Wang GL. Collision attack on the full extended MD4 and pseudo-preimage attack on RIPEMD. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(1): 129–143 Jan. 2013. DOI 10.1007/s11390-013-1317-5

Collision Attack on the Full Extended MD4 and Pseudo-Preimage Attack on RIPEMD

Gao-Li Wang (王高丽)

School of Computer Science and Technology, Donghua University, Shanghai 201620, China

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences Beijing 100093, China

E-mail: wanggaoli@dhu.edu.cn

Received January 11, 2012; revised September 12, 2012.

Abstract The cryptographic hash functions Extended MD4 and RIPEMD are double-branch hash functions, which consist of two parallel branches. Extended MD4 was proposed by Rivest in 1990, and RIPEMD was devised in the framework of the RIPE project (RACE Integrity Primitives Evaluation, 1988∼1992). On the basis of differential analysis and meet-in-the-middle attack principle, this paper proposes a collision attack on the full Extended MD4 and a pseudo-preimage attack on the full RIPEMD respectively. The collision attack on Extended MD4 holds with a complexity of 2^{37}, and a collision instance is presented. The pseudo-preimage attack on RIPEMD holds with a complexity of $2^{125.4}$, which optimizes the complexity order for brute-force attack. The results in this study will also be beneficial to the analysis of other double-branch hash functions such as RIPEMD-160.

Keywords collision attack, preimage attack, hash function, Extended MD4, RIPEMD

1 Introduction

Cryptographic hash functions remain one of the most prevailing cryptographic primitives, and they can guarantee the security of many cryptosystems and protocols such as digital signature, message authentication code, and so on. In 1990, Rivest introduced the first dedicated hash function MD4[1]. After the publication of MD4, several dedicated hash functions were proposed, and these functions are called MD-family. Depending on the methods of the message expansion and the number of parallel branches, the MD-family is divided into three subfamilies. The first subfamily is MD4-family, which consists of MD4[1], MD5[2] and HAVAL[3]. The characteristics of MD4-family are using roundwise permutations for the message expansion and only one branch of computation. The second subfamily is RIPEMD-family, which consists of RIPEMD[4], RIPEMD-{128, 160, 256, 320}[5] and Extended MD4①[1]. The crucial difference between MD4-family and RIPEMD-family is that RIPEMD-family uses two parallel branches of computations. The third subfamily is SHA-family, which consists of SHA-{0, 1, 224, 256, 384, 512}[6-8]. These functions use only one branch of computation, but the message expansion is achieved by some recursively defined functions. Several important breakthroughs have been made in the cryptanalysis of hash functions and they imply that most of the current standard hash functions are vulnerable. In this circumstance, National Institute of Standards and Technology (NIST) launches the NIST Hash Competition②, a public competition to develop a new hash standard, which is called SHA-3 and was announced at 2012.

From the security perspective, a cryptographic hash function should satisfy several properties such as preimage resistance, second-preimage resistance and collision resistance. A hash function is considered academically broken if it is possible to find a collision or (second) preimage faster than birthday attack or brute

Regular Paper

This work was supported by the National Natural Science Foundation of China under Grant No. 61103238, the "Chen Guang" project of Shanghai Municipal Education Commission and Shanghai Education Development Foundation of China under Grant No. 09CG29, and the Fundamental Research Funds for the Central Universities of China.

①Extended MD4 has two copies of (modified) MD4, and the hash value is obtained by concatenating the results of both copies of MD4. However, its security against collision attack is much stronger than that of MD4, so we classify it as a double-branch hash function.

②NIST. Cryptographic hash project, http://csrc.nist.gov/groups/ST/hash/index.html, September 2012.

force attack respectively. The first analysis of MD4 and MD5 was made by Vaudenay[9] and by den Boer and Bosselaers[10]. Along with the development of the hash functions, there is some continuous analysis on them and the work reveals that most of the hash functions are not so secure as claimed[11-21]. Wang *et al.* presented a series collision attacks on the most prevailing ARX-type (modular addition, rotation and bitwise XOR) hash functions including MD4[22], RIPEMD[22], RIPEMD-128[23], MD5[24], SHA-0[25], SHA-1[26], HAVAL[27-28], SIMD[29] and Skein[30], etc. using an attack technique which is based on differential cryptanalysis[31]. Wang's method was also adopted in searching the second-preimage of MD4[32], and was further developed in the analysis of SHA-1[33-34]. With the collision attacks on some dedicated hash functions, more attentions are payed to evaluate the preimage resistance of hash functions. So far, in the sense of preimage resistance, full MD2[35-36], MD4[14,37-41], MD5[42], HAVAL[43], Tiger[39], and step-reduced RIPEMD[44-45], RIPEMD-128[46], RIPEMD-160[46], SHA-0[47], SHA-1[47], SHA-2[39,48], GOST[49], Skein[50], Whirlpool[51], Grøstl[52], etc. have been broken. Most of these preimage attacks follow a technique of meet-in-the-middle[53-54]. It is worth noting that, in FSE 2012, Li *et al.* converted meet-in-the-middle pseudo-preimage attack into pseudo collision attack, and gave applications to SHA-2, etc[55].

Extended MD4 was proposed in the original article[1] by Rivest in 1990 with 256-bit hash value. Its compression function consists of two parallel branches called left branch and right branch. The left branch is the standard MD4 algorithm, and the right branch is a modified MD4 algorithm. The initial values of left branch and right branch are denoted by IV_0 and IV_1 ($IV_0 \neq IV_1$) respectively. Dobbertin proposed a pseudo-collision attack for Extended MD4 with a complexity of 2^{40} under the condition that $IV_0 = IV_1$ is prescribed. However, no collision attack on Extended MD4 under the standard initial values was proposed yet. RIPEMD[4] was developed in the European RIPE project (RACE Integrity Primitives Evaluation, 1988∼1992), and was designed by Dobbertin, Bosselaers, and Preneel. Its compression function consists of two parallel branches of transformations and generates the output by mixing the results of the two branches. Wang *et al.* presented a collision attack on RIPEMD[22]. As for the preimage attack, the security of step-reduced RIPEMD has been analyzed in [44-45③, 56].

Many studies have been conducted on the security of ARX-type hash functions using Wang's method. However, owing to the two parallel branches of RIPEMD-family, it is difficult to deduce the correct differential path and to use message modification technique to improve the success probability for RIPEMD-family. The security of RIPEMD-family hash functions against collision attack has been strengthened greatly. [22] reports that among 30 selected collision differential paths, only one can produce a real collision, and in other paths, the conditions of both branches in some step cannot hold simultaneously. Extended MD4 is such a representative hash function of RIPEMD-family. It is difficult to deduce a correct differential path for both branches and to modify the messages to greatly improve the success probability of the attack, so to find a practical collision. In Section 3, we propose a practical collision attack on Extended MD4 under the standard initial values, and present a collision instance for Extended MD4. By choosing a proper message difference, we can find a differential path of both left branch and right branch, and deduce the corresponding sufficient conditions that ensure the differential path hold. We use the message modification techniques to modify the messages so that almost all sufficient conditions hold. Our attack requires less than 2^{37} computations to get a collision of the full Extended MD4. To the best of our knowledge, this is the first work that a practical collision attack on the full Extended MD4 has been proposed.

The meet-in-the-middle technique works efficiently on narrow-pipe Merkle-Damgård hash functions. However, because the size of the internal state of RIPEMD is 256 bits which is a double of the size of hash value, if we apply the meet-in-the-middle technique directly, it will has no advantage compared with the complexity 2^{128} of the brute force attack. So it is difficult to apply the meet-in-the-middle technique to propose preimage attacks on RIPEMD directly. In Sections 4 and 5, we find some new observations and propose the first pseudo-preimage attack on the full RIPEMD using the meet-in-the-middle principle combined with some other techniques such as initial structure, partial-matching, partial-fixing. The complexity of our attack to find a pseudo-preimage of RIPEMD is $2^{125.4}$. The attack optimizes the complexity order for brute-force attack. See Table 1 for a summary of our results in Sections 4 and 5 and the comparison with the previous attacks.

The rest of the paper is organized as follows. Section 2 introduces some notations, describes the Extended MD4 and RIPEMD algorithms, and summarizes some useful properties of the Boolean functions in two hash functions. The next section proposes the detailed description of the collision attack on Extended MD4.

③The attack contains a flaw on the message-word order. If the correct order is used, the attack can work on the first 31 steps instead of 26 steps.

Table 1. Comparison of Our Results with Previous (Pseudo-)Preimage Attacks

Number of Steps	Pseudo-Preimage Attacks Complexity Time/Memory	(Second) Preimage Attacks Complexity Time/Memory	Reference
26/29*	$2^{110}/2^{33}$	$2^{115.2}/2^{33}$	[45]
33	$2^{121}/2^{10}$	$2^{125.5}/2^{10}$	[44]
35*	$2^{96}/2^{35}$	$2^{113}/2^{35}$	[44]
47⋆	$2^{119}/2^{10.5}$	$2^{124.5}/2^{10.5}$	[56]
48	$2^{125.4}/2^{58}$		Ours

Note: $*$: the attacked steps start from some intermediate step. $\star$: the attack is only applicable to find second preimages.

Sections 4 and 5 describe the procedure of our pseudo-preimage attack on the full RIPEMD compression function. Finally, Section 6 concludes the paper.

2 Preliminary

2.1 Notations

In order to describe our analysis conveniently, we introduce some notations, where $0 \leqslant j \leqslant 31$.

1) $M = (m_0, m_1, \ldots, m_{15})$ represents a 512-bit block, where m_i $(0 \leqslant i \leqslant 15)$ is a 32-bit word.

2) $\neg, \wedge, \oplus, \vee$ denote bitwise complement, AND, XOR and OR respectively.

3) $\lll s$ ($\ggg s$) is circular shift s-bit positions to the left (right).

