*

$$\Delta X := \Delta(X, X^*) = (x_{n-1} - x^*_{n-1}, \ldots, x_0 - x^*_0)$$

respectively $(\delta x_{n-1}, \ldots, \delta x_0) := (\delta(x_{n-1}, x^*_{n-1}), \ldots, \delta(x_0, x^*_0)) = (x_{n-1} - x^*_{n-1}, \ldots, x_0 - x^*_0)$

$\delta x_i = x_i - x^*_i \in \{-1, 0, +1\}, 0 \leq i \leq n-1$ is also called a single bit signed difference.

In [Dau05] Daum proves that a signed difference ΔX uniquely determines the XOR difference $\Delta^{\oplus} X$ and the modular difference $\Delta^{+} X$.

In [CR06] De Cannière and Rechberger generalized the concept of signed differences even further by allowing characteristics to impose arbitrary conditions on values of pairs of bits. These conditions take all 16 possible conditions on a pair of bits into account.

Definition 4.4. *Let X and X^* be two n-bit vectors and x_i respectively x^*_i the i-th bit of those vectors. Then all 16 possible conditions on a pair of bits (x_i, x^*_i) are called generalized conditions on a pair of bits and denoted by ∇x_i. The generalized conditions on a pair of n-bit vectors (X, X^*) are then defined as*

$$\nabla X := [\nabla x_{n-1}, \ldots, \nabla x_0]$$

∇X i.e. represents a set, containing the values for which the conditions are satisfied.

To write ∇X in a compact way the notation in Table 4.1 is used, which lists all possible conditions on a pair of bits $(x_i, x^*_i) = \nabla x_i$. Sometimes a differential characteristic using generalized conditions instead of simple differences is also called generalized characteristic.

Example 4.1: A short example should illustrate the concept of generalized conditions and its notation. Let X and X^* be two 8-bit vectors and ∇X the following set:

$$\nabla X = \{(X, X^*) \mid x_7 \vee x^*_7 = 1, x_5 \wedge x^*_5 = 0, x_4 = x^*_4 = 1, x_3 \neq x^*_3, x_i = x^*_i \text{ for } 0 \leq i \leq 2\}$$

This can be rewritten using the notation of Table 4.1 as:

$$\nabla X = [\texttt{E?71x---}]$$

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

Table 4.1: Notation for possible generalized conditions on a pair of bits

As can be easily seen the conditions $\{\texttt{x}, \texttt{-}\}$ represent the XOR differences $\{\texttt{1}, \texttt{0}\}$ and $\{\texttt{n}, \texttt{-}, \texttt{u}\}$ the signed differences $\{\texttt{-1}, \texttt{0}, \texttt{+1}\}$.

Other commonly used terms in connection with differential cryptanalysis are zero difference and non-zero difference, which are defined as follows:

Definition 4.5. *Let ΔX be a difference, then ΔX is a zero difference if $\Delta X = 0$ and ΔX is a non-zero difference if $\Delta X \neq 0$.*
Furthermore, a zero difference in step i of a hash function contains zero differences in all state variables of step i, e.g. for RIPEMD-160:

$$(\Delta A_{i-4}, \Delta A_i, \Delta A_{i-1}, \Delta A_{i-2}, \Delta A_{i-3}) = (0,0,0,0,0)$$
$$\textit{and} \quad (\Delta B_{i-4}, \Delta B_i, \Delta B_{i-1}, \Delta B_{i-2}, \Delta B_{i-3}) = (0,0,0,0,0)$$

A collision is then a zero difference after the last step. But not only this "global" collision is considered in differential attacks. Very often step-reduced collisions are considered, so collisions up to a certain step i, which means a zero difference in step i. Another commonly used term is the local collision, which can be defined as follows:

Definition 4.6. *A local collision is a collision over a certain (small) number of steps $i_1, \ldots, i_k$ of the compression function. Hence in step $i_1 - 1$ and step i_k there are zero differences, whereas between i_1 and i_k there are non-zero differences.*

In differential cryptanalysis the propagation of differences through functions are considered which leads to the following definitions:

Definition 4.7. *A differential for a function f consists of an input difference ΔX and an output difference ΔY and is written as*

$$\Delta X \xrightarrow{f} \Delta Y$$

Very often functions can be split into smaller sub-functions, *e.g.* the compression function into step functions. The sequence of differences in these sub-functions is usually called a differential characteristic or differential path.

Definition 4.8. *A differential characteristic through a function with k sub-functions f_i and $f = f_k \circ f_{k-1} \circ \ldots \circ f_2 \circ f_1$ consists of $k+1$ differences ΔX_i and is denoted by*

$$\Delta X_0 \xrightarrow{f_1} \Delta X_1 \xrightarrow{f_2} \Delta X_2 \xrightarrow{f_3} \ldots \xrightarrow{f_{k-1}} \Delta X_{k-1} \xrightarrow{f_k} \Delta X_k$$

An alternative way to describe a differential characteristic is based on the step structure of the compression function of the MD-family. Hence a differential characteristic is a certain sequence of differences in the state variables over a certain number of steps. Sometimes also the differences in the (expanded) message words are included in the differential characteristic.

Also the differential probability of a differential for a function f plays an important role in differential cryptanalysis. Informally the differential probability can be described as follows. Let $\Delta X \rightarrow \Delta Y$ be a differential and k be the number of n-bit vector pairs (X, X^*) with input difference ΔX and output difference ΔY. Then the differential probability can be defined as $DP(\Delta X \rightarrow \Delta Y) = k \cdot 2^{-n}$. As at most 2^n pairs exist, the differential probability lies in the range $[0, 1]$.
More formally the differential probability is defined as follows depending on the considered difference:

Definition 4.9. *For XOR differences the differential probability of a differential is defined as*

$$DP(\Delta^{\oplus} X \xrightarrow{f} \Delta^{\oplus} Y) = 2^{-n} \cdot \#\{X \mid f(X \oplus \Delta^{\oplus} X) \oplus f(X) = \Delta^{\oplus} Y\}$$

For modular differences the differential probability is defined as

$$DP(\Delta^{+} X \xrightarrow{f} \Delta^{+} Y) = 2^{-n} \cdot \#\{X \mid f(X + \Delta^{+} X) - f(X) = \Delta^{+} Y\}$$

4.2 Basic Operations Addition and Rotation

In this section the propagation of differences through the basic operations addition and rotation is analyzed in detail. Also some observations about the carries are given.

4.2.1 Rotation

As in each step of RIPEMD-160 rotations are used, the rotation of signed differences respectively generalized conditions should be shortly described here.

First of all the rotation is non-redundant. To rotate 32-bit signed differences ΔX or generalized conditions ∇X over s bits to left, each element δx_i respectively ∇x_i is rotated as follows:

$$\delta x_i \lll s = \delta x_{(i+s) \mod 32}$$
$$\nabla x_i \lll s = \nabla x_{(i+s) \mod 32}$$

Example 4.2: $\nabla X = [$`??--0D--x-3--7-n-u?-????-1C-7A??`$]$ is rotated over 10 bits to the left, which results in:

$$\begin{aligned}\nabla X \lll 10 &= [\texttt{??--0D--x-3--7-n-u?-????-1C-7A??}] \lll 10 \\ &= [\texttt{3--7-n-u?-????-1C-7A????--0D--x-}]\end{aligned}$$

4.2.2 Addition

Let single signed bit differences be uniquely determined, in the notation of generalized conditions `0`, `n`, `u`, `1`. Assuming there is no carry at the considered bit position the addition results in

$$\delta s = \delta x + \delta y = \begin{cases} \texttt{n} & \text{if } (\delta x, \delta y) = (\texttt{0},\texttt{n}) \text{ or } (\texttt{n},\texttt{0}) \text{ or } (\texttt{1},\texttt{u}) \text{ or } (\texttt{u},\texttt{1}) \\ \texttt{-} & \text{if } (\delta x, \delta y) = (\texttt{n},\texttt{n}) \text{ or } (\texttt{n},\texttt{u}) \text{ or } (\texttt{u},\texttt{n}) \text{ or } (\texttt{u},\texttt{u}) \\ \texttt{u} & \text{if } (\delta x, \delta y) = (\texttt{0},\texttt{u}) \text{ or } (\texttt{u},\texttt{0}) \text{ or } (\texttt{1},\texttt{n}) \text{ or } (\texttt{n},\texttt{1}) \end{cases}$$

with the corresponding carry

$$\delta c = \begin{cases} \texttt{n} & \text{if } (\delta x, \delta y) = (\texttt{n},\texttt{n}) \text{ or } (\texttt{1},\texttt{n}) \text{ or } (\texttt{n},\texttt{1}) \\ \texttt{0} & \text{if } (\delta x, \delta y) = (\texttt{0},\texttt{n}) \text{ or } (\texttt{n},\texttt{0}) \text{ or } (\texttt{n},\texttt{u}) \text{ or } (\texttt{u},\texttt{n}) \text{ or } (\texttt{0},\texttt{u}) \text{ or } (\texttt{u},\texttt{0}) \\ \texttt{u} & \text{if } (\delta x, \delta y) = (\texttt{u},\texttt{u}) \text{ or } (\texttt{1},\texttt{u}) \text{ or } (\texttt{u},\texttt{1}) \end{cases}$$

The outcomes of the addition of single signed bit differences can be divided into the following four classes [McD10]:

1. Identity: The addition of a zero has no effect.
2. Static Carry: The addition of a one carries the difference up one bit position and inverts the difference at the current bit position.

3. Dynamic Carry: The addition of the same difference causes the replacement of the difference at the current bit position by the difference carried up one bit position.
4. Inverse: The addition of two inverse differences cancels the difference out.

Example 4.3*:* The following example should illustrate the four classes of addition. 0's are used to assure that no carry occurs. Let X and X^* be two 4-bit vectors and ∇X_1 and ∇X_2 the two differences.

$$\begin{aligned}\nabla X_1 + \nabla X_2 &= [\texttt{---n}] + [\texttt{---0}] = [\texttt{---n}]\\ \nabla X_1 + \nabla X_2 &= [\texttt{--01}] + [\texttt{--0n}] = [\texttt{--nu}]\\ \nabla X_1 + \nabla X_2 &= [\texttt{--0u}] + [\texttt{--0u}] = [\texttt{--u0}]\\ \nabla X_1 + \nabla X_2 &= [\texttt{---u}] + [\texttt{---n}] = [\texttt{---1}]\end{aligned}$$

A useful fact is that the carry, which forms through the addition at the position of the most significant bit, is discarded because of the modular addition. This possibility to cancel a difference by using the carry should be illustrated in the following example.

Example 4.4*:* Let X and X^* be two 4-bit vectors, ∇X_1 and ∇X_2 the two differences and the addition be performed modulo 2^4.

$$\begin{aligned}\nabla X_1 + \nabla X_2 &= [\texttt{u---}] + [\texttt{u---}] = [\texttt{----}]\\ \text{or}\quad \nabla X_1 + \nabla X_2 &= [\texttt{n---}] + [\texttt{n---}] = [\texttt{----}]\end{aligned}$$

4.2.3 Carry

In the previous observations of the modular addition it was assumed that there is no carry at the considered bit position. In this subsection the "expansion" effect of a difference is discussed as well as some general notes about the carry.
For comprehension reasons a first example with a single difference should illustrate the possibilities of the expansion effect.

Example 4.5*:* Let X and X^* be two 4-bit vectors, ∇X_1 and ∇X_2 the two differences and the addition be performed modulo 2^4.

1. $\nabla X_1 + \nabla X_2 = [\texttt{----}] + [\texttt{---u}] = [\texttt{---u}]$ with probability 1/2
2. or $\nabla X_1 + \nabla X_2 = [\texttt{----}] + [\texttt{---u}] = [\texttt{--un}]$ with probability 1/4
3. or $\nabla X_1 + \nabla X_2 = [\texttt{----}] + [\texttt{---u}] = [\texttt{-unn}]$ with probability 1/8
4. or $\nabla X_1 + \nabla X_2 = [\texttt{----}] + [\texttt{---u}] = [\texttt{unnn}]$ with probability 1/16
5. or $\nabla X_1 + \nabla X_2 = [\texttt{----}] + [\texttt{---u}] = [\texttt{nnnn}]$ with probability 1/16

In this example there exist five possible results for the modular addition due to the expansion of the difference, but the probabilities for the different cases are quite different.
The first case is as likely as the cases 2 to 5 together. The reason for this is that the first case occurs if the least significant bit is `0` and all the other cases depend on the least significant bit to be `1`.
If the least significant bit is `1`, the second case is as likely as the cases 3 to 5, because the sum of the two second bits to be `0` ($\texttt{0}+\texttt{0}$, $\texttt{1}+\texttt{1}$) is as likely as the sum to be `1` ($\texttt{0}+\texttt{1}$, $\texttt{1}+\texttt{0}$).
The exact description of the calculation of probabilities can be found in [Dau05].

If the possibilities should be summarized in the notation of the generalized conditions, Example 4.5 would yield to:

$$\begin{aligned} \nabla X_1 + \nabla X_2 = [\texttt{----}] + [\texttt{---u}] = [\texttt{???x}] \text{ with probability } 1 \\ \text{but already } \nabla X_1 + \nabla X_2 = [\texttt{----}] + [\texttt{---u}] = [\texttt{--Bx}] \text{ with probability } 3/4 \\ \text{and } \nabla X_1 + \nabla X_2 = [\texttt{----}] + [\texttt{---u}] = [\texttt{-B?x}] \text{ with probability } 7/8 \end{aligned}$$

The reason for this behavior lies in the redundancy of the representation of a modular difference in a bitwise manner and hence as a XOR difference, signed difference or bitwise defined generalized condition.
To continue the example from before ∇X_1 has the modular difference $\Delta^+ X_1 = 0$ and ∇X_2 the modular difference $\Delta^+ X_2 = 1$. If the addition is considered with modular differences, this results in $\Delta^+ X_1 + \Delta^+ X_2 = 1$. But there exist 16 different pairs of 4-bit value with a modular difference of 1 and those pairs have one of the following forms: [`---u`] ($2^3 = 8$ pairs), [`--un`] ($2^2 = 4$ pairs), [`-unn`] ($2^1 = 2$ pairs), [`unnn`] (1 pair) or [`nnnn`] (1 pair).
The detailed description of the transformation of a modular difference into a signed difference can be found in [Dau05].

Up to here, only single differences were considered. The situation is more complex if more than one difference is involved, especially if the considered bit ranges are not disjoint and/or the number of expansion steps is not limited.
In practical attacks bits are often chosen at random and hence cases with higher probability (*e.g.* shorter carries) are more likely to occur. Therefore the number of expansion steps is often limited.

