Distinguishers on double-branch compression Function and applications to round-reduced RIPEMD-128 and RIPEMD-160

Sasaki, Yu; Wang, Lei

2014

Sasaki, Y., & Wang, L. (2014). Distinguishers on Double-Branch Compression Function and Applications to Round-Reduced RIPEMD-128 and RIPEMD-160. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E97-A (1), 177-190.

<https://hdl.handle.net/10356/103817>

<https://doi.org/10.1587/transfun.E97.A.177>

© 2014 The Institute of Electronics, Information and Communication Engineers. This paper was published in IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences and is made available as an electronic reprint (preprint) with permission of The Institute of Electronics, Information and Communication Engineers. The paper can be found at the following official DOI: [<http://dx.doi.org/10.1587/transfun.E97.A.177>]. One print or electronic copy may be made for personal use only. Systematic or multiple reproduction, distribution to multiple locations via electronic or other means, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper is prohibited and is subject to penalties under law.

Distinguishers on Double-Branch Compression Function and Applications to Round-Reduced RIPEMD-128 and RIPEMD-160*

Yu SASAKI^{†a)}, Member and Lei WANG^{††b)}, Nonmember

SUMMARY This paper presents differential-based distinguishers against double-branch compression functions and applies them to ISO standard hash functions RIPEMD-128 and RIPEMD-160. A double-branch compression function computes two branch functions to update a chaining variable and then merges their outputs. For such a compression function, we observe that second-order differential paths will be constructed by finding a sub-path in each branch independently. This leads to 4-sum attacks on 47 steps (out of 64 steps) of RIPEMD-128 and 40 steps (out of 80 steps) of RIPEMD-160. Then new properties called a (partial) 2-dimension sum and a q -multi-second-order collision are considered. The partial 2-dimension sum is generated on 48 steps of RIPEMD-128 and 42 steps of RIPEMD-160, with complexities of 2^{35} and 2^{36} , respectively. Theoretically, the 2-dimension sum is generated faster than the brute force attack up to 52 steps of RIPEMD-128 and 51 steps of RIPEMD-160, with complexities of 2^{101} and 2^{158} , respectively. The results on RIPEMD-128 can also be viewed as q -multi-second-order collision attacks. The practical attacks have been implemented and examples are presented. We stress that our results do not impact to the security of full RIPEMD-128 and RIPEMD-160 hash functions.

key words: RIPEMD-128, RIPEMD-160, double-branch structure, 2-dimension sum, q -multi-second-order collision

1. Introduction

Hash functions are taking important roles in various aspects of the cryptography. Since the collision resistance of MD5 and SHA-1 were broken by Wang et al. [36], [37], cryptographers have looked for stronger hash function designs. While various new designs are discussed in the SHA-3 competition [32], some of existing hash functions seem to have much higher security than the MD4-family. Evaluating such hash functions are useful especially if they have been standardized internationally.

RIPEMD-128 and RIPEMD-160 [8] are hash functions standardized by ISO [13]. RIPEMD-160 is also included in the recommended ciphers list of the Cryptography Research and Evaluation Committees (CRYPTREC) set up by the Japanese Government [7]. RIPEMD-160 is standardized in the SSL protocol [9], and is included in the OpenSSL Cryptography and SSL/TLS Toolkit [25]. RIPEMD-128 and

RIPEMD-160 are implemented in various cryptographic libraries, for example, BouncyCastle [18], FlexiProvider [31], and GNU Crypto [10] for JAVA, and Crypto++ [6] for C++. The use of RIPEMD-128 and RIPEMD-160 in HMAC is explained in RFC [14], [15].

Several papers have been published on the security evaluation of RIPEMD-128 and RIPEMD-160 [17], [19]–[21], [24], [35]. For collision resistance, the attacked steps on RIPEMD-128 reach 36 for hash function [20] and 64 (full) for compression function [17]. The attacked steps on RIPEMD-160 reach 36 out of 80 for compression function [19]. For preimage resistance, the attacks on RIPEMD-128 and RIPEMD-160 reach 36 steps [35] and 31 steps [24] respectively. For other non-randomness distinguisher resistance, the attacks on RIPEMD-128 and RIPEMD-160 reach 64 (full) steps [17] and 48 steps [19] respectively.

This paper deals with non-randomness distinguishers on the high-level compression function structure of RIPEMD-128 and RIPEMD-160. The compression functions of RIPEMD-128 and RIPEMD-160 adopt the double-branch structure, which takes a previous chaining variable H_{i-1} and a message block M_{i-1} as input and computes two functions $CF^L(H_{i-1}, M_{i-1})$ and $CF^R(H_{i-1}, M_{i-1})$. The output H_i is computed by merging H_{i-1} , $CF^L(H_{i-1}, M_{i-1})$, and $CF^R(H_{i-1}, M_{i-1})$. Note that, in order to prevent the recent meet-in-the-middle preimage attacks [11], [28], [29], some hash functions adopt a structure, which applies the feed-forward function several times, e.g. ARIRANG [5]. Ohtahara et al. pointed out that such a structure can be viewed as the double-branch structure [23]. Hence, analyzing the double-branch compression function is very necessary both for currently standardized hash functions such as RIPEMD-128 and RIPEMD-160 and for future hash function designs.

More precisely, in this paper boomerang type differential properties on double-branch compression functions are discussed. The boomerang attack was first proposed by Wagner for analyzing block-ciphers [33]. It divides the function $E(\cdot)$ into two subparts E_0 and E_1 such that $E(\cdot) = E_1 \circ E_0(\cdot)$. Let the probabilities of differential paths for E_0 and E_1 be p and q , respectively. The boomerang attack exploits the fact that a second order differential property with a probability p^2q^2 exists for the entire function E . Aumasson et al. [1] applied the boomerang attack to the internal cipher of the hash function Skein. However, the goal of the attack is still recovering the secret key. After that Biryukov et al. [4] and Lamberger and Mendel [16] independently applied this property on the compression function

Manuscript received March 29, 2013.

Manuscript revised July 1, 2013.

[†]The author is with NTT Secure Platform Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.

^{††}The author is with the Nanyang Technological University, Singapore.

*A preliminary version was presented in ACNS 2012 [30]. More detailed explanation of attacks on RIPEMD-160 is added in Sect. 6.3. More experimental results are added.

a) E-mail: sasaki.yu@lab.ntt.co.jp

b) E-mail: wang.lei@ntu.edu.sg

DOI: 10.1587/transfun.E97.A.177

so as to mount distinguishers. Then, Sasaki [27] showed a straight-forward application of the framework of [4], [16] to the MD4-family (using the single-branch structure) consisting of up to 5 rounds. With the straight-forward technique by [27], it is unclear how to attack the single-branch structure consisting of more than 5 rounds.

Our Contributions

This paper presents differential distinguishers against the double-branch compression functions. Our first target is the 4-sum property. Then, new differential properties called a *2-dimension sum* and a *q-multi-second-order collision* are considered. Note that the partial 4-sum and partial 2-dimension sum can be introduced naturally.

The differential path construction of our distinguishers is based on the framework of the boomerang distinguisher by [4], [16]. Our strategy is regarding CF^L as the first part of the boomerang attack (E_0) and CF^R as the second part (E_1), hence the entire compression function (E) is viewed as the single-branch structure. This simplifies the differential path construction because the differential paths for CF^L and CF^R can be analyzed almost independently. However, because the straight-forward application of the framework to the MD4-family [27] can only work up to 5 rounds, several improvements are necessary to extend the number of attacked rounds.

As an illustration, we apply the attack to RIPEMD-128 and RIPEMD-160. The results are summarized in Table 1. On RIPEMD-128, we use the local collision to construct differential paths. This leads to a long differential path satisfied with a high probability. As a result, 4-sums and partial 2-dimension sums are generated with a practical complexity up to 47 and 48 steps, respectively. In addition, 2-dimension sums are theoretically generated faster than the brute force attack up to 52 steps.

On RIPEMD-160, the local collision involves more message words than RIPEMD-128, and thus using the local collision is inefficient. Instead, we show an interesting non-linear differential property of RIPEMD-160 which can avoid the quick propagation of the difference. As a result,

4-sums and partial 2-dimension sums are generated with a practical complexity up to 38 and 40 steps, respectively. In addition, 2-dimension sums are theoretically generated faster than the brute force attack up to 43 steps. If the attack target starts from the second round, the numbers of attacked steps become 40, 42, and 51 for 4-sums, partial 2-dimension sums, and theoretical 2-dimension sums, which attack 3 steps more compared with other distinguishers [19].

