On the Collision Resistance of RIPEMD-160 *

Florian Mendel, Norbert Pramstaller,
Christian Rechberger, and Vincent Rijmen

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, A-8010 Graz, Austria
Florian.Mendel@iaik.tugraz.at

Abstract. In this article, the RIPEMD-160 hash function is studied in detail. To analyze the hash function, we have extended existing approaches and used recent results in cryptanalysis. While RIPEMD and RIPEMD-128 reduced to 3 rounds are vulnerable to the attack, it is not feasible for RIPEMD-160. Furthermore, we present an analytical attack on a round-reduced variant of the RIPEMD-160 hash function. To the best of our knowledge this is the first article that investigates the impact of recent advances in cryptanalysis of hash functions on RIPEMD-160.

Keywords: RIPEMD-160, low-weight codewords, hash function, cryptanalysis, collision attack, differential attack

1 Introduction

Recent results in cryptanalysis show weaknesses in commonly used hash functions, such as RIPEMD, MD5, Tiger, SHA-0, and SHA-1 [1,2,9,11,12,13,14]. Therefore, the analysis of alternative hash functions, like RIPEMD-160, the SHA-2 family, and Whirlpool is of great interest. Since RIPEMD-160 is part of the ISO/IEC 10118-3:2003 standard on dedicated hash functions, it is used in many applications and is recommended in several other standards as an alternative to SHA-1. Based on the similar design of RIPEMD-160, MD5, SHA-1, and its predecessor RIPEMD, one might doubt the security of RIPEMD-160. Therefore, we investigated the impact of recent attack methods on RIPEMD-160 in detail. We are not aware of any other published analysis with respect to collision attacks of the RIPEMD-160 hash function. In the analysis of the RIPEMD-160 hash function we have extended existing approaches using recent results in cryptanalysis. In the analysis, we show that methods successfully used to attack SHA-1 are not applicable to full RIPEMD-160. Furthermore, we use analytical methods to produce a collision in a RIPEMD-160 variant reduced to 3 rounds. However, no attack has been found for the original RIPEMD-160 hash function. In summary, we can state that RIPEMD-160 is secure against known attack methods. Nevertheless, further analysis is required to get a good view on the security of RIPEMD-160.

* The work in this paper has been supported by the Austrian Science Fund (FWF), project P18138.

S.K. Katsikas et al. (Eds.): ISC 2006, LNCS 4176, pp. 101–117, 2006.
http://dx.doi.org/10.1007/11836810_8

Table 1. Notation

Notation	Meaning
$A \oplus B$	logical XOR of two bit-strings A and B
m_i	input message word i (32-bits)
w_i	expanded input message word i (32-bits)
$A \ll n$	bit-rotation of A by n positions to the left
$A \gg n$	bit-rotation of A by n positions to the right
step	single execution of the step function
round	set of consecutive *steps*, has a size of 16 (1 *round* = 16 *steps*)

The remainder of this article is structured as follows. A description of the RIPEMD-160 hash function is given in Section 2.1. In Section 2.2, we give an overview of existing attacks on RIPEMD, the predecessor of RIPEMD-160. In Section 2.3, the basic attack strategy we use in our analysis is described. Section 3 presents the results of the analysis following this attack strategy. In Section 4, we describe some methods for improving the results of the analysis. Moreover, we present a theoretical attack on a simplified variant of RIPEMD-160 reduced to 3 rounds using analytical methods in Section 5. We conclude in Section 6.

2 Finding Collisions for RIPEMD-160

In this section, we will give a short description of the RIPEMD-160 hash function. We will present the basic strategy we used for the attack on RIPEMD-160 and we will show why existing attacks on RIPEMD are not applicable to RIPEMD-160. For the remainder of the article we will follow the notation given in Table 1.

2.1 Short Description of RIPEMD-160

The RIPEMD-160 hash function was proposed by Hans Dobbertin, Antoon Bosselaers and Bart Preneel in [8] to replace RIPEMD. It is an iterative hash function that processes 512-bit input message blocks and produces a 160-bit hash value. Like its predecessor RIPEMD, it consists of two parallel streams. In each stream the state variables are updated according to the expanded message word w_i and combined with the initial value IV after the last step, depicted in Figure 1. While RIPEMD consists of two parallel streams of MD4, the two streams are designed differently in the case of RIPEMD-160.

In the following, we briefly describe the RIPEMD-160 hash algorithm. The hash function basically consists of two parts: message expansion and state update transformation. A detailed description is given in [8].

Message Expansion. The message expansion of RIPEMD-160 is a permutation of the message words in each round, where different permutations are used for the left and the right stream.