4) $x \parallel y$ denotes concatenation of the two bit strings x and y.

5) $+, -$ denote addition and subtraction modulo 2^{32} respectively.

6) The least significant bit is the first bit (0th bit) and the most significant bit is the last bit (31st bit).

7) $x_{i,j}$ denotes the j-th bit of 32-bit word x_i.

8) $\Delta x_i = x'_i - x_i$ is the modular subtraction difference of two words x'_i and x_i.

9) $x'_i = x_i[j]$ is the value obtained by modifying the j-th bit of x_i from 0 to 1, i.e., $x_{i,j} = 0$, $x'_{i,j} = 1$, and the other bits of x_i and x'_i are all equal. Similarly, $x_i[-j]$ is the value obtained by modifying the j-th bit of x_i from 1 to 0.

10) $x_i[\pm j_1, \pm j_2, \ldots, \pm j_k]$ denotes the value obtained by modifying the bits in positions $j_1, \ldots, j_k$ of x_i according to the $\pm$ signs.

11) In Section 3, (a_i, b_i, c_i, d_i) and (aa_i, bb_i, cc_i, dd_i) $(0 \leqslant i \leqslant 12)$ represent the chaining variables corresponding to the message block M_1 of left branch and right branch respectively.

12) In Section 3, (a'_i, b'_i, c'_i, d'_i) and $(aa'_i, bb'_i, cc'_i, dd'_i)$ $(0 \leqslant i \leqslant 12)$ represent the chaining variables corresponding to the message block M'_1 of left branch and right branch respectively.

13) In Sections 4 and 5, (a_i, b_i, c_i, d_i) and (aa_i, bb_i, cc_i, dd_i) $(0 \leqslant i \leqslant 48)$ represent the chaining variables of left branch and right branch respectively.

14) $0^{\alpha\sim\beta}$ means that the bits from the β-th bit to the α-th bit are all 0, where $\alpha > \beta$.

15) $w^{\alpha\sim\beta}$ mean that the bits from the β-th bit to the α-th bit of the variable w are arbitrary bits.

16) $[\alpha \sim \beta]$ means that the bits from the β-th bit to the α-th bit are known, and all the other bits are unknown.

17) $[\alpha \sim \beta, \gamma \sim \delta]$ means that the bits from the β-th bit to the α-th bit and from the δ-th bit to the γ-th bit are known, and all the other bits are unknown.

Note that the differential definition in Wang's method[24] is a kind of precise differential which uses the difference in terms of integer modular subtraction and the difference in terms of XOR. The combination of both kinds of differences gives attackers more information. For example, the output difference in step 1 of Table 2 (collision differential path of Extended MD4) is $\Delta a_1 = a'_1 - a_1 = 2^{19}$, for the specific differential path, we need to expand the one-bit difference in bit 19 into a three-bit differences in bits 19, 20 and 21. That is, we expand $a_1[19]$ to $a_1[-19, -20, 21]$, which means the 19th and 20th bits of a_1 are 1, and the 21st bit of a_1 is 0, while the 19th and 20th bits of a'_1 are 0, and the 21st bit of a'_1 is 1.

2.2 Description of Extended MD4

The hash function Extended MD4 compresses a message of length less than 2^{64} bits into a 256-bit hash value. Firstly, the algorithm pads any given message into a message with the length of 512-bit multiple. We do not describe the padding process here because it has little relation with our attack, and the details of the message padding can refer to [1]. Each 512-bit message block invokes a compression function of Extended MD4. The compression function takes a 256-bit chaining value and a 512-bit message block as input and outputs another 256-bit chaining value. The initial chaining value is a set of fixed constants. The compression function consists of two parallel branches named left branch and right branch. Left branch is the standard MD4 and right branch is a modified MD4. Each branch has three rounds, and the nonlinear functions in each round are as follows:

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z),$$
$$G(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z),$$
$$H(X, Y, Z) = X \oplus Y \oplus Z.$$

Here X, Y, Z are 32-bit words. The operations of the three functions are all bitwise. Each round of the compression function in each branch consists of 16 similar

Table 2. Collision Differential Path of Extended MD4

Step	Chaining Variable for M	m_i	Shift	Δm_i	Step Difference	Chaining Variable for M'
1	a_1	m_0	3	2^{16}	2^{19}	$a_1[-19,-20,21]$
2	d_1	m_1	7		2^{27}	$d_1[-27,28]$
3	c_1	m_2	11		2^6	$c_1[-6,-7,8]$
4	b_1	m_3	19			b_1
5	a_2	m_4	3		$-2^{10}+2^{22}-2^{30}$	$a_2[-10,22,-30]$
6	d_2	m_5	7		2^2	$d_2[-2,-3,4]$
7	c_2	m_6	11		2^{17}	$c_2[17]$
8	b_2	m_7	19		-2^{21}	$b_2[21,-22]$
9	a_3	m_8	3		$-2-2^{13}$	$a_3[1,-2,-13]$
10	d_3	m_9	7		2^9	$d_3[-9,10]$
11	c_3	m_{10}	11		2^{28}	$c_3[28]$
12	b_3	m_{11}	19		-2^8	$b_3[8,-9]$
13	a_4	m_{12}	3		-2^4-2^{16}	$a_4[-4,-16]$
14	d_4	m_{13}	7			d_4
15	c_4	m_{14}	11		2^7	$c_4[-7,8]$
16	b_4	m_{15}	19			b_4
17	a_5	m_0	3	2^{16}	-2^7	$a_5[7,-8]$
18	d_5	m_4	5			d_5
19	c_5	m_8	9			c_5
20	b_5	m_{12}	13			b_5
21	a_6	m_1	3		-2^{10}	$a_6[-10]$
22	d_6	m_5	5			d_6
23	c_6	m_9	9			c_6
24	b_6	m_{13}	13			b_6
25	a_7	m_2	3		-2^{13}	$a_7[-13]$
26	d_7	m_6	5			d_7
27	c_7	m_{10}	9			c_7
28	b_7	m_{14}	13			b_7
29	a_8	m_3	3		-2^{16}	$a_8[-16]$
30	d_8	m_7	5			d_8
31	c_8	m_{11}	9			c_8
32	b_8	m_{15}	13			b_8
33	a_9	m_0	3	2^{16}		a_9
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
48	b_{12}	m_{15}	15			b_{12}

steps, and in each step one of the four chaining variables a, b, c, d is updated.

$$
\begin{aligned}
\phi_0(a,b,c,d,m_k,s) &= (a+F(b,c,d)+m_k) \lll s,\\
\phi_1(a,b,c,d,m_k,s) &= (a+G(b,c,d)+m_k+5a827999)\\
&\quad \lll s,\\
\phi_2(a,b,c,d,m_k,s) &= (a+H(b,c,d)+m_k+6ed9eba1)\\
&\quad \lll s,\\
\psi_0(a,b,c,d,m_k,s) &= (a+F(b,c,d)+m_k) \lll s,\\
\psi_1(a,b,c,d,m_k,s) &= (a+G(b,c,d)+m_k+50a28be6)\\
&\quad \lll s,\\
\psi_2(a,b,c,d,m_k,s) &= (a+H(b,c,d)+m_k+5c4dd124)\\
&\quad \lll s.
\end{aligned}
$$

The initial value of left branch is (a_0, b_0, c_0, d_0) = (67452301, efcdab89, 98badcfe, 10325476). The initial value of right branch is (aa_0, bb_0, cc_0, dd_0) = (33221100, 77665544, bbaa9988, ffeeddcc).

Compression Function of Extended MD4. For a 512-bit message block $M = (m_0, m_1, \ldots, m_{15})$ of the padded message $\overline{M}$, the compression function consists of left branch and right branch.

Left Branch. For the 512-bit block M, left branch is as follows:

1) Let (a_0, b_0, c_0, d_0) be the input of left branch for M. If M is the first message block to hashed, then (a_0, b_0, c_0, d_0) are set to be the initial value. Otherwise it is the output from compressing the previous message block by left branch.

2) Perform the following 48 steps (three rounds):

For $j = 0, 1, 2$,

For $i = 0, 1, 2, 3$,

$a = \phi_j(a, b, c, d, m_{ord(j,16j+4i+1)}, s_{j,16j+4i+1})$,

$d = \phi_j(d, a, b, c, m_{ord(j,16j+4i+2)}, s_{j,16j+4i+2})$,
$c = \phi_j(c, d, a, b, m_{ord(j,16j+4i+3)}, s_{j,16j+4i+3})$,
$b = \phi_j(b, c, d, a, m_{ord(j,16j+4i+4)}, s_{j,16j+4i+4})$,
where m_{ord} means order of message.

The compressing result of left branch is $(A, B, C, D) = (a_0 + a, b_0 + b, c_0 + c, d_0 + d)$.

Right Branch. For the 512-bit block M, right branch is as follows:

1) Let (aa_0, bb_0, cc_0, dd_0) be the input of right branch for M. If M is the first block to be hashed, (aa_0, bb_0, cc_0, dd_0) are the initial value. Otherwise it is the output from compressing the previous message block by right branch.

2) Perform the following 48 steps (three rounds):
For $j = 0, 1, 2$,
For $i = 0, 1, 2, 3$,
$aa = \psi_j(aa, bb, cc, dd, m_{ord(j,16j+4i+1)}, s_{j,16j+4i+1})$,
$dd = \psi_j(dd, aa, bb, cc, m_{ord(j,16j+4i+2)}, s_{j,16j+4i+2})$,
$cc = \psi_j(cc, dd, aa, bb, m_{ord(j,16j+4i+3)}, s_{j,16j+4i+3})$,
$bb = \psi_j(bb, cc, dd, aa, m_{ord(j,16j+4i+4)}, s_{j,16j+4i+4})$.

The compressing result of right branch is $(AA, BB, CC, DD) = (aa_0 + aa, bb_0 + bb, cc_0 + cc, dd_0 + dd)$.

