2-Dimension Sums: Distinguishers Beyond Three Rounds of RIPEMD-128 and RIPEMD-160*

Yu Sasaki[1] and Lei Wang[2]

[1] NTT Information Sharing Platform Laboratories, NTT Corporation
[2] The University of Electro-Communications

Abstract. This paper presents differential-based distinguishers against ISO standard hash functions RIPEMD-128 and RIPEMD-160. The compression functions of RIPEMD-128/-160 adopt the double-branch structure, which updates a chaining variable by computing two functions and merging their outputs. Due to the double size of the internal state and difficulties of controlling two functions simultaneously, only few results were published before. In this paper, second-order differential paths are constructed on reduced RIPEMD-128 and -160. This leads to a practical 4-sum attack on 47 steps (out of 64 steps) of RIPEMD-128 and 40 steps (out of 80 steps) of RIPEMD-160. We then extend the distinguished property from the 4-sum to other properties, which we call *a 2-dimension sum* and *a partial 2-dimension sum*. As a result, the practical partial 2-dimension sum is generated on 48 steps of RIPEMD-128 and 42 steps of RIPEMD-160, with a complexity of 2^{35} and 2^{36}, respectively. Theoretically, 2-dimension sums are generated faster than the exhaustive search up to 52 steps of RIPEMD-128 and 51 steps of RIPEMD-160, with a complexity of 2^{101} and 2^{158}, respectively. The practical attacks are implemented, and examples of generated (partial) 2-dimension sums are presented.

Keywords: RIPEMD-128, RIPEMD-160, double-branch structure, N-dimension sum, distinguisher

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.* [21, 22], cryptographers have looked for stronger hash function designs. While various new designs are discussed in the SHA-3 competition [16], some of existing hash functions seem to have much higher security than MD4-family. Evaluating the security of such hash functions are important especially if they are standardized internationally.

RIPEMD-128 and RIPEMD-160 designed by [4] are hash functions standardized by ISO [7]. They are designed to have the same digest size as MD5 and SHA-1 so that MD5 and SHA-1 can be replaced with them. In addition, because SHA-3 will not produce 128- and 160-bit digests, RIPEMD-128 and -160 might be used in the future to provide secure 128- and 160-bit digests.

RIPEMD-128 and -160 adopt the narrow-pipe Merkle-Damgård structure. Their compression functions 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 compression functions $\mathrm{CF}^{\mathrm{L}}(H_{i-1}, M_{i-1})$ and $\mathrm{CF}^{\mathrm{R}}(H_{i-1}, M_{i-1})$. The chaining variable output H_i is computed by merging H_{i-1}, $\mathrm{CF}^{\mathrm{L}}(H_{i-1}, M_{i-1})$, and $\mathrm{CF}^{\mathrm{R}}(H_{i-1}, M_{i-1})$. Due to the double size of the internal state and the difficulties of controlling the two functions simultaneously, only few results were published before. Note that, in order to prevent the recent meet-in-the-middle preimage attacks [5, 14, 15], some hash functions adopt a structure, which applies the feed-forward function several times, e.g. ARIRANG [3]. Sasaki pointed

* The manuscript was submitted to FSE 2012. No change from the submitted manuscript is made in this version. The review comments will be reflected in the future revised version.

out that such a structure can also be viewed as the double-branch structure [13]. Hence, the analysis on the double-branch structure seems useful even for the future hash function design.

RIPEMD-128 produces 128-bit digests and its compression function consists of 4 rounds, 64 steps. RIPEMD-160 produces 160-bit digests and its compression function consists of 5 rounds, 80 steps. Mendel *et al.* investigated the differential property of the compression functions of RIPEMD-128 and -160 [9]. They applied the linear approximation to the compression functions of RIPEMD-128 and -160, and showed that low Hamming weight differential paths for pseudo-near-collisions exist up to 3 rounds (48 steps) for RIPEMD-128 and up to some intermediate step in the third round (steps 33 – 48) for RIPEMD-160. Although [9] is useful to obtain some intuition on collision attacks against RIPEMD-128 and -160, a lot of work is necessary to complete the attacks and the complexity and even the possibility of the attacks are unclear. More details will be discussed in Sect. 3. Another previous work is the ones by Ohtahara *et al.* [10] and Wang *et al.* [19], which investigated preimage attacks on RIPEMD-128 and -160. [10] showed that the first 33 steps of RIPEMD-128 and the first 31 steps of RIPEMD-160 can be attacked while [19] showed that the intermediate 35 steps of RIPEMD-128 can be attacked. Their complexities are very close to the one by the brute force attack.

In this paper, boomerang type differential properties are discussed. The boomerang attack was first proposed by Wagner as a tool for analyzing block ciphers [17]. It divides the entire 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 the 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. Recently, several researchers have applied this property on compression functions so as to mount distinguishers [1, 2, 8, 12]. [12] showed the framework to attack the MD4-family (using the single-branch structure) consisting of up to 5 rounds.

Our Contributions

In this paper, differential-based distinguishers against the compression functions of RIPEMD-128 and RIPEMD-160 are presented.

The first target of this research is the 4-sum property. The 4-sum indicates 4 different inputs where the XOR sum of the corresponding outputs is 0. The current best generic attack to find 4-sums is the generalized birthday attack [18], which requires $2^{n/3}$ computations for n-bit output. Then, the 4-sum property is extended to a new differential property, which we call *a 2-dimension sum*, or more generally *an N-dimension sum*. The current best generic attack to find 2-dimension sums requires 2^n computations. Note that the partial 4-sum and partial N-dimension sum can be introduced naturally.

Then, differential paths are constructed on reduced RIPEMD-128 and -160. The differential path construction is based on the idea of the boomerang distinguisher against the MD4-family by Sasaki [12]. Our strategy is regarding CF^{L} as the first part of the boomerang attack (E_0) and CF^{R} as the second part of the boomerang attack (E_1), hence the entire function (E) consists of 8 and 10 rounds for RIPEMD-128 and -160, respectively. This simplifies the differential path construction because the differential paths for CF^{L} and CF^{R} can be analyzed almost independently. However, because the framework by [12] can only work up to 5 rounds, several improvements are necessary to maximize the number of attacked rounds.

On RIPEMD-128, to construct differential paths, we combine the local collision with the framework by [12]. This leads to a long differential path which is satisfied with a high probability. As

Table 1. A summary of the attack on the compression functions

Target	#Steps	Property	Complexity	Reference	Remarks
RIPEMD-128	33	preimage	2^{119}	[10]	
(64 steps in total)	36	preimage	2^{123}	[19]	starting from an intermediate step
	46	4-sum	2^{34}	Ours	
	47	4-sum	2^{39}	Ours	
	48	2-dimension sum	2^{48}	Ours	
	48	partial 2-dimension sum	2^{35}	Ours	
	52	2-dimension sum	2^{101}	Ours	
RIPEMD-160	31	preimage	2^{148}	[10]	
(80 steps in total)	38	4-sum	2^{42}	Ours	
	40	partial 2-dimension sum	2^{42}	Ours	
	43	2-dimension sum	2^{151}	Ours	
	40	4-sum	2^{36}	Ours	starting from the second round
	42	partial 2-dimension sum	2^{36}	Ours	starting from the second round
	51	2-dimension sum	2^{158}	Ours	starting from the second round

a result, 4-sums and partial 2-dimension sums are generated with a practical complexity up to 47 steps and 48 steps of RIPEMD-128, respectively. In addition, 2-dimension sums are theoretically generated faster than the brute force attack up to 52 steps of RIPEMD-128.