We stress that the above non-randomness distinguisher cannot be adapted trivially to attack the classical security notions of hash functions, and thus has rather limited impacts. Nevertheless, such distinguishers indeed reveal the security margin of hash functions. We believe that our work proposes a new approach to evaluate the security robustness of double-branch hash functions and particularly provides a better understanding of the security margin of RIPEMD-160. Moreover, we find an observation to efficiently control the difference propagation in RIPEMD-160 (see examples in Figs. 2 and 3), which may also be exploited for finding collision attacks.

Paper Outline

In Sect. 2, differential properties are discussed. In Sect. 3, the specification of RIPEMD-128 and -160 are explained. In Sect. 4, our attack strategy is explained. In Sect. 5, attacks on RIPEMD-128 are explained. In Sect. 6, attacks on RIPEMD-160 are explained. Finally, the paper is concluded in Sect. 7.

2. Differential Properties to be Distinguished

We summarize differential properties discussed in previous papers, which are 4-sums and second-order collisions, and introduce new properties called *2-dimension sums* and *q-multi-second-order collision*.

2.1 Previously Discussed Properties

4-sum is a set of 4 different inputs (I_0, I_1, I_2, I_3) where the sum of the corresponding outputs is 0, namely $CF(I_0) \oplus$

Table 1 Distinguishers on the compression functions. “2D-sum” and “q-multi-2nd” denote 2-dimension sum and q-multi-second-order collision respectively.

Target	#Steps	Property	Information Theoretic Bound	Generic Attack	Complexity
RIPEMD-128 (total 64 steps)	45	4-sum	2^{32}	2^{42}	2^{27}
	47	4-sum	2^{32}	2^{42}	2^{39}
	48	q-multi-2nd	2^{55} for $q = 17$	-	2^{53}
	52	q-multi-2nd	2^{108} for $q = 53$	-	2^{107}
RIPEMD-160 (total 80 steps)	38	4-sum	2^{40}	2^{53}	2^{42}
	40	partial 2D sum	2^{64}	2^{64}	2^{42}
	43	2D sum	2^{160}	2^{160}	2^{151}
	40 [†]	4-sum	2^{40}	2^{53}	2^{36}
	42 [†]	partial 2D sum	2^{64}	2^{64}	2^{36}
	51 [†]	2D sum	2^{160}	2^{160}	2^{158}

[†]: The attacked steps start from the second round.

$CF(I_1) \oplus CF(I_2) \oplus CF(I_3) = 0$. If the function is ideal, finding 4-sums requires at least $2^{n/4}$ queries for n -bit output. Therefore, if the 4-sum can be generated with the complexity less than $2^{n/4}$ computations, CF is regarded as non-ideal. Apart from the information theoretic bound ($2^{n/4}$), the current best generic attack to find 4-sums is a generalized birthday attack [34], which requires $2^{n/3}$ computations and $2^{n/3}$ memory. Hence, if 4-sums are generated with a complexity between $2^{n/4}$ and $2^{n/3}$, CF is said to be weak because the same property cannot be detected on other functions with the current knowledge. 4-sum may be defined for other group operations such as the modulo addition. However, for different group operations, generic attack complexity will change slightly. So far, 4-sum is always discussed for the exclusive-or operation.

The *second-order collision* [3], [16] is a special form, in other words, a subset of the 4-sum. It can be viewed as limiting the form of input values on the 4-sum property. [3], [16] defined the derivative at a point α for a function f as

$$\Delta_{(\alpha)}f(y) = f(y + \alpha) - f(y).$$

Then, the second-order derivative[†] at (α, β) is defined as

$$\begin{aligned} \Delta_{(\alpha, \beta)}f(y) &= \Delta_{(\beta)}(\Delta_{(\alpha)}f(y)) \\ &= f(y + \alpha + \beta) - f(y + \beta) - f(y + \alpha) + f(y). \end{aligned}$$

The second-order collision attack is finding (α, β, y) such that $\Delta_{(\alpha, \beta)}f(y) = 0$. Previous work [3], [16] showed that the information theoretic bound is $3 \cdot 2^{n/3}$ because the problem is essentially finding three parameters α, β, y with an n -bit relation. On the other hand, the current best generic attack requires $2^{n/2}$.

2.2 2-Dimension Sums and Suitability for Double-Branch Structure

Similarly to the framework of the rebound attack [22], which limits the input and output differences before making any query, we introduce a new differential property, which we call *2-dimension sums*. The 2-dimension sum is a special form, in other words, a subset of the second-order collision. We further introduce limitations to the form of input values on the second-order collision. The problem is changed to find a value y such that $\Delta_{(\alpha, \beta)}f(y) = 0$ for two pre-specified values α and β . The 2-dimension sum is different from the second-order collision only in the sense that α and β are pre-specified. The information theoretic bound for the 2-dimension sum is 2^n because the problem is essentially finding an n -bit value satisfying an n -bit condition. A generic attack for this problem is also 2^n computations; choose the value of y and check that the corresponding $CF(y + \alpha + \beta) - CF(y + \beta) - CF(y + \alpha) + CF(y)$ is 0.

The 2-dimension sum is particularly useful to attack the double-branch structure. The attacker can construct a pseudo-near-collision path for CF^L with setting input chaining variable difference to α . Then, a pseudo-near-collision path for CF^R is independently constructed with setting other

difference β . If the product of the probability of each path (after the message modification) is higher than $2^{-n/2}$, the 2-dimension sum can be generated with the complexity less than 2^n by using the boomerang attack approach [4], [16]. Therefore, the independence of the path construction for CF^L and CF^R greatly helps the attacker. A detailed discussion is given in Sect. 4.

Note that the *partial 2-dimension sum* is naturally introduced, where $CF(y + \alpha + \beta) - CF(y + \beta) - CF(y + \alpha) + CF(y)$ becomes 0 only for the specified partial bits, say d bits. In this case, the complexity of the generic attack is 2^d and thus a valid distinguisher must find it with a complexity less than 2^d computations.

2.2.1 N -Dimension Sum for N -Branch Compression Functions

The concept of the 2-dimension sums can be naturally expanded to an *N -dimension sum*. Let a set of N pre-specified difference be $(\Delta_1, \Delta_2, \dots, \Delta_N)$. Let a notation $\Delta[\delta_1\delta_2 \cdots \delta_N]$, where $\delta_x \in \{0, 1\}$ represent $\bigoplus_{x=1}^N (\Delta_x \cdot \delta_x)$, where $\Delta_x \cdot \delta_x$ is Δ_x for $\delta_x = 1$ and is 0 otherwise. The property of the N -dimension sum is defined to finding an input value I such that, for all possible 2^N patterns of $(\delta_1\delta_2 \cdots \delta_N)$, $\bigoplus CF(I \oplus \Delta[\delta_1\delta_2 \cdots \delta_N]) = 0$. A generic attack on N -dimension sum is the same as the one for 2-dimension sum, which requires 2^n computations.

Similarly to the 2-dimension sum, the N -dimension sum is useful to attack the N -branch structure, because differential paths can be constructed independently for each of N branches. If the overall probability of N -sum is higher than 2^{-n} , for example the probability of each path is higher than $2^{-\frac{n}{N \times 2^{N-1}}}$, the N -dimension sum can be generated with a complexity less than 2^n . Hash function FORK-256 [12] adopts the 4-branch structure and the computation of each branch is light, and thus it seems a suitable application target of the N -dimension sum.

2.3 q -Multi-Second-Order Collision

By following the framework of the q -multicollision [2], we introduce a notion of a q -multi-second-order collision on a function $f : \{0, 1\}^X \rightarrow \{0, 1\}^Y$, which is a set of two non-zero differences and q distinct inputs $\{\Delta, \nabla, x_1, x_2, \dots, x_q\}$ satisfying

$$\begin{aligned} f(x_1) - f(x_1 + \Delta) + f(x_1 + \Delta + \nabla) - f(x_1 + \nabla) &= 0, \\ \dots \\ f(x_q) - f(x_q + \Delta) + f(x_q + \Delta + \nabla) - f(x_q + \nabla) &= 0. \end{aligned}$$

2.3.1 Information-Theoretic Bound

Let an adversary make k distinct queries $x_1, \dots, x_k$ to a random function. For any specified Δ and ∇ , k distinct queries

[†]In [3], [16], the n -th order derivative is discussed rather than the specific case $n = 2$.

Table 2 Boolean functions, message expansions, and rotation numbers.