Note that after every 16-word block is processed, the values of the a register in left branch and the aa register in right branch are exchanged. The ordering of message words and the details of the shift positions can be seen in Table 3.

Table 3. Order of the Message Words and Shift Positions in Extended MD4

Step i	Order of Message $ord(0,i)$	Shift $s_{0,i}$	Step i	Order of Message $ord(1,i)$	Shift $s_{1,i}$	Step i	Order of Message $ord(2,i)$	Shift $s_{2,i}$
1	0	3	17	0	3	33	0	3
2	1	7	18		5	34	8	9
3	2	11	19	8	9	35	4	11
4	3	19	20	12	13	36	12	15
5	4	3	21	1	3	37	2	3
6	5	7	22	5	5	38	10	9
7	6	11	23	9	9	39	6	11
8	7	19	24	13	13	40	14	15
9	8	3	25	2	3	41	1	3
10	9	7	26	6	5	42	9	9
11	10	11	27	10	9	43	5	11
12	11	19	28	14	13	44	13	15
13	12	3	29	3	3	45	3	3
14	13	7	30	7	5	46	11	9
15	14	11	31	11	9	47	7	11
16	15	19	32	15	13	48	15	15

If M is the last block of $\overline{M}$, $(A \parallel B \parallel C \parallel D \parallel AA \parallel BB \parallel CC \parallel DD)$ is the hash value of the message $\overline{M}$. Otherwise, repeat the above process with the next 512-bit message block by taking (A, B, C, D) and (AA, BB, CC, DD) as the input chaining variables of left branch and right branch respectively.

2.3 Description of RIPEMD

The hash function RIPEMD compresses any arbitrary length message into a message with the length of 128 bit. Firstly RIPEMD pads any given message into a message with the length of 512 bit multiple. For each 512-bit message block, RIPEMD compresses it into a 128-bit hash value by a compression function. The compression function consists of two parallel operations, which are denoted by left branch and right branch respectively. The nonlinear functions are the same as the functions F, G, H in Extended MD4.

Left Branch. For a 512-bit block $M = (m_0, m_1, \ldots, m_{15})$, left branch is as follows:

1) Let (a_0, b_0, c_0, d_0) be the input of left branch for M. If M is the first block to be hashed, (a_0, b_0, c_0, d_0) is the initial value. Otherwise it is the output of the previous block compressing.

2) Perform the following 48 steps (three rounds):

a) For $i = 1, \cdots, 16$, do the following 16 operations: $a_i = d_{i-1}$, $b_i = (a_{i-1} + F(b_{i-1}, c_{i-1}, d_{i-1}) + m_{\sigma(i)}) \lll s_i$, $c_i = b_{i-1}$, $d_i = c_{i-1}$.

b) For $i = 17, \cdots, 32$, do the following 16 operations: $a_i = d_{i-1}$, $b_i = (a_{i-1} + G(b_{i-1}, c_{i-1}, d_{i-1}) + m_{\sigma(i)} + \text{5a827999}) \lll s_i$, $c_i = b_{i-1}$, $d_i = c_{i-1}$.

c) For $i = 33, \cdots, 48$, do the following 16 operations: $a_i = d_{i-1}$, $b_i = (a_{i-1} + H(b_{i-1}, c_{i-1}, d_{i-1}) + m_{\sigma(i)} + \text{6ed9eba1}) \lll s_i$, $c_i = b_{i-1}$, $d_i = c_{i-1}$.

Right Branch. For a 512-bit block $M = (m_0, m_1, \ldots, m_{15})$, right branch is as follows:

1) Let (aa_0, bb_0, cc_0, dd_0) be the input of right branch for M. If M is the first block to be hashed, $(aa_0,\ bb_0,\ cc_0,\ dd_0)$ is the initial value. Otherwise it is the output of the previous block compressing.

2) Perform the following 48 steps (three rounds):

a) For $i = 1, \cdots, 16$, do the following 16 operations: $aa_i = dd_{i-1}$, $bb_i = (aa_{i-1} + F(bb_{i-1}, cc_{i-1}, dd_{i-1}) + m_{\sigma(i)} + \text{50a28be6}) \lll s_i$, $cc_i = bb_{i-1}$, $dd_i = cc_{i-1}$.

b) For $i = 17, \cdots, 32$, do the following 16 operations: $aa_i = dd_{i-1}$, $bb_i = (aa_{i-1} + G(bb_{i-1}, cc_{i-1}, dd_{i-1}) + m_{\sigma(i)}) \lll s_i$, $cc_i = bb_{i-1}$, $dd_i = cc_{i-1}$.

c) For $i = 33, \cdots, 48$, do the following 16 operations: $aa_i = dd_{i-1}$, $bb_i = (aa_{i-1} + H(bb_{i-1}, cc_{i-1}, dd_{i-1}) + m_{\sigma(i)} + \text{5c4dd124}) \lll s_i$, $cc_i = bb_{i-1}$, $dd_i = cc_{i-1}$.

Note that the initial values of left branch and right branch are identical. The orders of message words and the details of the shift positions can be seen in Table 4. Add the output of left branch to the output of right branch as follows: $H_0 = b_0 + c_{48} + dd_{48}, H_1 = c_0+d_{48}+aa_{48}, H_2 = d_0+a_{48}+bb_{48}, H_3 = a_0+b_{48}+cc_{48}$. If M is the last message block of the message MM, then $H(MM) = H_0 \parallel H_1 \parallel H_2 \parallel H_3$ is the hash value for the message MM. Otherwise repeat the compression process for the next 512-bit message block and (H_0, H_1, H_2, H_3) as input. For the completed specification, refer to [4].

Table 4. Word Processing Orders and Shift Positions in RIPEMD

Step i	Order of Message $\sigma(i)$	Shift s_i	Step i	Order of Message $\sigma(i)$	Shift s_i	Step i	Order of Message $\sigma(i)$	Shift s_i
1	0	11	17	7	7	33	3	11
2	1	14	18	4	6	34	10	13
3	2	15	19	13	8	35	2	14
4	3	12	20	1	13	36	4	7
5	4	5	21	10	11	37	9	14
6	5	8	22	6	9	38	15	9
7	6	7	23	15	7	39	8	13
8	7	9	24	3	15	40	1	15
9	8	11	25	12	7	41	14	6
10	9	13	26	0	12	42	7	8
11	10	14	27	9	15	43	0	13
12	11	15	28	5	9	44	6	6
13	12	6	29	14	7	45	11	12
14	13	7	30	2	11	46	13	5
15	14	9	31	11	13	47	5	7
16	15	8	32	8	12	48	12	5

2.4 Some Basic Conclusions of the Three Nonlinear Functions

We will recall some well-known properties of the three nonlinear Boolean functions because they are very helpful for determining the collision differential path, the corresponding sufficient conditions and the initial structure. In the following, $x \in \{0, 1\}$, $y \in \{0, 1\}$ and $z \in \{0, 1\}$.

Proposition 1. *For the nonlinear function $F(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$, there are the following properties.*

1) a) $F(x, y, z) = F(\neg x, y, z)$ *if and only if* $y = z$.
   b) $F(x, y, z) = x$ *and* $F(\neg x, y, z) = \neg x$ *if and only if* $y = 1$ *and* $z = 0$.
   c) $F(x, y, z) = \neg x$ *and* $F(\neg x, y, z) = x$ *if and only if* $y = 0$ *and* $z = 1$.
2) a) $F(x, y, z) = F(x, \neg y, z)$ *if and only if* $x = 0$.
   b) $F(x, y, z) = y$ *and* $F(x, \neg y, z) = \neg y$ *if and only if* $x = 1$.
3) a) $F(x, y, z) = F(x, y, \neg z)$ *if and only if* $x = 1$.
   b) $F(x, y, z) = z$ *and* $F(x, y, \neg z) = \neg z$ *if and only if* $x = 0$.

Proposition 2. *For the nonlinear function $G(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$, there are the following properties*:

1) a) $G(x, y, z) = G(\neg x, y, z)$ *if and only if* $y = z$.
   b) $G(x, y, z) = x$ *and* $G(\neg x, y, z) = \neg x$ *if and only if* $y \neq z$.
2) a) $G(x, y, z) = G(x, \neg y, z)$ if and only if $x = z$.
   b) $G(x, y, z) = y$ and $G(x, \neg y, z) = \neg y$ if and only if $x \neq z$.
3) a) $G(x, y, z) = G(x, y, \neg z)$ if and only if $x = y$.
   b) $G(x, y, z) = z$ and $G(x, y, \neg z) = \neg z$ if and only if $x \neq y$.
4) $G(x, 0, 0) = G(0, y, 0) = G(0, 0, z) = 0$.

Proposition 3. *For the nonlinear function $H(x, y, z) = x \oplus y \oplus z$, there are the following properties*:

1) $H(x, y, z) = \neg H(\neg x, y, z) = \neg H(x, \neg y, z) = \neg H(x, y, \neg z)$.
2) $H(x, y, z) = H(\neg x, \neg y, z) = H(x, \neg y, \neg z) = H(\neg x, y, \neg z)$.