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 property of the non-linear differential path 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 steps and 40 steps of RIPEMD-160, respectively. In addition, with an infeasible complexity, 2-dimension sums are generated faster than the brute force attack up to 43 steps of RIPEMD-160. 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. The attack results are summarized in Table 1.

Paper Outline. In Sect. 2, the specification of RIPEMD-128 and -160 are explained. In Sect. 3, related work is summarized. In Sect. 4, a new differential property called 2-dimension sum is introduced. In Sect. 5, attacks on RIPEMD-128 are explained. In Sect. 6, attacks on RIPEMD-160 are explained. Finally, we conclude this paper in Sect. 7.

2 Specifications

RIPEMD-128 and RIPEMD-160 are hash functions proposed by Dobbertin *et al.* [4] as stronger hash functions than RIPEMD [11], which take a message of arbitrary length as input and produce a 128-bit and a 160-bit hash digests, respectively. Because our attack target is their compression functions, we omit the description of their domain extensions. The differences between RIPEMD-128 and RIPEMD-160 are the digest size and the compression function structure. In the following sections, the compression functions are described.

2.1 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 $m_0, m_1, \ldots, m_{14}$ and 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.

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

where SF_j^L and SF_j^R are step functions for $\mathrm{CF^L}$ and $\mathrm{CF^R}$ respectively, and perform the following computations.

$$\begin{aligned} a_{j+1}^L &\leftarrow d_j^L, & a_{j+1}^R &\leftarrow d_j^R, \\ b_{j+1}^L &\leftarrow (a_j^L + f_j(b_j^L, c_j^L, d_j^L) + m_{\pi^L(j)} + k_j^L) \lll s_j^L, & b_{j+1}^R &\leftarrow (a_j^R + f_{63-j}(b_j^R, c_j^R, d_j^R) + m_{\pi^R(j)} + k_j^R) \lll s_j^R, \\ c_{j+1}^L &\leftarrow b_j^L, & a_{j+1}^R &\leftarrow d_j^R, \\ d_{j+1}^L &\leftarrow c_j^L, & a_{j+1}^R &\leftarrow d_j^R. \end{aligned}$$

'+' represents the addition on modulo 2^{32}, '$\lll x$' represents left cyclic shift by x bits, f_x is a Boolean function, π^L and π^R are the message expansion, and k_j^L and k_j^R are the constant. Details of these values are listed in Tables 2.

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

2.2 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.

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

Most of the parameters are shared with RIPEMD-128. The details are described in Table 2. In the computations of SF^R, b_{j+1}^R are computed as follows;

$$b_{j+1}^R \leftarrow ((a_j^R + f_{79-j}(b_j^R, c_j^R, d_j^R) + m_{\pi^R(j)} + k_j^R) \lll s_j^R) + e_j^R.$$

The other variables $a_{j+1}^R, c_{j+1}^R, d_{j+1}^R$, and e_{j+1}^R are computed similarly to SF^L. Finally, the output chaining variable H_i is computed as shown in Table 3.

Table 2. Boolean functions, message expansions, rotation numbers of RIPEMD-128 and RIPEMD-160

	$f_x(X,Y,Z)$
$x = 0, 1, \ldots, 15$	$X \oplus Y \oplus Z$
$x = 16, 17, \ldots, 31$	$(X \wedge Y) \vee (\neg X \wedge Z)$
$x = 32, 33, \ldots, 47$	$(X \vee \neg Y) \oplus Z$
$x = 48, 49, \ldots, 63$	$(X \wedge Z) \vee (Y \wedge \neg Z)$
$x = 64, 65, \ldots, 79$	$X \oplus (Y \vee \neg Z)$

	$\pi^L(j)$																$\pi^R(j)$															
$j = 0, 1, \ldots, 15$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	5	14	7	0	9	2	11	4	13	6	15	8	1	10	3	12
$j = 16, 17, \ldots, 31$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8	6	11	3	7	0	13	5	10	14	15	8	12	4	9	1	2
$j = 32, 33, \ldots, 47$	3	10	14	4	9	15	8	1	2	7	0	6	13	11	5	12	15	5	1	3	7	14	6	9	11	8	12	2	10	0	4	13
$j = 48, 49, \ldots, 63$	1	9	11	10	0	8	12	4	13	3	7	15	14	5	6	2	8	6	4	1	3	11	15	0	5	12	2	13	9	7	10	14
$j = 64, 65, \ldots, 79$	4	0	5	9	7	12	2	10	14	1	3	8	11	6	15	13	12	15	10	4	1	5	8	7	6	2	13	14	0	3	9	11

	s_j^L																s_j^R															
$j = 0, 1, \ldots, 15$	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8	8	9	9	11	13	15	15	5	7	7	8	11	14	14	12	6
$j = 16, 17, \ldots, 31$	7	6	8	13	11	9	7	15	7	12	15	9	11	7	13	12	9	13	15	7	12	8	9	11	7	7	12	7	6	15	13	11
$j = 32, 33, \ldots, 47$	11	13	6	7	14	9	13	15	14	8	13	6	5	12	7	5	9	7	15	11	8	6	6	14	12	13	5	14	13	13	7	5
$j = 48, 49, \ldots, 63$	11	12	14	15	14	15	9	8	9	14	5	6	8	6	5	12	15	5	8	11	14	14	6	14	6	9	12	9	12	5	15	8
$j = 64, 65, \ldots, 79$	9	15	5	11	6	8	13	12	5	12	13	14	11	8	5	6	8	5	12	9	12	5	14	6	8	13	6	5	15	13	11	11

	RIPEMD-128		RIPEMD-160	
	k_j^L	k_j^R	k_j^L	k_j^R
$j = 0, 1, \ldots, 15$	`0x00000000`	`0x50a28be6`	`0x00000000`	`0x50a28be6`
$j = 16, 17, \ldots, 31$	`0x5a827999`	`0x5c4dd124`	`0x5a827999`	`0x5c4dd124`
$j = 32, 33, \ldots, 47$	`0x6ed9eba1`	`0x6d703ef3`	`0x6ed9eba1`	`0x6d703ef3`
$j = 48, 49, \ldots, 63$	`0x8f1bbcdc`	`0x00000000`	`0x8f1bbcdc`	`0x7a6d76e9`
$j = 64, 65, \ldots, 79$	-	-	`0xa953fd4e`	`0x00000000`

3 Related Work

At ISC 2006, Mendel *et al.* presented differential properties of linearized RIPEMD-128 and RIPEMD-160 [9]. Their work can be summarized as follows;

1. Replace the addition operation with the XOR operation.
2. Replace all Boolean functions with the XOR of three input words.
3. Ignore the first round under the assumption that any differential path can be satisfied by using the message modification technique [21].
4. Find a path which has a low Hamming weight on chaining variable a_j where $j > 16$, by using algorithms from coding theory.

Table 3. Computations for the Output of the Compression Function

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

5. Estimate the attack complexity up to step x as $2^{2 \cdot \Sigma_{j=17}^{x} HW(a_j)}$.

As a result, the authors discovered a near-pseudo-collision differential path up to 3 rounds (48 steps) for RIPEMD-128 and RIPEMD-160 with the Hamming weight of 18 and 224, respectively. According to their counting method, these indicate that near-pseudo-collisions might be found up to round 3 with 2^{36} computations on RIPEMD-128 and 2^{448} computations on RIPEMD-160. This indicates that attacking 3 rounds of RIPEMD-160 would be impossible. Note that as the authors mentioned [9, Sect.3.2], the estimation of the attack complexity is quite conservative. The final attack complexity might be higher in practice.