	$f_x(X, Y, Z)$														
$x = 0, \dots, 15$	$X \oplus Y \oplus Z$														
$x = 16, \dots, 31$	$(X \wedge Y) \vee (\neg X \wedge Z)$														
$x = 32, \dots, 47$	$(X \vee \neg Y) \oplus Z$														
$x = 48, \dots, 63$	$(X \wedge Z) \vee (Y \wedge \neg Z)$														
$x = 64, \dots, 79$	$X \oplus (Y \vee \neg Z)$														
	$\pi^L(j)$														
$j = 0, \dots, 15$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$j = 16, \dots, 31$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11
$j = 32, \dots, 47$	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5
$j = 48, \dots, 63$	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6
$j = 64, \dots, 79$	4	0	5	9	7	12	2	10	14	1	3	8	11	6	15
	$\pi^L(R)$														
$j = 0, \dots, 15$	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3
$j = 16, \dots, 31$	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1
$j = 32, \dots, 47$	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4
$j = 48, \dots, 63$	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10
$j = 64, \dots, 79$	12	15	10	4	1	5	8	7	6	2	13	14	0	3	9

at most contribute to k quartets as below; $(x_1, x_1 + \Delta, x_1 + \nabla, x_1 + \Delta + \nabla), \dots, (x_k, x_k + \Delta, x_k + \nabla, x_k + \Delta + \nabla)$. So the number of q -tuple quartets is $\binom{k}{q}$ for any specified Δ and ∇ . There are less than 2^{2X} values for (Δ, ∇) . Thus in total the number of q -tuple quartets satisfying the form is at most $2^{2X} \cdot \binom{k}{q}$. One such q -tuple has a probability $2^{-q \times Y}$. We get the following inequality.

$$\begin{aligned}
 2^{2X} \cdot \binom{k}{q} &\geq 2^{qY}, \\
 \frac{k(k-1) \cdots (k-(q-1))}{q!} &\geq 2^{qY-2X}, \\
 k &> \sqrt[q]{q!} \cdot 2^{Y-\frac{2X}{q}}.
 \end{aligned} \tag{1}$$

3. Specifications of RIPEMD-128 and RIPEMD-160

RIPEMD-128 and RIPEMD-160 were proposed by Dobbertin et al. [8] as stronger hash functions than RIPEMD [26]. They take a message of arbitrary length as input and produce 128-bit and 160-bit hash digests, respectively. Because our attack target is the compression function, we omit the description of the domain extension.

3.1 RIPEMD-128

The compression function of RIPEMD-128 takes a 128-bit chaining variable H_{i-1} and a 512-bit message block M_{i-1} as input and outputs a 128-bit chaining variable H_i . M_i is divided into sixteen 32-bit message words $m_0, m_1, \dots, m_{15}$. Let p_j^L be a 128-bit chaining variable and a_j^L, b_j^L, c_j^L and d_j^L be 32-bit variables satisfying $p_j^L = a_j^L \| b_j^L \| c_j^L \| d_j^L$, where $0 \leq j \leq 64$. Similarly, p_j^R and $a_j^R, b_j^R, c_j^R, d_j^R$ are defined. The computation for CF^L is as follows.

$$\begin{aligned}
 p_0^L &\leftarrow H_{i-1}, \\
 p_{j+1}^L &\leftarrow \text{SF}_j^L(p_j^L, m_{\pi^L(j)}) \text{ for } j = 0, 1, \dots, 63,
 \end{aligned}$$

Table 3 Computations for the output of the compression function.

	RIPEMD-128	RIPEMD-160
$H_i^{(a)}$	$H_{i-1}^{(b)} + c_{64}^L + d_{64}^R$	$H_{i-1}^{(b)} + c_{80}^L + d_{80}^R$
$H_i^{(b)}$	$H_{i-1}^{(c)} + d_{64}^L + a_{64}^R$	$H_{i-1}^{(c)} + d_{80}^L + e_{80}^R$
$H_i^{(c)}$	$H_{i-1}^{(d)} + a_{64}^L + b_{64}^R$	$H_{i-1}^{(d)} + e_{80}^L + a_{80}^R$
$H_i^{(d)}$	$H_{i-1}^{(a)} + b_{64}^L + c_{64}^R$	$H_{i-1}^{(e)} + a_{80}^L + b_{80}^R$
$H_i^{(e)}$	–	$H_{i-1}^{(a)} + b_{80}^L + c_{80}^R$

where SF_j^L is a step function for CF^L and performs the following computation.

$$\begin{aligned}
 a_{j+1}^L &\leftarrow d_j^L, \\
 b_{j+1}^L &\leftarrow (a_j^L + f_j(b_j^L, c_j^L, d_j^L) + m_{\pi^L(j)} + k_j^L) \lll s_j^L, \\
 c_{j+1}^L &\leftarrow b_j^L, \\
 d_{j+1}^L &\leftarrow c_j^L,
 \end{aligned}$$

where ‘+’ represents the addition on modulo 2^{32} , ‘ $\lll s$ ’ represents left cyclic shift by s bits, f_x is a Boolean function, $\pi^L(j)$ is the message expansion, and k_j^L is a constant. CF^R is similarly described. The values of $s_j^R, \pi^R(j), k_j^R$ are different and f_{63-j} is used in step j . Details of $f_x, \pi^L(j), \pi^R(j)$ are in Table 2.

Finally, the output $H_i = H_i^{(a)} \| H_i^{(b)} \| H_i^{(c)} \| H_i^{(d)}$ is computed as shown in Table 3.

3.2 RIPEMD-160

The compression function of RIPEMD-160 is almost the same as the one for RIPEMD-128. The chaining variable size is 160 bits, and thus 160-bit intermediate states are represented by five 32-bit variables, e.g. $p_j^L = a_j^L \| b_j^L \| c_j^L \| d_j^L \| e_j^L$. The step functions SF^L and SF^R are iteratively computed 80 times ($0 \leq j \leq 79$). The details of the computation of SF^L are as follows.

$$a_{j+1}^L \leftarrow e_j^L,$$

Fig. 1 Overview of our attack strategy.

$$\begin{aligned}
 b_{j+1}^L &\leftarrow ((a_j^L + f_j(b_j^L, c_j^L, d_j^L) + m_{\pi^L(j)} + k_j^L) \lll s_j^L) + e_j^L, \\
 c_{j+1}^L &\leftarrow b_j^L, \\
 d_{j+1}^L &\leftarrow c_j^L \lll 10, \\
 e_{j+1}^L &\leftarrow d_j^L.
 \end{aligned}$$

Most of the parameters are shared with RIPEMD-128. The details are described in Table 2. In the computations of SF^R , the Boolean function in step j is $f_{79-j}(b_j^R, c_j^R, d_j^R)$. The other computations are similarly specified as SF^L . Finally, the output chaining variable H_i is computed as shown in Table 3.

4. Our Attack Strategy on Double-Branch Compression Function

A graphical description of our strategy is given in Fig. 1. First, we construct a Δ -differential-path $\Delta \xrightarrow{\Delta^M} \Delta'$ in the left branch, and a ∇ -differential-path $\nabla \xrightarrow{\nabla^M} \nabla'$ in the right branch. Then we try to search for an input of the compression function (H, M) such that (H, M) , $(H + \Delta, M + \Delta^M)$, $(H + \nabla, M + \nabla^M)$, and $(M + \Delta + \nabla, H + \Delta^M + \nabla^M)$ satisfy the following conditions.

- The difference propagations between (H, M) and $(H + \Delta, M + \Delta^M)$ and between $(H + \nabla, M + \nabla^M)$ and $(H + \nabla + \Delta, M + \nabla^M + \Delta^M)$ follow Δ -differential-path in the left branch.
- The difference propagations between (H, M) and $(H + \nabla, M + \nabla^M)$ and between $(H + \Delta, M + \Delta^M)$ and $(H + \Delta + \nabla, M + \Delta^M + \nabla^M)$ follow ∇ -differential-path in the right branch.

For such a (H, M) , we obtain the relationship; $CF(H, M) + CF(H + \Delta + \nabla, M + \Delta^M + \nabla^M) - CF(H + \Delta, M + \Delta^M) - CF(H + \nabla, M + \nabla^M) = 0$, where CF denotes the compression function.

Note that the terminology “4-sum” is somehow strange to discuss the above modular subtraction. However, the terminology “second-order collision” usually considers the limitation of the inputs, and discusses essentially different properties. Hence, to avoid the confusion, we use “4-sum” for the above property.

5. Attacks on RIPEMD-128

We construct differential distinguishers against compression

function of RIPEMD-128. Hereafter, we compute the difference in modular subtraction because the main operation of RIPEMD-128 and RIPEMD-160 is the modular addition. The message differences, differential path, and sufficient conditions against RIPEMD-128 are shown in Tables 4, 5 and 6 respectively.

5.1 Constructing Δ -Differential-Path

We should keep the differential path as simple as possible in order to maximize its probability. One natural approach is to restrict the difference propagations. Particularly, we expect that f functions do not produce new differences.