Further in practical cases very often more than two values are added in one considered operation. Hence also the number of possible carries increases if more than two values are added. If *e.g.* four 1's are added in one step the carry can already be 2 and not only 0 or 1 even if the carry at the bit position was 0. If additionally a carry at the position

occurs, the carry can get higher. So it is possible that the carry not only affects the next bit position, but also higher bit positions.

4.3 Propagation Properties of Boolean Functions

In this section the propagation of differences through the five in RIPEMD-160 used Boolean functions are described in detail. First some basic properties of the functions are reviewed and extended considering XOR differences `x` and `-`. Later the results are described in more detail also considering signed differences `n`, `-` and `u`.

The propagation can in general be controlled by conditions on the input values, but each of the used functions has certain limits in order to manipulate the output of the function.

4.3.1 The XOR Function

The XOR function $XOR(x, y, z) = x \oplus y \oplus z$ is used as $f_1(x, y, z) = XOR(x, y, z)$ in RIPEMD-160 in the first round of the left stream and last round of the right stream. There exist the following properties for the XOR function:

1. $XOR(x, y, z) = \neg XOR(\neg x, y, z) \quad = \neg XOR(x, \neg y, z) = \neg XOR(x, y, \neg z)$
2. $XOR(x, y, z) = XOR(\neg x, \neg y, z) \quad = XOR(x, \neg y, \neg z) = XOR(\neg x, y, \neg z)$
3. $XOR(x, y, z) = \neg XOR(\neg x, \neg y, \neg z)$

But as not only XOR differences but also signed differences should be considered, a more detailed analysis is necessary.
A zero output difference happens if and only if exactly two non-zero input differences occur, hence it does not matter which non-zero input difference. Of course the trivial case of three zero input differences as well leads to a zero output differences.
Another observation is that by imposing conditions, it is possible to determine the signed output difference of the XOR function if and only if there is exactly one input difference. The disadvantage is that these conditions always affect both other input bits. Such conditions affecting two bits are also called two-bit conditions and are discussed in Subsection 6.3.2.
If there are differences in each input, the output difference is fully determined and cannot be altered.

4.3.2 The IF Function

The IF function $IF(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$ is used as $f_2(x, y, z) = IF(x, y, z)$ and as $f_4(x, y, z) = IF(z, x, y)$ in RIPEMD-160. For the IF function there exist the following properties:

1. $IF(x, y, z) = IF(\neg x, y, z)$ if and only if $y = z$
2. $IF(x, y, z) = IF(x, \neg y, z)$ if and only if $x = 0$
3. $IF(x, y, z) = IF(x, y, \neg z)$ if and only if $x = 1$
4. $IF(x, y, z) = IF(\neg x, \neg y, z)$ if and only if $x \oplus y \oplus z = 1$
5. $IF(x, y, z) = \neg IF(x, \neg y, \neg z)$
6. $IF(x, y, z) = IF(\neg x, y, \neg z)$ if and only if $x \oplus y \oplus z = 0$
7. $IF(x, y, z) = IF(\neg x, \neg y, \neg z)$ if and only if $y \neq z$

As can be seen above, the IF function can produce zero output differences in almost all different cases of input differences (cases 1.-4. and 6.-7.).
An interesting case is the case of a signed input difference on x, as by imposing conditions the output difference can be fully determined. The disadvantage of determining the non-zero output difference in this special case is that two conditions are needed, so $y = 1$ and $z = 0$ respectively $y = 0$ and $z = 1$. In all the other cases, where the non-zero input difference can produce a zero output difference, the non-zero output difference cannot be altered.
In case 5. from above the non-zero output difference can only be altered if the non-zero input differences are different, *i.e.* if $y \neq z$.
It is worth noting that the IF function is the only used function that can produce a zero output difference for all three single non-zero input differences.

4.3.3 The ONX Function

The ONX function $ONX(x, y, z) = (x \vee \neg y) \oplus z$ is used as $f_3(x, y, z) = ONX(x, y, z)$ and as $f_5(x, y, z) = ONX(y, z, x)$ in RIPEMD-160. For the ONX function there exist

the following properties:

1. $ONX(x, y, z) = ONX(\neg x, y, z)$ if and only if $y = 0$
2. $ONX(x, y, z) = ONX(x, \neg y, z)$ if and only if $x = 1$
3. $ONX(x, y, z) = \neg ONX(x, y, \neg z)$
4. $ONX(x, y, z) = ONX(\neg x, \neg y, z)$ if and only if $x = y$
5. $ONX(x, y, z) = ONX(x, \neg y, \neg z)$ if and only if $x = 0$
6. $ONX(x, y, z) = ONX(\neg x, y, \neg z)$ if and only if $y = 1$
7. $ONX(x, y, z) = ONX(\neg x, \neg y, \neg z)$ if and only if $x \neq y$

Also the ONX function can produce zero output differences in most cases. Further the non-zero output difference can be fully determined for single non-zero input differences by imposing conditions. Again the disadvantage is that two distict conditions are needed.
In case 4. the non-zero output difference can be altered through conditions on z if and only if the non-zero input differences are different. In the remaining cases of two non-zero input differences, the non-zero output difference cannot be altered. The output differences of three non-zero input differences are determined and cannot be altered by imposing conditions.

4.4 Differential Attacks

As already mentioned the collision resistance is an intuitive target for differential attacks, because two distinct messages $M \neq M' \Leftrightarrow \Delta M \neq 0$ should be found, which produce the same hash value $H(M) = H(M') \Leftrightarrow \Delta H(M) = 0$.
Basically all current attacks can be divided in at least two separate parts. In the first part a sequence of differences is determined or chosen. In the second part the collisions are determined by searching for message pairs, which follow the sequence of differences. But there exist exceptions as well, *e.g.* in [MNS11] the search for differential characteristics and confirming message pairs is already combined.

Usually the attacker tries first to choose differences in the message words, or more precisely in the expanded message words, such that the differences behave in a desired way throughout the hash function or in certain parts of the hash function. This is probably the most important, but also the most critical part. Already here the different types of attacks start to differ.

As differences are considered, the attacks already differ in the choice of the considered differences. While *e.g.* Dobbertin mainly used modular differences in his attacks on MD_4, MD_5 and RIPEMD, Chabaud and Joux mainly considered XOR differences in their attack on SHA-0. Wang *et al.* used modular differences but also XOR differences or more precisely signed differences, when they needed it.
Both differences have their advantages and disadvantages. On one hand the modular addition is often used extensively in the step operations, *e.g.* the step operation of RIPEMD-160 consists of four modular additions, and hence considering modular differences is in some sense more intuitive. But bit-wise defined functions cannot be analyzed by only considering modular differences as the bit-wise representation of a modular difference is redundant. On the other hand by only using XOR differences the whole function cannot be analyzed either. In this case approximations of the functions are needed [Dau05].

Also the methods to construct the differential characteristic differ. For a long time they were mostly constructed by hand before Chabaud and Joux started to use automated tools of coding theory and linear algebra to search for differential characteristics by using linear differentials. De Cannière and Rechberger used an even more complex tool to search for non-linear differential characteristics [Sch11].

Nevertheless the different methods have the same goal, to find a differential characteristic with a high probability. Often the Hamming weight of the differences in the differential characteristic is used to estimate this probability, hence the less differences the higher the probability. So in general attackers search for sparse differential characteristics, but such sparse differential characteristics are not easy to find.
Since the message can be chosen in most attacks (*e.g.* collision attack), Chabaud and Joux observed that the message can be chosen according to the differential characteristic [CJ98]. Hence it is possible to allow differential characteristics with low probability as long as the message can be chosen. In most hash functions the message can be chosen up to a certain step of the hash functions respectively compression function. If *e.g.* round based structures are considered the message can be chosen in the first round or first few rounds. By allowing low probability in parts, where the message can be chosen, it is often easier to find a differential characteristics with high probability in the remaining parts. So for most hash functions this means that a good differential characteristic has low probability at the beginning and high probability towards the end. By using such differential characteristics Wang *et al.* were able to attack several hash functions, like MD_4, MD_5 etc. In their attacks they further developed advanced message modification techniques to increase the overall success probability of the attack. In their attacks they used the following three-step approach:

1. Find a message difference, which produces a collision (with high probability).

2. Derive a differential characteristic for the message difference, resulting in a set of sufficient conditions for the characteristic to hold.
3. Use message modification techniques, such that almost all sufficient conditions hold.

Since then many attacks based on this approach have been published improving the message modification and differential characteristic search in different ways.

Also the differential attack of this thesis is based on this three step-approach:

1. Finding a starting point.
2. Finding a differential characteristic.
3. Finding a confirming message pair.

5 Finding a Starting Point

In this thesis we consider differential attacks, which start with one or more message words containing differences. This part is crucial as it influences not only the success of finding a differential characteristic for the hash function, but also the probability of finding a message pair that leads to a collision.
The compression function of RIPEMD-160 consists of two streams of five rounds each, but at the moment it is infeasible to attack the whole compression function. Hence in this thesis round reduced variants of RIPEMD are discussed, to be precise the attack concentrates on two rounds.

First of all the impact of a difference in a message word on the state variables should be clarified. Therefore the update process for one step of one stream of RIPEMD-160 (3.2) is reviewed in order to make the following observation:

Proposition 5.1. *If there are zero differences in all input state variables,* $\Delta A_i = 0$ *with* $k-5 \leq i \leq k-1$*, then*

$$\Delta A_k = 0 \Leftrightarrow \Delta W_k = 0.$$

Further if $\Delta A_i = 0$ *for* $k-4 \leq i \leq k$*, then*

$$\Delta A_{k-5} = 0 \Leftrightarrow \Delta W_k = 0.$$

A similar observation for MD5 can be found in [Dau05].

This observation motivates the following two observations about local collisions.

Proposition 5.2. *A local collision can only begin and end in a step, where a non-zero message word difference is used in the update process. More formally, for a local collision over* k *steps from step* i *to step* $i+k$ *the following holds*

$$\Delta W_i \neq 0 \text{ and } \Delta W_{i+k} \neq 0.$$

Proposition 5.3. *The minimum number of steps of a local collision in RIPEMD-160 is five.*

The success probability of an attack is generally higher if the number of conditions over the considered area and therefore the number of conditions on the state variables of the considered area is small. Further the goal is to find high probability differential characteristics in later rounds of both streams. A high probability differential characteristic would only contain a few conditions in the later rounds. Hence if attacking the first two rounds, the differential characteristic should be sparse in the second round of both streams.

One way to achieve such high probability differential characteristics is to cancel differences quickly using local collisions over a small number of steps. Another way is to eliminate the differences as early as possible in the later round, so in the considered case early in the second round.

5.1 Using a Single Message Word

Due to the different message permutations in each stream it is difficult to find a single message word that can produce short local collisions in both streams concurrently. Nevertheless a few possible candidates should be given here. The idea of using a single message word is illustrated in Figure 5.1.

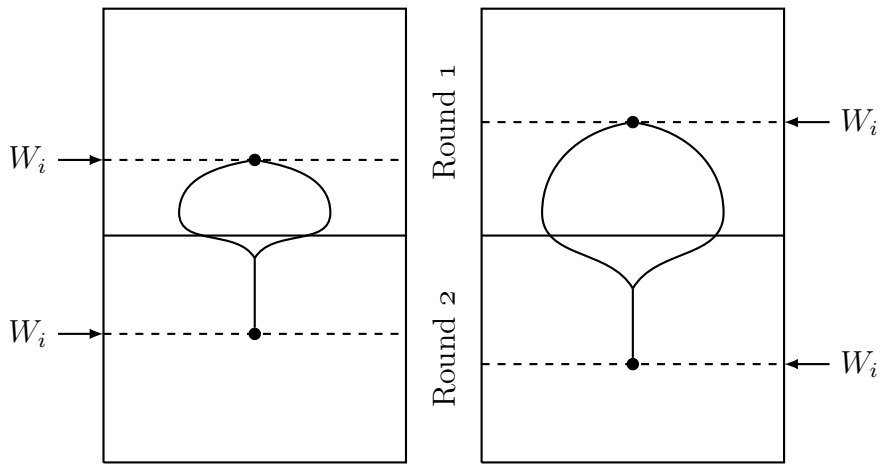


Figure 5.1: Local Collisions using a single message word over two rounds

The best candidate seems to be W_{13}, with a 5-step local collision in the left stream and a 13-step local collision in the right stream between round 1 and 2. Also both local collisions end quite early in the second round. But there exists a strong constraint for a

local collision over five steps. The consequence for using a single message word should be illustrated by the following observation.

Proposition 5.4. *The shortest local collision, which uses a single message word, has to be constructed over six steps.*

The reason for this behavior lies in the update process again. Hence the update process for one step of one stream of RIPEMD-160 (3.2) is reviewed again for a 5-step local collision.
A 5-step local collision only allows a non-zero difference in a single state variable, more precisely the state variable where the difference is introduced using the message word difference. This difference could be canceled five steps later using the same message word. But due to the construction of the RIPEMD-160 state update transformation, a closer look at step four of the 5-step local collision is needed. In step four of a local collision over five steps the following setting is given:

$$\Delta A_{i-5} = 0, \Delta A_{i-3} = 0, \Delta A_{i-2} = 0, \Delta A_{i-1} = 0 \text{ and } \Delta A_{i-4} \neq 0 \text{ and } \Delta W_i = 0$$

Hence the following equation leads to a contradiction:

$$\begin{aligned}\Delta A_i = &\overbrace{(((\Delta A_{i-5} \lll 10) + f(\Delta A_{i-1}, \Delta A_{i-2}, (\Delta A_{i-3} \lll 10)) + \Delta W_i + K_i) \lll s)}^{\Delta D_i} \\ &+ (\Delta A_{i-4} \lll 10) \\ \underbrace{\Delta A_i}_{=0} = &\underbrace{\Delta D_i}_{=0} + \underbrace{(\Delta A_{i-4} \lll 10)}_{\neq 0}\end{aligned}$$

A difference can only be canceled through a modular addition, if there are differences in both summands. Therefore the equation above leads to a contradiction.

Further due to the XOR function, which is used in the left stream in round 1, the difference cannot be eliminated immediately, which is another reason, why a 5-step local collision for W_{13} in the left stream between round 1 and 2 is impossible. The round dependent behavior of differences is discussed later in this thesis.

Table 5.1 presents single message words that can produce local collisions over less than 16 steps in each stream between round 1 and 2.

Some of the local collisions end rather late in the second round in at least one stream. *E.g.* the local collision produced by W_{12} in the right stream ends in step 27. The consequence is that differences in more state variables of the second round have to be expected.