Although [9] is useful to obtain the intuition for the differential attack on RIPEMD-128/-160, there are many gaps between their estimation and attacks in practice.

- Differential path construction for the first round is non-trivial.
- Wang *et al.* pointed out the difficulties of applying the message modification to the double branch structure [20], and thus it is unclear that both of the paths in the first rounds of CF^{L} and CF^{R} can be satisfied efficiently,
- Difference propagations in the linearized mode and in the original mode are different. For example, let Δ_1 be $0x01$ and Δ_2 be $0x01$. $\Delta_1 \oplus \Delta_2$ is always $0x00$, but $\Delta_1 + \Delta_2$ might be $0x10$ with a probility $1/2$.
- Several Boolean functions do not behave as linear, e.g., $f_{16}(\texttt{0x00}, \texttt{0x01}, \texttt{0x01})$ is always $\texttt{0x01}$.
- [9] only gives the minimum Hamming weight. Other crucial information such as message differences is not available.

In summary, [9] tells that the differential attack on 3 rounds of RIPEMD-160 would be impossible. However, only from the results of [9], it is hard to guess the actual attack complexity or even the possibility of the attack on reduced RIPEMD-128/160.

4 Differential Properties to be Distinguished

In this section, we give a summary and introduce a new differential property to be used to distinguish the target compression function from the ideal function. Because our attacks are based on the boomerang attack, differential properties with 2 differences are mainly discussed.

4.1 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 $\mathrm{CF}(I_0) \oplus \mathrm{CF}(I_1) \oplus \mathrm{CF}(I_2) \oplus \mathrm{CF}(I_3) = 0$. The current best generic attack to find a 4-sum is a generalized birthday attack [18], which requires $2^{n/3}$ computations and $2^{n/3}$ memory for n-bit output. Hence, if 4-sums are generated faster than $2^{n/3}$ computations, CF is not regarded as ideal.

It is possible to limit the form of input values on 4-sum properties. For example, the problem is changed to finding a set of 4 different inputs (I_0, I_1, I_2, I_3), where two pairs have a pre-specified difference Δ and the sum of the corresponding outputs is 0. Namely, $I_0 \oplus I_1 = I_2 \oplus I_3 = \Delta$ and $\mathrm{CF}(I_0) \oplus \mathrm{CF}(I_1) \oplus \mathrm{CF}(I_2) \oplus \mathrm{CF}(I_3) = 0$. As far as we know, the current best generic attack for this problem requires $2^{n/2}$ computations;

1. Choose $2^{n/2}$ different I_0s. Compute $I_1 \leftarrow I_0 \oplus \Delta$ and then compute $\mathrm{CF}(I_0) \oplus \mathrm{CF}(I_1)$.
2. Choose $2^{n/2}$ different I_2s. Compute $I_3 \leftarrow I_2 \oplus \Delta$ and then compute $\mathrm{CF}(I_2) \oplus \mathrm{CF}(I_3)$.
3. Find a collision between $\mathrm{CF}(I_0) \oplus \mathrm{CF}(I_1)$ and $\mathrm{CF}(I_2) \oplus \mathrm{CF}(I_3)$.

Therefore, if such 4 inputs are generated faster than $2^{n/2}$ computations, CF is not regarded as ideal.

4.2 2-Dimension Sums and Suitability for the Double-Branch Structure

We introduce a new differential property, which we call *2-dimension sum*. In this notion, we set more limitations to the form of input values on 4-sum properties. The problem is changed to finding a set of 4 different inputs (I_0, I_1, I_2, I_3) which satisfy $I_1 = I_0 \oplus \Delta, I_2 = I_0 \oplus \nabla, I_3 = I_0 \oplus \Delta \oplus \nabla$, and $\mathrm{CF}(I_0) \oplus \mathrm{CF}(I_1) \oplus \mathrm{CF}(I_2) \oplus \mathrm{CF}(I_3) = 0$, for two pre-specified differences Δ and ∇. A generic attack for this problem seems to require 2^n computations; choose the value of I_0 and check that the corresponding $\mathrm{CF}(I_0) \oplus \mathrm{CF}(I_0 \oplus \Delta) \oplus \mathrm{CF}(I_0 \oplus \nabla) \oplus \mathrm{CF}(I_0 \oplus \Delta \oplus \nabla)$ is 0.

2-dimension sums are 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 Δ. Then, a pseudo-near-collision path for CF^{R} is independently constructed with setting other difference ∇. If the product of the probability of each path (after message modification) is higher than $2^{-n/2}$, the 2-dimension sum can be generated faster than 2^n by using the boomerang attack approach [12]. Different from the original RIPEMD, RIPEMD-128 and -160 adopt very different functions as CF^{L} and CF^{R}. Therefore, the independence of the path construction for CF^{L} and CF^{R} greatly helps the attacker. More discussion is given in Sect. 5 and 6.

Note that the *partial 2-dimension sum* is naturally introduced, where $\mathrm{CF}(I_0) \oplus \mathrm{CF}(I_0 \oplus \Delta) \oplus \mathrm{CF}(I_0 \oplus \nabla) \oplus \mathrm{CF}(I_0 \oplus \Delta \oplus \nabla)$ 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 faster than 2^d.

4.3 *N*-Dimension Sum for *N*-Branch Hash 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 $(\Delta_1, \Delta_2, \ldots, \Delta_N)$. Let a notation $\Delta[\delta_1\delta_2 \cdots \delta_N]$, where $\delta_x \in \{0, 1\}$ represent $\bigoplus_{x=1}^{N}(\Delta_x \cdot \delta_x)$. The property of the N-dimension sum is defined to finding an input value I such that, for all possible 2^N patterns of $(\delta_1\delta_2 \cdots \delta_N)$, $\bigoplus \mathrm{CF}(I \oplus \Delta[\delta_1\delta_2 \cdots \delta_N]) = 0$. 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 faster than 2^n. Hash function FORK-256 [6] 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.

5 Attacks on RIPEMD-128

We construct 2-dimension sums against the compression function of RIPEMD-128. Hereafter, we compute the difference in modular subtraction because the main operation of RIPEMD-128/-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 Overall Strategy

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

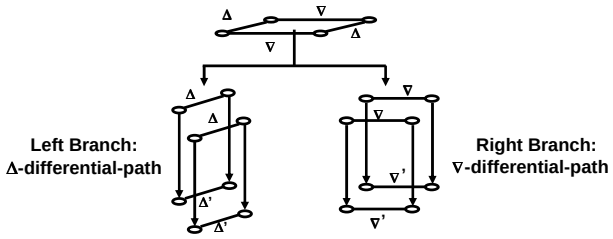

Fig. 1. Our Attack Strategy on RIPEMD-128 and -160

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

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

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.

- Both the difference between (H, M) and $(H + \Delta, M + \Delta^M)$ and the difference between $(H + \nabla, M + \nabla^M)$ and $(H + \nabla + \Delta, M + \nabla^M + \Delta^M)$ follow Δ-differential-path in the left branch.
- Both the difference between (H, M) and $(H + \nabla, M + \nabla^M)$ and the difference 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 following relationship;

$$\mathrm{CF}(H, M) + \mathrm{CF}(H + \Delta + \nabla, M + \Delta^M + \nabla^M) - \mathrm{CF}(H + \Delta, M + \Delta^M) - \mathrm{CF}(H + \nabla, M + \nabla^M) = 0,$$

where CF is the compression function of RIPEMD-128. The proof is simple algebra and is omitted.