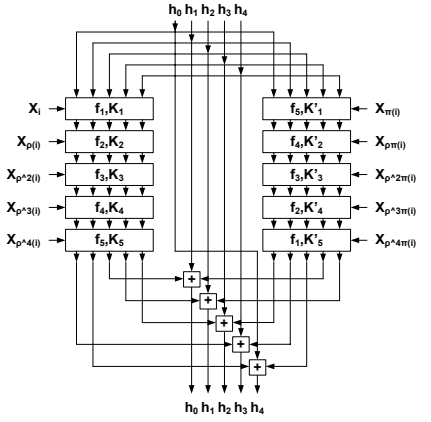

Fig. 1. The RIPEMD-160 compression function.

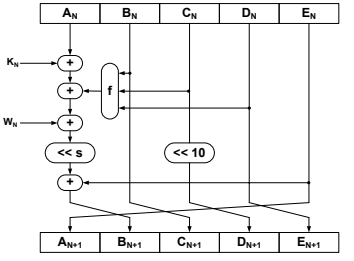

Fig. 2. The step function of RIPEMD-160.

State Update Transformation. The state update transformation starts from a (fixed) initial value IV of five 32-bit registers and updates them in 5 rounds of 16 steps each by using the expanded message word w_i in step i. Figure 2 shows one step of the state update transformation of RIPEMD-160. The function f is different in each round. f_j is used for the j-th round in the left stream, f_{6-j} is used for the j-th round in the right stream ($j = 1, \ldots, 5$). A step constant K_j is added in every step; the constant is different for each round and for each stream. Different rotation values s are used in each step and in both streams. After the last step of the state update transformation, the initial value and the values of the right and the left stream are combined, resulting in the final value of one iteration (feed forward). In detail, the feed forward is a modular addition of the permutations of the IV and the output of the left and right stream (see Figure 1). The result is the final hash value or the initial value for the next message block.

For the analysis of RIPEMD-160 in Section 3, we use a linearized variant of the state update transformation. Every addition identified in the hash function is

replaced by an XOR and the nonlinear functions f_2, f_3, f_4, f_5 are approximated by a 3-input XOR; f_1 is already an XOR.

2.2 Existing Attacks on the Predecessor RIPEMD

In this section, we will discuss the results in cryptanalysis of RIPEMD, the predecessor of RIPEMD-160. We will describe the attack of Dobbertin and Wang *et al.* and discuss why these attacks are not applicable to RIPEMD-160. A detailed description of both attack strategies is given in [6].

Attack of Dobbertin [7]. In 1997, Hans Dobbertin presented an attack on RIPEMD reduced to 2 rounds with complexity about 2^{31} hash computations. The basic idea of the attack is to find an inner collision for the compression function using a very simple input differential pattern (having only a difference in one message word m_i). Hence, there are differences in the state variables after step i. Since m_i has to be applied in the second round as well, it is chosen in such a way that the differences in state variables cancel out and the remaining steps are equal. Once an inner collision has been found, the remaining free variables have to be determined to meet the IV by calculating backward from step i in both streams.

In the attack, Dobbertin uses modular differences to describe the whole hash function by a system of equations. In general, such a system is too large to be solved, but Dobbertin used several constraints to extremely simplify the system such that it becomes solvable in practice. In the attack, he exploits the fact that the left and the right stream of RIPEMD are quite similar. A detailed description of the attack is given in [7].

However, applying the attack to RIPEMD-160 might be impractical. Due to the different permutation and rotation values used in the left and the right stream of RIPEMD-160 and due to the increased number of rounds, the system of equations would be too large to be solvable in practice.

Attack of Wang *et al.* [12]. In 2004, Wang *et al.* presented collision attacks on MD4, MD5, and RIPEMD. The attack on RIPEMD has a complexity of about 2^{18} hash computations. The basic idea of all attacks is to use differences in more than one message word to find an inner collision within a few steps in the last round and then find a suitable characteristic for the remaining steps. Hash functions with only 3 rounds seem to be vulnerable to this method in general. Hash functions with more than 3 rounds can only be broken if it is possible to exploit weaknesses of the design [6]. For instance, in the case of RIPEMD, Wang *et al.* take advantage of the similar design of the two streams of the hash function. Since the permutation and rotation values are equal for both streams, it is sufficient to find a collision-producing characteristic for one stream (3 rounds) and apply it simultaneously to both streams. Nevertheless, the number of necessary conditions increases for two streams. Hence, it is more likely to have

contradicting conditions. In fact, Wang *et al.* reported that among 30 selected collision-producing characteristics only one can produce the real collision.