Message Word	Local Collision Lengths	
	Left Stream	Right Stream
W_{10}	10 steps (step 10 to 20)	10 steps (step 13 to 23)
W_{12}	12 steps (step 12 to 24)	12 steps (step 15 to 27)
W_6	15 steps (step 6 to 21)	7 steps (step 9 to 16)
W_{15}	7 steps (step 15 to 22)	15 steps (step 10 to 25)

Table 5.1: Local collision candidates (single message word)

5.2 Using Multiple Message Words

If a single message word is used, there is the restriction that the local collision has to be constructed between two rounds. Due to this restriction only a few possible message words are left, that can be used to construct a rather short local collision. So the idea is to use more than one message word, such that local collisions within one round can be constructed.
Also this approach aims for high probability differential characteristics in the later rounds. Hence the local collision should be short respectively a slightly longer local collision should end very early in the later round. In the considered case that would be the second round.

5.2.1 Using two Message Words

A first approach would be to use two distinct message words to build short collisions in the second round. Proposition 5.3 suggests that the shortest local collision is one over five steps, but the same restriction proposed in Proposition 5.4 has to be applied. Hence the shortest local collision using two message words has a minimum number of six steps.
Figure 5.2 shows the construction of four local collisions without considering the order of the two message words in the different rounds.

Only considering the second round, *e.g.* the pair (W_5, W_6) seems to be a good idea, as in both streams the local collision would span over six steps. But if two local collisions in the second round should be constructed, also two local collisions in the first round are needed. Hence it is impossible to use the pair (W_5, W_6) as the local collision in the left stream in round 1 would only have one step.
So the goal is to find pairs of message words, which have short local collisions in the second round, but also possible local collisions in the first round, preferable also

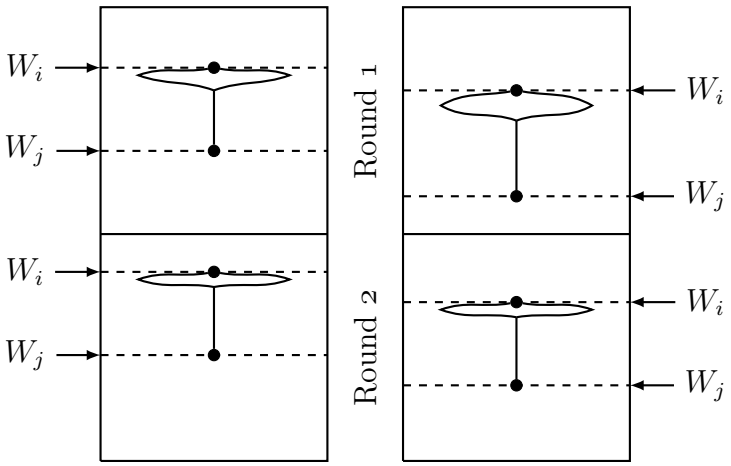


Figure 5.2: Four Local Collisions using a two message words

short ones. Already 12 of the 20 considered message pairs of the second round lead to impossible local collisions in the first round.
The pairs of message words, that can construct 6-step local collisions in the second round in both streams and local collisions in the first round are presented in Table 5.2.

Pairs of Message Words	Local Collision Lengths			
	Left Stream		Right Stream	
	Round 1	Round 2	Round 1	Round 2
(W_0, W_8)	8 steps (0 - 8)	6 steps (25 - 31)	8 steps (3 - 11)	6 steps (20 - 26)
(W_7, W_{15})	8 steps (7 - 15)	6 steps (16 - 22)	8 steps (2 - 10)	6 steps (19 - 25)
(W_3, W_{14})	11 steps (3 - 14)	6 steps (23 - 29)	13 steps (1 - 14)	6 steps (18 - 24)

Table 5.2: Local collision candidates (pairs of message words)

5.2.2 Using three Message Words

Another approach to avoid the restriction of Proposition 5.4 is to use differences in a third message word, which is used in the forth of the five local collision steps to avoid the contradiction.
Unfortunately no such triple of message words exists for both streams of round 2, but there exist several triples, which have a 7-step local collision in the second stream. Again the first round has to be considered as well. Seven out of the 22 in the second round considered cases lead to impossible local collisions in the first round.

The two best triples considering both rounds in both streams are presented in Table 5.3.

Triples of Message Words	Local Collision Lengths			
	Left Stream		Right Stream	
	Round 1	Round 2	Round 1	Round 2
(W_4, W_{10}, W_{12})	8 steps (4 - 12)	7 steps (17 - 24)	8 steps (7 - 15)	5 steps (23 - 28)
(W_5, W_9, W_{15})	10 steps (5 - 15)	5 steps (22 - 27)	10 steps (0 - 10)	7 steps (22 - 29)

Table 5.3: Local collision candidates (triples of message words)

5.2.3 Using three Local Collisions

A third approach is to use one long local collision in one stream instead of two short ones. A constraint would be that the long collision should cancel differences as early as possible in the second round. This approach can be done again for two or three message words. Figure 5.3 should illustrate the construction with two message words and without considering the order of the message words in the different rounds.

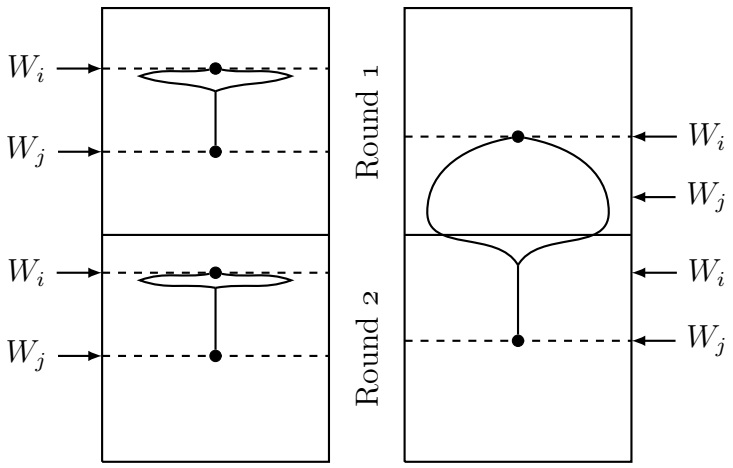


Figure 5.3: Three Local Collisions using a two message words

Considering two message words only three pairs are left that have a relatively short local collision in the first round of the same stream and a not too long local collision between round 1 and round 2 in the other stream. Those three pairs can be found in

Table 5.4.
Considering three message words the situation gets worse, because there is only one reasonable candidate, which can be found in Table 5.4 as well.
All long local collisions between round 1 and 2 in the left stream end in the middle of the second round, which would lead to differences in two, four or six state variables of the second round.

Pairs/Triples of Message Words	Local Collision Lengths			
	Left Stream		Right Stream	
	Round 1	Round 2	Round 1	Round 2
(W_{12}, W_{13})	12 steps (12 - 24)		7 steps (8 - 15)	6 steps (21 - 27)
(W_7, W_{15})	15 steps (7 - 22)		8 steps (2 - 10)	6 steps (19 - 25)
(W_9, W_{10})	17 steps (9 - 26)		9 steps (4 - 13)	6 steps (23 - 29)
(W_4, W_{10}, W_{12})	20 steps (4 - 24)		8 steps (7 - 15)	5 steps (23 - 28)

Table 5.4: Candidates producing three local collisions

5.3 RIPEMD-160 Message Pattern

In the previous sections, message words were obtained which can be used to attack the first two rounds of RIPEMD-160 as their distance between the rounds or within the rounds fulfill certain requirements. As the permutation of message words of RIPEMD-160 follows some mathematical structure and is not random, there exists a repeating pattern considering those distances. Hence it is possible to find other message words with the same distances between rounds 2 and 3 and so on.

To derive the pattern, the message words at a certain step i of rounds 1 to 4 have to be considered. The same sequence of message words between round 1 and 4 can be found again in a certain step j between round 2 to 5. To be more precise, if message word W_k can be found in step i of round n and step j of round $n+1$, then the message word W_l in step i of round $n+1$ can also be found in step j of round $n+2$. This pattern holds for each step and each stream.

So *e.g.* the 10-step local collision message word W_{10} between round 1 and 2 becomes the 10-step local collision message word W_9 between round 2 and 3. Table 5.5 should illustrate, how message words of the same distance can be found.

	Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Left Stream	Round 1	0	1	2	3	4	5	6	7	8	9	**10**	11	12	13	14	15
	Round 2	7	4	13	1	**10**	6	15	3	12	0	**9**	5	2	14	11	8
	Round 3	3	10	14	4	**9**	15	8	1	2	7	**0**	6	13	11	5	12
	Round 4	1	9	11	10	**0**	8	12	4	13	3	**7**	15	14	5	6	2
	Round 5	4	0	5	9	**7**	12	2	10	14	1	3	8	11	6	15	13
Right Stream	Round 1	5	14	7	0	9	2	11	4	13	6	15	8	1	**10**	3	12
	Round 2	6	11	3	7	0	13	5	**10**	14	15	8	12	4	**9**	1	2
	Round 3	15	5	1	3	7	14	6	**9**	11	8	12	2	10	**0**	4	13
	Round 4	8	6	4	1	3	11	15	**0**	5	12	2	13	9	**7**	10	14
	Round 5	12	15	10	4	1	5	8	**7**	6	2	13	14	0	3	9	11

Table 5.5: Index of the message block words with distance patterns

As can be easily seen, message words with same distances can be found by simply going through the columns. If *e.g.* instead of rounds 1 and 2 the rounds 2 and 3 should be attacked, without considering round 1, the observations from the previous sections can be used to derive the message words for the considered rounds.

6 Finding a Differential Characteristic

After a starting point is fixed, a differential characteristic needs to be found. This part is the most difficult one, as the differential characteristic should fulfill certain properties.
First an introduction to the used notation for differential characteristics is provided. Then some general considerations about the RIPEMD-160 specific difficulties in finding a differential characteristic are given. Further the used automatic search tool and the used algorithms are described considering different search strategies.

6.1 Notation

To find a differential characteristic an automatic search tool is used. It is based on the work of De Cannière and Rechberger, who presented a method to search for characteristics in an automatic way. They used it to construct a 64-step collision for SHA-1 [CR06]. The tool was improved and extended by Mendel *et al.* to find complex non-linear differential characteristics and confirming message pairs for a 27-step SHA-2 collision [MNS11]. Recently it was used to successfully construct a 38-step collision for RIPEMD-128 [MNS12].

Remark: The description of the automatic search tool as well as the used search strategies for SHA-1 and SHA-2 can be found in Section 6.3.

As this tool is also used in this thesis, Table 6.1 should illustrate the notation of differential characteristics used in combination with the tool and RIPEMD-160.

In each row of the distinct areas the generalized conditions of the current variables can be found, while i denotes the respective step. The message words are only presented once.

<table>
<tr><th>i</th><th>∇A_i</th><th>∇B_i</th><th>∇W_i</th></tr>
<tr><td>-5
⋮
-1</td><td colspan="3">initial value</td></tr>
<tr><td>0
⋮
15
16
⋮
79</td><td>state
variables
of the
left
stream</td><td>state
variables
of the
right
stream</td><td>message
words</td></tr>
<tr><td>80
⋮
84</td><td colspan="3">hash value</td></tr>
</table>

Table 6.1: Notation of differential characteristics

6.2 General considerations

In this section some general considerations on the search for a differential characteristic in RIPEMD-160 are given. They should illustrate the RIPEMD-160 specific difficulties.

To simplify further considerations the update process of one step of each stream is divided into three sub-steps. A similar approach was used *e.g.* in the attacks on SHA-2 and RIPEMD-128 [MNS11; MNS12]. Hence the update process is rewritten using the following notation for the left stream

$$\begin{aligned} F_i &= f(A_{i-1}, A_{i-2}, (A_{i-3} \lll 10)) \\ D_i &= (((A_{i-5} \lll 10) + F_i + W_i + K_i) \lll s) \\ A_i &= D_i + (A_{i-4} \lll 10) \end{aligned}$$

and the following one for the right stream

$$\begin{aligned} G_i &= f(B_{i-1}, B_{i-2}, (B_{i-3} \lll 10)) \\ E_i &= (((B_{i-5} \lll 10) + G_i + W_i + K_i) \lll s) \\ B_i &= E_i + (B_{i-4} \lll 10) \end{aligned}$$

6.2.1 Canceling differences in RIPEMD-160

Like Proposition 5.4 states, the shortest collision using a single message word or two message words within one round has a minimum length of six steps, because non-zero differences in two subsequent state variables are needed. Considering the modular addition in step four and five of the 6-step local collision is interesting. The start setting of such a 6-step local collision is given in Table 6.2.

i	∇A_i	∇W_i
1	????????????????????????????????	????????????????????????????????x
2	????????????????????????????????	--------------------------------
3	--------------------------------	--------------------------------
4	--------------------------------	--------------------------------
5	--------------------------------	--------------------------------
6	--------------------------------	--------------------------------
7	--------------------------------	????????????????????????????????

Table 6.2: 6-step local collision starting point using two message words

Canceling differences in the modular addition

First some general observations on canceling differences in the modular addition are provided. We consider bitwise defined differences in combination with the modular addition. As already mentioned in Subsection 4.2.2, a non-zero bit difference can only be canceled by another non-zero bit difference at the same position. This difference can either be a non-zero bit difference in the value at the considered position or a non-zero bit difference in the carry bit. But as non-zero differences in the carry bit can only occur after being introduced by a non-zero bit difference in the value, the following observation can be made.

Proposition 6.1. *Let ∇X_1 and ∇X_2 be two non-zero differences. The modular addition $\nabla X_1 + \nabla X_2$ of ∇X_1 and ∇X_2 can only be a zero difference, if the position of the first non-zero bit difference in ∇X_1 equals the position of the first non-zero bit difference in ∇X_2.*

The following examples should illustrate the proposition.

Example 6.1*:* Let ∇X_1 and ∇X_2 be two 8-bit differences and the addition be performed modulo 2^8.

$$\begin{aligned}
\nabla X_1 + \nabla X_2 &= [\texttt{------n-}] + [\texttt{------u-}] = [\texttt{--------}] \\
\text{or } \nabla X_1 + \nabla X_2 &= [\texttt{-----u--}] + [\texttt{-nuuuu--}] = [\texttt{--------}] \\
\text{or } \nabla X_1 + \nabla X_2 &= [\texttt{--nu--u-}] + [\texttt{----uuu-}] = [\texttt{--------}] \\
\text{or } \nabla X_1 + \nabla X_2 &= [\texttt{nnnn----}] + [\texttt{---n----}] = [\texttt{--------}]
\end{aligned}$$

Step 5 of a 6-step Local Collision

Figure 6.1 should illustrate the situation in step five and hence the following observations. $\delta[k]$ denotes a non-zero bit difference at bit position k.