5.2 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 no or weak absorption property, which means these f functions produce new differences with a high probability. 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 appear at a very beginning step in the first round and at a very late step in the third round. Therefore the whole differential path consists of 3 sub-paths: a short path at the beginning steps in the first round; a local collision in the second round; and a short path at the late steps in the third round. Such a strategy will maximize the probability of the whole differential path. Finally we choose to use message difference Δ^M as below

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

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

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

The value of Δ' changes with the number of the attacked steps. Moreover, we do not need to specify Δ' according to *amplified probability* using multiple outside differential paths [12].

Remark 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.

5.3 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. In order to achieve it, since the f function in the first round of the right branch has strong absorption property, 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 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 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

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

5.4 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 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 another valid (H, M) by modifying m_{12}. As a summary, the complexity for satisfying the differential path for the first 17 steps 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.

5.5 Complexity Evaluation and Experiments

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

Attack on 46 steps. There are 11 and 8 sufficient conditions in the Δ- and ∇-differential-paths, respectively. Each condition must be satisfied in two pairs and thus its probability is 2^{-2}. However, 2 conditions in the Δ-differential-path are the one discussed in the remarks in Sect. 5.2, which satisfied with probability 2^{-1}. Overall, the complexity is $2^{36(=2\times(9+8)+2)}$. We then experimentally check the amplified probability, and the final complexity is 2^{34}.

Attack on 48 steps. We experimentally checked the amplified probability, and the final complexity is 2^{48}.

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 instead of constructing a specific differential path. As a result, the complexity to obtain a 4-sum is 2^{101}.

The attack was implemented on single PC. The generated 46-step 2-dimension sum, which is also 3-round (48-step) partial 2-dimension sum, is shown in Table. 8 in Appendix.

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, we need the difference in 3 message-words due to the direct addition from chaining variable e_j. Hence, we avoid using the local-collision. Instead, we insert the difference only into 1 message word that appears in a latter 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 an 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.

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

Table 5. Differential Paths for 2-Dimension Sums on 3-Round RIPEMD-128

$*$ means that the $+/-$sign of difference is not specified. Input chaining variable difference for $\mathrm{CF^L}$ is that $\Delta a_0^L = 2^8$, $\Delta b_0^L = 0$, $\Delta c_0^L = *2^{31} + 2^{16}$, and $\Delta d_0^L = *2^{31} + 2^{16}$. Input chaining variable difference for $\mathrm{CF^R}$ is that $\Delta a_0^R = 2^{20}$, $\Delta b_0^R = 0$, $\Delta c_0^R = 0$, and $\Delta d_0^R = 2^6$.

Path for $\mathrm{CF^L}$				Path for $\mathrm{CF^R}$			
j	Δb_j^L	$\Delta m_{\pi^L(j)}$	$\pi^L(j)$	j	∇b_j^R	$\nabla m_{\pi^R(j)}$	$\pi^R(j)$
1		-2^8	0	1	2^{28}		5
2	$*2^{31}$		1	2	2^{15}		14
3	$*2^{31}$		2	3			7
4			3	4			0
5			4	5	2^9		9
6			5	6	2^{30}		2
7		$*2^{31}$	6	7			11
8			7	8			4
9			8	9	2^{16}		13
10			9	10		-2^{30}	6
11			10	11			15
12			11	12			8
13			12	13	2^{30}		1
14			13	14			10
15			14	15			3
16			15	16			12
17			7	17		-2^{30}	6
...	...	...	...	...	...	...	...
22	2^8	$*2^{31}$	6	22			13
23			15	23			5
24			3	24			10
25			12	25			14
26		-2^8	0	26			15
...	...	...	...	...	...	...	...
39			8	39	-2^4	-2^{30}	6
40			1	40			9
41			2	41			11
42			7	42			8
43	-2^{21}	-2^8	0	43	-2^9		12
44	$*2^5$	$*2^{31}$	6	44			2
45			11	45			10
46	$*2^{26}$		13	46			0
47	$-2^{28} * 2^{12}$		5	47	-2^{16}		4
48	$*2^{10}$		12	48			13

Table 6. Sufficient Conditions of Attacks on 3-Round RIPEMD-128

j	Δ Conditions for Step j in CF^{L}	∇ Conditions for Step j in CF^{R}
0	$c^L_{0,16} = 0$, $d_{0,16} = 0$	$b^R_{0,6} = c^R_{0,6}$, $b^R_{0,28} = 1$, $b^R_{0,15} = 0$, $c^R_{0,28} = 0$, $d^R_{0,6} = 0$
1	$b^L_{1,16} = b^L_{0,16}$	$b^R_{1,28} = 0$, $b^R_{1,15} = 1$
2	no carry in b^L_2	$b^R_{2,15} = 0$
3	no carry in b^L_3	$b^R_{3,9} = 0$
4		$b^R_{4,15} = b^R_{3,15}$, $b^R_{4,30} = 0$, $b^R_{4,9} = 1$
5		$b^R_{5,9} = 0$, $b^R_{5,30} = 1$
6		$b_{6,30} = 0$
7		$b^R_{7,9} = b^R_{6,9}$, $b^R_{7,16} = 0$
8		$b^R_{8,30} = b^R_{7,30}$, $b^R_{8,16} = 1$
9		$b^R_{9,16} = 0$
11		$b^R_{11,16} = b^R_{10,16}$
12		$b^R_{12,30} = 0$
13		$b^R_{13,30} = 0$
15		$b^R_{15,30} = b^R_{14,30}$
22	$b^L_{22,8} = 0$	
23	$c^L_{22,8} = d^L_{22,8}$	
24	$b^L_{24,8} = 0$	
25	$b^L_{25,8} = 1$	
39		$b^R_{39,4} = 1$
40		$b^R_{40,4} = 0$, $c^R_{39,4} = d^R_{39,4}$
41		$b^R_{41,4} = 1$
42		$b^R_{42,9} = b^R_{41,9}$
43	$b^L_{43,21} = 1$, $b^L_{43,5} = 0$	$b^R_{43,9} = 1$
44	$d^L_{43,21} = 0$, $b^L_{44,21} = 1$	$b^R_{44,9} = 0$
45	$b^L_{45,26} = 0$, $b^L_{45,5} = 1$	$b^R_{45,9} = 0$
46	$b^L_{46,28} = 0$, $b^L_{46,12} = 0$, no carry in b^L_{46}	$b^R_{46,16} = b^R_{45,16}$
47	$b^L_{47,28} = 1$, $b^L_{47,26} = 1$, no carry in b^L_{47}	$b^R_{47,16} = 1$
48	no carry in b^L_{48}	
	Extra conditions on (H, M) and $(H + \nabla, M + \nabla^M)$	
44	Share the same bit value $b^L_{44,5}$	
46	Share the same bit value $b^L_{46,26}$	
47	Share the same bit value $b^L_{47,12}$	

through the f function in the following 3 steps, only chaining variables e will have 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}, and thus the difference $+2^n$ is always propagated to b_{j+1}. However, 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 position has the difference. In step $j + 1$, if the difference in the n-th bit of b_{j+1} is output through 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. Therefore, $Y = 1$ 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 $(X \wedge Z) \vee (Y \wedge \neg Z)$. In this case, the sign of Δf_{j+1} cannot be opposite of Δe_j, and we need another technique explained in the next paragraph.