The f functions of the first and the third rounds in the left branch have weak absorption property. By weak absorption property, we mean that a bit difference is produced by f with the probability 1 if a (particular) input variable has a difference at that bit. So we must make the Δ -differential-path short in these two rounds. In order to achieve it, we generate a *local collision* in the second round of the left branch, and pick a message difference Δ^M which appears at a very beginning step in the first round and at a very late step in the third round. Such a strategy maximizes the probability of the whole differential path. Finally we choose the message difference Δ^M as below

$$\Delta m_0 = -2^{10}; \text{ and } \Delta m_6 = 2^{31}.$$

Δ is determined backwards according to the differential path in the first round.

$$\Delta a_0 = 2^8; \Delta b_0 = 0; \Delta c_0 = 2^{31} + 2^{16}; \text{ and } \Delta d_0 = 2^{31} + 2^{16}.$$

Δ' changes with the number of the attacked steps. Moreover, we do not specify Δ' according to *amplified probability* using multiple outside differential paths.

5.1.1 Remarks on Multiple Differential Path

At step 44 of Δ -differential-path, a difference of $*2^5$ is produced by the f function. For this difference, we do not limit the sign for each pair, but we need the condition that the signs are identical between two pairs. Hence, the probability to satisfy this condition is 2^{-1} rather than 2^{-2} . We also set 2 similar conditions at steps 46 and 47.

5.2 Constructing ∇ -Differential-Path

The f function of the second round in the right branch has weak absorption property. So we must make ∇ -differential-path short in the second round of the right branch. At the same time, the f function in the first round of the right branch has strong absorption property. By strong absorption property, we mean that no bit difference is produced by f by setting conditions if only one input variant has a difference on that bit. So we decide to generate a relatively long but simple sub-path in the first round, which should be ended by the message difference in the second round. We should

Table 4 Differential path construction for 3-round RIPEMD-128.

round	$\pi^L(j)$															
1	①	1	2	3	4	5	⑥	7	8	9	10	11	12	13	14	15
	Δ			MM		$\leftarrow$	Δ					constant				
2	7	4	13	1	10	⑥	15	3	12	①	9	5	2	14	11	8
			constant					$\leftarrow$ LC $\rightarrow$					constant			
3	3	10	14	4	9	15	8	1	2	7	①	⑥	13	11	5	12
					constant						Δ	Δ	$\rightarrow$			
	$\pi^R(j)$															
1	5	14	7	0	9	2	11	4	13	⑥	15	8	1	10	3	12
								MM								$\leftarrow$
2	⑥	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
	∇								constant							
3	15	5	1	3	7	14	⑥	9	11	8	12	2	10	0	4	13
			constant				∇	$\rightarrow$								

Table 5 Differential paths for 2-dimension sums on 3-round RIPEMD-128.

Path for CF^L		Path for CF^R	
$\Delta a_0^L = 2^8; \Delta b_0^L = 0;$ $\Delta c_0^L = *2^{31} + 2^{16};$ $\Delta d_0^L = *2^{31} + 2^{16};$		$\Delta a_0^R = 2^{20}; \Delta b_0^R = 0;$ $\Delta c_0^R = 0; \Delta d_0^R = 2^6;$	
Step j	Δb_j^L	Step j	Δb_j^R
2	$*2^{31}$	1	2^{28}
3	$*2^{31}$	2	2^{15}
22	2^8	5	2^9
43	-2^{21}	6	2^{30}
44	$*2^5$	9	2^{16}
46	$*2^{26}$	13	2^{30}
47	$-2^{28} * 2^{12}$	39	-2^4
48	$*2^{10}$	43	-2^9
		47	-2^{16}

Table 6 Sufficient conditions of attacks on 3-round RIPEMD-128.

Left Branch	
$c_{0,16}^L = 0; d_{0,16}^L = 0; b_{1,16}^L = b_{0,16}^L;$ no carry in b_2^L ; no carry in $b_3^L;$ $b_{22,8}^L = 0; c_{22,8}^L = d_{22,8}^L; b_{24,8}^L = 0; b_{25,8}^L = 1$ $b_{43,21}^L = 1; b_{43,5}^L = 0; d_{43,21}^L = 0; b_{44,21}^L = 1;$ $b_{45,26}^L = 0; b_{45,5}^L = 1; b_{46,28}^L = 0; b_{46,12}^L = 0;$ no carry in $b_{46}^L; b_{47,28}^L = 1; b_{47,26}^L = 1;$ no carry in b_{47}^L ; no carry in $b_{48}^L;$	
Right Branch	
$b_{0,6}^R = c_{0,6}^R; b_{0,28}^R = 1; b_{0,15}^R = 0; c_{0,28}^R = 0;$ $d_{0,6}^R = 0; b_{1,28}^R = 0; b_{1,15}^R = 1; b_{2,15}^R = 0$ $b_{3,9}^R = 0; b_{4,15}^R = b_{3,15}^R; b_{4,30}^R = 0; b_{4,9}^R = 1;$ $b_{5,9}^R = 0; b_{5,30}^R = 1; b_{6,30}^R = 0; b_{7,9}^R = b_{6,9}^R;$ $b_{7,16}^R = 0; b_{8,30}^R = b_{7,30}^R; b_{8,16}^R = 1; b_{9,16}^R = 0;$ $b_{11,16}^R = b_{10,16}^R; b_{12,30}^R = 0; b_{13,30}^R = 0;$ $c_{39,4}^R = d_{39,4}^R; b_{41,4}^R = 1; b_{42,9}^R = b_{41,9}^R; b_{43,9}^R = 1;$ $b_{44,9}^R = 0; b_{45,9}^R = 0; b_{46,16}^R = b_{45,16}^R; b_{47,16}^R = 1;$	
(H, M) and $(H + \nabla, M + \nabla^M)$: share same bit values on $b_{44,5}^L, b_{46,26}^L, b_{47,12}^L;$	

long path going through the whole first round and ending at a beginning step in the second round; and a short path at the late steps in the third round. Finally we choose the message difference ∇^M and corresponding ∇ as below

$$\nabla m_6 = -2^{30}; \nabla a_0 = 2^{20}; \nabla b_0 = 0; \nabla c_0 = 0; \text{ and } \nabla d_0 = 2^6.$$

5.3 Searching for (H, M)

Firstly, we search for (H, M) s which satisfy the differential path for the first 17 steps in both branches. The complexity of finding such (H, M) s, i.e. the complexity for satisfying the first 17 steps can be ignored by applying the message modification technique (e.g. [36], [37]) and optimizing the computation order. More precisely, the message modification is a technique to efficiently satisfy all conditions in the first round. It exploits the property that each step in the first round is computed with a message word which is not fixed yet. For example, to satisfy the conditions of the variable b_{j+1} in the first round, you can iterate the computation in step j many times by only changing the value of $m_{\pi(j)}$ without influencing the previous steps. Hence, by satisfying the conditions step by step, the complexity is greatly reduced. Moreover, modifying the message word m_{12} never impacts to the sufficient conditions in the first round. This is because m_{12} is used in late steps of the first round in both branch (See Table 4). Thus, once we obtain an (H, M) satisfying the differential path up to step 17, we can generate up to 2^{32} valid (H, M) by modifying m_{12} . As a summary, the complexity for satisfying the differential path for the first 17 steps is (approximately) 2^{-32} times of the complexity of satisfying the whole differential path, and thus can be ignored.

For the remaining steps (after step 17), we simply satisfy the path in the brute-force manner. Hence, the entire attack complexity only depends on the number of conditions after step 17.

5.4 Complexity Evaluation and Experiments

pick a message difference which appears at a very beginning step in the second round and at a very late step in the third round. The whole differential path consists of 2 sub-paths: a

Besides counting the number of conditions in Δ - and ∇ -differential-paths after step 17, we also experimentally verify the amplified probability for outside paths.

Attack on 45 steps. There are 9 and 7 conditions in the Δ - and ∇ -differential-paths, respectively. Each condition must be satisfied in two pairs and thus its probability is 2^{-2} . However, 1 condition in the Δ -path is the one discussed in the remarks in Sect. 5.1, which satisfied with probability 2^{-1} . Overall, the complexity is $2^{31(=2(8+7)+1)}$. We then experimentally check the amplified probability, and the final complexity is 2^{27} . This implies that 45 steps of the compression function is non-ideal with respect to the 4-sum property.

Attack on 46 and 47 steps. We experimentally checked the amplified probability, and the final complexities on 46 and 47 steps are 2^{34} and 2^{39} respectively. With respect to the 4-sum property, our attack on 47 steps is faster than the current best generic attack [34].

Attack on 48 steps. The complexity to satisfy the path is 2^{48} . We then introduce q -multi-second-order collisions. Generating them by following the path requires $q \cdot 2^{48}$. When $q = 17$, the complexity is less than 2^{53} . On the other hand, from Eq. (1), the generic case requires more than $\sqrt[17]{17!} \cdot 2^{128 - \frac{2(512+128)}{17}} \approx 2^{55.55}$. Hence, our attack is faster than the generic case.