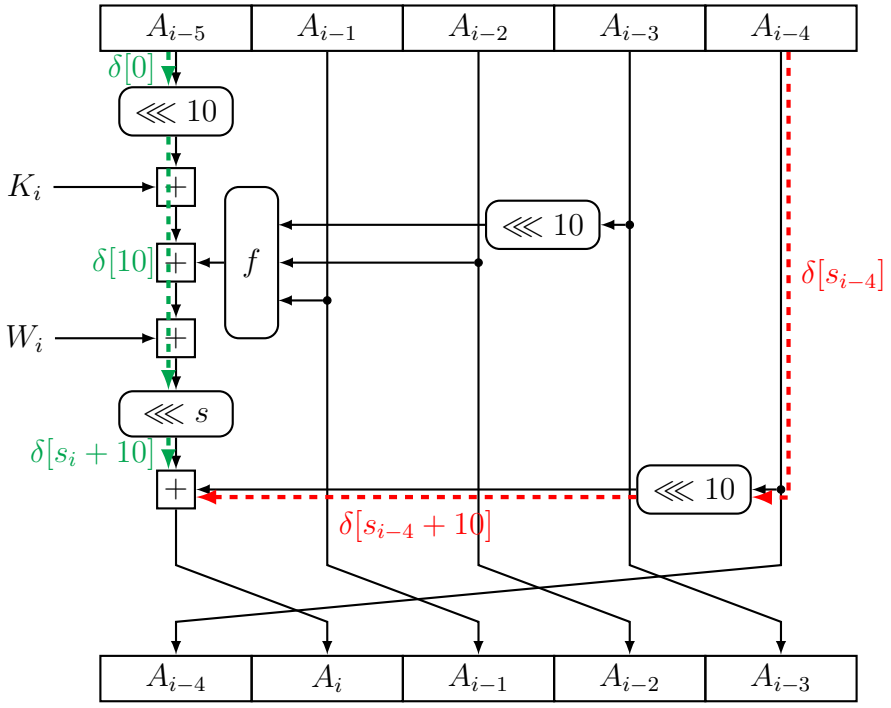


Figure 6.1: Differences in step 5 of a 6-step local collision

In step five there are non-zero differences in the two state variables A_{i-4} and A_{i-5}, so $\Delta A_{i-4} \neq 0$ and $\Delta A_{i-5} \neq 0$, which should lead to a zero difference in A_i, so $\Delta A_i = 0$. Using the sub-steps from above the non-zero differences in D_i and A_{i-4} are of interest. Considering modular differences we get $\Delta^+ D_i = -\Delta^+ A_{i-4}$. As we are considering bit-wise defined differences, the situation is a bit more complex due to the previous gained observations.

As not only ΔD_i, but also ΔA_{i-4} is influenced by ΔA_{i-5}, it is worth to have first a closer look on how both values are influenced by ΔA_{i-5}. For simplicity reason a single non-zero bit difference in A_{i-5} at the least significant bit position ($\nabla x_0 = \mathtt{x}$) is considered.

Since

$$\boldsymbol{F_{i-4}} = f(\boldsymbol{A_{i-5}}, A_{i-6}, (A_{i-7} \lll 10)),$$

a single non-zero bit difference in A_{i-5} leads to a single non-zero bit difference in F_{i-4}. Through

$$\boldsymbol{D_{i-4}} = (((A_{i-9} \lll 10) + \boldsymbol{F_{i-4}} + W_{i-4} + K_{i-4}) \lll s_{i-4})$$

this either leads to a single non-zero bit difference in D_{i-4} or more consecutive non-zero bit differences in D_{i-4} through the expansion effect of the carry. Those differences are rotated over s_{i-4} bits to the left, where s_{i-4} varies between 5 and 15. Finally the modular addition

$$\boldsymbol{A_{i-4}} = \boldsymbol{D_{i-4}} + (A_{i-8} \lll 10)$$

is performed to result in A_{i-4} with a single non-zero bit difference or more "carry expanded" consecutive non-zero bit differences. Note that this last modular addition may also lead to carry expanded differences.

A_{i-5} is used as well in the calculation of D_i, as

$$\boldsymbol{D_i} = (((\boldsymbol{A_{i-5}} \lll 10) + F_i + W_i + K_i) \lll s_i)$$

First the non-zero bit difference in A_{i-5} is rotated 10 bits to the left. Then the modular additions lead to one or more consecutive non-zero bit differences, which are rotated over s_i bits to the left. In total the non-zero bit difference of A_{i-5} is rotated over 15 to 25 bits to the left to result in the non-zero bit difference(s) of D_i.
For simplicity reason we assume that both D_i and A_{i-4} only have a single non-zero bit difference. So for D_i we have $\nabla x_{s_i+10} = \mathtt{x}$ and for A_{i-4} we have $\nabla x_{s_{i-4}} = \mathtt{x}$.

Now step five of the local collision and hence the equation

$$A_i = \boldsymbol{D_i} + (\boldsymbol{A_{i-4}} \lll 10)$$

is considered again.
Due to Proposition 6.1 we need $s_i + 10 = s_{i-4} + 10$ to cancel the difference. But usually $s_{i-4} \neq s_i$ and therefore the difference cannot be canceled.
As a consequence we can assume that we need more than a single non-zero bit difference in A_{i-5} and/or A_{i-4} to fulfill Proposition 6.1 in step five.

Remark: The observation that the positions of first non-zero bit differences of D_i and $(A_{i-4} \lll 10)$ have to be equal holds for any local collision in the four steps prior to the last step, not only for the 6-step local collisions. But in longer local collisions the non-zero bit differences in A_{i-4} are determined by non-zero bit differences in more than one state variable (up to five). Hence there is more freedom in constructing the desired non-zero bit differences.

Implication on message words The non-zero bit differences of A_{i-5} are introduced by the non-zero bit difference(s) of a message word in the first step of the local collision. Further the non-zero bit differences of A_{i-4} are canceled by the non-zero bit difference(s) of a message word in the last step of the local collision (Proposition 5.1 and 5.2). If the non-zero bit differences of the state variable are separated by zero bit differences, they cause multiple non-zero bit differences in the corresponding message word. Theoretically a single non-zero bit difference in the message word can produce 32 consecutive non-zero bit differences. In this case the non-zero bit difference would have to be at the least significant bit of the message word and the added value would have to consist of 1's except for the most significant bit. The probability for this case to happen without influencing the added value is 2^{-31}.

Step 4 of a 6-step Local Collision

Figure 6.2 should illustrate the situation in step four and the following observations. Again $\delta[k]$ denotes a non-zero bit difference at bit position k.

In step four there are non-zero differences in two state variables, namely $\Delta A_{i-3} \neq 0$ and $\Delta A_{i-4} \neq 0$, which should lead to a zero difference in A_i, so $\Delta A_i = 0$.
Again A_{i-4} and A_{i-3} are related as described above.
The non-zero bit differences in A_{i-3} lead to non-zero bit differences in F_i and hence in D_i. As A_{i-3} is rotated 10 bits to the left first, also the non-zero bit differences in F_i are rotated 10 bits to the left. Therefore the non-zero bit differences in D_i are rotated 15 to 25 bits to the left compared to A_{i-3}. As modular additions are used to compute D_i, carry expanded differences are possible.
For simplicity reasons we continue the example from above. So we have A_{i-4} with $\nabla x_0 = \mathtt{x}$, A_{i-3} with $\nabla x_{s_{i-3}} = \mathtt{x}$ and D_i with $\nabla x_{s_i+10+s_{i-3}} = \mathtt{x}$.

Now again the modular addition of D_i and $(A_{i-4} \lll 10)$ is the interesting part. As Proposition 6.1 states, the bit positions of the first non-zero bit differences in both variables have to be the same. Hence we need $s_i + 10 + s_{i-3} = 10$ to cancel the difference. As $5 \leq s_i \leq 15$ and $5 \leq s_{i-3} \leq 15$ we get $10 \leq s_i + s_{i-3} \leq 30$. So there is no way

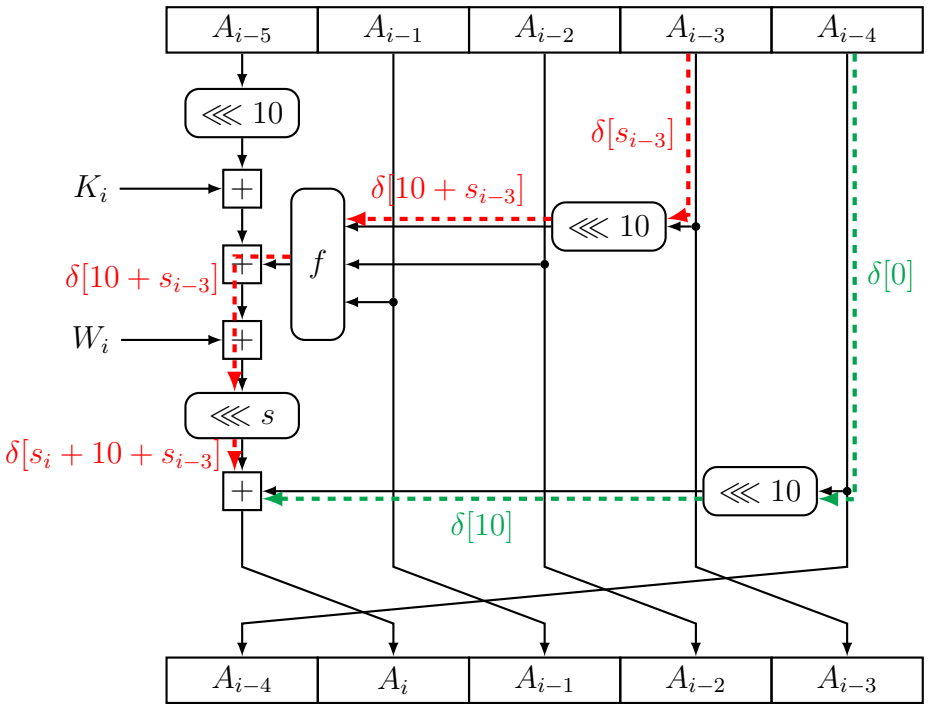


Figure 6.2: Differences in step 4 of a 6-step local collision

to cancel the difference.

Hence it is not possible to construct a 6-step local collision using only a single non-zero bit difference in each of the two state variables with non-zero differences.

6.2.2 Boolean Functions

In the previous subsection some general considerations about the modular addition were given. These observations are independent of the round and stream. In this subsection some complications due to the Boolean functions are considered. The properties of the propagation of differences through the different Boolean function are given in Section 4.3.

First we consider the XOR function, which is used in the first round of the left stream and the last round of the right stream. Again a 6-step local collision is discussed. Here the steps two and three are of interest.

In step three of the 6-step local collision there are non-zero differences in A_{i-2} and A_{i-3}, so $\Delta A_{i-2} \neq 0$ and $\Delta A_{i-3} \neq 0$.

First we consider the relation between the differences of A_{i-3} and A_{i-2}. In the first step the state variable A_{i-3} is used as input of the XOR function. As it is the only input containing non-zero bit differences, those non-zero bit differences cannot lead to zero bit differences in the output of the XOR function. Since

$$\boldsymbol{F_{i-2}} = XOR(\boldsymbol{A_{i-3}}, A_{i-4}, (A_{i-5} \lll 10))$$

each non-zero bit difference of A_{i-3} results in a non-zero bit difference in F_{i-2}. Through

$$\boldsymbol{A_{i-2}} = (((A_{i-7} \lll 10) + \boldsymbol{F_{i-2}} + W_{i-2} + K_{i-2}) \lll s_{i-2}) + (A_{i-6} \lll 10)$$

this on the other hand leads to non-zero bit differences in A_{i-2} rotated over s_{i-2} bits to the left and possibly carry expanded.

In step two of the 6-step local collision A_{i-3} and A_{i-2} are used in the equation

$$F_{i-1} = XOR(\boldsymbol{A_{i-2}}, \boldsymbol{A_{i-3}}, (A_{i-4} \lll 10))$$

and should lead to a zero difference in F_{i-1}. In order to produce a zero bit output difference exactly two non-zero bit input differences of the XOR function are needed. Hence, the non-zero bit differences of A_{i-3} and A_{i-2} are needed at exactly the same bit positions in order to lead to zero bit differences in F_{i-1}.

In step three then A_{i-3} and A_{i-2} are used in

$$F_i = XOR(A_{i-1}, \boldsymbol{A_{i-2}}, (\boldsymbol{A_{i-3}} \lll 10))$$

and should lead to a zero difference in F_i. As can be seen A_{i-3} is rotated 10 bits to the left before it is used as input of the XOR function, whereas A_{i-2} is not rotated. Again non-zero input bit differences at exactly the same bit positions are needed. But as A_{i-3} is rotated 10 bits to the left, at least 16 differences in each of the two state variables are needed due to the rotation over 10 bits. To be more precise at least non-zero bit differences on every second bit are needed, because $32 \mod 10 = 2$.

The XOR function also has further restrictions in order to produce zero output differences, *e.g.* also in step four the non-zero input bit differences are passed on, as non-zero bit differences in only one input variable of the XOR function always produces non-zero output bit differences.

Also the ONX function has restrictions in order to produce zero output differences. Consider *e.g.* a 6-step local collision in the first round of the right stream. The non-zero bit differences in A_i, the first state variable containing differences, are always passed on to A_{i+1}, the second state variable containing differences.

The IF function can be considered as the nicest function producing zero output differences. Consider *e.g.* round two of the left stream. The only case, where zero output differences cannot be produced, is if there are non-zero bit differences in A_{i-2} and $(A_{i-3} \lll 10)$ and a zero bit differences in A_{i-1} at the same bit position. This is pretty unlikely in sparse differential characteristics, but can become relevant in dense ones.

6.3 Automatic Search Tool

Altogether the previous section suggests, that there are multiple interactions between the different state variables and too many influences, such that finding differential paths by hand is almost impossible. Hence an automatic search tool is needed, which can be used to find complex nonlinear differential characteristics. Therefore the tool introduced in Section 6.1 is used in this thesis.

6.3.1 Propagating Generalized Conditions

The generalized conditions are propagated the same way as in the SHA-2 variant of the tool. Basically there exist three cases [CR06]:

1. conditions are inconsistent
2. conditions are consistent
3. conditions are consistent, if additional bit conditions are fulfilled (the conditions propagate)

As mentioned in [MNS11] the complexity of propagating generalized conditions increases exponentially with the number of input bits and additions. In RIPEMD-160 there are six input bits excluding the carry. To reduce the complexity the whole update process of RIPEMD-160 is split into three sub-steps, which have already been mentioned at the beginning of the previous section. This way the number of input bits reduces to at most three. Further the sub-steps F_i and G_i take advantage of the for SHA-2 developed optimizations for sub-steps without modular addition. Those can be precomputed for all generalized input conditions as a speedup method. The drawback of the method of splitting is that the relation between the sub-steps compared to the combined propagation is lost.