However, due to different permutation and rotation values in the left and the right stream of RIPEMD-160 and the increased number of rounds (each stream has 5 rounds), this attack is not applicable to RIPEMD-160.

2.3 Our Attack Strategy

In the following, we will present our attack strategy against RIPEMD-160 based on recent results in cryptanalysis of SHA-1. All attacks basically use the same strategy:

1. Find a collision-producing characteristic that holds with high probability.
2. Find values for the message bits such that the message follows the characteristic.

There are several methods for finding a characteristic, *i.e.* the propagation of input differences through the compression function of the hash function. In the following, we will describe the method of Chabaud and Joux [5] and the method of Wang *et al.* [15] which is used in their attack on SHA-1.

Method of Chabaud and Joux [5]. In 1998, Chabaud and Joux presented an attack on the SHA-0 hash function. In this attack a linearization of the hash function was used to obtain a characteristic (in this paper referred to as *L-characteristic*). The probability that the characteristic holds in the original hash function is related to the Hamming weight of the characteristic. In general, a characteristic with low Hamming weight has a higher probability than one with a high Hamming weight.

Remark 1 *For the first steps, the probability of the characteristic is not important, because the conditions that have to be satisfied such that the characteristic holds can be easily fulfilled for these steps [5].*

Method of Wang *et al.* [15]. Considering the recent results of Wang *et al.*, it seems to be a good approach to use a general (possibly non-linear) characteristic for the first 16 steps and a characteristic that follows the linear approximation for the remaining steps. This is shown in Figure 3. For the remainder of this article the first 16 steps are referred to as V_1 and the remaining steps are referred to as V_2. The basic idea of this method is to maximize the probability of the *L-characteristic* in V_2 and to ignore the probability of the characteristic in V_1. This is based on the fact that the probability of V_1 can be neglected (see Remark 1).

Observation 1 *Wang's method to find a characteristic for the hash function can be generalized as follows:*

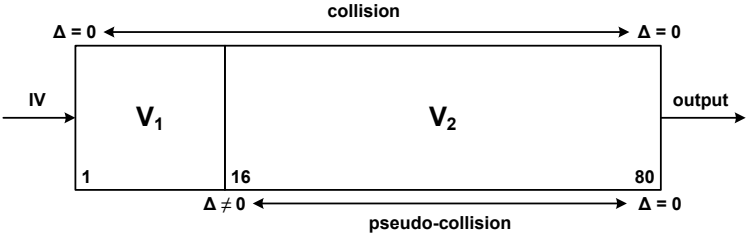

Fig. 3. Method of Wang *et al.*

1. *Find an L-characteristic with good probability resulting in a pseudo-collision for* V_2.
2. *Find a general characteristic for* V_1 *to turn a pseudo-collision into a collision.*

Observation 2 *Multi-block messages can be used to turn near-collisions into collisions.*

Since Biham and Chen observed in [1] that near-collisions are easier to find than collisions, we will use Observation 2 in Section 3.3 to improve the attack.

3 Finding an *L-characteristic* with Good Probability

Finding an *L-characteristic* for V_2 with good probability is the most important part of the attack. Since the first 16 steps (V_1) can be fulfilled by using message modification techniques [13,14] and neutral bits [1], the attack complexity only depends on the probability of the chosen *L-characteristic* in V_2. A common approach to find an *L-characteristic* with good probability is to search for one with low Hamming weight. In [10,11], algorithms from coding theory were used to obtain an *L-characteristic* for SHA-1 with low Hamming weight, *i.e.* an *L-characteristic* with good probability. Even if these algorithms are probabilistic and do not guarantee to find the best *L-characteristic*, they are expected to produce good results as they did in the case of SHA-1.

For the remainder of this article, we will only give the Hamming weight of the state variable A of the *L-characteristic*, since this gives us a good heuristic for its probability. More precisely, we use $2^{-2 \cdot \mathrm{HW}(A)}$, where $\mathrm{HW}(A)$ is the Hamming weight of the state variable A. Note that this is a quite conservative method to estimate the probability of the *L-characteristic*. The probability might be lower in practice.