Proposition 4[56]. *Suppose (a_1, b_1, c_1, d_1) is known and m_0 is unknown, then we can get b_0, c_0, d_0 and $a_0 + m_0 = (b_1 \ggg 11) - F(b_0, c_0, d_0)$. Because the input of left branch and right branch is identical, we can get $bb_0 = b_0$, $cc_0 = c_0$, $dd_0 = d_0$. According to $bb_1 = (aa_0 + m_0 + F(bb_0, cc_0, dd_0) + \text{50a28be6}) \lll 11 = (a_0 + m_0 + F(b_0, c_0, d_0) + \text{50a28be6}) \lll 11$, we can get $bb_1 = ((b_1 \ggg 11) + \text{50a28be6}) \lll 11$. Therefore, (aa_1, bb_1, cc_1, dd_1) can be obtained.*

3 Practical Collision Attack Against Extended MD4

In this section, we present a practical collision attack on Extended MD4. Each message in the collision includes two 512-bit message blocks. We search the collision pair $(M_0 \parallel M_1, M_0 \parallel M_1')$ in the following four parts:

1) Denote Extended MD4 by h and the hash value $h(M_0)$ by $(a \parallel b \parallel c \parallel d \parallel aa \parallel bb \parallel cc \parallel dd)$. (a, b, c, d) and (aa, bb, cc, dd) are also the input chaining variables of left branch and right branch of the next compression function respectively. Find a message block M_0 such that $h(M_0)$ satisfies some conditions which are part of the sufficient conditions that ensure the differential path hold, and the conditions of $h(M_0)$ are $b_i = c_i (i = 19, 21)$, $b_i = 0 (i = 20, 27, 28)$, $c_{0,20} = 1$, $bb_i = cc_i (i = 19, 21)$, $bb_i = 0 (i = 20, 27, 28)$ and $cc_{0,20} = 1$.

2) Choose an appropriate message difference $\Delta M_1 = M_1' - M_1$ and deduce the differential path according to the specified message difference.

3) Derive a set of sufficient conditions which ensure the differential path hold. This means that if $h(M_1)$ satisfies all the conditions in Table 5, then $(M_0 \parallel M_1, M_0 \parallel M_1')$ consist of a collision.

4) Modify the message M_1 to fulfill most of the sufficient conditions.

Obviously the first part is easy to be carried out. We will describe the last three parts in details.

Table 5. Set of Suffcient Conditions for the Differential Path Given in Table 2

c_0	$c_{0,20}=1$
b_0	$b_{0,19}=c_{0,19}$, $b_{0,20}=0$, $b_{0,21}=c_{0,21}$, $b_{0,27}=0$, $b_{0,28}=0$
a_1	$a_{1,19}=1$, $a_{1,20}=1$, $a_{1,21}=0$, $a_{1,27}=1$, $a_{1,28}=1$
d_1	$d_{1,6}=a_{1,6}$, $d_{1,7}=a_{1,7}$, $d_{1,8}=a_{1,8}$, $d_{1,19}=0$, $d_{1,20}=0$, $d_{1,21}=0$, $d_{1,27}=1$, $d_{1,28}=0$
c_1	$c_{1,6}=1$, $c_{1,7}=1$, $c_{1,8}=0$, $c_{1,19}=1$, $c_{1,20}=1$, $c_{1,21}=1$, $c_{1,27}=0$, $c_{1,28}=0$
b_1	$b_{1,6}=0$, $b_{1,7}=1$, $b_{1,8}=0$, $b_{1,10}=c_{1,10}$, $b_{1,22}=c_{1,22}$, $b_{1,27}=0$, $b_{1,28}=1$, $b_{1,30}=c_{1,30}$
a_2	$a_{2,2}=b_{1,2}$, $a_{2,3}=b_{1,3}$, $a_{2,4}=b_{1,4}$, $a_{2,6}=1$, $a_{2,7}=1$, $a_{2,8}=1$, $a_{2,10}=1$, $a_{2,22}=0$, $a_{2,30}=1$
d_2	$d_{2,2}=1$, $d_{2,3}=1$, $d_{2,4}=0$, $d_{2,10}=0$, $d_{2,17}=a_{2,17}$, $d_{2,22}=0$, $d_{2,30}=0$
c_2	$c_{2,2}=1$, $c_{2,3}=0$, $c_{2,4}=0$, $c_{2,10}=1$, $c_{2,17}=0$, $c_{2,21}=d_{2,21}$, $c_{2,22}=1$, $c_{2,30}=1$
b_2	$b_{2,1}=c_{2,1}$, $b_{2,2}=1$, $b_{2,3}=1$, $b_{2,4}=1$, $b_{2,13}=c_{2,13}$, $b_{2,17}=0$, $b_{2,21}=0$, $b_{2,22}=1$
a_3	$a_{3,1}=0$, $a_{3,2}=1$, $a_{3,9}=b_{2,9}$, $a_{3,10}=b_{2,10}$, $a_{3,13}=1$, $a_{3,17}=1$, $a_{3,21}=0$, $a_{3,22}=0$
d_3	$d_{3,1}=0$, $d_{3,2}=0$, $d_{3,9}=1$, $d_{3,10}=0$, $d_{3,13}=0$, $d_{3,21}=1$, $d_{3,22}=1$, $d_{3,28}=a_{3,28}$
c_3	$c_{3,1}=1$, $c_{3,2}=1$, $c_{3,8}=d_{3,8}$, $c_{3,9}=0$, $c_{3,10}=0$, $c_{3,13}=1$, $c_{3,28}=0$
b_3	$b_{3,4}=c_{3,4}$, $b_{3,8}=0$, $b_{3,9}=1$, $b_{3,10}=1$, $b_{3,16}=c_{3,16}$, $b_{3,28}=0$
a_4	$a_{4,4}=1$, $a_{4,8}=0$, $a_{4,9}=1$, $a_{4,16}=1$, $a_{4,28}=1$
d_4	$d_{4,4}=0$, $d_{4,7}=a_{4,7}$, $d_{4,8}=1$, $d_{4,9}=1$, $d_{4,16}=0$
c_4	$c_{4,4}=1$, $c_{4,7}=1$, $c_{4,8}=0$, $c_{4,16}=1$
b_4	$b_{4,7}=d_{4,7}$, $b_{4,8}=d_{4,8}$
a_5	$a_{5,7}=0$, $a_{5,8}=1$
d_5	$d_{5,7}\neq b_{4,7}$, $d_{5,8}\neq b_{4,8}$
c_5	$c_{5,7}=d_{5,7}$, $c_{5,8}=d_{5,8}$
b_5	$b_{5,10}=c_{5,10}$
a_6	$a_{6,10}=1$
d_6	$d_{6,10}=b_{5,10}$
c_6	$c_{6,10}=d_{6,10}$
b_6	$b_{6,13}=c_{6,13}$
a_7	$a_{7,13}=1$
d_7	$d_{7,13}=b_{6,13}$
c_7	$c_{7,13}=d_{7,13}$
b_7	$b_{7,16}=c_{7,16}$
a_8	$a_{8,16}=1$
d_8	$d_{8,16}=b_{7,16}$
c_8	$c_{8,16}=d_{8,16}$

3.1 Collision Differential Path for Extended MD4

Constructing the differential path and deriving the sufficient conditions go on simultaneously. On one hand, we derive the sufficient conditions according to the differential path. On the other hand, we adjust the differential path to avoid the contradictory conditions. If the sufficient conditions in some steps of left branch and right branch contradict each other, the corresponding differential path is an error and no collision can be found, then we must search other differential paths from scratch.

Almost all the conditions in the first round and some conditions in the second round can be modified to hold by the message modification technique, the other conditions in the last rounds are difficult to be modified to hold. Therefore, we will ensure the sufficient conditions in the last rounds to be as less as possible. In order to find such a differential path, we select a difference between two messages as follows: $\Delta M_1 = M_1' - M_1 = (\Delta m_0, \Delta m_1, \ldots, \Delta m_{15})$, where $\Delta m_0 = 2^{16}$, $\Delta m_i = 0, 0 < i \leqslant 15$.

The whole differential path is shown in Table 2. The first column denotes the step, the second is the chaining variable in each step for M_1, the third is the message word of M_1 in each step, the fourth is the shift rotation, the fifth is the message difference between M_1 and M_1', the sixth is the chaining variable difference for M_1 and M_1', and the seventh is the chaining variable for M_1'. The empty items both in the fifth and the sixth columns denote zero differences, and steps which are not listed in the table have zero differences for message words and chaining variables.

3.2 Deriving the Sufficient Conditions for Differential Path

In light of the propositions of the nonlinear Boolean functions given in Subsection 2.4, we can derive the conditions that guarantee the differential path in Table 2 hold. A set of sufficient conditions is shown in Table 5.

We give an example to describe how to derive a set of sufficient conditions that guarantees the differential path in step 9 of Table 2 hold. Other conditions can be derived similarly. The differential path in step 9 of Table 2 is:

$$(a_2[-10,22,-30], d_2[-2,-3,4], c_2[17], b_2[21,-22]) \longrightarrow (d_2[-2,-3,4], c_2[17], b_2[21,-22], a_3[1,-2,-13]).$$

1) According to $a_3 = (a_2 + F(b_2, c_2, d_2) + m_8) \lll 3$ and b) of Proposition 1 1), the conditions $c_{2,22} = 1$ and $d_{2,22} = 0$ ensure that the change of $b_{2,22}$ results

in $F(b_2[-22], c_2, d_2) - F(b_2, c_2, d_2) = -2^{22}$, combined with $\Delta a_2 = 2^{22}$, which will lead to no change in a_3.

2) According to a) of Proposition 1 1), the condition $c_{2,21} = d_{2,21}$ ensures that the change of $b_{2,21}$ results in no change in a_3.

3) According to a) of Proposition 1 2), the condition $b_{2,17} = 0$ ensures that the change of $c_{2,17}$ results in no change in a_3.

4) According to a) of Proposition 1 3), the conditions $b_{2,2} = 1$, $b_{2,3} = 1$ and $b_{2,4} = 1$ ensure that the changes in the 2nd, 3rd and 4th bits of d_2 result in no change in a_3.

5) Because the shift is 3 in step 9, $\Delta a_2 = -2^{10}$ must lead to $\Delta a_3 = -2^{13}$, and the condition $a_{3,13} = 1$ results in $a'_{13} = a_{13}[-13]$.

6) Similarly, $\Delta a_2 = -2^{30}$ must lead to $\Delta a_3 = -2$, and the condition $a_{3,1} = 0$ and $a_{3,2} = 1$ result in $a'_3 = a_3[1, -2]$.