Attack on 52 steps. ∇ -differential-path in the fourth round of the right branch becomes very complicated because the f function does not have the absorption property. Thus we only verified the amplified probability in the fourth round. As a result, the complexity to obtain a 4-sum is 2^{101} . We then consider the q -multi-second-order collision for $q = 53$. Our attack requires $53 \cdot 2^{101} < 2^{107}$, while the generic case in Eq. (1) requires $2^{108.21}$.

The attack was implemented on single PC. The generated 46-step 2-dimension sum, which is 3-round (48-step) partial 2-dimension sum, is shown in Table A.1.

6. Attacks on RIPEMD-160

Two attacks are presented on RIPEMD-160; in the first scenario, the attack target is starting from the first round and in the second scenario, the attack target is starting from the second round.

In the first scenario, the f functions of both branches do not have the absorption property in the first and third rounds. This makes the efficient differential path construction hard. On the other hand, in the second scenario, the absorption property is available in both branches in the first and third rounds. The differential path is more efficient than the first scenario, and the number of attacked steps is beyond 3 rounds (up to 52 steps).

6.1 Overall Strategy and Relatively Slow Differential Propagation

Different from RIPEMD-128, using the local-collision to construct the path is not efficient in RIPEMD-160. For RIPEMD-128, the local-collision is formed only with differences in 2 message words. However, in RIPEMD-160,

Fig. 2 Difference cancellation between e_j and f_{j+1} . The sign of Δf_{j+1} must be opposite of Δe_j . Information in '[ ]' is the bitwise difference. '[any]' represents that the bitwise difference is irrelevant.

we need the difference in 3 message-words due to the direct addition from chaining variable e_j . Hence, we stop using the local-collision. Instead, we insert the difference only into 1 message word that appears in a late step of the second round, and just propagate it to the third round as much as possible. The problem is that the differential propagation in RIPEMD-160 seems much quicker than RIPEMD-128 due to the direct addition from chaining variable e_j , and thus not so many steps can be attacked. However, we explain a useful property of RIPEMD-160 where we can limit the impact of the differential propagation. In fact, this is the main reason why we can attack more than 3 rounds in the second scenario.

6.1.1 Cancelling Differences between e_j and f_{j+1}

Assume that, in some step, there is no difference in chaining variables and a message difference is inserted. If the difference is not propagated through f in the following 3 steps, only chaining variable e has the difference. This situation is illustrated in Fig. 2. Let j be the step index of this chaining variable and $(\Delta a_j, \Delta b_j, \Delta c_j, \Delta d_j, \Delta e_j) = (0, 0, 0, 0, +2^n)$. In step j , e_j is directly added to compute b_{j+1} , thus the difference $+2^n$ is always propagated to b_{j+1} . As shown in Fig. 2, Δe_j and Δb_{j+1} can cancel each other in step $j+1$ through f_{j+1} .

Assume that the difference 2^n in b_{j+1} does not cause the carry, and thus only n -th bit has the difference. In step $j+1$, if the difference in the n -th bit of b_{j+1} is output from f_{j+1} , moreover if its sign is opposite (-2^n), the cancellation occurs.

In the attack starting from the first round, we utilize this property in the third round where $f_x(X, Y, Z)$ is $(X \vee \neg Y) \oplus Z$ in both branches. $Y = 1$ and $Z = 1$ are the conditions for this event. On the other hand, in the attack starting from the second round, $f_x(X, Y, Z)$ in the left branch is $(X \wedge Z) \vee (Y \wedge$

tial propagation can be controlled more efficiently and this enables us to attack more rounds. The complexity to generate 4-sums up to 40 steps is 2^{36} computations, which is faster than the generic 4-sum attack using the generalized birthday attack. A 40-step 2-dimension sum, which can also be regarded as 42-step partial 2-dimension sum, was gener-

Table 9 Sufficient conditions on RIPEMD-160 in the first scenario.

Left Branch	Right Branch
$b_{0,10}^L = c_{0,10}^L$; no carry in $d_{0,0}^L$ and $e_{0,0}^L$; no carry in b_{29}^L ; $b_{28,21}^L = b_{27,21}^L$; $b_{30,21}^L = 0$; $b_{31,31}^L = 1$; no carry in b_{33}^L ; $b_{32,31}^L = 1$; $b_{34,31}^L = 1$; $b_{35,31}^L \vee \neg b_{34,31}^L = 1$; $b_{35,16}^L = 0$; no carry in b_{36}^L and b_{37}^L ; $b_{37,16}^L = 1$; $b_{36,9}^L = 1$; $b_{35,9}^L = 1$; $b_{38,9}^L = 1$; $b_{38,26}^L \vee \neg b_{37,26}^L = 1$; $b_{38,7}^L = 0$; no carry in b_{39}^L ; $b_{39,19}^L \vee \neg b_{38,19}^L = 1$; $b_{40,7}^L = 1$; no carry in b_{40}^L ; $b_{39,26}^L = 1$; $b_{39,2}^L = 0$; $b_{38,26}^L = 1$; no carry in b_{41}^L ; $b_{41,26}^L = 1$; $b_{41,2}^L = 1$; $b_{40,26}^L = 0$; $b_{40,24}^L = 1$; $b_{39,24}^L = 1$; $b_{41,17}^L \vee \neg b_{40,17}^L = 1$; no carry in b_{42}^L ; $b_{42,24}^L = 1$; $b_{42,19}^L = 1$; $b_{42,4}^L = 0$; $b_{41,25}^L = 0$; $b_{41,4}^L = 1$; $b_{42,12}^L \vee \neg b_{41,12}^L = 1$;	$b_{0,9}^R = 0$; $c_{0,19}^R = 1$; $d_{0,8}^R = 0$; no carry in $b_{0,0}^R$, $c_{0,0}^R$ and $d_{0,0}^R$; $b_{0,0}^R = 0$; $b_{0,18}^R = 1$; $c_{0,9}^R = 1$; $c_{0,22}^R = 1$; $b_{0,22}^R = 0$; $b_{1,29}^R = 1$; $b_{2,10}^R = 1$; no carry in b_{32}^R ; $b_{31,21}^R = 0$; $b_{33,21}^R = 1$; $b_{34,31}^R \vee \neg b_{33,31}^R = 1$; no carry in b_{35}^R ; $b_{34,14}^R = 0$; no carry in b_{36}^R ; $b_{34,14}^R = 0$; no carry in b_{36}^R ; $b_{36,14}^R = 1$; $b_{35,31}^R = 0$; $b_{37,31}^R = 1$; $b_{37,24}^R \vee \neg b_{36,24}^R = 1$; no carry in b_{38}^R ; $b_{37,30}^R = 0$; $b_{38,9}^R \vee \neg b_{37,9}^R = 1$; no carry in b_{39}^R ; $b_{39,30}^R = 1$; $b_{38,24}^R = 1$; $b_{38,15}^R = 0$; $b_{37,24}^R = 1$; $b_{40,24}^R = 1$; no carry in b_{40}^R ; $b_{40,15}^R = 1$; $b_{39,9}^R = 1$; $b_{38,9}^R = 1$; $b_{40,8}^R \vee \neg b_{39,8}^R = 1$; no carry in b_{41}^R ; $b_{41,9}^R = 1$; $b_{40,20}^R = 0$; $b_{41,25}^R \vee \neg b_{40,25}^R = 1$; $b_{41,2}^R \vee \neg b_{40,2}^R = 1$; no carry in b_{42}^R ; $b_{42,20}^R = 1$; $b_{41,15}^R = 0$; $b_{41,8}^R = 1$; $b_{42,19}^R \vee \neg b_{41,19}^R = 1$;

ated in our experiment. The experiment result is given in Appendix.

7. Concluding Remarks

We presented distinguishers against double-branch compression functions. Differential paths were constructed by regarding CF^L as the first part and CF^R as the second part. This enabled us to analyze CF^L and CF^R independently. Moreover we have applied the attack to the compression functions of RIPEMD-128 and -160. On RIPEMD-128, the local collision was applied to construct differential paths. Partial 2-dimension sums were generated for 48 steps. Theoretically, the attack works up to 52 steps. On RIPEMD-160, the difference cancelation between Δe_j and Δf_{j+1} was exploited. Partial 2-dimension sums were generated up to 40 steps. Theoretically, the attack works up to 43 steps. If the attack starts from the second round, more rounds can be attacked. We stress that our results do not impact to the security of full RIPEMD-128 and RIPEMD-160 hash functions.

Table 11 Differential paths for 2D sums on RIPEMD-160 in the second scenario.