3.1 Collision and Near-Collision Producing Characteristics

To find a collision-producing characteristic with good probability (low Hamming weight), we use algorithms from coding theory like it is done in [10,11] for SHA-1. To construct the generator matrix G, we use the linearized variant of the state

Table 2. Hamming weight of A using an *NL-characteristic* in V_1.

	type	Hamming weight	projection*	steps	stream
RIPEMD - 160	pseudo-collision	1471	704	16–80	both
	near-pseudo-collision	907	480	16–80	both
	pseudo-collision	657	352	16–80	left
	pseudo-collision	665	352	16–80	right
	pseudo-collision	959	384	16–64	both
	near-pseudo-collision	675	352	16–64	both
	pseudo-collision	432	192	16–64	left
	pseudo-collision	424	192	16–64	right
	pseudo-collision	458	256	16–48	both
	near-pseudo-collision	428	224	16–48	both
	pseudo-collision	187	96	16–48	left
	pseudo-collision	180	128	16–48	right
RIPEMD - 128	pseudo-collision	659	448	16–64	both
	near-pseudo-collision	561	448	16–64	both
	pseudo-collision	298	256	16–64	left
	pseudo-collision	311	192	16–64	right
	pseudo-collision	178	-	16–48	both
	near-pseudo-collision	**18**	-	16–48	both
	pseudo-collision	**28**	-	16–48	left
	pseudo-collision	**10**	-	16–48	right
RIPEMD	pseudo-collision	**20**	-	16–48	both

(*)Results achieved by using a projection as described in Section 4.

update transformation having zero differences as input in the first step and forcing zero differences after the feed forward (a collision). To keep the generator matrix and the search space small, only state variable A of each step is used. B_i, C_i, D_i, E_i and W_i of step i can be reconstructed from $A_i, \ldots, A_{i+5}$. The Hamming weight of the codewords found and hence the attack complexity is too high for an attack on RIPEMD-160. In Appendix C, the Hamming weight of the codewords found for RIPEMD-160, RIPEMD-128 and round-reduced variants is shown. Considering these results, we conclude that the final attack complexity would be too high for a reasonable attack.

3.2 Pseudo-Collision Producing Characteristics

Since we assume that we are able to turn a pseudo-collision into a collision within V_1 (see Observation 1), we can restrict the low-weight search to pseudo-collisions in V_2. As we want zero differences in the end (after the feed forward) the generator matrix G is constructed by going backwards in V_2, having zero differences after the feed forward. More precisely, this is done by going backwards in the left and the right stream using the linearized inverse state update transformation. We have a difference δ^L in the left stream and a difference δ^R in the right stream after step 80, where the differences δ^L and δ^R cancel out due to the feed forward.

Table 2 lists the Hamming weight of the codewords found for RIPEMD-160, RIPEMD-128 and round-reduced variants. Note that this weight includes the weight of variable A in the left and the right stream without considering the weight of the first 16 steps. As can be seen in Table 2, we found a codeword for RIPEMD with weight of 20, which might be low enough for an attack following the attack strategy described in this article. Based on the assumed heuristic, we estimate the final attack complexity to be $2^{2 \cdot 20}$. Since the heuristic for the estimation of the attack complexity is quite conservative, the final attack complexity might be higher in practice. Note that the round-reduced variant of the left and the right stream of RIPEMD-128 is very close to an MD4 computation. This explains the low Hamming weight of the codewords found. The results of the left and the right stream differ, because different permutations are used in the message expansion for both streams. However, the probability of the found *L-characteristic* is too low for an attack on RIPEMD-160 following the strategy described in Section 2.3.

3.3 Near-Pseudo-Collision Producing Characteristics

The results of Section 3.2 can be further improved by extending our search to near-collisions. In [13], Wang *et al.* show how this can be done for SHA-1 by using 2 message blocks. They use different characteristics in V_1, but the same *L-characteristic* in V_2 in both blocks. Due to the permutation of the state variables of the left and the right stream before the addition of both streams and the initial value in the feed forward, we would need 5 instead of 2 message blocks to turn a near-collision into a collision for RIPEMD-160 if we use the same *L-characteristic* in V_2 in each message block.

The results of the low-weight search are shown in Table 2. We found a codeword with weight of 18 for RIPEMD-128 reduced to 3 rounds which is comparable to the result of RIPEMD for a pseudo-collision. However, the Hamming weight of the codewords for RIPEMD-160 is still too high for a reasonable attack complexity. This has several reasons:

- The search space is very large and the problem of finding low-weight codewords in a linear code is NP-hard.
- We do not know any lower bound for the Hamming weight in the code defined by the generator matrix G.
- The search algorithms are probabilistic and certain parameters need to be tuned to optimize the performance. While there exist guidelines, which values to chose for a random code [4], we do not know which values would be optimal in the case of RIPEMD-160.

4 Improving Search Algorithms

Considering the results from the previous section, we have to think about improvements of the probabilistic algorithms. There are several possibilities to increase the speed (success probability for finding a codeword with low Hamming weight) of the algorithms.