The above 10 conditions consist of a set of sufficient conditions for the differential path in step 9.

3.3 Message Modification

In order to improve the collision probability, we modify M_1 so that most of the sufficient conditions in Table 5 hold. The modification includes basic and advanced techniques. Because Extended MD4 has two branches, the modification is much more complicated than that of MD4, MD5, HAVAL, etc. which only contain one branch operation.

1) We modify M_1 word by word so that both branches with the modified M_1 satisfy almost all the conditions in the first round.

a) By using the basic modification technique, we modify m_{i-1} such that the i-th step conditions in the first round of left branch hold. For example, to ensure the eight conditions of d_1 in Table 5 hold, we modify m_1 as follows:

$d_1 \longleftarrow d_1 \oplus (d_{1,19} \lll 19) \oplus (d_{1,20} \lll 20) \oplus (d_{1,21} \lll 21) \oplus (d_{1,28} \lll 28) \oplus ((d_{1,27} \oplus 1) \lll 27) \oplus ((d_{1,6} \oplus a_{1,6}) \lll 6) \oplus ((d_{1,7} \oplus a_{1,7}) \lll 7) \oplus ((d_{1,8} \oplus a_{1,8}) \lll 8)$,

$m_1 \longleftarrow (d_1 \ggg 7) - d_0 - F(a_1, b_0, c_0)$.

b) By using the advanced modification technique, we modify the message word from low bit to high bit to correct the corresponding conditions in the first round of right branch.

• Firstly, we can correct the conditions by bit carry. For example, because there are no constraint conditions in $a_{1,18}$ in Table 5, we can correct $aa_{1,19} = 0$ to $aa_{1,19} = 1$ as follows. If $a_{1,18} = 0$ and $aa_{1,18} = 1$, let $m_0 \longleftarrow m_0 + 2^{15}$, then there is a bit carry in right branch and no bit carry in left branch, so the condition in $aa_{1,19}$ can be corrected, and the corrected $a_{1,19}$ will not be changed. Similarly, if $a_{1,18} = 1$ and $aa_{1,18} = 0$, let $m_0 \longleftarrow m_0 - 2^{15}$, then $aa_{1,19}$ can be corrected and $a_{1,19}$ will not be changed. If $a_{1,18} = aa_{1,18}$, we can use the lower bit carry to change $a_{1,18}$ or $aa_{1,18}$ such that $a_{1,18} \neq aa_{1,18}$, and then use the bit carry. The details for correcting $aa_{1,19}$ are given in Table 6.

• Secondly, we can correct the condition on $a_{i,j}$ by changing the corresponding variables in the previous steps. For example, we can correct $dd_{1,20} = 1$ to $dd_{1,20} = 0$ as follows. If $b_{0,13} \oplus c_{0,13} \neq bb_{0,13} \oplus cc_{0,13}$ (which means when $b_{0,13} = c_{0,13}$, then $bb_{0,13} \neq cc_{0,13}$; when $b_{0,13} \neq c_{0,13}$, then $bb_{0,13} = cc_{0,13}$), let $m_0 \longleftarrow m_0 \pm 2^{10}$, then $a_{1,13}$ and $aa_{1,13}$ will be changed, and the changed $a_{1,13}, aa_{1,13}$ only cause one of $d_{1,20}$ and $dd_{1,20}$ to change according to 1) of Proposition 1. Then if $d_{1,20} = dd_{1,20} = 1$, let $m_1 \longleftarrow m_1 - 2^{13}$, if $d_{1,20} = dd_{1,20} = 0$, modify the next bit of dd_1. Note that there is no condition in $a_{1,13}$ and $aa_{1,13}$ in Table 5, so the changes in $a_{1,13}$ and $aa_{1,13}$ do not invalidate the differential path. The details for correcting $dd_{1,20}$ are given in Table 7.

2) There are $18 \times 2 = 36$ conditions in total in the second round in both branches. We can utilize some more precise modification techniques to correct some conditions in the second round. Sometimes, it needs to add some extra conditions in the first round in advance such that the change of any condition does not affect all the corrected conditions.

Table 6. Message Modification for Correcting $aa_{1,19} = 0$ to $aa_{1,19} = 1$

Known Conditions	Modified m_0	New Chaining Variables
$a_{1,18} = 0, aa_{1,18} = 1$	$m_0 \longleftarrow m_0 + 2^{15}$	$aa_{1,19} = 1$, $a_{1,19}$ unchanged
$a_{1,18} = 1, aa_{1,18} = 0$	$m_0 \longleftarrow m_0 - 2^{15}$	$aa_{1,19} = 1$, $a_{1,19}$ unchanged
$a_{1,18} = aa_{1,18} = 1$	$m_0 \longleftarrow m_0 + 2^{14}$	$aa_{1,19} = 1$, $a_{1,19}$ unchanged
$a_{1,17} = 0, aa_{1,17} = 1$		
$a_{1,18} = aa_{1,18} = 1$	$m_0 \longleftarrow m_0 - 2^{14} - 2^{15}$	$aa_{1,19} = 1$, $a_{1,19}$ unchanged
$a_{1,17} = 1, aa_{1,17} = 0$		
$a_{1,18} = aa_{1,18} = 0$	$m_0 \longleftarrow m_0 + 2^{14} + 2^{15}$	$aa_{1,19} = 1$, $a_{1,19}$ unchanged
$a_{1,17} = 0, aa_{1,17} = 1$		
$a_{1,18} = aa_{1,18} = 0$	$m_0 \longleftarrow m_0 - 2^{14}$	$aa_{1,19} = 1$, $a_{1,19}$ unchanged
$a_{1,17} = 1, aa_{1,17} = 0$		

Table 7. Message Modification for Correcting $dd_{1,20} = 1$ to $dd_{1,20} = 0$

Step	Case	m_i	Shift	Modified m_i	Chaining Variables	Conditions
1		m_0	3	$m_0 \longleftarrow m_0 \pm 2^{10}$	$a_{1,13}$, $aa_{1,13}$ changed	
2	Case 1	m_1	7	$m_1 \longleftarrow m_1 - 2^{13}$	$d_{1,20}$ changed, i.e., $d_{1,20} = 1$	$b_{0,13} \neq c_{0,13}$
					$dd_{1,20}$ unchanged, i.e., $dd_{1,20} = 1$	$bb_{0,13} = cc_{0,13}$
					$d_{1,20} = 0, dd_{1,20} = 0$	
2	Case 2	m_2	7		$d_{1,20}$ unchanged, i.e., $d_{1,20} = 0$	$b_{0,13} = c_{0,13}$
					$dd_{1,20}$ changed, i.e., $dd_{1,20} = 0$	$bb_{0,13} \neq cc_{0,13}$

3.4 Overview of the Collision Attack Algorithm

From the above description, an overview of the collision attack algorithm on Extended MD4 can be expressed as follows.

1) Find a message block M_0 such that $h(M_0) = (a \parallel b \parallel c \parallel d \parallel aa \parallel bb \parallel cc \parallel dd)$ satisfies $b_i = c_i (i = 19, 21)$, $b_i = 0 (i = 20, 27, 28)$, $c_{20} = 1$, $bb_i = cc_i (i = 19, 21)$, $bb_i = 0 (i = 20, 27, 28)$ and $cc_{20} = 1$.

2) Repeat the following steps until we can find a message block M_1 which satisfies all the sufficient conditions in the first round of left branch and right branch in Table 5.

a) Select a random message block M_1.

b) Modify M_1 by step 1 of message modification described above.

c) Test if the hash value of M_1 satisfies all the sufficient conditions in the first round in both branches in Table 5.

3) Repeat the following steps until a collision $(M_0 \parallel M_1, M_0 \parallel M_1')$ is found.

a) Select random message words m_{14} and m_{15} of M_1.

b) Modify M_1 by step 1 of message modification described above such that all the conditions in c_4, b_4, cc_4 and bb_4 satisfied.

c) Modify M_1 by step 2 of message modification described above such that some conditions in the second round satisfied.

d) Then M_1 and $M_1' = M_1 + \Delta M_1$ satisfy all the sufficient conditions in both branches in Table 5 with the probability higher than 2^{-36}.

e) Test if the hash value of M_1 is equal to the hash value of M_1'.

It is easy to find proper M_0 in step 1 and to find M_1 which satisfies all the sufficient conditions in the first round in both branches in Table 5, and the complexity can be neglected. There are 36 conditions in total in the second round in both branches, so M_1 and M_1' lead to a collision with probability higher than 2^{-36}, and the complexity to find a collision $(M_0 \parallel M_1, M_0 \parallel M_1')$ is less than 2^{37} Extended MD4 computations. A collision for Extended MD4 can be seen in Table 8.

4 Strategies of the Pseudo-Preimage on the Compression Function of RIPEMD

Suppose CF is the compression function and y is a given targeting hash value, a pseudo-preimage is a pair (x, M) that satisfies $CF(x, M) = y$, where x is not required to be equal to the standard initial value. This section studies the pseudo-preimage resistance against the full RIPEMD. We choose proper neutral words for the first chunk and the second chunk. By constructing proper initial structure and applying the partial-fixing and partial-matching techniques, a few steps can be skipped. Combined with the exhaustive search, we can propose a pseudo-preimage on the full RIPEMD algorithm.