Path for CF ^L		Path for CF ^R	
$\Delta a_{16}^L = +2^3 + 2^2 + 2^0$,		$\Delta a_{16}^R = -2^{26} + 2^{21}$,	
$\Delta b_{16}^L = -2^{28}$,		$\Delta b_{16}^R = +2^{23}$,	
$\Delta c_{16}^L = 0$,		$\Delta c_{16}^R = -2^{16}$,	
$\Delta d_{16}^L = +2^{15}$,		$\Delta d_{16}^R = +2^{11}$,	
$\Delta e_{16}^L = -2^{10} - 2^9$.		$\Delta e_{16}^R = -2^{30}$.	
Step j	Δb_j^L	Step j	Δb_j^R
17	$+2^7$	17	-2^3
18	-2^{16}	48	$+2^{18}$
19	$+2^{23}$	52	$+2^{28}$
20	-2^6	56	$+2^6$
48	$+2^{21}$	60	$2^{22} + 2^{16}$
52	$+2^{31}$	64	$2^{26} + 2^0$
55	$+2^{25}$	66	$*2^{31} * 2^5$
56	$+2^9$	67	$*2^{22} * 2^{17} * 2^{16} * 2^{11}$
59	$+2^3$		
60	$+2^{29}$		
63	$+2^{13}$		
64	$+2^{29}$		
67	$+2^{13}$		

Table 10 Differential path construction for the intermediate 3-rounds of RIPEMD-160.

round	$\pi^L(j)$																
2	7	4	13	1	10	6	15	3	0	9	5	2	14	11	8		
3	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12	
4	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2	
	→						Δ										
	$\pi^R(j)$																
2	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2	
3	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13	
4	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14	
	→										∇						

Table 12 Sufficient conditions on RIPEMD-160 in the second scenario.

Left Branch
$b_{16,15}^L = 1, c_{16,28}^L = d_{16,28}^L$, no carry in b_{16}^L and d_{16}^L , $b_{17,28}^L = 0, c_{16,28}^L = d_{16,28}^L$, no carry in b_{18}^L , $b_{18,7}^L = 0, b_{18,6}^L = 1, c_{18,16}^L = d_{18,16}^L$, no carry in b_{19}^L , $b_{19,17}^L = 1, b_{19,16}^L = 0, c_{19,23}^L = d_{19,23}^L$, no carry in b_{20}^L , $b_{20,26}^L = 1, b_{20,23}^L = 0, c_{20,6}^L = d_{20,6}^L, b_{21,6}^L = 0, b_{21,1}^L = 1$, $b_{21,6}^L = 0, b_{21,1}^L = 1, b_{22,16}^L = 1$, no carry in b_{48}^L , $d_{48,21}^L = 0, d_{49,21}^L = 1, b_{50,31}^L = c_{50,31}^L$, no carry in $b_{52}^L, d_{52,31}^L = 1, b_{53,31}^L = 1$, $b_{54,9}^L = c_{54,9}^L$, no carry in $b_{55}^L, d_{55,25}^L = 0$, 1-bit carry in $b_{56}^L, d_{56,25}^L = 1, d_{56,10}^L = 0, d_{56,9}^L = 1$, $b_{57,3}^L = c_{57,3}^L, d_{57,10}^L = 1, d_{57,9}^L = 1$, $b_{58,20}^L = c_{58,20}^L, b_{58,19}^L = c_{58,19}^L$, 1-bit carry in b_{59}^L , $d_{59,4}^L = 0, d_{59,3}^L = 1$, 1-bit carry in b_{60}^L , $d_{60,20}^L = 0, d_{60,19}^L = 1, d_{60,4}^L = 1, d_{60,3}^L = 1$, $b_{61,14}^L = c_{61,14}^L, b_{61,13}^L = c_{61,13}^L, d_{61,20}^L = 1, d_{61,19}^L = 1$, $b_{62,30}^L = c_{62,30}^L, b_{62,29}^L = c_{62,29}^L$, 1-bit carry in b_{63}^L , $d_{63,14}^L = 0, d_{63,13}^L = 1$, no carry in b_{64}^L , $c_{64,29}^L \vee \neg d_{64,29}^L = 1, d_{64,14}^L = 0, d_{64,13}^L = 0$, $c_{65,14}^L = 1, c_{65,13}^L = 1, d_{65,29}^L = 0, c_{66,29}^L = 1$
Right Branch
$b_{16,11}^R = c_{16,11}^R, d_{16,23}^R = 0, d_{16,16}^R = 1$, no carry in b_{16}^R, c_{16}^R , and d_{16}^R , $b_{17,26}^R = c_{17,26}^R, d_{17,23}^R = 1, d_{17,3}^R = 0$, $b_{18,1}^R = c_{18,1}^R, d_{18,3}^R = 1, b_{19,13}^R = c_{19,13}^R$, no carry in $b_{48}^R, c_{48,18}^R = d_{48,18}^R, b_{49,18}^R = 0$, $b_{50,28}^R = 0$, no carry in $b_{52}^R, c_{52,28}^R = 0, d_{52,28}^R = 1$, $b_{53,28}^R = 0, b_{54,6}^R = 1$, no carry in b_{56}^R , $c_{56,6}^R = 0, d_{56,6}^R = 1, b_{57,6}^R = 0, b_{58,16}^R = 1$, no carry in $b_{60}^R, c_{60,22}^R = d_{60,22}^R, c_{60,16}^R = 0, d_{60,16}^R = 1$, $b_{61,22}^R = 0, b_{61,16}^R = 0, b_{62,26}^R = 1, b_{62,0}^R = 1$, no carry in $b_{64}^R, c_{64,26}^R \oplus d_{64,26}^R = 1, c_{64,0}^R \oplus d_{64,0}^R = 1$, no carry after adding f , no carry in b_{66}^R , signs of $*2^5$ in b_{66}^R are identical between two pairs. signs of $*2^{22} * 2^{17} * 2^{16} * 2^{11}$ in b_{67}^R are identical between two pairs.