4.1 Optimization of the Algorithms/Implementation

Since these algorithms are well known and have been studied by many researchers, we can assume that they are almost optimal in the general case (for a random code). There is still space for some optimizations in the implementation of the algorithms, but the speedup we can obtain is not significant enough.

4.2 Reducing the Search Space

Reducing the search space might be the best way to increase the speed of the probabilistic algorithms we used in the analysis. Since the code describing the linearized hash function is not a random code, its structure can be exploited to reduce the search space, *i.e.* size of the generator matrix describing the linear code. This method was successfully used for SHA-1. It was observed that differences in the expanded message words and state variables occur in bands of successive ones [11]. For RIPEMD-160, no structure in the low-weight codewords could be found so far. Nevertheless, several methods can be applied to reduce the size of the generator matrix and/or the search space of the algorithms. Some of these methods are:

1. Restricting the analysis to the left (right) stream of the hash function.
2. Looking at round-reduced variants of RIPEMD-160.
3. Using other linearizations for non-linear functions f_2, f_3, f_4 and f_5.
4. Forcing zero bits (like it is done in [10] for SHA-1). In detail the search space is reduced by setting certain bits (differences) to zero before doing the low-weight search.
5. Reducing the search space by using a projection, $P(w) = \sum_{i=1}^{32} b_i > 0$, where b_i is the i-th bit of the word w. The main idea is to reduce the search space by looking at words instead of bits. In detail, $P(w)$ is 1 if there are differences in the word w and 0 if there are no differences. This reduces the number of columns and rows of the matrix by a factor of 32.

Some of the methods described in this section substantially increase the quality of the results. While the improvements are marginal for reducing the search space by forcing (random) zero bits in the generator matrix or using other linearizations for f_2, f_3, f_4 and f_5, the other methods worked quite well as shown in Table 2. On the one hand, codewords with lower Hamming weight can be found by reducing the search space but on the other hand the Hamming weight of the codewords found is still too high for an attack on RIPEMD-160 or round-reduced variants. Therefore, we need other (analytic) methods to improve the results.

5 A Variant of RIPEMD-160

In this section, we will describe an approach using analytic methods to find a characteristic with low Hamming weight through the hash function. Since this is very difficult for the original hash function, we concentrate the analysis on a

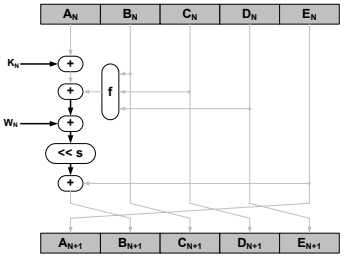

Fig. 4. A fixed-point for one step of the RIPEMD-160 variant.

variant of RIPEMD-160, where the rotation of register C is removed, as shown in Figure 4. For this variant, reduced to 3 rounds, we can find a collision using fixed-points.

5.1 Fixed-Points in the RIPEMD-160 Variant

By removing the rotation of register C, it is possible to construct *fixed-points* in one and two steps of the hash function, where a fixed-point is defined as: $\delta = g(\delta)$. In our scope, g either is a single step or two steps of the RIPEMD-160 variant. In Figure 4, a fixed-point for one step of the RIPEMD-160 variant is shown, while Figure 5 shows fixed-points for two steps of the RIPEMD-160 variant. The gray lines and shadowed rectangles indicate a difference in the MSB. These fixed-points can be used to produce a collision in the RIPEMD-160 variant reduced to 3 rounds with complexity 2^{64} and 2^{51}.

Note that in [3] a similar attack has been applied to MD5 and we can assume that the designers of RIPEMD-160 included the rotation of register C to prevent this kind of attack.

From a Fixed-Point to an Attack. In the analysis, we assume that the conditions for the first 16 steps (V_1) of the hash function can be fulfilled and we can construct differences in the MSB in arbitrary state variables of the left and the right stream after V_1 using a general characteristic. More precisely, if we have differences in the MSB in all state variables of both streams at the first step of V_2 then we can use the fixed-point shown in Figure 4 for the remaining 64 steps in V_2. The output difference of f with input differences $\delta = (1, 1, 1)$ is 1 or 0, depending on the values of the input variables. Since the difference in the MSB of A_i can be canceled by f, the difference in E_i propagates to B_{i+1}. This results in a collision after the feed forward of the RIPEMD-160 variant. By choosing the differences in the MSB, we reduce the complexity of the attack enormously, since the modular addition behaves linearly for differences in the MSB. So only the conditions for the nonlinear functions f_2, f_3, f_4, f_5 have to be considered for the attack complexity. In detail, one condition has to be fulfilled for the nonlinear functions f_j in each step of the left and the right stream in V_2.