Table 8. Collision of Extended MD4

M_0	a4eff7cd 87afe33e b96f8657 1054fe49 8397de8d 23bc04b8 b683a020 3b2a5d9f c69d71b3 f9e99198 d79f805e a63bb2e8 45dd8e31 97e31fe5 2794bf08 b9e8c3e9
H_0	b5aac7e7 c1664fe2 01705583 ac3cc062 65c931e6 452829ae 527e12c7 30fafffb
M_1	54b7b7e1 65336f98 7621fe73 ffa42822 13dda7e1 1c28a008 bf20c341 3e8e28f2 578bdb87 afd42be4 b9ecab2e 0aaa9293 02e7070b eab6f4cf 2e96aaf7 5ab41efd
M_1'	54b8b7e1 65336f98 7621fe73 ffa42822 13dda7e1 1c28a008 bf20c341 3e8e28f2 578bdb87 afd42be4 b9ecab2e 0aaa9293 02e7070b eab6f4cf 2e96aaf7 5ab41efd
H	3055a689 7fe0b4a6 88d59251 af8afd0f 3826bda2 942f0939 c2673493 a6c56bac

Note: H_0 is the hash value for the message block M_0 with little-endian and no message padding. H is the common hash value for the message $M_0 \parallel M_1$ and $M_0 \parallel M_1'$ with little-endian and no message padding.

Left	0	1	2	$\boxed{3}$	4	5	6	7	8	9	10	11	12	13	14	15
								⟵ First Chunk ⟵								
	7	4	13	1	10	6	15	$\boxed{3}$	14	0	9	5	14	2	11	8
			⟵ First Chunk ⟵						*Start*$_1$	⟶ Initial Structure ⟵						
	$\boxed{3}$	10	2	4	9	15	8	1	14	7	0	6	11	13	5	12
	⟵	*Start*$_2$						⟶ Second Chunk ⟶								
Right	0	1	2	$\boxed{3}$	4	5	6	7	8	9	10	11	12	13	14	15
								⟶ First Chunk ⟶								
	7	4	13	1	10	6	15	$\boxed{3}$	14	0	9	5	14	2	11	8
			⟶ First Chunk ⟶							⟶ Match						
	$\boxed{3}$	10	2	4	9	15	8	1	14	7	0	6	11	13	5	12
								⟵ Exhaustive Search ⟵								

Fig.1. Overview of pseudo-preimage attack on RIPEMD.

4.1 Choice of Chunks and Neutral Words

As shown in Fig.1, we construct the initial structure in the left branch. We choose (m_3, b_{25}) as the neutral words for the first chunk, and (m_0, b_{33}) as the neutral words for the second chunk. The initial structure consists of eight steps in the left branch. The partial-fixing and partial-matching techniques enable us to skip seven steps in the right branch by considering two possible carried number patterns.

The first chunk is independent of (m_0, b_{33}). It starts from step 25 in the left branch with a backward computation, and ends with a forward computation until step 25 in the right branch because the initial values of both branches are identical. The second chunk is independent of (m_3, b_{25}), and it is from step 33 to step 48 in the left branch with a forward computation.

We can get partial bits of $aa_{32}(= bb_{29})$ from the computation of the first chunk. Performing the consistency check together with the partial-matching test and exhaustive search guarantees our attack hold even if the number of matched bits is small. See Fig.2 for a pictorial depiction of the attack, where *IS* means initial structure, and *ES* means exhaustive search.

4.2 Details of Initial Structure, Partial-Fixing and Partial-Matching

In Table 9, the initial structure is constructed in the left branch. (m_3, b_{25}) are the neutral words for the first chunk, and (m_0, b_{33}) are the neutral words for the second chunk. Considering the initial structure, partial-fixing and partial-matching techniques, we fix the the bit positions $27 \sim 24, 19 \sim 0$ of m_0, the bit positions $15 \sim 0$ of b_{25}, and all the other bits of m_0, b_{25} are free bits. All the 32 bits of m_3, b_{33} are free bits. We try all the values of free bits in the meet-in-the-middle attack.

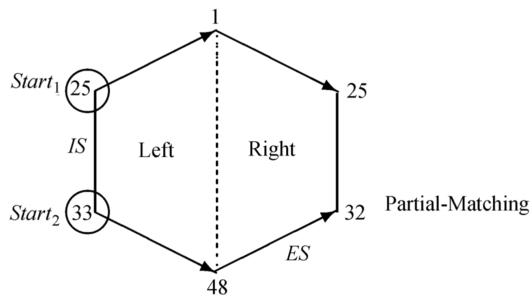


Fig.2. Pseudo-preimage attack on RIPEMD.

Let $x_{\rm 1st}$ represents a free bit of m_0 and $y_{\rm 2nd}$ represents a free bit of (m_3, b_{25}). The free bits $'y'_{\rm 2nd}$s in m_3 only impact b_{33} which is satisfied with probability 2^{-32} in the consistency check of the initial structure. Thus in order to construct the initial structure, the remaining work is to guarantee that $(a_{25}, b_{25}, c_{25}, d_{25})$ are independent of $'x'_{\rm 1st}$s, and $(a_{33}$, c_{33}, $d_{33})$ are independent of $'y'_{\rm 2nd}$s. We choose $a_{25} = -$5a827999, $c_{25} = 0$, $d_{25} = 0$ and $b_{25} = y_{\rm 2nd}^{31\sim16}\, 0^{15\sim0}$, which means that

Table 9. Initial Structure

i	m_i	a_i	b_i	c_i	d_i	Shift
25		$-k_1$	$y_{\mathrm{2nd}}^{31\sim16}0^{15\sim0}$	0	0	
26	$m_0=x_{\mathrm{1st}}^{31\sim28}0^{27\sim24}x_{\mathrm{1st}}^{23\sim20}0^{19\sim0}$	0	$m_0 \lll 12 = 0^{31\sim12}x_{\mathrm{1st}}^{11\sim8}0^{7\sim4}x_{\mathrm{1st}}^{3\sim0}$	$y_{\mathrm{2nd}}^{31\sim16}0^{15\sim0}$	0	12
27	$m_9=-k_1$	0	0	$0^{31\sim12}x_{\mathrm{1st}}^{11\sim8}0^{7\sim4}x_{\mathrm{1st}}^{3\sim0}$	$y_{\mathrm{2nd}}^{31\sim16}0^{15\sim0}$	15
28	$m_5=-k_1$	$y_{\mathrm{2nd}}^{31\sim16}0^{15\sim0}$	0	0	$0^{31\sim12}x_{\mathrm{1st}}^{11\sim8}0^{7\sim4}x_{\mathrm{1st}}^{3\sim0}$	9
29	$m_{14}=-k_1$	$0^{31\sim12}x_{\mathrm{1st}}^{11\sim8}0^{7\sim4}x_{\mathrm{1st}}^{3\sim0}$	$a_{28} \lll 7 = y_{\mathrm{2nd}}^{31\sim23}0^{22\sim7}y_{\mathrm{2nd}}^{6\sim0}$	0	0	7
30	$m_2=-k_1$	0	$a_{29} \lll 11 = 0^{31\sim23}x_{\mathrm{1st}}^{22\sim19}0^{18\sim15}x_{\mathrm{1st}}^{14\sim11}0^{10\sim0}$	$y_{\mathrm{2nd}}^{31\sim23}0^{22\sim7}y_{\mathrm{2nd}}^{6\sim0}$	0	11
31	$m_{11}=-k_1$	0	0	$0^{31\sim23}x_{\mathrm{1st}}^{22\sim19}0^{18\sim15}x_{\mathrm{1st}}^{14\sim11}0^{10\sim0}$	$y_{\mathrm{2nd}}^{31\sim23}0^{22\sim7}y_{\mathrm{2nd}}^{6\sim0}$	13
32	$m_8=-k_1$	$y_{\mathrm{2nd}}^{31\sim23}0^{22\sim7}y_{\mathrm{2nd}}^{6\sim0}$	0	0	$0^{31\sim23}x_{\mathrm{1st}}^{22\sim19}0^{18\sim15}x_{\mathrm{1st}}^{14\sim11}0^{10\sim0}$	12
33	$m_3=y_{\mathrm{2nd}}^{31\sim0}$	$0^{31\sim23}x_{\mathrm{1st}}^{22\sim19}0^{18\sim15}x_{\mathrm{1st}}^{14\sim11}0^{10\sim0}$	?	0	0	11

Note: ?: all possible values of b_{33}.

$(a_{25}, b_{25}, c_{25}, d_{25})$ is independent of $'x'_{\mathrm{1st}}$s. Therefore the remaining work is to guarantee that (a_{33}, c_{33}, d_{33}) can be computed independently of $'y'_{\mathrm{2nd}}$s in b_{25}. Let $m_0 = x_{\mathrm{1st}}^{31\sim28}0^{27\sim24}x_{\mathrm{1st}}^{23\sim20}0^{19\sim0}$, $m_3 = y_{\mathrm{2nd}}^{31\sim0}$, $b_{25} = y_{\mathrm{2nd}}^{31\sim16}0^{15\sim0}$, we give a detailed explanation in the following Algorithm 1.

Algorithm 1.

1) In step 26, we choose $c_{25} = 0$ and $d_{25} = 0$ such that the free bits $'y'_{\mathrm{2nd}}$s of b_{25} do not affect the variable b_{26}. Choose $a_{25} = -5\mathrm{a}827999$ to cancel the addition of 5a827999 for simplicity. Then the variable $b_{26} = (m_0 + a_{25} + G(b_{25}, c_{25}, d_{25}) + 5\mathrm{a}827999) \lll 12 = (x_{\mathrm{1st}}^{31\sim28}0^{27\sim24}x_{\mathrm{1st}}^{23\sim20}0^{19\sim0} + (-5\mathrm{a}827999) + G(b_{25}, 0, 0) + 5\mathrm{a}827999) \lll 12 = (x_{\mathrm{1st}}^{31\sim28}0^{27\sim24}x_{\mathrm{1st}}^{23\sim20}0^{19\sim0}) \lll 12 = 0^{31\sim12}x_{\mathrm{1st}}^{11\sim8}0^{7\sim4}x_{\mathrm{1st}}^{3\sim0}$, which is independent of $'y'_{\mathrm{2nd}}$s.