6.3.2 Propagating Conditions on two Bits

But also additional conditions are present in the differential characteristic, that are not covered by generalized conditions.
As an example we consider the IF function, which is used in the second round of the left stream (Section 4.3). Assume we have a non-zero bit difference in the first input and a zero bit difference in the second and third input. If the output should be a zero bit difference, there exists the condition that the second and the third input bit have to be equal. Therefore conditions on two bits need to be considered as well such that the differential characteristic is valid. Such two-bit conditions are needed to control the propagation of differences through the Boolean functions.
Moreover, two-bit conditions also occur in modular additions. The following example should illustrate how two-bit conditions are used in modular additions.

Example 6.2*:* Let ∇X_1, ∇X_2 and ∇X_3 be two 4-bit differences and the addition be performed modulo 2^4. The following result of the addition is desired:

$$\nabla X_1 + \nabla X_2 = [\texttt{--n-}] + [\texttt{----}] = [\texttt{--n-}] = \nabla X_3$$

It is known from Section 4.2, that $\texttt{n} + \texttt{0} = \texttt{n}$, but this does not take the carry into account. The following additions lead to the desired result:

$\nabla X_1 =$	[--n0]	or [--n0]	or [--n1]	or [--n1]
$+\ \nabla X_2 =$	[--00]	[--01]	[--00]	[--11]
$Carry =$	[--00]	[--00]	[--00]	[--11]
$\nabla X_3 =$	[--n0]	[--n1]	[--n1]	[--n0]

The additional condition on two bits is, that the carry at the second bit position is equal to bit value of ∇X_2 at the second bit position.

It is worth to mention, that already Wang *et al.* used two-bit conditions in their attacks on MD_4 and RIPEMD [Wan+05].

These two-bit conditions can lead to further inconsistencies, as two contradicting conditions may occur. While in SHA-2 such inconsistencies can happen quite often, this is not the case in RIPEMD-160, where they only occur rarely. Hence only one simple check is used to verify if the differential characteristic is valid. These checks are only done at certain points, more precisely only at the end of the search for a local collision in a certain area, *e.g.* one after the local collision over two rounds in one stream is found and a second one after the local collision over two rounds in second stream is

found. As check the Complete Condition Check of SHA-2 on bits, which contain two-bit conditions, is used. It is an expensive check, as for every bit restricted to '-' it is checked, whether both choices, so '0' and '1', are still valid. If both choices are invalid, the whole characteristic is impossible. Further details on the check can be found in [MNS11].

6.3.3 Search Strategy

The search technique is based on the technique, that was used by De Cannière and Rechberger for SHA-1 [CR06] and adopted and extended by Mendel *et al.* for SHA-2 [MNS11]. The search strategy can in general be divided into three parts: decision, deduction and backtracking.
In the first part is decided, which bit is chosen and which condition is imposed. In the second part the propagation of the imposed condition is computed it is checked if a contradiction through the imposed condition occurs. If there is a contradiction, the third part does the backtracking by undoing decisions [MNS11].

SHA-1 Search Strategy The basic search strategy to find differentials, which has been proposed in [CR06] can be described as follows. This description can be found in [MNS11].

Let U be the set of all '?' and 'x', then repeat the following until U is empty.
Decision

1. Pick randomly a bit in U.
2. Impose a '-' for a '?' or randomly a sign ('u' or 'n') for 'x'.

Deduction

3. Compute the propagation.
4. If a contradiction is detected start backtracking, else go to step 1.

Backtracking

5. Jump back to an earlier state of the search and go to step 1.

This is already an optimized version of the in [CR06] proposed algorithm. In step 2 of the not optimized version only '?'-bits are picked (*i.e.* bits which are not restricted). Furthermore, the backtracking part (step 5) does not exist in the not optimized version. If a contradiction occurs, simply a restart is done. Also the original procedure does not take the different sub-steps into account.
A nice property of this search strategy is, that sparser differential characteristics are discovered with a higher probability, if they exist.

SHA-2 Search Strategy The same strategy was also applied to SHA-2 in [MNS11], but no valid differential characteristic could be found. Hence a more sophisticated search strategy was developed to find valid differential characteristics. Basically the search for a confirming message pair with the search for a differential characteristic was combined by considering bits with a critical amount of two-bit conditions, *i.e.* bits involved in many relations with other bits, much earlier. Further the backtracking part was improved by remembering critical bits during the search process, which improves the search speed significantly. A decision bit is marked critical, if a contradiction cannot be resolved by trying the second choice for the bit (see step 5 of the backtracking part described below). Further optional additional checks were done to detect invalid characteristics earlier, which were caused by the high number of related two-bit conditions in SHA-2.

As already mentioned in RIPEMD-160 such complex conditions occur rarely. Hence it is possible to reduce those optional additional checks respectively only use them to verify characteristics.
Since the search process benefits from the improved backtracking of the SHA-2 variant of the algorithm, it is also used for RIPEMD-160. The backtracking part is described as follows [MNS11].

Backtracking

5. If the decision bit is 'x' try the second choice for the sign or if the decision bit is '?' impose a 'x'.

6. If still a contradiction occurs mark bit as critical.

7. Jump back until the critical bit can be resolved.

8. Continue with step 1.

RIPEMD-160 Search Strategy The SHA-2 strategy was applied to RIPEMD-160, but no valid characteristic could be found in a certain amount of time. The reason for this lies in another problem, that arises through the used separation into three sub-steps in RIPEMD-160, which is not present in the separation into sub-steps in SHA-2. In SHA-2 the final sub-step in the state update process is basically one big modular addition consisting of summands that are either input state variables or function outputs of functions that do not use modular additions. The situation is similar in SHA-1. In RIPEMD-160 on the other hand, the modular addition of one step in one stream is divided by a rotation in between. Hence in the update process (3.2), which is

$$A_i = (((A_{i-5} \lll 10) + f(A_{i-1}, A_{i-2}, (A_{i-3} \lll 10)) + W_i + K_i) \lll s) + (A_{i-4} \lll 10)$$

the modular additions were divided into the two sub-steps:

$$D_i = (((A_{i-5} \lll 10) + F_i + W_i + K_i) \lll s)$$
$$A_i = D_i + (A_{i-4} \lll 10)$$

Therefore two different carry expansions, the one in sub-step D_i and the one in A_i may occur. Through guessing only bits in the main steps A_i, like in the SHA-2 variant, the conditions propagate very slowly and the contradictions are detected very late in the search process. Hence the search strategy of SHA-2 is not suitable for RIPEMD-160. A similar problem was detected by Mendel *et al.* for MD5 [MRS09].

Remark: The use of $F_i = f(A_{i-1}, A_{i-2}, (A_{i-3} \lll 10))$ does not pose a problem.

A further optimization approach, which was proposed in [MRS09], was taken into account in the search process. It was suggested, that '`x`'-bits should be picked as soon as they appear. The sign of the difference should be guessed, so '`u`' or '`n`', and backtracking should be done, if the chosen sign does not lead to consistent conditions. They observed that contradictions are discovered considerably earlier this way.

Remark: Although in the SHA-2 search strategy '`x`'-bits are also picked, they are in the same set U with the unrestricted '`?`'-bits and the bits of U are chosen by random.

It turns out that this for MD5 proposed approach is also more effective for RIPEMD-160.

In RIPEMD-160 the search is done separately for different search areas, *e.g.* only one round and/or only one stream. There exist different refinements considering those search areas for the different approaches discussed in this thesis, which can be found in Section 6.4 in the respective Subsections.
The previous observations lead to the following overall search strategy for RIPEMD-160.

Let U be the set of all '?' in the defined search area and D be the set of all 'x' in the defined search area, then repeat the following until both U and D are empty.

Decision

1. Pick randomly a bit in D and go to step 2. If D is empty go to step 3.
2. Impose randomly a sign ('u' or 'n') for 'x' and go to step 5.
3. Pick randomly a bit in U and go to step 4.
4. Impose a '-' for a '?'.

Deduction

5. Compute the propagation.
6. If no contradiction is detected, go to step 1, else start backtracking.

Backtracking

7. If the decision bit is 'x' try the second choice for the sign or if the decision bit is '?' impose a 'x'.
8. If still a contradiction occurs mark bit as critical.
9. Jump back until the critical bit can be resolved.
10. Continue with step 1.

Using this strategy contradictions that lead to an impossible differential characteristic are detected earlier during the search process. The drawback of this strategy is that the resulting characteristics are less sparse, as long carry expansions are more likely to occur.

Remark: The whole search process is restarted after a certain amount of contradictions. This should end states in the search that are stuck and far from being a possible differential characteristic.

6.4 Differential Characteristic Search

6.4.1 Multiple Short Local Collisions

A first approach is to find multiple short local collisions, *e.g.* 6-step local collisions. This way only a few state variables would contain differences, considering *e.g.* a 6-step local collision, only two state variables contain differences. Although for one round and one stream such local collision could be found, it was not possible to find more of them. Further those local collisions tend to be very dense, which is not the optimal

case regarding the search for a confirming message pair, especially in round 2.

In the search for *e.g.* a 6-step local collision further adjustments to the overall search strategy described in 6.3 are made, which seem to improve the search.

Remark: For simplicity reasons only the variables A_i, D_i and F_i of the left stream are used to describe the modifications, although they were used as well in the right stream and in both streams simultaneously.

As also suggested in [MRS09] the following setting of search areas is used to attack one stream:

- U_1 contains the unrestricted bits of the D_i and A_i
- D_1 contains the bits of the D_i and A_i imposing a difference
- U_2 contains both the unrestricted bits and the bits imposing a difference of F_i

The sequence of using the sets is as follows:

1. Use set U_1 and D_1 and apply the overall search strategy.
2. If U_1 and D_1 both are empty, use U_2 and pick randomly, preferring neither '`x`'-bits nor '`?`'-bits.

The reason for using one set U_2 is that only little freedom in F_i is possible after all D_i and A_i are determined.

Further modifications concerning the sub-step D_i are done to improve the search process. As longer trails of differences could be expected, the backtracking part is slightly modified. If the decision bit is '`-`' in D_i also the second choice, so '`x`' is tried. For longer collisions that would be ineffective, but in a 6-step local collision only four of these sub-steps have to be considered.

As the differential characteristics tend to be extremely dense, a further modification is done in order to get sparser A_i. The set U_1 is split into two sets U_A and U_D, so into the unrestricted bits of A_i and the unrestricted bits of D_i. Then the sequence is modified as follows:

1. Use set D_1 as usual.
2. Use a control sequence to choose between U_A and U_D, if one of them is empty continue using the other one.
3. If U_A, U_D and D_1 are empty, use U_2 and pick randomly, preferring neither '`x`'-bits nor '`?`'-bits.

Another observation considering 6-step local collisions is that trying to make the A_i more sparse resulted in more isolated differences in the main state variables A_i. This on the other hand caused more differences in the message words.

For *e.g.* the message pair (W_7, W_{15}) a local collision in round 2 of the right stream could be found, which is shown in Table 6.3 but this local collision already contains 16 differences in two state variables and further fixed bits. Also the construction of this local collision leads to a fully determined message pair (W_7, W_{15}) with four differences in W_7 and six differences in W_{15}. The fully determined message words make the search in other areas more difficult.

i	∇B_i	∇W	
14	---------1----------	----------	$= W_3$
15	--0----------	----------	$= W_{12}$
16	---------1----------	----------	$= W_6$
17	----1----------	----------	$= W_{11}$
18	----1----00000000000----0----011	----------	$= W_3$
19	---------11111111111un--1-n--1--	-n----------un--n	$= W_7$
20	uuuuuuuuuu----n----u----------n	----------	$= W_0$
21	----------1---01----	----------	$= W_{13}$
22	----------0---10----	----------	$= W_5$
23	----------	----------	$= W_{10}$
24	----------0	----------	$= W_{14}$
25	----------	----u----n----------unnn	$= W_{15}$

Table 6.3: 6-step local collision differential characteristic within the second round of the right stream

6.4.2 Longer Local Collisions

Slightly longer local collisions seem to work better, due to the additional freedom, especially in the last five steps of the local collisions, where the differences have to be canceled. Hence starting points, that only use a single message word to produce local collisions between two rounds are used in the search for differential characteristics for both streams.
Also here several modifications considering search areas and search strategy were tried to improve the search process.

As very short local collisions are already hard to find with two distinct message words and result in multiple separated differences or pretty long trails of differences in the used message words, it seems almost impossible to find a very short local collision using a single message word. Hence for message words, which have a short 7-step local

collision in one stream, it seems hard to find a local collision. Such a short local collision could not be found with the used strategies. Message words, which seem more suitable, are those having similar length in both streams.

Round 2 and 3

As the Boolean functions in the second and third round have nicer properties considering influencing its output, first those rounds are attacked. The most suitable message word considering the length of the local collisions between round 2 and 3 is W_9, which can produce 10-step local collisions in both streams.
In first experiments it was tried to attack both streams simultaneously. Unfortunately getting close to a solution in one stream resulted practically always in contradictions in the second stream. Hence in further attempts the left and right stream are considered separately.

The differences in the message word W_9 lead to differences in the state variables $A_{26}, \dots A_{31}$ in the left stream and to differences in the state variables $B_{29}, \dots B_{34}$ in the right stream of RIPEMD-160. The differences in the state variables of the left stream are all in the second round, whereas in the right stream differences in three state variables of the third round occur.

Using the default setting of search areas described in Subsection 6.4.1 and the described sequence of using the sets, a differential characteristic could be found for both the left stream and the right stream. Unfortunately the found differential characteristic is dense in both streams. So further adjustments are needed. Table 6.4 shows the relevant area of the state variables as well as the used message words.