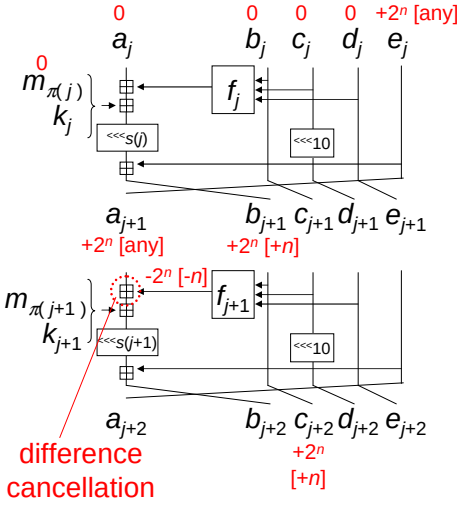

Fig. 2. Difference cancellation between e_j and f_{j+1}. The sign of Δf_{j+1} must be opposite of Δe_j. Information in '[ ]' represents the bitwise difference. '[any]' represents that the bitwise difference is irrelevant.

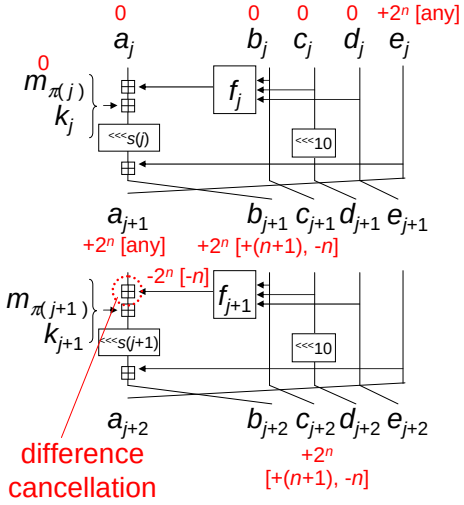

Fig. 3. Difference cancellation with considering the carry effect for the case that the sign of Δf_{j+1} is always the same as Δe_j

Table 7. Differential path construction for the first 3-rounds of RIPEMD-160

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

Cancelling differences with considering the carry effect. This is a technique for the case that Δf_{j+1} is always the same as Δe_j. In step j, we make a carry in b_{j+1} as shown in Fig. 3. Therefore, n-th bit position changes in the opposite direction as Δe_j. Finally, in step $j+1$, by propagating the difference in the n-th bit and by absorbing the difference in the $(n+1)$-th bit, the cancellation occurs.

6.2 Scenario 1: Attack from the First Round

The message differences for the attack starting from the first round is shown in Table 7.

We need to insert both of the Δ-difference and ∇-difference in the 10th bit of m_2. To avoid the contradiction of two paths, the differences and the values of m_2 must be carefully chosen. We choose the following message differences;

$$\Delta m_2 = m_2^2 - m_2^1 = m_2^4 - m_2^3 = +2^{10}, \nabla m_2 = m_2^3 - m_2^1 = m_2^4 - m_2^2 = -2^{10}. \quad (1)$$

To achieve this, we first choose m_2^1 and then compute $m_2^2 \leftarrow m_2^1 + 2^{10}$ and $m_2^3 \leftarrow m_2^1 - 2^{10}$. m_2^4 should be $m_2^1 + 2^{10} - 2^{10}$ and thus identical with m_2^1. Hence, the attack only requires 3 messages

m_2^1, m_2^2, m_2^3 rather than the standard message quartet. It seems strange but the Δ and ∇-differences in (1) are surely satisfied. In fact, our experiment shows that the attack can work even if $m_2^1 = m_2^4$.

The differential paths for CF^{L} and CF^{R}, and their sufficient conditions are shown in Tables 11 and 12 in Appendix.

For the differential path in Table 11, the first round of CF^{L} and CF^{R} can be guaranteed with the message modification technique in negligible time. Hence, the attack cost only depends on the differential path from the second round. The probability of satisfying the differential path up to step 38 is 2^{-42}. Hence, we can generate 4-sums up to 38 steps with 2^{42} computations. Because $2^{42} < 2^{160/3}$, the attack runs faster than the generic 4-sum attack using the generalized birthday attack. This can be regarded as partial 2-dimension sums up to 40 steps because the newly computed values in the last 2 steps are not used to compute two output chaining variables $H_{i-1}^{(b)}$ and $H_{i-1}^{(c)}$. The attack was implemented on a PC. The generated 38-step 4-sum (or 40-step partial 2-dimension sum) is shown in Table 9 in Appendix. Theoretically, 2-dimension sums can be generated up to 43 steps with 2^{151} computations.

6.3 Scenario 2: Attack from the Second Round

The overall strategy is the same as the first scenario. The details of the attack such as the choice of the message difference, differential path construction, and sufficient conditions are optimized for this scenario. The biggest difference from the first scenario is that the f function in the third round (round 4) have the absorption property. Hence, the differential propagation can be controlled more efficiently and this enables us to attack more rounds. Due to the limited space, we only show the data to launch the attack. The message differences, differential paths, and sufficient conditions are shown in Table 13, 14 and 15 in Appendix, respectively. 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. The generated 40-step 2-dimension sum, which can also be regarded as 42-step partial 2-dimension sum, is shown in Table 10 in Appendix.

7 Concluding Remarks

We presented differential-based distinguishers against compression functions of RIPEMD-128 and RIPEMD-160. The 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} almost independently.

On RIPEMD-128, to construct differential paths, we combined the local collision with the framework by [12]. Partial 2-dimension sums were practically generated for 3 rounds (48 steps) of RIPEMD-128. Theoretically, 2-dimension sums are generated faster than the brute force attack up to 52 steps. On RIPEMD-160, we exploited the cancelling property of the differences between Δe_j and Δf_{j+1}. Partial 2-dimension sums were practically generated up to 40 steps of RIPEMD-160. Theoretically, 2-dimension-sum attack can work up to 43 steps. If the attack starts from the second round, partial 2-dimension sums were practically generated up to 42 steps and theoretically, 2-dimension-sum attack can work up to 51 steps.

References

1. Alex Biryukov, Mario Lamberger, Florian Mendel, and Ivica Nikolić. Second-order differential collisions for reduced SHA-256. In *Advances in Cryptology — Asiacrypt 2011*. Springer-Verlag, 2011. to appear.