2) In step 27, we choose $m_9 = -5\mathrm{a}827999$ to cancel the addition of 5a827999 for simplicity. According to $b_{27} = (m_9 + a_{26} + G(b_{26}, c_{26}, d_{26}) + 5\mathrm{a}827999) \lll 15$, we can get $b_{27} = G(0^{31\sim12}x_{\mathrm{1st}}^{11\sim8}0^{7\sim4}x_{\mathrm{1st}}^{3\sim0}, y_{\mathrm{2nd}}^{31\sim16}0^{15\sim0}, 0) \lll 15$. According to the property of the nonlinear function G, we can get $b_{27} = 0$.

3) In step 28, we choose $m_5 = -5\mathrm{a}827999$, and according to $b_{28} = (m_5 + a_{27} + G(b_{27}, c_{27}, d_{27}) + 5\mathrm{a}827999) \lll 9$, we can get $b_{28} = 0$.

4) In step 29, we choose $m_{14} = -5\mathrm{a}827999$ to cancel the addition of 5a827999 for simplicity. According to $b_{29} = (m_{14} + a_{28} + G(b_{28}, c_{28}, d_{28}) + 5\mathrm{a}827999) \lll 7$, we can get $b_{29} = (a_{28} + G(0, 0, d_{28})) \lll 7 = a_{28} \lll 7 = y_{\mathrm{2nd}}^{31\sim23}0^{22\sim7}y_{\mathrm{2nd}}^{6\sim0}$.

5) In step 30, we choose $m_2 = -5\mathrm{a}827999$ to cancel the addition of 5a827999 for simplicity. According to $b_{30} = (m_2 + a_{29} + G(b_{29}, c_{29}, d_{29}) + 5\mathrm{a}827999) \lll 11$, we can get $b_{30} = (a_{29} + G(b_{29}, 0, 0)) \lll 11 = a_{29} \lll 11 = b_{26} \lll 11 = 0^{31\sim23}x_{\mathrm{1st}}^{22\sim19}0^{18\sim15}x_{\mathrm{1st}}^{14\sim11}0^{10\sim0}$.

6) In step 31, we choose $m_{11} = -5\mathrm{a}827999$ to cancel the addition of 5a827999 for simplicity. According to $b_{31} = (m_{11} + a_{30} + G(b_{30}, c_{30}, d_{30}) + 5\mathrm{a}827999) \lll 13$, we can get $b_{31} = G(0^{31\sim23}x_{\mathrm{1st}}^{22\sim19}0^{18\sim15}x_{\mathrm{1st}}^{14\sim11}0^{10\sim0}, y_{\mathrm{2nd}}^{31\sim23}0^{22\sim7}y_{\mathrm{2nd}}^{6\sim0}, 0) \lll 13$. According to the property of the nonlinear function G, we can get $b_{31} = 0$.

7) In step 32, we choose $m_8 = -5\mathrm{a}827999$, and according to $b_{32} = (m_8 + a_{31} + G(b_{31}, c_{31}, d_{31}) + 5\mathrm{a}827999) \lll 12$, we can get $b_{32} = 0$.

Thus, $a_{33} = bb_{30} = 0^{31\sim23}x_{\mathrm{1st}}^{22\sim19}0^{18\sim15}x_{\mathrm{1st}}^{14\sim11}0^{10\sim0}$, $c_{33} = bb_{32} = 0$, $d_{33} = bb_{31} = 0$ can be computed independently of the free bits $'y'_{\mathrm{2nd}}$s of b_{25}.

Owe to the bit positions $27 \sim 24, 19 \sim 0$ of m_0 are fixed, and $(aa_{25}, bb_{25}, cc_{25}, dd_{25})$ in the right branch can be computed in the first chunk, we can obtain partial bits of aa_{32} in the forward computation in the following Algorithm 2. Table 10 illustrates the partial-matching process.

Algorithm 2.

1) In the forward computation for step 26 in the right branch, because $bb_{26} = (aa_{25} + G(bb_{25}, cc_{25}, dd_{25}) + m_0) \lll 12$ and aa_{25}, bb_{25}, cc_{25}, dd_{25}, the bits positions $27 \sim 24, 19 \sim 0$ of m_0 are fixed, we can get two candidates of bits $31 \sim 12$ and $7 \sim 4$ of bb_{26} for each m_0 by considering two possible carried number patterns from the 23nd bit to the 24th bit.

2) In the forward computation for step 27, aa_{26}, cc_{26},

Table 10. Partial Matching

i	m_i	aa_i	bb_i	cc_i	dd_i	Shift
25		aa_{25}	bb_{25}	cc_{25}	dd_{25}	
26	$m_0[27 \sim 24, 19 \sim 0]$	dd_{25}	$[31 \sim 12, 7 \sim 4]$	bb_{25}	cc_{25}	12
27	m_9	cc_{25}	$[31 \sim 27, 22 \sim 19, 14 \sim 0]$	$[31 \sim 12, 7 \sim 4]$	bb_{25}	15
28	m_5	bb_{25}	$[31 \sim 28, 8 \sim 4]$	$[31 \sim 27, 22 \sim 19, 14 \sim 0]$	$[31 \sim 12, 7 \sim 4]$	9
29	m_{14}	$[31 \sim 12, 7 \sim 4]$	$[14 \sim 11, 6 \sim 3]$	$[31 \sim 28, 8 \sim 4]$	$[31 \sim 27, 22 \sim 19, 14 \sim 0]$	7
30	m_2	$[31 \sim 27, 22 \sim 19, 14 \sim 0]$		$[14 \sim 11, 6 \sim 3]$	$[31 \sim 28, 8 \sim 4]$	11
31	m_{11}	$[31 \sim 28, 8 \sim 4]$			$[14 \sim 11, 6 \sim 3]$	13
32	m_8	$[14 \sim 11, 6 \sim 3]$				12
32	m_8	aa_{32}	bb_{32}	cc_{32}	dd_{32}	12
33	m_3	aa_{33}	bb_{33}	cc_{33}	dd_{33}	11
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
48	m_{12}	aa_{48}	bb_{48}	cc_{48}	dd_{48}	5

dd_{26}, m_9 and bits $31 \sim 12, 7 \sim 4$ of bb_{26} are known. According to $bb_{27} = (aa_{26} + G(bb_{26}, cc_{26}, dd_{26}) + m_9) \lll 15$, we can get 2^2 candidates of bits $31 \sim 27, 22 \sim 19, 14 \sim 0$ of bb_{27} for each (m_0, bb_{26}) by considering two possible carried number patterns from the 3rd bit to the 4th bit and two possible carried number patterns from the 11th bit to the 12th bit.

3) In the forward computation for step 28, aa_{27}, dd_{27}, m_5, bits $31 \sim 27, 22 \sim 19$ of bb_{27} and bits $31 \sim 12, 7 \sim 4$ of cc_{27} are known. According to $bb_{28} = (aa_{27} + G(bb_{27}, cc_{27}, dd_{27}) + m_5) \lll 9$, we can get 2^2 candidates bb_{28} of bit positions $31 \sim 28, 8 \sim 4$ for each (m_0, bb_{27}, cc_{27}) by considering two possible carried number patterns from the 18th bit to the 19th bit and two possible carried number patterns from the 26th bit to the 27th bit.

4) In the forward computation for step 29, aa_{28}, m_{14}, bits $31 \sim 28, 8 \sim 4$ of bb_{28}, bits $31 \sim 27, 22 \sim 19, 14 \sim 0$ of cc_{28} and bits $31 \sim 12, 7 \sim 4$ of dd_{28} are known. According to $bb_{29} = (aa_{28} + G(bb_{28}, cc_{28}, dd_{28}) + m_{14}) \lll 7$, we can get 2^2 candidates bb_{29} of bit positions $14 \sim 11, 6 \sim 3$ for each $(m_0, bb_{28}, cc_{28}, dd_{28})$.

5) In the backward computation from step 48 to step 33 in the right branch, by exhaustively search m_0 and m_3, we can get the value $(aa_{32}, bb_{32}, cc_{32}, dd_{32})$.

So far, we have successfully obtained eight bits values of $aa_{32} = bb_{29}$ with seven unknown carried numbers in the forward computation, which is compared with the value aa_{32} computed from the backward direction.

5 Pseudo-Preimage Attack on the Full RIPEMD

In this section, we present the pseudo-preimage attack on the full RIPEMD compression function based on the initial structure and partial-matching proposed in Section 4.

5.1 Pseudo-Preimage Attack Procedure

The procedure of our pseudo-preimage attack on the full RIPEMD compression function is as follows.

1) Set chaining variables in the initial structure in the left branch as shown in Table 9. Set m_0, m_2, m_3, m_5, m_8, m_9, m_{11} and m_{14} as shown in Table 9.

2) Set randomly chosen values to other message words, and let m_{15} satisfy the padding.

3) For all possible values of m_3 and bit positions $31 \sim 16$ of b_{25}, in total 48 free bits, do the following:

a) Compute $(b_{25} \lll 7) + m_3$ for efficient consistency check. Denote this value as $C^{1\text{st}}$.

b) Compute from step 25 in the backward direction in the left branch until the initial state. The values of b_0, c_0, d_0 can be computed. $a_0 + m_0 = (b_1 \ggg 11) - F(b_0, c_0, d_0)$ can also be computed, denote this value as D.

c) Because the initial value of the right branch is the same as the initial value of the left branch, the value of bb_1 in the right branch can be computed as $bb_1 = (a_0 + m_0 + F(b_0, c_0, d_0) + \text{50a28be6}) \lll 11 = ((b_1 \ggg 11) + \text{50a28be6}) \lll 11$. Therefore, we obtain the internal state at step 1 in the right branch.

d) Compute from step 2 in the right branch in the forward direction until step 25 to get the internal state $(aa_{25}, bb_{25}, cc_{25}, dd_{25})$.

e) From Algorithm 2, we can obtain 8 bit positions $14 \sim 11, 6 \sim 3$ of aa_{32}.

f) Make a table of $(m_3, b_{25}, C^{1\text{st}}, aa_{32}, D, b_0, c_0, d_0)$.