As first approach, the method described in Subsection 6.4.1 of splitting the unrestricted sets was tried without success. Hence a modification using an additional splitting of sets was tried. Not only the set U_1 (unrestricted bits '?') is split into U_A and U_D, but also the set D_1 (bits with difference 'x') is split into D_A and D_D. While U_A and U_D are chosen using a certain ratio, D_A is chosen until its empty before D_D is chosen.
Although this approach did not lead to a solution, intermediate states of the search process indicated that quite often trails of differences respectively contradictions were found quicker. But as the method in general did not lead to results, thorough experiments were not done.

i	∇A_i	W	∇B_i	W	∇W	W
24	----------11111----------	W_{12}	----------	W_{14}	----------	W_0
25	10----0-11------110-101100----00	W_0	----------	W_{15}	----------1-00--10	W_1
26	------n10-1011000---0101----0---	W_9	----------	W_8	----------	W_2
27	nnnnnnnnnnnnnnnnnnnnnnnnnnnnnn00	W_5	--------1-0010111---------	W_{12}	10011100010011100100--0100011001	W_3
28	111110u111nun1111000000001111100	W_2	0110-----10000000-10111110110110	W_4	----------	W_4
29	111111100u-0-1111n10-un111111u1n	W_{14}	uuuuuuu111-----0111-------011111	W_9	00001----------1	W_5
30	101nuuu110-101000000000000000000	W_{11}	----0110unnnnnnnnnnnnnu1-n01111-	W_1	----------	W_6
31	---111100--unnnnnnnnnnnnnnnnnnn	W_8	0111uun1nn11111100000011-00n1un1	W_2	----------11011001000-----0	W_7
32	111111111--00111111011101001111	W_3	000000001111000u0nn1uu0110-1uuun	W_{15}	111101----------0---	W_8
33	-001000001-100111111011111111100	W_{10}	0000nu11-1-010110011101111111110	W_5	----------nuuu----------	W_9
34	----------0-------	W_{14}	----1110-1-unnnnnnnnnnnnnnnnnnn	W_1	-0000011------1-1011000100001111	W_{10}
35	----------	W_4	011---0111-111010111100000010110	W_3	----------0----------1	W_{11}
36	----------	W_9	-00---0010100101-1100000000000--	W_7	----------	W_{12}
37	----------	W_{15}	----------	W_{14}	----------	W_{13}
38	----------	W_8	----------	W_6	----------	W_{14}
39	----------	W_1	----------	W_9	----------	W_{15}

Table 6.4: 10-step local collision differential characteristic between the second and the third round of both streams

Round 1 and 2

As differential characteristics between round 2 and 3 can be found, the next target is to find differential characteristics between round 1 and 2. In order to achieve sparser differential characteristics, slightly longer local collisions may help. Hence message word W_{12} is used, which can produce 12-step local collisions between the first and the second round. A drawback of using W_{12} is, that four state variables with differences are in the second round of the left stream and seven state variables contain differences in the second round of the right stream. On the other hand a local collision produced by W_{12} in both streams would lead to a step-reduced collision up to step 42, as W_{12} is introduced late in the third round.
In this search process the search areas were refined again. The goal is to make at least the last few state variables sparse, so the last few A_i containing differences. As the most sensitive step considering the last A_i containing differences is four steps later, where a zero difference has to be produced, A_i and D_{i+4} are put together in one set. The same can be done up to the last four state variables containing differences. It can be observed that the sensitivity producing a zero output difference through the sum of $A_i \lll 10$ and D_{i+4} reduces, the further the steps go back. Hence first it was tried to fully determine A_i and D_{i+4} before searching in A_{i-1} and D_{i+3} and so on. Hence the set U_1 is divided into the small sets $U_1^*, \ldots, U_k^*$ with $k \leq 4$ containing the unrestricted bits of up to the last four state variables and U_1' containing the remaining unrestricted bits of U_1.
Although the approach works as expected and produces the desired sparse state variables considering differences, it was not possible in the experiments to connect these state variables with the rest of the local collision, at least not in both streams. Further trying to get state variables sparse considering differences, multiple other bits in the state variables get fixed to '1' or '0'. A differential characteristic *e.g.* for the right

stream could be found using all sets, so U_1^*, U_2^*, U_3^*, U_4^*, U_1' and U_2. Table 6.5 illustrates the relevant area.

i	∇B_i	∇W	
12	----------0111-01001001------	----------	$= W_1$
13	1110110--01000001100001111111111	----------	$= W_{10}$
14	1-1111100-01000000111110110100000	011-----011100-1----------0	$= W_3$
15	uuuuuuuuuuuuuuuuuuuuuuuuuuuuu--0	--n----------	$= W_{12}$
16	1un0111-01001001n-n11-nu1-00-uu0	----------	$= W_6$
17	00u0111-nnu111100-100-101-n0-00u	----------110----------	$= W_{11}$
18	nn---00101--10100u0000-0nuuuu1u1	011-----011100-1----------0	$= W_3$
19	011-nuu111----10000-010-1---0111	---0100-0---000----------	$= W_7$
20	001001-0-----1011111110----unnnn	0----------0--0	$= W_0$
21	----------1010101nu--11--0	----------0-11	$= W_{13}$
22	--------0000u--1000011----------	----------00----------0	$= W_5$
23	----------	----------	$= W_{10}$
24	----------	----------	$= W_{14}$
25	----------11----------	----------	$= W_{15}$
26	----------	----------	$= W_8$
27	----------	--n----------	$= W_{12}$

Table 6.5: 12-step local collision differential characteristic between the first and the second round of the right stream using multiple subsets of unrestricted bits

Hence the sequence of choosing the different sets was adopted as follows:

1. Use set D_1 as usual.
2. Use a control sequence to choose between the different U_i^*.
3. If all U_i^* are empty use U_1'.
4. If U_1' is empty as well and also D_1, use U_2 and pick randomly, preferring neither 'x'-bits nor '?'-bits.

The control sequence was influenced by the above made observations, so the set containing the last state variable with differences was preferred over the set containing the second last, and so on. But as again no results could be achieved the restriction, that U_1' is chosen after all U_i^* are empty was weakened and another control sequence was used to pick bits from U_1' before the U_i^* are empty respectively not all four sets were used.

Using this strategy by only considering four separated sets for the unrestricted bits, so U_1^*, U_2^*, U_1' and U_2 a differential characteristic for the left stream could be found for an already found differential characteristic of the right stream, where the default setting of search areas described in Subsection 6.4.1 was used. It was tried to find another differential characteristic for the right stream using the same strategy as for

the left stream, which was unfortunately not possible. As can be seen in Table 6.6 the characteristic is dense again and contains multiple trailing differences. But these results were the only relevant results that could be produced between the first and second round and were therefore as well reused to find a confirming message pair. The starting point and the full differential characteristic can be found in Tables 1 and 2 in the Appendix.

i	∇A_i	W	∇B_i	W	∇W	W
10	----------1----------	W_{10}	----------	W_{15}	----------	W_0
11	----------0---------00---	W_{11}	----------	W_8	-----10----------	W_1
12	----------0011----un-00---	W_{12}	----------	W_1	----------	W_2
13	-------100-----0-uu0----------	W_{13}	0000-----1----------0000000000	W_{10}	----------0	W_3
14	--1--0--1u---nuuu111---000------	W_{14}	1111----------0111111111	W_3	----------	W_4
15	-n0--u-1nnn--00100001111100-----	W_{15}	0000000000--unnnnnnnnnnnn----00	W_{12}	----------	W_5
16	-0100-1u0nnnnnn0------10--0-111	W_7	0111111111---------0000000000011	W_6	----------	W_5
17	0----1nnunuuuuuuuuuuuuuuuuuuuu	W_4	--01111111---------0unnnnnnnnnn	W_{11}	--01-1----------1	W_6
18	0000101110010011111111 0un1111-11	W_{13}	---------nuuuuuuuuu111111------	W_3	----------000----------	W_7
19	10000nuuuu100000011111-10--10010	W_1	-----00001100000111111--------01	W_7	----------	W_8
20	-----00110---11----------00	W_{10}	-----000010000000001----------11	W_0	----------	W_9
21	----------0----10100	W_6	-----111111111111111un----------	W_{13}	---------0----------	W_{10}
22	----------1----------	W_{15}	uuuuuuuuuu----------nuuuu	W_5	----------	W_{11}
23	----------0	W_3	----------1----------	W_{10}	----------u	W_{12}
24	----------	W_{12}	----------0----------	W_{14}	----------	W_{13}
25	----------	W_0	----------	W_{15}	----------	W_{14}
26	----------	W_9	----------0	W_8	----------	W_{15}
27	----------	W_5	----------	W_{12}		

Table 6.6: 12-step local collision differential characteristic between the first and the second round of both streams

Considering the found differential characteristic, it is interesting that there are only a few bits fixed in the message words. Also worth mentioning is that in the left stream in the state variables B_{19} and B_{20} there are no differences.

To complete the observations, it was also tried to limit the occurring differences to areas within the state variables without any success.

Conclusion

While it seems almost impossible to find multiple short local collisions, longer local collisions in both streams can be found. Due to the structure of RIPEMD-160 the found differential characteristics tend to be dense.
Using different strategies it is possible to reduce the number of differences in certain areas of the local collisions, but really sparse differential characteristics have not been found. Moreover, predictions on how strong the different strategies influenced the whole process of finding a differential characteristic are hard to make, as not many results

could be produced. Practically all observations that have been made are based on intermediate results, that lead to contradictions at some stage. Therefore only the effects of the considerations concerning the goals of the used strategies, *e.g.* getting certain state variables more sparse concerning differences, could be verified in the different cases, but it is not clear if the described methods would lead to a differential characteristic for both streams.

7 Finding a Confirming Message Pair

In the previous chapter a differential characteristic for both streams could be found together with conditions on the state variables. In order to find a confirming message pair the conditions on single bits of the differential characteristic need to be fulfilled. This is not an easy task as the found characteristic is very dense. But there are also plenty of two-bit conditions, which are not obvious in the differential characteristic. So the task is to find a message pair, which fulfills all conditions, such that the differential characteristic leads to a collision. As one independent condition is fulfilled with probability 1/2, a random message fulfills all conditions with probability 2^{-n}, where n denotes the number of conditions. Hence the complexity can be denoted as 2^n. As there are plenty of conditions, randomly searching for confirming message pairs would result in higher complexity than a birthday attack. In order to reduce this complexity, message modification techniques are used, such that the modified message pair fulfills as many conditions as possible.
In this chapter the different attempts to find a confirming message pair for the obtained differential characteristics are described. Further an approach is presented, which combines the search for a differential characteristic with the search for an confirming message pair.

7.1 Separated Search

In this section the different attempts to find a confirming message pair for the differential characteristic found in the previous chapter are described. If such a message pair can be found, a step-reduced collision for RIPEMD-160 can be produced. Note that the found differential characteristics are very dense in both streams as well as in the second attacked round.

Furthermore, there are numerous two-bit conditions, which mostly concern the state variables within the local collisions in the start setting of the message search, *i.e.* the differential characteristic.

Due to the Boolean functions also the two state variables prior to the beginning of the local collision may be heavily affected.
As a simple example we take the IF function used in the second round of the left stream. Suppose that the local collision starts at step i and that there is a single non-zero bit difference in A_i. In step $i+1$ the IF function, *i.e.* $\mathrm{IF}(A_i, A_{i-1}, (A_{i-2} \lll 10))$ should lead to a zero output difference. Hence we review the following property of the IF function:

$$IF(x, y, z) = IF(\neg x, y, z) \text{ if and only if } y = z$$

Now we consider the state variables again. For a single non-zero bit difference in A_i at position k the bits of A_{i-1} and $(A_{i-2} \lll 10)$ at position k have to be equal in order to lead to a zero bit difference at position k in the output of the IF function. Therefore we get one two-bit condition affecting one bit of A_{i-1} and one bit of A_{i-2}. Note that a single independent two-bit condition is fulfilled with probability $1/2$.
Assume now that there are eight non-zero bit differences in A_i and in step $i+1$ the IF function should lead to a zero output difference. Then, the observations made above suggest that already eight two-bit conditions are needed to produce the desired zero output difference.

Furthermore, the other state variables prior to the first step of the local collision may be affected by multiple two-bit conditions due to the carry. Further details on propagating conditions on two bits can be found in Subsection 6.3.2.
Also the message words that are used within the local collision steps may already contain multiple bits that need to fulfill two-bit conditions. Moreover, bits are often not only related to one other bit through a two-bit condition, but to multiple other bits through two-bit conditions.
Hence the search gets even harder, as in order to lead to a confirming message pair, not only the bit conditions but also those two-bit conditions have to be fulfilled.

Message Modification Techniques

In the first round, *i.e.* for the first 16 steps, the freedom of all message words W_i can be used to find a confirming message pair. If only a single stream is considered, a basic message modification technique would be sufficient for the first round. The method can be shortly described as follows:

1. Choose confirming state variables
2. Invert the update process of each step to calculate the confirming message word

This technique is also called single-message modification and can be found in [Wan+05].

But as RIPEMD-160 uses two parallel streams, message modification is more complicated as the conditions on both streams have to be fulfilled by using the same message words. This was also already noted by Wang *et al.* in [Wan+05]. There they overcome the problem for RIPEMD by applying two further modification techniques to the second stream. The first one uses carry effects to fulfill conditions. The second one basically uses the Boolean function to fulfill the condition by modifying bits of the input state variables of the function, which do not have to fulfill a condition. Those bits in the input state variables are modified by previous message words.
But while in RIPEMD the same permutation of message words is used in both streams, *i.e.* the same message word in the same step, RIPEMD-160 uses different permutations in each stream. Furthermore, in RIPEMD also the same Boolean function is used in each round of both streams, which is not the case in RIPEMD-160. The different permutation of message words increases the complexity of applying message modification significantly. Also the same rotation values in both streams of RIPEMD simplify the used technique. In this case, message modification can be applied from the least significant to the most significant bit. But as in RIPEMD-160 different rotation values are used in each stream, it is possible that previously corrected conditions may become invalid again. These complications have already been noted in [MNS12] for RIPEMD-128.

RIPEMD-160 Message Modification

There exist several dedicated advanced message modification techniques that can be used to fulfill conditions on two state variables by using a single message word. *E.g.* for MD_5 the multi-message modification by Wang and Yu [WY05] or the multi-message modification by Klíma [Klí06] have been proposed.
In this thesis a more generalized approach proposed in [MNS11; MNS12] is used to simplify the message modification. The same automatic search tool, which is used to find differential characteristics, is used for message modification instead of dedicated and complicated techniques. Instead of imposing '`-`' or '`x`' for '`?`' respectively '`u`' or '`n`' for '`x`' like in the search for a differential characteristic, in the message search we look for valid assignments of '`-`' to '`0`' or '`1`' in the first round. As this is a bit-wise approach the different message word permutations and different rotation values as well as carry effects are handled by the tool automatically. The drawback is the slightly higher complexity compared to word-wise approaches. Similar observations have been made on RIPEMD-128 in [MNS12].

The overall search strategy for a confirming message pair can be described as follows:

Let U be the set of all '-' in the defined search area and repeat the following until U is empty.

Decision

1. Pick randomly a bit in U.
2. Impose randomly a '0' or '1'.

Deduction

3. Compute the propagation.
4. If no contradiction is detected, go to step 1, else start backtracking.

Backtracking

5. Try the second choice of the decision bit.
6. If still a contradiction occurs mark bit as critical.
7. Jump back until the critical bit can be resolved.
8. Continue with step 1.

Again the optimized backtracking part of the algorithm used to attack SHA-2 is used [MNS11].

Message Search Based on Two-bit Conditions

An observation made in [MNS11] influences the choice of the decision bit in the message search. It seems wise to first pick bits that are related to multiple other bits by two-bit conditions. Through fixing the value of such a bit multiple other bits are determined as well, which may even lead to further determined bits through distinct two-bit conditions on those other bits. Through the propagation of all those bits also new two-bit conditions may arise. So a single bit with many two-bit conditions has many consequences. Therefore they should be picked first.