2. Alex Biryukov, Ivica Nikolić, and Arnab Roy. Boomerang attacks on BLAKE-32. In Antoine Joux, editor, *Fast Software Encryption (FSE) 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 218–237, Berlin, Heidelberg, New York, 2011. Springer-Verlag.
3. Donghoon Chang, Seokhie Hong, Changheon Kang, Jinkeon Kang, Jongsung Kim, Changhoon Lee, Jesang Lee, Jongtae Lee, Sangjin Lee, Yuseop Lee, Jongin Lim, and Jaechul Sung. *ARIRANG*. Available at NIST home page: `http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions\_rnd1.html%`.
4. Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *Fast Software Encryption Third International Workshop (FSE 1996)*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82, Berlin, Heidelberg, New York, 1996.
5. Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. Cryptology ePrint Archive, Report 2010/016, 2010. `http://eprint.iacr.org/`.
6. Deukjo Hong, Donghoon Chang, Jaechul Sung, Sangjin Lee, Seokhie Hong, Jaesang Lee, Dukjae Moon, and Sungtaek Chee. A new dedicated 256-bit hash function: FORK-256. In Matthew J. B. Robshaw, editor, *Fast Software Encryption (FSE 2006)*, volume 4047 of *Lecture Notes in Computer Science*, pages 195–209, Berlin, Heidelberg, New York, 2006.
7. International Organization for Standardization. *ISO/IEC 10118-3:2004, Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions*, 2004.
8. Mario Lamberger and Florian Mendel. Higher-order differential attack on reduced SHA-256. Cryptology ePrint Archive, Report 2011/037, 2011. `http://eprint.iacr.org/2011/037`.
9. Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. On the collision resistance of RIPEMD-160. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference; ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 101–116, Berlin, Heidelberg, New York, 2006.
10. Chiaki Ohtahara, Yu Sasaki, and Takeshi Shimoyama. Preimage attacks on step-reduced RIPEMD-128 and RIPEMD-160. In Xuejia Lai, Moti Yung, and Dongdai Lin, editors, *Inscrypt 2010*, volume 6584 of *Lecture Notes in Computer Science*, pages 169–186, Berlin, Heidelberg, New York, 2011. Springer-Verlag.
11. 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.
12. Yu Sasaki. Boomerang distinguishers on MD4-family: First practical results on full 5-pass HAVAL. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography SAC 2011*, volume 7118 of *Lecture Notes in Computer Science*, pages 1–18, Berlin, Heidelberg, New York, 2011. Springer-Verlag.
13. Yu Sasaki. Recent advances in MitM preimage attacks. Invited Talk at ECRYPT II Hash Workhop 2011, 2011.
14. Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Antoine Joux, editor, *Advances in Cryptology — EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152, Berlin, Heidelberg, New York, 2009. Springer-Verlag.
15. Yu Sasaki and Kazumaro Aoki. Meet-in-the-middle preimage attacks on double-branch hash functions: Application to RIPEMD and others. In Colin Boyd and J. González Nieto, editors, *Information Security and Privacy, 14th Australasian Conference, ACISP 2009*, volume 5594 of *Lecture Notes in Computer Science*, pages 214–231, Berlin, Heidelberg, New York, 2009. Springer-Verlag.
16. U.S. Department of Commerce, National Institute of Standards and Technology. *Federal Register /Vol. 72, No. 212/Friday, November 2, 2007/Notices*, 2007. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.
17. David Wagner. The boomerang attack. In Lars R. Knudsen, editor, *Fast Software Encryption 1999*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170, Berlin, Heidelberg, New York, 1999.
18. David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303, Berlin, Heidelberg, New York, 2002. Springer-Verlag.
19. Lei Wang, Yu Sasaki, Wataru Komatsubara, Kazuo Ohta, and Kazuo Sakiyama. (Second) preimage attacks on step-reduced RIPEMD/RIPEMD-128 with a new local-collision approach. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 197–212, Berlin, Heidelberg, New York, 2011. Springer-Verlag.
20. 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*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, Berlin, Heidelberg, New York, 2005.

21. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, Berlin, Heidelberg, New York, 2005.
22. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In Victor Shoup, editor, *Advances in Cryptology — CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16, Berlin, Heidelberg, New York, 2005. Springer-Verlag.

A Examples of Generated Data and Details of the Attacks on RIPEMD-160

Table 8. An example of 4-sum on 46-steps and partial 2-simension sum on 3-rounds (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}^3	0x8f9ccc30; 0x072fcbb2; 0x153c19c8; 0x79b1632a;
3-Round H_{i+1}^3	0xcb04f456; 0xc0465964; 0x2e0ae04f; 0x0afb77c2;
46-Step 4-sum	0x00000000; 0x00000000; 0x00000000; 0x00000000;
3-Round 4-Sum	0xffffff704; 0x00000000; 0x00000000; 0x00065801;

Table 9. A 38-step 2-dimension sum and 40-step partial 2-dimension sum on RIPEMD-160 from the first round.

H_i^1	0x4144c3a7; 0x8a965cea; 0x647e4d03; 0x04e7a03c; 0x18814c3e;
M_i^1	0x15f04e2b; 0xb2c328cd; 0x8eea7e12; 0xa1d55a25; 0xbff66c59; 0x399570f1; 0x997d1fd4; 0xea8f403e; 0xab607923; 0x712bc2a1; 0x32c11766; 0xafaf4bfe; 0x7297f9d4; 0xe2c4573f; 0x96dc27dc; 0x566d9d73;
38 Steps H_{i+1}^1	0x33d3fb92; 0x753e7a17; 0x50fb34b1; 0x1874be98; 0x48de951c;
H_i^2	0x4146bfa7; 0x8a965cea; 0x647e4d03; 0x04e7a43c; 0x08814c3e;
M_i^2	0x15f04e2b; 0xb2c328cd; 0x8eea7a12; 0xa1d55a25; 0xbff66c59; 0x399570f1; 0x997d1fd4; 0xea8f403e; 0xab607923; 0x712bc2a1; 0x32c11766; 0xafaf4bfe; 0x7297f9d4; 0xe2c4573f; 0x96dc27dc; 0x566d9d73;
38 Steps H_{i+1}^2	0x07e43a48; 0x1482c47c; 0x473df79e; 0xefed372c; 0x55037e85;
H_i^3	0x404cc5a7; 0x8a9e5cea; 0x647e4c03; 0x04e7a23c; 0x18814c3e;
M_i^3	0x15f04e2b; 0xb2c328cd; 0x8eea8212; 0xa1d55a25; 0xbff66c59; 0x399570f1; 0x997d1fd4; 0xea8f403e; 0xab607923; 0x712bc2a1; 0x32c11766; 0xafaf4bfe; 0x7297f9d4; 0xe2c4573f; 0x96dc27dc; 0x566d9d73;
38 Steps H_{i+1}^3	0xad046562; 0x61c2166a; 0x9d7dc2b3; 0x901e2f34; 0x12d8aa5c;
H_i^4	0x404ec1a7; 0x8a9e5cea; 0x647e4c03; 0x04e7a63c; 0x08814c3e;
M_i^4	0x15f04e2b; 0xb2c328cd; 0x8eea7e12; 0xa1d55a25; 0xbff66c59; 0x399570f1; 0x997d1fd4; 0xea8f403e; 0xab607923; 0x712bc2a1; 0x32c11766; 0xafaf4bfe; 0x7297f9d4; 0xe2c4573f; 0x96dc27dc; 0x566d9d73;
38 Steps H_{i+1}^3	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 10. A 40-step 2-dimension sum and 42-step partial 2-dimension sum for RIPEMD-160 from the 2nd round.