References

- [1] J.-P. Aumasson, Ç. Çalik, W. Meier, O. Özen, R.C.-W. Phan, and K. Varici, "Improved cryptanalysis of Skein," in M. Matsui, ed., *Advances in Cryptology — ASIACRYPT 2009*, Lect. Notes Comput. Sci., vol.5912, pp.542–559, Springer-Verlag, Berlin, Heidelberg, New York, 2009. Extended version is available at Cryptology ePrint Archive: Report 2009/438.
- [2] A. Biryukov, D. Khovratovich, and I. Nikolić, "Distinguisher and related-key attack on the full AES-256," in S. Halevi, ed., *CRYPTO 2009*, Lect. Notes Comput. Sci., vol.5677, pp.231–249, Springer-Verlag, Berlin, Heidelberg, New York, 2009.
- [3] A. Biryukov, M. Lamberger, F. Mendel, and I. Nikolić, "Second-order differential collisions for reduced SHA-256," in D.H. Lee and X. Wang, eds., *Advances in Cryptology — ASIACRYPT 2011*, Lect. Notes Comput. Sci., vol.7073, pp.270–287, Springer-Verlag, Berlin, Heidelberg, New York, 2011.
- [4] A. Biryukov, I. Nikolić, and A. Roy, "Boomerang attacks on BLAKE-32," in A. Joux, ed., *Fast Software Encryption (FSE) 2011*, Lect. Notes Comput. Sci., vol.6733, pp.218–237, Springer-Verlag, Berlin, Heidelberg, New York, 2011.
- [5] D. Chang, S. Hong, C. Kang, J. Kang, J. Kim, C. Lee, J. Lee, J. Lee, S. Lee, Y. Lee, J. Lim, and J. Sung, "ARIRANG," Available at NIST home page: http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html
- [6] Crypto++, Crypto++ Library 5.6.1 API Reference, <http://www.cryptopp.com/>
- [7] Cryptography Research and Evaluation Committees (CRYPTREC), e-Government recommended ciphers list, 2003, http://www.cryptrec.go.jp/english/images/cryptrec_01en.pdf
- [8] H. Dobbertin, A. Bosselaers, and B. Preneel, "RIPEMD-160: A strengthened version of RIPEMD," in D. Gollmann, ed., *Fast Software Encryption Third International Workshop (FSE 1996)*, Lect. Notes Comput. Sci., vol.1039, pp.71–82, Berlin, Heidelberg, New York, 1996.
- [9] A. Freier, P. Karlton, and P. Kocher, *The Secure Sockets Layer (SSL) Protocol Version 3.0*, Internet Engineering Task Force (IETF), RFC 6101, 2001. <http://www.ietf.org/rfc/rfc6101.txt>
- [10] The GNU Crypto project, GNU Crypto, <http://www.gnu.org/software/gnu-crypto/>
- [11] J. Guo, S. Ling, C. Rechberger, and H. Wang, "Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2," in M. Abe, ed., *Advances in Cryptology — ASIACRYPT 2010*, Lect. Notes Comput. Sci., vol.6477, pp.56–75, Springer-Verlag, Berlin, Heidelberg, New York, 2010.
- [12] D. Hong, D. Chang, J. Sung, S. Lee, S. Hong, J. Lee, D. Moon, and S. Chee, "A new dedicated 256-bit hash function: FORK-256," in M.J.B. Robshaw, ed., *Fast Software Encryption (FSE 2006)*, Lect. Notes Comput. Sci., vol.4047, pp.195–209, Berlin, Heidelberg, New York, 2006.
- [13] International Organization for Standardization, ISO/IEC 10118-3:2004, Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions, 2004.
- [14] J. Kap, Test Cases for HMAC-RIPEMD160 and HMAC-RIPEMD128, Internet Engineering Task Force (IETF), RFC 2286, 1998, <http://www.ietf.org/rfc/rfc2286.txt>
- [15] A. Keromyti and N. Provos, The Use of HMAC-RIPEMD-160-96 within ESP and AH, Internet Engineering Task Force (IETF), RFC 2857, 2001, <http://www.ietf.org/rfc/rfc2857.txt>
- [16] M. Lamberger and F. Mendel, "Higher-order differential attack on reduced SHA-256," *Cryptology ePrint Archive*, Report 2011/037, 2011, <http://eprint.iacr.org/2011/037>
- [17] F. Landelle and T. Peyrin, "Cryptanalysis of full ripemd-128," in T. Johansson and P.Q. Nguyen, eds., *EUROCRYPT*, Lect. Notes Comput. Sci., vol.7881, pp.228–244, Springer, 2013.
- [18] The Legion of the Bouncy Castle, Bouncy Castle Crypto APIs, <http://www.bouncycastle.org/>
- [19] F. Mendel, T. Nad, S. Scherz, and M. Schläffer, "Differential attacks on reduced RIPEMD-160," in D. Gollmann and F.C. Freiling, eds., *ISC*, Lect. Notes Comput. Sci., vol.7483, pp.23–38, Springer, 2012.
- [20] F. Mendel, T. Nad, and M. Schläffer, "Collision attacks on the reduced dual-stream hash function RIPEMD-128," in A. Canteaut, ed., *FSE*, Lect. Notes Comput. Sci., vol.7549, pp.226–243, Springer, 2012.
- [21] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen, "On the collision resistance of RIPEMD-160," in S.K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, eds., *Information Security*, 9th International Conference; ISC 2006, Lect. Notes Comput. Sci., vol.4176, pp.101–116, Berlin, Heidelberg, New York, 2006.
- [22] F. Mendel, C. Rechberger, M. Schläffer, and S.S. Thomsen, "The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl," in O. Dunkelman, eds., *Fast Software Encryption 16th International Workshop, FSE 2009*, Lect. Notes Comput. Sci., vol.5665, pp.260–276, Springer-Verlag, Berlin, Heidelberg, New York, 2009.

- [23] C. Ohtahara, K. Okada, Y. Sasaki, and T. Shimoyama, "Preimage attacks on full-ARIRANG: Analysis of DM-mode with middle feed-forward," in S. Jung and M. Yung, eds., WISA 2011, Lect. Notes Comput. Sci., vol.7115, pp.40–54, Springer-Verlag, Berlin, Heidelberg, New York, 2012.
- [24] C. Ohtahara, Y. Sasaki, and T. Shimoyama, "Preimage attacks on step-reduced RIPEMD-128 and RIPEMD-160," in X. Lai, M. Yung, and D. Lin, eds., Inscrypt 2010, Lect. Notes Comput. Sci., vol.6584, pp.169–186, Springer-Verlag, Berlin, Heidelberg, New York, 2011.
- [25] The OpenSSL Project, crypto — OpenSSL cryptographic library, <http://www.openssl.org/docs/crypto/ripemd.html>
- [26] RIPE Integrity Primitives, Berlin, Heidelberg, New York, Integrity Primitives for Secure Information Systems, Final RIPE Report of RACE Integrity Primitives Evaluation, RIPE-RACE 1040, 1995.
- [27] Y. Sasaki, "Boomerang distinguishers on MD4-family: First practical results on full 5-pass HAVAL," in A. Miri and S. Vaudenay, eds., Selected Areas in Cryptography SAC 2011, Lect. Notes Comput. Sci., vol.7118, pp.1–18, Springer-Verlag, Berlin, Heidelberg, New York, 2011.
- [28] Y. Sasaki and K. Aoki, "Finding preimages in full MD5 faster than exhaustive search," in A. Joux, ed., Advances in Cryptology — EUROCRYPT 2009, Lect. Notes Comput. Sci., vol.5479, pp.134–152, Springer-Verlag, Berlin, Heidelberg, New York, 2009.
- [29] Y. Sasaki and K. Aoki, "Meet-in-the-middle preimage attacks on double-branch hash functions: Application to RIPEMD and others," in C. Boyd and J.G. Nieto, eds., Information Security and Privacy, 14th Australasian Conference, ACISP 2009, Lect. Notes Comput. Sci., vol.5594, pp.214–231, Springer-Verlag, Berlin, Heidelberg, New York, 2009.
- [30] Y. Sasaki and L. Wang, "Distinguishers beyond three rounds of the ripemd-128/-160 compression functions," in F. Bao, P. Samarati, and J. Zhou, eds., ACNS, Lect. Notes Comput. Sci., vol.7341, pp.275–292, Springer, 2012.
- [31] Technische Universität Darmstadt, FlexiProvider, <http://www.flexiprovider.de/>
- [32] U.S. Department of Commerce, National Institute of Standards and Technology. Federal Register /vol.72, no. 212/Friday, Nov. 2, 2007/Notices, 2007. http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf
- [33] D. Wagner, "The boomerang attack," in L.R. Knudsen, ed. Fast Software Encryption 1999, Lect. Notes Comput. Sci., vol.1636, pp.156–170, Berlin, Heidelberg, New York, 1999.
- [34] D. Wagner, "A generalized birthday problem," in M. Yung, ed., Advances in Cryptology — CRYPTO 2002, Lect. Notes Comput. Sci., vol.2442, pp.288–303, Springer-Verlag, Berlin, Heidelberg, New York, 2002.
- [35] L. Wang, Y. Sasaki, W. Komatsubara, K. Ohta, and K. Sakiyama, "(Second) preimage attacks on step-reduced RIPEMD/RIPEMD-128 with a new local-collision approach," in A. Kiayias, ed., Topics in Cryptology — CT-RSA 2011, Lect. Notes Comput. Sci., vol.6558, pp.197–212, Springer-Verlag, Berlin, Heidelberg, New York, 2011.
- [36] X. Wang, Y. Lisa Yin, and H. Yu, "Finding collisions in the full SHA-1," in V. Shoup, ed., Advances in Cryptology — CRYPTO 2005, Lect. Notes Comput. Sci., vol.3621, pp.17–36, Springer-Verlag, Berlin, Heidelberg, New York, 2005.
- [37] X. Wang and H. Yu, "How to break MD5 and other hash functions," in R. Cramer, ed., Advances in Cryptology — EUROCRYPT 2005, Lect. Notes Comput. Sci., vol.3494, pp.19–35, Springer-Verlag, Berlin, Heidelberg, New York, 2005.

Appendix: Experiment Results

Table A·1 4-sum on 46-steps and partial 2-dimension sum on 48-steps of RIPEMD-128.