The reasoning should be illustrated by the following example.
Assume we are picking bits with no conditions as well and randomly set them to '1' or '0'. After several bits with no conditions have been set we pick a bit, which is related to k other bits. By setting this decision bit multiple other bits are determined (at least k). Through propagating those bits it is likely that multiple new conditions on other bits arise, also on bits with previously no condition, which have already been randomly set. Through the multiple new conditions, it is likely that a contradiction occurs somewhere. Hence backtracking needs to be done. If the contradiction affects a bit with previously no condition, the contradiction could have been avoided by not

setting this bit randomly but according to the condition. Therefore backtracking could have been avoided as well.

Another observation is that the later a bit causing many consequences is chosen, the more likely it gets that both choices ('1' and '0') lead to a contradiction somewhere. Hence the message search can be carried out more effectively, if the bits are chosen from the bits with the most two-bit conditions to the ones with few two-bit conditions.

A further advantage of first picking bits with many relations to other bits is, that multiple bits can be marked as critical in the backtracking part of the algorithm.

Therefore the set U of '-'-bits in the overall search strategy is split into k subsets containing the bits with two-bit conditions and one subset containing the bits with no two-bit conditions. As already mentioned two-bit conditions are created dynamically in the search process by determining bits, hence also the different subsets change dynamically. So the overall search strategy is slightly adapted by using the following subsets in the decision part:

- For $1 \leq i \leq k$: U_i contains the '-'-bits, which are related to i other bits.
- U' contains the '-'-bits, which are related to no other bits.

Those subsets are chosen in the following order:

1. Pick randomly a bit in U_k. If U_k is empty, pick randomly a bit in U_{k-1} and so on.
2. If all U_i, so $U_k, \ldots, U_1$ are empty, pick randomly a bit in U'

Using this strategy for the whole collision area, so from step 0 to step 41, it is possible to find message words, which are able to follow the differential characteristic roughly in the local collision areas of both streams. This local collision areas extend from step 12 to 24 in the left stream and from step 15 to 27 in the right stream. What remains is to connect this area with the initial values. Unfortunately there is not enough freedom left in the remaining message words to achieve this goal in the first round of both streams. An example of such a try can be found in Table 3 in the Appendix.

Sequential Message Modification

Another approach is to do message modification sequentially. So the used search areas to apply the overall search strategy are the state variables A_i and B_i for step i. The message words are then modified according to the state variables. If a contradiction is

encountered, backtracking is done until the contradiction can be resolved.

A first simple approach would be to start at step 0 and modify the message words W_0 and W_5 according to randomly chosen A_0 and B_0. Then go to step 1 and do the same. But this way it is not even possible to get near the local collision areas.
Another approach is to start at step 15 and do the modifications in backward direction instead of forward direction. Also this way the search is not getting far.
Considering only one stream first, using the same methods does not lead to satisfying results either.

Furthermore, we use the same approaches in combination with the two-bit based search strategy. *I.e.* the message modification is done sequentially, but within the state variables we pay attention to the bits with two-bit conditions. Again no solution can be achieved.

Altogether none of the described methods leads to a confirming message pair for the first round.

The problem is that some message words have to be used to fulfill many conditions in the local collision areas of both streams and can therefore barely be used for message modification in the first round. As each message word is used in both streams, it is at least used twice until the end of the local collisions. Hence, there seems to be not enough freedom in the message words to find a confirming message pair for such a dense differential characteristic in both streams.
Therefore it is tried to simplify the problem by not starting at the beginning of round 1 but at a certain step of round 1, such that the gained freedom in some message words allows to produce a collision from step i in the first round to step 41 in the third round. If we start *e.g.* at step 11, W_4 is only used once until the end of the local collisions in both streams. Some are only used twice instead of three or four times (*e.g.* W_7) and some only three instead of four times (*e.g.* W_{13}).
But again no confirming message pair could be found for both streams using the described methods. It is worth to note that considering only one stream and starting at step 11, message pairs could be found quite fast, although some message words are used twice within the local collisions.

Moreover, it is tried to use the found differential characteristic to construct a semi-free start collision for RIPEMD-160 without any success.
To simplify the problem we consider a RIPEMD-320 variant. In order to use the found differential characteristic the interaction between the two lines after each round of

RIPEMD-320 is removed. This way it is possible to construct a 42-step semi-free start collision for this RIPEMD-320 variant, which can be found in Table 4 in the Appendix.

7.2 Combined Search

As the separated search for differential characteristics and confirming message pairs turned out to be infeasible with the used methods, the idea is to combine the search for a differential characteristic with the search for a confirming message pair.
Therefore a similar search strategy to the one in the SHA-2 attack [MNS11] is used. Here also the whole search process is split into two phases and it is switched between both phases dynamically. Generally speaking bits related to many other bits are already considered at a certain point in the search for a differential characteristic.

As a first modification the two separated modular additions, so

$$D_i = (((A_{i-5} \lll 10) + F_i + W_i + K_i) \lll s)$$
$$A_i = D_i + (A_{i-4} \lll 10)$$

respectively

$$E_i = (((B_{i-5} \lll 10) + G_i + W_i + K_i) \lll s)$$
$$B_i = E_i + (B_{i-4} \lll 10)$$

are combined into one modular addition:

$$A_i = (((A_{i-5} \lll 10) + F_i + W_i + K_i) \lll s) + (A_{i-4} \lll 10)$$

respectively

$$B_i = (((B_{i-5} \lll 10) + G_i + W_i + K_i) \lll s) + (B_{i-4} \lll 10)$$

D_i respectively E_i are used as intermediate variables.

Basically the overall search strategy for the differential characteristic denoted as phase 1 stays the same except for refining the different sets and the switching point to phase 2. In phase 2 the overall search strategy for the confirming message pair is used with adapted sets and a change in the last step of the backtracking process.
The last step of the backtracking can be described as follows:

8. If it is necessary jump back to phase 1, otherwise continue with step 1.

Switching Point

It makes sense to start the second phase after the state variables A_i respectively B_i are determined or almost determined. Furthermore, in phase 2 not too many bits should be determined in order to find a differential characteristic. Therefore, in phase 2 only bits in A_i respectively B_i are picked, which are at least related to *e.g.* 3 other bits.

Early Switching Point Not taking D_i (and F_i) respectively E_i (and G_i) into account in phase 1 leads faster to a possible solution for A_i respectively B_i. Therefore we can switch to the second phase very early. But then again contradictions are detected very late (Subsection 6.3.3).

Late Switching Point On the other hand taking all D_i respectively E_i into account leads to later determination of A_i respectively B_i and hence to a later meaningful switch to the second phase. Furthermore, it is more likely, that A_i respectively B_i already contain multiple bits with many relations to other bits, such that the second phase leads to many further determined bits. Then again contradictions are encountered, because there is not enough freedom to find a differential characteristic.

So neither a very early nor a very late switching point leads to any results in the experiments in reasonable time.

Hence it is tried to take D_i respectively E_i into account, but not in the whole search area, *e.g.* only over a certain amount of steps. This way contradictions at least in that certain area are detected early, before we start switching to phase 2. Unfortunately also using this approach no differential characteristic could be found in reasonable time.

Finally we try to construct longer local collisions hoping to find a differential characteristic with not too many conditions on single bits and fewer bits with relations to many other bits. So *e.g.* message word W_{14} is used between round 1 and 3 in the left stream and between round 2 and 3 in the right stream. Although intermediate states of the search process show that there are less bits, which are related to many other bits, there are still plenty of conditions on single bits to be fulfilled. Also not even a differential characteristic could be found in reasonable time. An intermediate state at a very late stage of the search before contradictions occur can be found in Table 5 in the Appendix.

8 Conclusion

This thesis concentrated on a differential attack on the collision resistance of RIPEMD-160. Therefore methods for differential cryptanalysis have been defined and discussed focusing on the used functions in RIPEMD-160. The main goal of the attack was to construct a collision for step-reduced but otherwise unmodified RIPEMD-160. The attack itself consists of three parts: finding a starting point, finding a differential characteristic and finding a confirming message pair.

There exist several RIPEMD-160 specific difficulties in the attack.
In the first part of the attack a good starting point needs to be found. Due to the dual-stream design of RIPEMD-160 and the different message word permutations in both streams it is not easy to find a single message word or more message words, that may lead to a collision with a reasonable probability.
Also in the second part different difficulties arise. Especially the extra modular addition in combination with the different rotations poses a problem. Not only an intermediate result in the update process of one step is rotated, but also other input variables. Due to those difficulties the found differential characteristics tend to be very dense.
Finally in the third part, the search for a confirming message pair turns out to be far from being easy. The different message word permutations for both streams make message modification for the first round already hard. Since only dense differential characteristics in both streams could be found, the many conditions prevented to find a confirming message pair.
Several distinct approaches for each part of the attack have been presented as well as their advantages and disadvantages. Also different intermediate results illustrating the different approaches have been provided.

In general the different message word permutations for both streams of RIPEMD-160 in combination with the modular addition and different rotations complicate attacks significantly.

Although no collision could be found, the used methods indicate that collisions for a step-reduced variant of RIPEMD-160 can be found, if it is possible to overcome certain problems, which arose in the methods used in this thesis.

Appendix

Results and Intermediate Results

i	∇A_i	∇B_i	∇W_i
-5	----------		
-4	----------		
-3	----------		
-2	----------		
-1	----------		
0	----------	----------	----------
1	----------	----------	----------
2	----------	----------	----------
3	----------	----------	----------
4	----------	----------	----------
5	----------	----------	----------
6	----------	----------	----------
7	----------	----------	----------
8	----------	----------	----------
9	----------	----------	----------
10	----------	----------	----------
11	----------	----------	----------
12	??????????????????????????????????	----------	??????????????????????????????????x
13	??????????????????????????????????	----------	----------
14	??????????????????????????????????	----------	----------
15	??????????????????????????????????	??????????????????????????????????	----------
16	??????????????????????????????????	??????????????????????????????????	
17	??????????????????????????????????	??????????????????????????????????	
18	??????????????????????????????????	??????????????????????????????????	
19	??????????????????????????????????	??????????????????????????????????	
20	----------	??????????????????????????????????	
21	----------	??????????????????????????????????	
22	----------	??????????????????????????????????	
23	----------	----------	
24	----------	----------	
25	----------	----------	
26	----------	----------	
27	----------	----------	
28	----------	----------	
29	----------	----------	
30	----------	----------	
31	----------	----------	
32	----------	----------	
33	----------	----------	
34	----------	----------	
35	----------	----------	
36	----------	----------	
37	----------	----------	
38	----------	----------	
39	----------	----------	
40	----------	----------	
41	----------	----------	

Table 1: 12-step local collision starting point

i	∇A_i	∇B_i	∇W_i
-5	----------		
-4	----------		
-3	----------		
-2	----------		
-1	----------		
0	----------	----------	----------
1	----------	----------	-----10----------
2	----------	----------	----------
3	----------	----------	----------0
4	----------	----------	----------
5	----------	----------	----------
6	----------	----------	--01-1----------1
7	----------	----------	----------000----------
8	----------	----------	----------
9	----------	----------	----------
10	----------1----------	----------	---------0----------
11	----------0---------00---	----------	----------
12	----------0011----un-00---	----------	----------u
13	-------100-----0-uu0----------	0000-----1----------0000000000	----------
14	--1--0--1u---nuuu111---000------	1111----------0111111111	----------
15	-n0--u-1nnn--00100001111100-----	0000000000--unnnnnnnnnnnn----00	----------
16	-0100-1u0nnnnnnn0------10--0-111	0111111111---------0000000000011	
17	0----1nnunuuuuuuuuuuuuuuuuuuuuu	--01111111---------0unnnnnnnnnn	
18	00001011100100111111110un1111-11	---------nuuuuuuuuuu111111------	
19	10000nuuuu100000011111-10--10010	-----00001100000111111--------01	
20	-----00110---11----------00	-----000010000000001----------11	
21	----------0----10100	-----111111111111111un----------	
22	----------1----------	uuuuuuuuuu----------nuuuu	
23	----------0	----------1----------	
24	----------	----------0----------	
25	----------	----------	
26	----------	----------0	
27	----------	----------	
28	----------	----------	
29	----------	----------	
30	----------	----------	
31	----------	----------	
32	----------	----------	
33	----------	----------	
34	----------	----------	
35	----------	----------	
36	----------	----------	
37	----------	----------	
38	----------	----------	
39	----------	----------	
40	----------	----------	
41	----------	----------	

Table 2: 12-step local collision differential characteristic

i	∇A_i	∇B_i	∇W_i
-5	01100111010001010010001100000001		
-4	11000011110100101110000111110000		
-3	00010000001100100101010001110110		
-2	10011000101110101101110011111110		
-1	11101111110011011010110001001		
0	11100011111000111100110000000010	11110100110011000110111111101010	01110011101110011011110000011011
1	01000000010010101001100010101000	10000001100001000100111000011110	01100101100110000100000010001000
2	--00011101001----------10-	01010110001001000111000111001101	------00----1100-110110001011011
3	----------0----0011001100	11110001010101011011111101110110	01101111111010111001000010111110
4	----------	11000010110101111001000101000110	11100010101111000010010110101110
5	----------	1-0110101001101----------010	00110010000101101111011000110000
6	10101-0--1----------01-	----------	01011100111111000010111110000011
7	10101010100010110001110111000111	----------	00011111011001110100001000100001
8	01010001010110101101011001011010	----------	10000001101011010010111011100-0
9	10110000010101000000011111001110	----------	10010100100110011000101010001111
10	00010111000000110111001100011101	---------1----------	10011100001000000000000100001111
11	10101000010111010011010100100100	---------10----00----------	----------01-11100-00-1
12	01011010101000000011101100000110	10110-01-1----------	11101100000001000111111111010011u
13	11111011000000101uu0001101110111	00000010110101001111110000000000	00000010110010101110001100001110
14	101110111u001nuuu111011000010000	11110110010010101011000111111111	10110100110000010111101010101111
15	1n001u01nnn0100100001111100111111	00000000001unnnnnnnnnnnnn100000	00110000001001001011010000010011
16	1010011u0nnnnnnn0111111100100111	01111111111000111110000000000011	
17	000011nnunuuuuuuuuuuuuuuuuuuuuu	11011111111000111010unnnnnnnnnn	
18	00001011100100111111110un1111011	110010111nuuuuuuuuuu111111011111	
19	10000nuuuu10000001111101010010	00010000011000001111111011000001	
20	00001001101011110001001101110000	00010000010000000001110100010011	
21	11011000100010100010100000110100	01100111111111111111un0001111101	
22	10100011001010111111100011111000	uuuuuuuuuu00011111001011100nuuuu	
23	10010001010100110010010101101000	01101110010001111111011101000010	
24	11001100011010001001110011011111	10101001100110111101011101000010	
25	11111110011000101111011001000110	00011001001101100101001100011010	
26	11100101110011001101010011100101	010111000000011100-1100101101110	
27	00101111001110001111010010001010	11110011110100100000111000011010	
28	-001--0110110100110----------	01111001111000110110111111110011	
29	00100001----------1-----1	10100100001101100001011001111111	
30	----------	11001100-11101111011001100010100	
31	----------1-------	------00100010001100----------	
32	----------	1100101011-----001---1-------010	
33	----------	------01011----------1	
34	----------	----------0----1----------	
35	----------	----------	
36	----------	----------	
37	----------	----------	
38	----------	----------	
39	----------	----------	
40	----------	----------	
41	----------	----------	