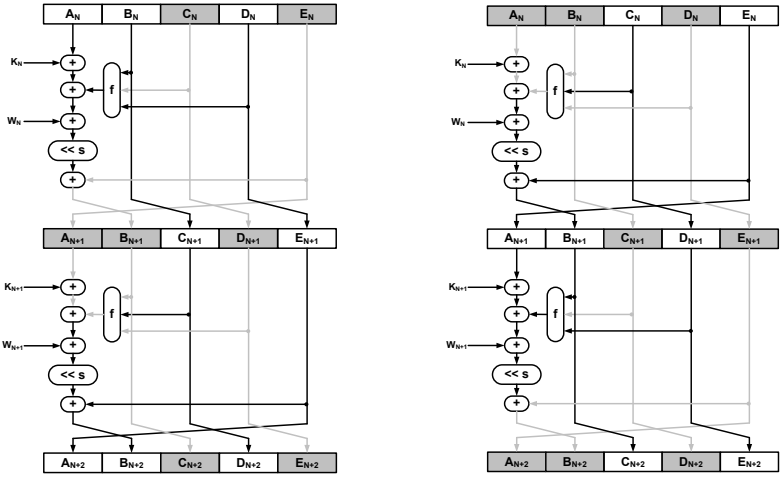

Fig. 5. Two fixed-points for two steps of the RIPEMD-160 variant.

To cancel a difference in the expanded message word w_i, we exploit the properties of the functions f_j. The output of the functions f_2, f_3, f_4 and f_5 is either 1 or 0 with probability 1/2 for an input difference $\delta = (1, 1, 1)$, which allows us to cancel differences in the expanded message words in round 2, 3, and 4 of the RIPEMD-160 variant. In the first round of the left stream and in the last round of the right stream, the linear function f_1 is used, making it impossible to cancel a difference there, because f_1 flips with probability 1 for $\delta = (1, 1, 1)$. Since there are differences in all message words in the MSB, f_2, f_3, f_4 have to be blocked in each round of V_2. We use another (general) characteristic in V_1. Hence, we have an attack on the RIPEMD-160 variant reduced to 3 rounds. We derive the following set of conditions for round 2 and 3 of the right and the left stream. Note that the conditions are equal for the right and the left stream.

$$\begin{aligned} B_{i,32} = \neg C_{i,32} = D_{i,32} \quad & i = 16 \\ B_{i,32} = \neg B_{i-1,32} \quad & i = 17, \ldots, 47 \end{aligned}$$

This results in a set of 64 conditions (32 for each stream). Satisfying all these conditions with the most naive method (random trials), we get a complexity close to 2^{64} hash computations. Note that no conditions are needed for the modular addition in the feed forward, since we have only differences in the MSB of all state variables of the left and the right stream.

Finding a pseudo-collision in the according RIPEMD-320 reduced to 4 rounds has a complexity of at most 2^{76} hash computations. RIPEMD-320 is an extension of RIPEMD-160 which has the same security level as RIPEMD-160, but produces a hash value of 320 bits. In Appendix B, the conditions for all 4 rounds as well as a sample pseudo-collision on a round-reduced variant (2 rounds) are given.

Improving the Attack Complexity. The attack complexity can be further improved by using one of the fixed-points shown in Figure 5 and by choosing differences in the MSB of w_i, for $i = 1, 4, 6, 7, 10, 11, 12, 15$. Using one of these fixed-points, we can construct an attack on the RIPEMD-160 variant reduced to the first 3 rounds with complexity close to 2^{51} hash computations. By choosing differences in the MSB of w_i, for $i = 1, 4, 6, 7, 10, 11, 12, 15$, only 8 conditions are needed instead of 16 in round 3 of the left and the right stream. This is due to the fact that the differences in the message words are chosen in such a way that only the even or odd words of the left and the right stream have differences in the MSB. Hence, the number of conditions is reduced from 64 to 48. In more detail, if f_3 flips for an input $\delta = (0, 1, 0)$, then it also flips in the next step with input $\delta = (1, 0, 1)$. Hence, round 3 has a probability of 2^{-8} and not 2^{-16} as one may expect. Since we need 5 message blocks to have a collision after the last block, the final attack complexity is $2^{48} \cdot 5$. Since all the differences in the state variables are in the MSB, no additional conditions have to be fulfilled for the feed forward. Note that the same *L-characteristic* is used in each message block and only the general characteristic is different for each block. The conditions for the used *L-characteristic* are given in Appendix A.