H_i^1	0x400268ec;	0x159b2e00	0x 6a66026;	0x268c3594;
M_i^1	0x7b69e00f;	0x7a1da89e;	0xb6760e61;	0x222860d3;
	0x20d0343c;	0x36333ccb;	0x44a67388;	0x9034d3f9;
	0x19810c83;	0x9cba7f1e;	0x43aa3459;	0xe8987de1;
	0xf48cfeb0;	0x36b56404;	0xca71b589;	0x4e4956b3;
46-Step H_{i+1}^1	0x8e492929;	0x34c37860;	0x085981da;	0x3a28780d;
3-Round H_{i+1}^1	0xf57d1452;	0x00cc6f47;	0x9c7fe2e0;	0x5cdae22d;
H_i^2	0x400269ec;	0x159b2e00;	0x86a76026;	0xa68d3594;
M_i^2	0x7b69df0f;	0x7a1da89e;	0xb6760e61;	0x222860d3;
	0x20d0343c;	0x36333ccb;	0xc4a67388;	0x9034d3f9;
	0x19810c83;	0x9cba7f1e;	0x43aa3459;	0xe8987de1;
	0xf48cfeb0;	0x36b56404;	0xca71b589;	0x4e4956b3;
46-Step H_{i+1}^2	0x4b37a7fb;	0xe0a9ebf0;	0x09e98a18;	0x17b730cd;
3-Round H_{i+1}^2	0x672c3692;	0x5e5c2707;	0xe9e5bbda;	0xc6e4b82a;
H_i^3	0x401268ec;	0x159b2e00;	0x06a66026;	0x268c35d4;
M_i^3	0x7b69e00f;	0x7a1da89e;	0xb6760e61;	0x222860d3;
	0x20d0343c;	0x36333ccb;	0x04a67388;	0x9034d3f9;
	0x19810c83;	0x9cba7f1e;	0x43aa3459;	0xe8987de1;
	0xf48cfeb0;	0x36b56404;	0xca71b589;	0x4e4956b3;
46-Step H_{i+1}^3	0xd2ae4d5e;	0x5b495822;	0x13ac118a;	0x9c22aa6a;
3-Round H_{i+1}^3	0x5955db12;	0x62b6a1a4;	0xe0a50755;	0xa0eb49c4;
H_i^4	0x401269ec;	0x159b2e00;	0x86a76026;	0xa68d35d4;
M_i^4	0x7b69df0f;	0x7a1da89e;	0xb6760e61;	0x222860d3;
	0x20d0343c;	0x36333ccb;	0x84a67388;	0x9034d3f9;
	0x19810c83;	0x9cba7f1e;	0x43aa3459;	0xe8987de1;
	0xf48cfeb0;	0x36b56404;	0xca71b589;	0x4e4956b3;
46-Step H_{i+1}^4	0x8f9ccc30;	0x072fcbb2;	0x153c19c8;	0x79b1632a;
3-Round H_{i+1}^4	0xcb04f456;	0xc0465964;	0x2e0ae04f;	0x0afb77c2;
46-Step 4-sum	0x00000000;	0x00000000;	0x00000000;	0x00000000;
3-Round 4-Sum	0xfffff704;	0x00000000;	0x00000000;	0x00065801;

Table A·2 38-step 4-sum and 40-step partial 2D sum on RIPEMD-160 from 1st round.

H_i^1	0x4144c3a7;	0x8a965cea;	0x647e4d03;	0x04e7a03c;	0x18814c3e;
M_i^1	0x15f04e2b; 0xbff66c59; 0xab607923; 0x7297f9d4;	0xb2c328cd; 0x399570f1; 0x712bc2a1; 0xe2c4573f;	0x8eea7e12; 0x997d1fd4; 0x32c11766; 0x96dc27dc;	0xa1d55a25; 0xea8f403e; 0xafaf4bfe; 0x566d9d73;	
38 Steps H_{i+1}^1	0x33d3fb92;	0x753e7a17;	0x50fb34b1;	0x1874be98;	0x48de951c;
H_i^2	0x4146bfa7;	0x8a965cea;	0x647e4d03;	0x04e7a43c;	0x08814c3e;
M_i^2	0x15f04e2b; 0xbff66c59; 0xab607923; 0x7297f9d4;	0xb2c328cd; 0x399570f1; 0x712bc2a1; 0xe2c4573f;	0x8eea7a12; 0x997d1fd4; 0x32c11766; 0x96dc27dc;	0xa1d55a25; 0xea8f403e; 0xafaf4bfe; 0x566d9d73;	
38 Steps H_{i+1}^2	0x07e43a48;	0x1482c47c;	0x473df79e;	0xefed372c;	0x55037e85;
H_i^3	0x404cc5a7;	0x8a9e5cea;	0x647e4c03;	0x04e7a23c;	0x18814c3e;
M_i^3	0x15f04e2b; 0xbff66c59; 0xab607923; 0x7297f9d4;	0xb2c328cd; 0x399570f1; 0x712bc2a1; 0xe2c4573f;	0x8eea8212; 0x997d1fd4; 0x32c11766; 0x96dc27dc;	0xa1d55a25; 0xea8f403e; 0xafaf4bfe; 0x566d9d73;	
38 Steps H_{i+1}^3	0xad046562;	0x61c2166a;	0x9d7dc2b3;	0x901e2f34;	0x12d8aa5c;
H_i^4	0x404ec1a7;	0x8a9e5cea;	0x647e4c03;	0x04e7a63c;	0x08814c3e;
M_i^4	0x15f04e2b; 0xbff66c59; 0xab607923; 0x7297f9d4;	0xb2c328cd; 0x399570f1; 0x712bc2a1; 0xe2c4573f;	0x8eea7e12; 0x997d1fd4; 0x32c11766; 0x96dc27dc;	0xa1d55a25; 0xea8f403e; 0xafaf4bfe; 0x566d9d73;	
38 Steps H_{i+1}^4	0x8114a418;	0x010660cf;	0x93c085a0;	0x6796a7c8;	0x1efd93c5;
38 Steps 4-sum	0x00000000;	0x00000000;	0x00000000;	0x00000000;	0x00000000;
40 Steps 4-Sum	noisy data	noisy data	0x00000000;	0x00000000;	noisy data

Table A·3 40-step 2D sum and 42-step partial 2D sum on RIPEMD-160 from 2nd round.

H_i^1	0x4d5d6030;	0x9a35fc23;	0x05f928c8;	0xa8310281;	0x4130be1e;
M_i^1	0xa067bf68; 0xe6ddd4e5; 0x3f474bb8; 0x9834c195;	0x24062cba; 0xa07f9fb5; 0x7dad7ea3; 0xf04d8c7e;	0x1fd1ec79; 0x383b2647; 0xa1ae1c49; 0xf49b5db9;	0x71c986a3; 0xc52d0687; 0xd8a4ce49; 0xe1d2e0bc;	
40 Steps H_{i+1}^1	0x3835744e;	0x4e2b4c05;	0xe4e4a765;	0xa4ac81eb;	0x51aab742;
H_i^2	0x4d5d603d;	0x8a35fc23;	0x05f928c8;	0xa8318281;	0x4130b81e;
M_i^2	0xa067bf68; 0xe6ddd4e5; 0x3f474bb8; 0x9835c195;	0x24062cba; 0xa07f9fb5; 0x7dad7ea3; 0xf04d8c7e;	0x1fd1ec79; 0x383b2647; 0xa1ae1c49; 0xf49b5db9;	0x71c986a3; 0xc52d0687; 0xd8a4ce49; 0xe1d2e0bc;	
40 Steps H_{i+1}^2	0xabeb44b4;	0xe3837f1b;	0xc3749168;	0xa4c998be;	0xc9469bfe;
H_i^3	0x497d6030;	0x9a35fc23;	0x05f828c8;	0xa8310a81;	0x0130be1e;
M_i^3	0xa067bf68; 0xe6ddd4e5; 0x3f474bb8; 0x9834c195;	0x24062cba; 0xa07f9fb5; 0x7dad7ea3; 0xf04dac7e;	0x1fd1ec79; 0x383b2647; 0xa1ae1c49; 0xf49b5db9;	0x71c986a3; 0xc52d0687; 0xd8a4ce49; 0xe1d2e0bc;	
40 Steps H_{i+1}^3	0x794898eb;	0x43abcddc;	0x1475d80b;	0xa9ee7d07;	0xf6f6119f;
H_i^4	0x497d603d;	0x8a35fc23;	0x05f828c8;	0xa8318a81;	0x0130b81e;
M_i^4	0xa067bf68; 0xe6ddd4e5; 0x3f474bb8; 0x9835c195;	0x24062cba; 0xa07f9fb5; 0x7dad7ea3; 0xf04dac7e;	0x1fd1ec79; 0x383b2647; 0xa1ae1c49; 0xf49b5db9;	0x71c986a3; 0xc52d0687; 0xd8a4ce49; 0xe1d2e0bc;	
40 Steps H_{i+1}^4	0xecfe6951;	0xd90400f4;	0xf305c20e;	0xaa0b93da;	0x6e91f65b;
40 Steps 4-sum	0x00000000;	0x00000000;	0x00000000;	0x00000000;	0x00000000;
42 Steps 4-Sum	noisy data	noisy data	0x00000000;	0x00000000;	noisy data

Yu Sasaki received the B.E., M.E. and Ph.D. from The University of Electro-Communications in 2005, 2007, and 2010. Since 2007, he has been a researcher at NTT Secure Platform Laboratories. His current research interests are in cryptography. He was awarded a paper prize from SCIS 2007 and IEICE Trans. in 2009. He also received a best paper award from IWSEC 2009, SECRIPT 2012, and IWSEC 2012.

Lei Wang received the M.E. and Ph.D. from The University of Electro-Communications in 2009, and 2011, respectively. His current research interests are in cryptography. He was awarded a paper prize from SCIS 2008 and IEICE Trans. in 2009. He also received a best paper award from IWSEC 2009 and IWSEC 2012.