Table 3: 12-step local collision message search at a late state before contradictions occur

i	∇A_i	∇B_i	∇W_i
-5	10011010011110100101111011101111	01001000101011101110101101001001	
-4	10011101101011010110111011010000	00110111110100100111010000110100	
-3	10111001111011010101100001111011	00000100111110001101110101000001	
-2	10000010111101111011111111001011	00100001100000010111111001100011	
-1	01100010001001110011000011000111	10100011010101110100000011001011	
0	00011011110101001011110100000011	10011011001011111110111111100110	11010010101110000100111011000011
1	11100011100100001111001011100101	11001011111001101100100010100011	00100111111011011001001111101111
2	10011011110100011100000010000010	11100001101000000111011000100000	00001011000001100011101001010010
3	10101000101100100001100100010110	01110011110111011001010100000110	00001011001101000000000101011100
4	01011100001110110101101001100101	00000001110000010011101010110100	10100010110111101100111011001101
5	00000100110010010011101101010100	00101101110110100101101111110010	11000000010000011010100000011000
6	10010001011100111111001110000100	11101101110111100100101001000010	01011110001100110001000010011000
7	11101101001000111101010010010100	01010011000110101001011111111111	00111001001111010000110000100001
8	01001111001101011000011010110110	01100001011110000110001000001010	00000111101010100111101101001101
9	01100101100000000101100110100000	10011100111100010000010111101100	10111110011101011111111011000100
10	00011110000010000101101010101010	00001010001010010100101011001101	00010110110000001000001101011011
11	11000100110111111000010000011110	00001111111000001110100001111110	11100011101001100001011111111111
12	10000100101110111110000000un101000	01010010101111010010111101000011	01001011010011010101000011000011u
13	11101100011101100uu0001111111111	00000011011110000100100000000000	10011100101010011000110010111010
14	011000101u010nuuu101101000111100	11111111101000101011110111111111	11001000011101011101101010111110
15	0n000u01nnn1100100001111110001001	00000000011unnnnnnnnnnnn001100	11110011011101010100010001001011
16	0010001u0nnnnnnn0010010100000111	01111111111011010000000000000011	
17	010011nnunuuuuuuuuuuuuuuuuuuuuu	10011111111011011110unnnnnnnnnn	
18	00001011100100111111110un1111011	000001000nuuuuuuuuuu010001000101	
19	10000nuuuu10000001111111101100010	00011000011000001111010000011001	
20	00001001100001111110111100110101	00011000010000000001111111110011	
21	01110100001010111100101100110100	00001111111111111111un0101000100	
22	00110111101110010001011100010101	uuuuuuuuuu00000000101111111nuuuu	
23	10010111001010101000001011111100	10000001000000000000100001110101	
24	10011100100100001001010111011001	11001010100111111110100001110101	
25	11110101000000101101010111100100	11110001001010100011001100100111	
26	01100101010111110000011111110101	10111010001100100001001011101000	
27	11101111110001001101110100101000	01000010101010111110000110110101	
28	10011111110010010100001000011001	00000001110111111011111101101110	
29	10011110100111111011111001111110	01101110000101000110111001101100	
30	10100001010000001001001011000010	01110110101101000000111001010100	
31	01101111100001000100100111100100	01001111011011010010101000111100	
32	01101110010010101000001011110101	00000100010000100101011101100101	
33	11110001001110001010111101010100	00011111010011000101100101000000	
34	10111011010101101111100011101010	00010110111110011110101100010000	
35	00000001001000001011010010010010	11010101000101111010101110000001	
36	01000010001010011000111100011100	11010010010110010111011100111000	
37	11010001100100111011111000100110	01010111100111100011101001010100	
38	01001010000101100110010111110101	01101101111011111010001100100010	
39	00100010100010111010010101111010	10111111010110101100101111001000101	
40	00001010110010010110111001111001	11001110101110010111101010111110	
41	00001101000011010011101100011010	10011010110110111001101011000011	

Table 4: 42-step semi-free start collision for RIPEMD-320 variant

i	∇A_i	∇B_i	∇W_i
-5	----------		
-4	----------		
-3	----------		
-2	----------		
-1	----------		
0	----------	----------	----------
1	----------	??????????????????????????????	----------
2	----------	??????????????????????????????	----------11
3	----------	??????????????????????????????	----------
4	----------	??????????????????????????????	----------
5	----------	??????????????????????????????	----------
6	----------	??????????????????????????????	----------
7	----------	??????????????????????????????	----------
8	----------	??????????????????????????????	----------
9	----------	??????????????????????????????	----------
10	----------	??????????????????????????????	----------
11	----------	??????????????????????????????	----------
12	---1-0----------	??????????????????????????????	----------
13	----------0----	??????????????????????????????	----------
14	--0n----------	??????????????????????????????	----------nu---------0---------
15	---1-----1----------1--u----	??????????????????????????????	----------
16	---1----------u1-0n-0-	??????????????????????????????	
17	--------00-------1u-111100111---	??????????????????????????????	
18	-------1------0-0011n---1n0--010	??????????????????????????????	
19	nn----uu100---0-un-11----0----un	??????????????????????????????	
20	10----00-n1---u111-n-011------00	??????????????????????????????	
21	n1-01-00u0u1101----100nu--1-11-u	??????????????????????????????	
22	nB--1-un0n1uu--10-----10-u1-11-0	??????????????????????????????	
23	n--1-001000001--n----11u-nn-nn1n	??????????????????????????????	
24	100--u11-1-----10-----1--10-0nuu	??????????????????????????????	
25	u----1-------1-01u11-101-u01-1n0	??????????????????????????????	
26	nuu---------00-1-0-1-10----1--1u	??????????????????????????????	
27	1000---1----uunnDx-u0nn001-0-011	??????????????????????????????	
28	---n--------1010-0-010011-------	??????????????????????????????	
29	--1100-0-1-11unnnnnnnnn---------	??????????????????????????????	
30	----11--1--1001000-1011--1------	?????????x????????????????????	
31	----00--0--0000----------------	----------??????????????????x-	
32	----------	----------???????????x---------	
33	----------	----------	
34	----------	----------	
35	----------	----------	
36	----------	----------	
37	----------	----------	
38	----------	----------	
39	----------	----------	
40	----------	----------	
41	----------	----------	
42	...	...	

Table 5: 20-step local collision differential characteristic search at a very late stage in the left stream before contradictions are encountered

Notation

List of Abbreviations

ARX	modular Addition, Rotation and XOR
MD	Message Digest
MDC	Modification Detection Code or Manipulation Detection Code
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
SHA	Secure Hash Algorithm

List of Used Symbols

$\neg X$	One's complement of X
$X \lll s$	X rotated over s bits to the left
$X \ggg s$	X rotated over s bits to the right
$X \oplus Y$	Bitwise exclusive OR (XOR) of X and Y
$X \wedge Y$	Bitwise AND of X and Y
$X \vee Y$	Bitwise OR of X and Y
$X + Y$	Addition of X and Y modulo 2^{32}
$X - Y$	Subtraction of X and Y modulo 2^{32}

List of Tables

List of Figures

Bibliography

[BB93] Bert den Boer and Antoon Bosselaers. "Collisions for the Compressin Function of MD5." In: *EUROCRYPT*. 1993, pp. 293–304. DOI: 10.1007/3-540-48285-7_26.

[BP95] Antoon Bosselaers and Bart Preneel, eds. *Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evaluation RIPE-RACE 1040*. Vol. 1007. Lecture Notes in Computer Science. Springer, 1995, pp. 69–111. ISBN: 3-540-60640-8. DOI: 10.1007/3-540-60640-8.

[BS90] Eli Biham and Adi Shamir. "Differential Cryptanalysis of DES-like Cryptosystems." In: *CRYPTO*. 1990, pp. 2–21. DOI: 10.1007/3-540-38424-3_1.

[CJ98] Florent Chabaud and Antoine Joux. "Differential Collisions in SHA-0." In: *CRYPTO*. 1998, pp. 56–71. DOI: 10.1007/BFb0055720.

[CR06] Christophe De Cannière and Christian Rechberger. "Finding SHA-1 Characteristics: General Results and Applications." In: *ASIACRYPT*. 2006, pp. 1–20. DOI: 10.1007/11935230_1.

[Dam89] Ivan Damgård. "A Design Principle for Hash Functions." In: *CRYPTO*. 1989, pp. 416–427. DOI: 10.1007/0-387-34805-0_39.

[Dau05] Magnus Daum. "Cryptanalysis of Hash Functions of the MD4-Family." PhD thesis. Ruhr-Universität Bochum, 2005. URL: www.cits.rub.de/imperia/md/content/magnus/dissmd4.pdf.

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. "RIPEMD-160: A Strengthened Version of RIPEMD." In: *FSE*. 1996, pp. 71–82. DOI: 10.1007/3-540-60865-6_44.

[Dob97] Hans Dobbertin. "RIPEMD with Two-Round Compress Function is Not Collision-Free." In: *J. Cryptology* 10.1 (1997), pp. 51–70. DOI: http://dx.doi.org/10.1007/s001459900019.

[GA11] E. A. Grechnikov and A. V. Adinetz. "Collision for 75-step SHA-1: Intensive Parallelization with GPU." In: *IACR Cryptology ePrint Archive* 2011 (2011), p. 641. URL: http://eprint.iacr.org/2011/641.

[Klí06] Vlastimil Klíma. "Tunnels in Hash Functions: MD5 Collisions Within a Minute." In: *IACR Cryptology ePrint Archive* 2006 (2006), p. 105. DOI: http://eprint.iacr.org/2006/105.

[McD10] Cameron McDonald. "Analysis of Modern Cryptographic Primitives." PhD thesis. Macquarie University, 2010. URL: http://web.science.mq.edu.au/groups/acac/researchstudent_completed/Cameron_thesis.pdf.

[Men+06] Florian Mendel et al. "On the Collision Resistance of RIPEMD-160." In: *ISC*. 2006, pp. 101–116. DOI: 10.1007/11836810_8.

[Mer89] Ralph C. Merkle. "One Way Hash Functions and DES." In: *CRYPTO*. 1989, pp. 428–446. DOI: 10.1007/0-387-34805-0_40.

[MNS11] Florian Mendel, Tomislav Nad, and Martin Schläffer. "Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions." In: *ASIACRYPT*. 2011, pp. 288–307. DOI: 10.1007/978-3-642-25385-0_16.

[MNS12] Florian Mendel, Tomislav Nad, and Martin Schläffer. "Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128." In: *to appear*. 2012.

[MOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. URL: http://www.cacr.math.uwaterloo.ca/hac/.

[MRS09] Florian Mendel, Christian Rechberger, and Martin Schläffer. "MD5 Is Weaker Than Weak: Attacks on Concatenated Combiners." In: *ASIACRYPT*. 2009, pp. 144–161. DOI: 10.1007/978-3-642-10366-7_9.

[OSS10] Chiaki Ohtahara, Yu Sasaki, and Takeshi Shimoyama. "Preimage Attacks on Step-Reduced RIPEMD-128 and RIPEMD-160." In: *Inscrypt*. 2010, pp. 169–186. DOI: http://dx.doi.org/10.1007/978-3-642-21518-6_13.

[Riv91] Ronald L. Rivest. "The MD4 Message Digest Algorithm." In: *Advances in Cryptology-CRYPTO' 90*. 1991, pp. 303–311. DOI: 10.1007/3-540-38424-3_22.

[Riv92a] Ronald L. Rivest. *The MD4 Message-Digest Algorithm*. RFC 1320 (Historic). Obsoleted by RFC 6150. Internet Engineering Task Force, Apr. 1992. URL: www.ietf.org/rfc/rfc1320.txt.

[Riv92b] Ronald L. Rivest. *The MD5 Message-Digest Algorithm*. RFC 1321 (Informational). Updated by RFC 6151. Internet Engineering Task Force, Apr. 1992. URL: www.ietf.org/rfc/rfc1321.txt.

[Romo4] Bart Van Rompay. "Analysis and Design of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers." PhD thesis. Katholieke Universiteit Leuven, 2004. URL: www.cosic.esat.kuleuven.be/publications/thesis-16.pdf.

[Sas+08] Yu Sasaki et al. "New Message Differences for Collision Attacks on MD4 and MD5." In: *IEICE Transactions* 91-A.1 (2008), pp. 55–63. DOI: 10.1093/ietfec/e91-a.1.55.

[Sch11] Martin Schläffer. "Cryptanalysis of AES-Based Hash Functions." PhD thesis. Graz University of Technology, 2011. URL: https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=58178.

[SO06] Martin Schläffer and Elisabeth Oswald. "Searching for Differential Paths in MD4." In: *FSE*. 2006, pp. 242–261. DOI: 10.1007/11799313_16.

[ST08] National Institute of Standards and Technology. *Federal Information Processing Standard (FIPS), Publication 180-3, Secure Hash Standard*. Tech. rep. Department of Commerce, 2008. URL: http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf.

[ST93] National Institute of Standards and Technology. *Federal Information Processing Standard (FIPS), Publication 180, Secure Hash Standard*. Tech. rep. Department of Commerce, 1993.

[Ste06] Marc Stevens. "Fast Collision Attack on MD5." In: *IACR Cryptology ePrint Archive* 2006 (2006), p. 104. URL: http://eprint.iacr.org/2006/104.

[Wan+05] Xiaoyun Wang et al. "Cryptanalysis of the Hash Functions MD4 and RIPEMD." In: *EUROCRYPT*. 2005, pp. 1–18. DOI: 10.1007/11426639_1.

[WY05] Xiaoyun Wang and Hongbo Yu. "How to Break MD5 and Other Hash Functions." In: *EUROCRYPT*. 2005, pp. 19–35. DOI: 10.1007/11426639_2.

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Efficient Collision Search Attacks on SHA-0." In: *CRYPTO*. 2005, pp. 1–16. DOI: 10.1007/11535218_1.

[WYY05b] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding Collisions in the Full SHA-1." In: *CRYPTO*. 2005, pp. 17–36. DOI: 10.1007/11535218_2.