5.2 Attack on the RIPEMD-160 Variant Using Fixed-Points

Since we assume that we use a general characteristic in V_1 (first round) to obtain the desired target differences at the input of the first step of V_2, we have an attack on the RIPEMD-160 variant reduced to the first 3 rounds using one of the fixed-points described before. Using one message block to construct a collision, the attack has complexity 2^{64} and complexity 2^{51} using 5 message blocks. Even though we cannot extend this attack to the full RIPEMD-160 variant, we conjecture that the rotation of state variable C in the state update transformation enhances the security of RIPEMD-160. The attack works as follows:

1. Choose differences in the MSB in message words w_i.
2. Use a general characteristic to construct differences in the MSB in the state variables at the input of the first step in V_2 (to match the desired target difference) and fulfill the conditions for the first 16 steps (V_1) using message modification techniques and neutral bits. Note that if more than one message block is needed to produce a collision then this step has to be repeated for each block.
3. Construct the set of conditions for the *L-characteristic* in V_2 corresponding to the chosen differences in the message words w_i.
4. Fulfill the conditions for V_2 by random trials. The final attack complexity is related to the number of conditions in V_2.

6 Conclusion

In this article, we used recent results in the cryptanalysis of hash functions to analyze the security of RIPEMD-160. We combined methods from coding

theory with recent attack techniques which were successfully used in the attack on SHA-1. While RIPEMD and RIPEMD-128 reduced to 3 rounds are vulnerable to this kind of attack, the attack is not suitable for RIPEMD-160.

Furthermore, we analyzed a variant of RIPEMD-160, where the rotation of state variable C was removed. We show that for this variant an attack on 3 rounds is possible using fixed-points. Hence, we conclude that the rotation of state variable C enhances the security level of RIPEMD-160.

We found no attack on the original RIPEMD-160 hash function including all 5 rounds. In summary, we state that RIPEMD-160 is secure against known attacks. Neither the attack of Dobbertin or Wang *et al.* on RIPEMD can be extended to RIPEMD-160, nor recent methods used in the cryptanalysis of SHA-1 are applicable to full RIPEMD-160. Even though this paper gives new insights on the security of RIPEMD-160, further analysis is required to get a good view on its security margin.

References

1. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *LNCS*, pages 290–305. Springer, 2004.
2. Eli Biham, Rafi Chen, Antoine hirose, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 36–57. Springer, 2005.
3. Bert den Boer and Antoon Bosselaers. Collisions for the Compression Function of MD5. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 1994.
4. Florent Chabaud. On the Security of Some Cryptosystems Based on Error-correcting Codes. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *LNCS*, pages 131–139. Springer, 1995.
5. Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *LNCS*, pages 56–71. Springer, 1998.
6. Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family.* PhD thesis, Ruhr Universität Bochum, 2005. Available at `http://www.cits.rub.de/imperia/md/content/magnus/dissmd4.pdf`.
7. Hans Dobbertin. Ripemd with two-round compress function is not collision-free. *J. Cryptology*, 10(1):51–70, 1997.
8. Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption,*

Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1996.

9. Krystian Matusiewicz and Josef Pieprzyk. Finding good differential patterns for attacks on SHA-1. Cryptology ePrint Archive, Report 2004/364, 2004. `http://eprint.iacr.org/`.
10. Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Exploiting Coding Theory for Collision Attacks on SHA-1. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *LNCS*, pages 78–95. Springer, 2005.
11. Vincent Rijmen and Elisabeth Oswald. Update on SHA-1. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *LNCS*, pages 58–71. Springer, 2005.
12. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.
13. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
14. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.
15. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *LNCS*, pages 1–16. Springer, 2005.

A Set of Sufficient Conditions for the *L-characteristic* of the RIPEMD-160 Variant Reduced to 3 Rounds

In this section, we will give the complete set of sufficient conditions for the attack on the RIPEMD-160 variant reduced to 3 rounds using a fixed-point for 2 steps as described in Section 5.

For the analysis, we assume that we can find a general characteristic for round 1 such that we have differences in state variable C of the left stream and the right stream in the input of the first step of round 2. Since there are differences in the message words w_i, $i = 1, 4, 6, 7, 10, 11, 12, 15$, the number of conditions is reduced, see Section 5. Hence, we derive the following set of equations for the *L-characteristic* for round 2 and 3 of the right and the left stream.