H_i^1	0x4d5d6030; 0x9a35fc23; 0x05f928c8; 0xa8310281; 0x4130be1e;
M_i^1	0xa067bf68; 0x24062cba; 0x1fd1ec79; 0x71c986a3; 0xe6ddd4e5; 0xa07f9fb5; 0x383b2647; 0xc52d0687; 0x3f474bb8; 0x7dad7ea3; 0xa1ae1c49; 0xd8a4ce49; 0x9834c195; 0xf04d8c7e; 0xf49b5db9; 0xe1d2e0bc;
40 Steps H_{i+1}^1	0x3835744e; 0x4e2b4c05; 0xe4e4a765; 0xa4ac81eb; 0x51aab742;
H_i^2	0x4d5d603d; 0x8a35fc23; 0x05f928c8; 0xa8318281; 0x4130b81e;
M_i^2	0xa067bf68; 0x24062cba; 0x1fd1ec79; 0x71c986a3; 0xe6ddd4e5; 0xa07f9fb5; 0x383b2647; 0xc52d0687; 0x3f474bb8; 0x7dad7ea3; 0xa1ae1c49; 0xd8a4ce49; 0x9835c195; 0xf04d8c7e; 0xf49b5db9; 0xe1d2e0bc;
40 Steps H_{i+1}^2	0xabeb44b4; 0xe3837f1b; 0xc3749168; 0xa4c998be; 0xc9469bfe;
H_i^3	0x497d6030; 0x9ab5fc23; 0x05f828c8; 0xa8310a81; 0x0130be1e;
M_i^3	0xa067bf68; 0x24062cba; 0x1fd1ec79; 0x71c986a3; 0xe6ddd4e5; 0xa07f9fb5; 0x383b2647; 0xc52d0687; 0x3f474bb8; 0x7dad7ea3; 0xa1ae1c49; 0xd8a4ce49; 0x9834c195; 0xf04dac7e; 0xf49b5db9; 0xe1d2e0bc;
40 Steps H_{i+1}^3	0x794898eb; 0x43abcdde; 0x1475d80b; 0xa9ee7d07; 0xf6f6119f;
H_i^4	0x497d603d; 0x8ab5fc23; 0x05f828c8; 0xa8318a81; 0x0130b81e;
M_i^4	0xa067bf68; 0x24062cba; 0x1fd1ec79; 0x71c986a3; 0xe6ddd4e5; 0xa07f9fb5; 0x383b2647; 0xc52d0687; 0x3f474bb8; 0x7dad7ea3; 0xa1ae1c49; 0xd8a4ce49; 0x9835c195; 0xf04dac7e; 0xf49b5db9; 0xe1d2e0bc;
40 Steps H_{i+1}^3	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

Table 11. Differential paths for 2-dimension sums on RIPEMD-160 in the first scenario.

Input chaining variable difference for CF^{L} is that $\Delta a_0^L = +2^{17} - 2^{10}$, $\Delta b_0^L = 0$, $\Delta c_0^L = 0$, $\Delta d_0^L = +2^{10}$, and $\Delta e_0^L = -2^{28}$. Input chaining variable difference for CF^{R} is that $\Delta a_0^R = -2^{24} + 2^{19} + 2^9$, $\Delta b_0^R = +2^{19}$, $\Delta c_0^R = -2^8$, $\Delta d_0^R = +2^9$, and $\Delta e_0^R = 0$.

Path for CF^{L} with $\Delta m_2 = -2^{10}$				Path for CF^{R} with $\nabla m_2 = +2^{10}$			
j	Δb_j^L	$\Delta m_{\pi^L(j)}$	$\pi^L(j)$	j	∇b_j^R	$\nabla m_{\pi^R(j)}$	$\pi^R(j)$
1			0	1	-2^0		5
2			1	2			14
3		-2^{10}	2	3			7
4			3	4			0
5			4	5			9
6			5	6		$+2^{10}$	2
7			6	7			11
...	...	...	...	...	...	...	...
28			5	28			12
29	-2^{21}	-2^{10}	2	29			4
30			14	30			9
31			11	31		$+2^{10}$	1
32			8	32	$+2^{21}$		2
33	-2^{31}		3	33			15
34			10	34			5
35			14	35	-2^{14}		1
36	-2^{16}		4	36	$+2^{31}$		3
37	$+2^9$		9	37			7
38			15	38	$+2^{30}$		14
39	$+2^7$		8	39	$-2^{24} - 2^{15}$		6
40	$-2^{26} - 2^2$		1	40	$+2^9$		9
41	$-2^{26} + 2^{19}$	-2^{10}	2	41	-2^{20}		11
42	-2^{25}		7	42	$+2^{15} + 2^8 + 2^6$		8
43	$+2^{25}$		0	43	$-2^{25} - 2^{24} - 2^2$		12

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

j	Δ Conditions for Step j in $\mathrm{CF^L}$	∇ Conditions for Step j in $\mathrm{CF^R}$
0	$b^L_{0,10} = c^L_{0,10}$, no carry in d^L_0 and e^L_0	$b^R_{0,9} = 0, c^R_{0,19} = 1, d^R_{0,8} = 0$, no carry in b^R_0, c^R_0 and d^R_0
1		$b^R_{0,0} = 0, b^R_{0,18} = 1, c^R_{0,9} = 1, c^R_{0,22} = 1$
2		$b^R_{0,22} = 0, b^R_{1,29} = 1$
3		$b^R_{2,10} = 1$
28	no carry in b^L_{29}	
29	$c^L_{29,21} = d^L_{29,21}$	
30	$b^L_{30,21} = 0$	
31	$b^L_{31,31} = 1$	carry does not occur in b^R_{32}
32	no carry in b^L_{33}	$c^R_{32,21} = 0$
33	$c^L_{33,31} = 1$	$b^R_{33,21} = 1$
34	$b^L_{34,31} = 1$	$b^R_{34,31} \vee \neg c^R_{34,31} = 1$, no carry in b^R_{35}
35	$b^L_{35,31} \vee \neg c^L_{35,31} = 1$, no carry in b^L_{36}	$c^R_{35,14} = 0$, no carry in b^R_{36}
36	$c^L_{36,16} = 0$, no carry in b^L_{37}	$b^R_{36,14} = 1, c^R_{36,31} = 0$
37	$b^L_{37,16} = 1, c^L_{37,9} = 1, d^L_{37,9} = 1$	$b^R_{37,31} = 1, b^R_{37,24} \vee \neg c^R_{37,24} = 1$, no carry in b^R_{38}
38	$b^L_{38,9} = 1, b^L_{38,26} \vee \neg c^L_{38,26} = 1$ no carry in b^L_{39}	$c^R_{38,30} = 0, b^R_{38,9} \vee \neg c^R_{38,9} = 1$, no carry in b^R_{39}
39	$c^L_{39,7} = 0, b^L_{39,19} \vee \neg c^L_{39,19} = 1$ no carry in b^L_{40}	$b^R_{39,30} = 1, c^R_{39,24} = 1, c^R_{39,15} = 0, d^R_{39,24} = 1$, no carry in b^R_{40}
40	$b^L_{40,7} = 1, c^L_{40,26} = 1, c^L_{40,2} = 0, d^L_{40,26} = 1$, no carry in b^L_{41}	$b^R_{40,24} = 1, b^R_{40,15} = 1, c^R_{40,9} = 1, d^R_{40,9} = 1$, $b^R_{40,8} \vee \neg c^R_{40,8} = 1$, no carry in b^R_{41}
41	$b^L_{41,26} = 1, b^L_{41,2} = 1, c^L_{41,26} = 0, c^L_{41,24} = 1, d^L_{41,24} = 1$, $b^L_{41,17} \vee \neg c^L_{41,17} = 1$, no carry in b^L_{42}	$b^R_{41,9} = 1, c^R_{41,20} = 0, b^R_{41,25} \vee \neg c^R_{41,25} = 1$, $b^R_{41,2} \vee \neg c^R_{41,2} = 1$, no carry in b^R_{42}
42	$b^L_{42,24} = 1, b^L_{42,19} = 1, b^L_{42,4} = 0, c^L_{42,25} = 0, c^L_{42,4} = 1$, $b^L_{42,12} \vee \neg c^L_{42,12} = 1$	$b^R_{42,20} = 1, c^R_{42,15} = 0, c^R_{42,8} = 1, c^R_{42,6} = 0, d^R_{42,8} = 1$, $b^R_{42,19} \vee \neg c^R_{42,19} = 1$

Table 13. Differential path construction for the intermediate 3-rounds of RIPEMD-160

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

Table 14. Differential paths for 2-dimension sums on RIPEMD-160 in the second scenario.