– Left Stream:

$$\begin{aligned} B_{i,32} &= 0 \quad i = 18, 26, 28 \\ B_{i,32} &= 1 \quad i = 16, 20, 22, 24, 30, 32, 34, 36, 38, 40, 42, 44, 46 \\ B_{i,32} &= B_{i-2,32} \quad i = 19, 23, 25, 31 \\ B_{i,32} &= \neg B_{i-2,32} \quad i = 17, 21, 27, 29 \end{aligned}$$

– Right Stream:

$$\begin{aligned} B_{i,32} &= 0 \quad i = 14, 26, 28, 32, 34, 36, 38, 40, 42, 44, 46 \\ B_{i,32} &= 1 \quad i = 16, 18, 20, 22, 24 \\ B_{i,32} &= B_{i-2,32} \quad i = 17, 19, 23, 25, 29 \\ B_{i,32} &= \neg B_{i-2,32} \quad i = 21, 27 \\ B_{i,32} &= B_{i-1,32} \oplus B_{i-2,32} \quad i = 31 \end{aligned}$$

B Set of Sufficient Conditions for a Pseudo-Collision in a Round-Reduced RIPEMD-320 Variant

In this section, we will give a set of sufficient conditions for a pseudo-collision in a RIPEMD-320 variant. Note that there are no differences in the message words and the IV has differences in the MSB of all words. This would result in an attack complexity of 2^{128} for a pseudo-collision in RIPEMD-320. Since we assume that we can fulfill the first 16 to 20 steps of the right stream (no conditions have to be fulfilled for the first 16 steps in the left stream), the attack complexity would be 2^{108}.

– Left Stream:

$$\begin{aligned} B_{i,32} = C_{i,32} = D_{i,32} &= 1 \quad i = 16 \\ B_{i,32} &= B_{i-1,32} \quad i = 17, \ldots, 79 \end{aligned}$$

– Right Stream:

$$\begin{aligned} B_{i,32} = C_{i,32} = D_{i,32} &= 1 \quad i = 1 \\ B_{i,32} &= B_{i-1,32} \quad i = 2, \ldots, 63 \end{aligned}$$

Below, a message and the according IV is given for a pseudo-collision in the first 2 rounds of the RIPEMD-320 variant which has a complexity of 2^{28} hash computations. A pseudo-collision for the first 3 rounds would require about 2^{60} hash computations.

i	M							
0-7	1330C95E	D6E82F5D	1902E1F8	040C42B4	F51D77D2	B8EF7ED0	D075FEE3	1CB083FD
8-15	37246C9D	72205B19	703A3DCD	E7E5AFFD	FD9D1E57	4C64C76F	4B424959	56B11DB4

i	IV				
0-4	A99DA4B3	257D7E0C	56D85144	8F93F035	79096694
5-9	58EEE5C0	AA910BAB	BD91DCA9	8D5BE12A	14C72EF0

C Hamming Weight of Codewords Found for using an *L-characteristic* in V_1 and V_2

In this section, we will give the Hamming weight of the codewords found for using an *L-characteristic* in V_1 and V_2 as described in Section 3.1. In Table 3, the Hamming weight of the codewords found for RIPEMD-160, RIPEMD-128 and round-reduced variants are shown. Since we assume that it is possible to turn near-collisions to collisions by using multi-block messages (see Observation 2), we can improve the Hamming weight of the codewords found and hence the probability of the characteristic. For a near-collision, the condition of having zero differences after the feed forward can be ignored. The Hamming weight of the codewords found are also shown in Table 3. Note that we only give the Hamming weight after step 16, since the first 16 steps (V_1) can be fulfilled in advance, and only the probability of V_2 is significant for the attack complexity. We conclude that the final attack complexity would be too high for a reasonable attack.

Table 3. Hamming weight of A using an *L-characteristic* in V_1 and V_2.

	type	Hamming weight	projection*	steps	stream
RIPEMD - 160	collision	1760	768	16–80	both
	near-collision	1568	768	16–80	both
	collision	895	448	16–80	left
	collision	848	576	16–80	right
	collision	1184	576	16–64	both
	near-collision	1184	576	16–64	both
	collision	608	320	16–64	left
	collision	644	352	16–64	right
	collision	863	384	16–48	both
	near-collision	768	352	16–48	both
	collision	421	160	16–48	left
	collision	414	128	16–48	right
RIPEMD - 128	collision	1303	640	16–64	both
	near-collision	880	512	16–64	both
	collision	602	256	16–64	left
	collision	576	320	16–64	right
	collision	800	256	16–48	both
	near-collision	640	256	16–48	both
	collision	377	64	16–48	left
	collision	374	128	16–48	right

(*)Results achieved by using a projection as described in Section 4.