'$*$' represents that the sign is not fixed for each pair, but must be identical between two pairs. Input chaining variable difference for $\mathrm{CF^L}$ is that $\Delta a_{16}^L = +2^3 + 2^2 + 2^0$, $\Delta b_{16}^L = -2^{28}$, $\Delta c_{16}^L = 0$, $\Delta d_{16}^L = +2^{15}$, and $\Delta e_{16}^L = -2^{10} - 2^9$. Input chaining variable difference for $\mathrm{CF^R}$ is that $\Delta a_{16}^R = -2^{26} + 2^{21}$, $\Delta b_{16}^R = +2^{23}$, $\Delta c_{16}^R = -2^{16}$, $\Delta d_{16}^R = +2^{11}$, and $\Delta e_{16}^R = -2^{30}$.

Path for $\mathrm{CF^L}$ with $\Delta m_{12} = +2^{16}$				Path for $\mathrm{CF^R}$ with $\nabla m_{13} = +2^{13}$			
j	Δb_j^L	$\Delta m_{\pi^L(j)}$	$\pi^L(j)$	j	∇b_j^R	$\nabla m_{\pi^R(j)}$	$\pi^R(j)$
17	$+2^7$		7	17	-2^3		6
18	-2^{16}		4	18			11
19	$+2^{23}$		13	19			3
20	-2^6		1	20			7
21			10	21			0
22			6	22		$+2^{13}$	13
23			15	23			5
24			3	24			10
25		$+2^{16}$	12	25			14
26			0	26			15
...	...	...	...	...	...	...	...
47			5	47			4
48	$+2^{21}$	$+2^{16}$	12	48	$+2^{18}$	$+2^{13}$	13
49			1	49			8
50			9	50			6
51			11	51			4
52	$+2^{31}$		10	52	$+2^{28}$		1
53			0	53			3
54			8	54			11
55	$+2^{25}$	$+2^{16}$	12	55			15
56	$+2^9$		4	56	2^6		0
57			13	57			5
58			3	58			12
59	$+2^3$		7	59			2
60	$+2^{19}$		15	60	$+2^{22} + 2^{16}$	$+2^{13}$	13
61			14	61			9
62			5	62			7
63	$+2^{13}$		6	63			10
64	$+2^{29}$		2	64	$+2^{26} + 2^0$		14
65			4	65			12
66			0	66	$*2^{31} * 2^5$		15
67	$+2^{13}$		5	67	$*2^{22} * 2^{17} * 2^{16} * 2^{11}$		10

Table 15. Sufficient conditions on RIPEMD-160 in the second scenario.

j	Δ Conditions for Step j in $\mathrm{CF^L}$	∇ Conditions for Step j in $\mathrm{CF^R}$
17	$b^L_{16,15}=1, c^L_{16,28}=d^L_{16,28}$, no carry in b^L_{16} and d^L_{16}	$b^R_{16,11}=c^R_{16,11}, d^R_{16,23}=0, d^R_{16,16}=1$, no carry in b^R_{16}, c^R_{16}, and d^R_{16}
18	$b^L_{17,28}=0, c^L_{16,28}=d^L_{16,28}$, no carry in b^L_{18}	$b^R_{17,26}=c^R_{17,26}, d^R_{17,23}=1, d^R_{17,3}=0$
19	$b^L_{18,7}=0, b^L_{18,6}=1, c^L_{18,16}=d^L_{18,16}$, no carry in b^L_{19}	$b^R_{18,1}=c^R_{18,1}, d^R_{18,3}=1$
20	$b^L_{19,17}=1, b^L_{19,16}=0, c^L_{19,23}=d^L_{19,23}$, no carry in b^L_{20}	$b^R_{19,13}=c^R_{19,13}$
21	$b^L_{20,26}=1, b^L_{20,23}=0, c^L_{20,6}=d^L_{20,6}$	
22	$b^L_{21,6}=0, b^L_{21,1}=1$	
23	$b^L_{22,16}=1$	
48	no carry in b^L_{48}	no carry in b^R_{48}
49	$d^L_{48,21}=0$	$c^R_{48,18}=d^R_{48,18}$
50	$d^L_{49,21}=1$	$b^R_{49,18}=0$
51	$b^L_{50,31}=c^L_{50,31}$	$b^R_{50,28}=0$
52	no carry in b^L_{52}	no carry in b^R_{52}
53	$d^L_{52,31}=1$	$c^R_{52,28}=0, d^R_{52,28}=1$
54	$b^L_{53,31}=1$	$b^R_{53,28}=0$
55	$b^L_{54,9}=c^L_{54,9}$ no carry in b^L_{55}	$b^R_{54,6}=1$
56	$d^L_{55,25}=0$, 1-bit carry in b^L_{56}	no carry in b^R_{56}
57	$d^L_{56,25}=1, d^L_{56,10}=0, d^L_{56,9}=1$	$c^R_{56,6}=0, d^R_{56,6}=1$
58	$b^L_{57,3}=c^L_{57,3}, d^L_{57,10}=1, d^L_{57,9}=1$	$b^R_{57,6}=0$
59	$b^L_{58,20}=c^L_{58,20}, b^L_{58,19}=c^L_{58,19}$, 1-bit carry in b^L_{59}	$b^R_{58,16}=1$
60	$d^L_{59,4}=0, d^L_{59,3}=1$, 1-bit carry in b^L_{60}	no carry in b^R_{60}
61	$d^L_{60,20}=0, d^L_{60,19}=1, d^L_{60,4}=1, d^L_{60,3}=1$	$c^R_{60,22}=d^R_{60,22}, c^R_{60,16}=0, d^R_{60,16}=1$
62	$b^L_{61,14}=c^L_{61,14}, b^L_{61,13}=c^L_{61,13}, d^L_{61,20}=1, d^L_{61,19}=1$	$b^R_{61,22}=0, b^R_{61,16}=0$
63	$b^L_{62,30}=c^L_{62,30}, b^L_{62,29}=c^L_{62,29}$, 1-bit carry in b^L_{63}	$b^R_{62,26}=1, b^R_{62,0}=1$
64	$d^L_{63,14}=0, d^L_{63,13}=1$, no carry in b^L_{64}	no carry in b^R_{64}
65	$c^L_{64,29} \vee \neg d^L_{64,29}=1, d^L_{64,14}=0, d^L_{64,13}=0$	$c^R_{64,26} \oplus d^R_{64,26}=1, c^R_{64,0} \oplus d^R_{64,0}=1$
66	$c^L_{65,14}=1, c^L_{65,13}=1, d^L_{65,29}=0$	no carry after adding f, no carry in b^R_{66}, the signs of $*2^5$ in b^R_{66} are identical between two pairs.
67	$c^L_{66,29}=1$	the signs of $*2^{22}*2^{17}*2^{16}*2^{11}$ in b^R_{67} are identical between two pairs.