4) For all possible values of bit positions $31 \sim 28, 23 \sim 20$ of m_0 and all bits of b_{33}, in total 40 free bits, do the following:

a) Compute a_{33} as shown in Table 9.

b) Compute $(b_{33} \ggg 11) - H(c_{33}, d_{33}, a_{33}) -$ 6ed9eba1 for the efficient consistency check. Denote the value as $C^{2\text{nd}}$.

c) Compute from step 33 until step 48 in the left branch to get the value $(a_{48}, b_{48}, c_{48}, d_{48})$.

d) i) Check whether $C^{2\text{nd}}$ is matched with $C^{1\text{st}}$ in the table.

ii) If they match, compute the value $aa_0 = a_0 = D - m_0$ by the value D in the table generated in f) of step 3. From a_0, b_0, c_0, d_0 in the table, we can get the initial value (aa_0, bb_0, cc_0, dd_0) of the right branch. Then compute the corresponding value of $(aa_{48}, bb_{48}, cc_{48}, dd_{48})$ according to the given target hash value and the initial value (aa_0, bb_0, cc_0, dd_0). Compute from step 48 to step 33 in the backward direction in the right branch to get aa_{32}. Check whether bit positions $14 \sim 11, 6 \sim 3$ of aa_{32} are matched in the table.

iii) If they match, for the remaining (m_0, b_{33}, m_3, b_{25}), compute aa_{31} in the backward direction, and check whether bits $31 \sim 28, 8 \sim 4$ of aa_{31} are matched.

iv) Similarly, for the remaining $(m_0, b_{33}, m_3,$ $b_{25})$, compute aa_{30}, aa_{29} and check the matching. If all bits are matched and the carried number assumptions are correct, a pseudo-preimage is found.

5) If no pseudo-preimage is found, change the values of the message words in step 2, and repeat steps 3∼4.

5.2 Complexity Evaluation

One step computation is regarded as $\frac{1}{48\times 2} = \frac{1}{96}$ RIPEMD compression function computation. The complexity of the attack can be evaluated as follows.

Step 1: Negligible.

Step 2: Negligible.

Step 3(a): $2^{48} \times \frac{1}{96}$ RIPEMD compression function.

Step 3(b): $2^{48} \times \frac{25}{96}$ RIPEMD compression function.

Step 3(c): $2^{48} \times \frac{1}{96}$ RIPEMD compression function.

Step 3(d): $2^{48} \times \frac{24}{96}$ RIPEMD compression function.

Step 3(e): $2^{48} \times 2 \times \frac{1}{96} + 2^{48} \times 2^3 \times \frac{1}{96} + 2^{48} \times 2^5 \times \frac{1}{96} + 2^{48} \times 2^7 \times \frac{1}{96}$ RIPEMD compression function.

Step 3(f): Negligible.

Step 4(a): $2^{40} \times \frac{1}{96}$ RIPEMD compression function.

Step 4(b): $2^{40} \times \frac{1}{96}$ RIPEMD compression function.

Step 4(c): $2^{40} \times \frac{15}{96}$ RIPEMD compression function.

Step 4(d) i: Negligible. The number of remaining pairs is $2^{56}(= 2^{48} \times 2^{40} \times 2^{-32})$.

Step 4(d) ii: $2^{56} \times \frac{16}{96}$ RIPEMD compression function. The number of remaining pairs is $2^{55}(= 2^{56} \times 2^{-8} \times 2^7)$.

Step 4(d) iii: $2^{55} \times \frac{1}{96}$ RIPEMD compression function. The number of remaining pairs is $2^{46}(= 2^{55} \times 2^{-9})$.

Step 4(d) iv: Approximately $2^{46} \times \frac{1}{96}$ RIPEMD compression function.

The overall complexity of the above computations is about $2^{56} \times \frac{16}{96}$. By performing the attack procedure $2^{72}(= 2^{128} \times 2^{32} \times 2^{-48} \times 2^{-40})$ times, we expect to get a pseudo-preimage. Therefore the overall complexity of finding a pseudo-preimage of RIPEMD is about $2^{125.4}(\approx 2^{72} \times 2^{56} \times \frac{16}{96})$ computations. The attack requires 2^{55} (m_3, b_{25}, $C^{1\text{st}}$, aa_{32}, D, b_0, c_0, d_0)s to be stored, and the memory complexity is $2^{55} \times 8 = 2^{58}$ words.

The padding rule forces some constraints on the last message words, and there are some restrictions on m_{14} in our attack. So the padding rule is an obstacle to extend the pseudo-preimage attack to preimage attack on RIPEMD.

6 Conclusions

In this paper, for two double-branch hash functions Extended MD4 and RIPEMD, we examined their security against collision attack and pseudo-preimage attack respectively. A practical attack on Extended MD4 for finding 2-block collision was proposed and a true collision instance of Extended MD4 was found. Based on 8-step initial structure, partial-fixing and partial-matching techniques, etc., a pseudo-preimage attack on RIPEMD was implemented.

A generic method to convert meet-in-the-middle preimage attack to pseudo collision attack was proposed in [55]. On one hand, it greatly improves the number of attacked steps of hash functions, compared with previous collision attack based on differentials and previous meet-in-the-middle preimage attack which needs to take into account padding bits. On the other hand, the method only converts the meet-in-the-middle preimage attack to a pseudo collision attack, not collision attack, and the time complexity is high. Future analysis should be able to explore the security of other double-branch hash functions, and to explore other practical relations between preimage attack and collision attack.

References

[1] Rivest R. The MD4 message digest algorithm. In *Proc. the 10th Int. Cryptology Conference* (*CRYPTO*), Aug. 1990, pp.303-311.

[2] Rivest R. The MD5 message-digest algorithm. 1992, http://www.ietf.org/rfc/rfc1321.txt.

[3] Zheng Y, Pieprzyk J, Seberry J. HAVAL—A one-way hashing algorithm with variable length of output. In *Proc. Workshop on the Theory and Application of Gyptographic Techniques: Advances in Cryptology* (*AUSCRYPT*), Dec. 1992, pp.81-104.

[4] Bosselaers A, Preneel B (eds.). Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evalution, Springer-Verlag, 1995.

[5] Dobbertin H, Bosselaers A, Preneel B. RIPEMD-160: A strengthened version of RIPEMD. In *Proc. the 3rd Int. Workshop on Fast Software Encryption*, Feb. 1996, pp.71-82.

[6] National Institute of Standards and Technology of USA. Secure hash standard. *Federal Information Processing Standard Publication*, FIPS-180, May 1993, http://www.mavi-1.org/web_security/cryptography/applied-crypto/fips180.txt.

[7] National Institute of Standards and Technology of USA. Secure hash standard. *Federal Information Processing Standards Publication*, FIPS-180-1, April 17, 1995, http://www.itl. nist.gov/fipspubs/fip/189-1.htm.

[8] National Institute of Standards and Technology of USA. Secure hash standard. *Federal Information Processing Standards Publication*, FIPS-180-2, August, 26, 2002, http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[9] Vaudenay S. On the need for multipermutations: Cryptanalysis of MD4 and SAFER. In *Proc. the 2nd Int. Workshop on Fast Software Encryption*, Dec. 1994, pp.286-297.

[10] den Boer B, Bosselaers A. Collisions for the compression function of MD5. In *Proc. Workshop on the Theory and Application of Cryptographic Techniques* (*EUROCRYPT*), May 1993, pp.293-304.

[11] Biham E, Chen R. Near-collisions of SHA-0. In *Proc. Int. Cryptology Conf.* (*CRYPTO*), Aug. 2004, pp.290-305.

[12] Biham E, Chen R, Joux A, Carribault P, Lemuet C, Jalby W. Collisions of SHA-0 and reduced SHA-1. In *Proc. the 24th Int. Conf. Theory and Applications of Cryptographic Techniques* (*EUROCRYPT*), May 2005, pp.36-57.

[13] Chabaud F, Joux A. Differential collisions in SHA-0. In *Proc. the 18th Int. Cryptology Conf.* (*CRYPTO*), Aug. 1998, pp.56-71.

[14] Dobbertin H. The first two rounds of MD4 are not one-way. In *Proc. the 5th Int. Workshop on Fast Software Encryption*, Mar. 1998, pp.284-292.

[15] Dobbertin H. Cryptanalysis of MD4. In *Proc. the 3rd Int. Workshop on Fast Software Encryption*, Feb. 1996, pp.53-69.

[16] Dobbertin H. Cryptanalysis of MD5 compress. In *Proc. Int. Conf. Theory and Application of Cryptology and Information Security* (*Rump Session*), May 1996, http://www.iacr.org/conferences/ec96/ec96rump.html.

[17] Dobbertin H. RIPEMD with two round compress function is not collision-free. *Journal of Cryptology*, 1997, 10(1): 51-70.

[18] Joux A. Collisions for SHA-0. In *Proc. of CRYPTO 2004* (*Rump Session*), Aug. 2004, http://www.iacr.org/conferences/ crypto2004/rump.html.

[19] Mendel F, Rechberger C, Rijmen V. Update on SHA-1. In *Proc. CRYPTO 2007* (*Rump Session*), Aug. 2007, http://rump2007.cr.yp.to.

[20] Rompay B, Biryukov A, Preneel B, Vandewalle J. Cryptanalysis of 3-pass HAVAL. In *Proc. the 9th Int. Conf. Theory and Application of Cryptology and Information Security* (*ASIACRYPT*), Nov. 30-Dec. 4, 2003, pp.228-245.

[21] Wang X Y, Feng D G, Lai X J, Yu H B. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. In *Proc. CRYPTO 2004 Rump Session*, Aug. 2004, http://www.iacr.org/conferences/crypto2004/rump.html.

[22] Wang X Y, Lai X J, Feng D G, Chen H, Yu X Y. Cryptanalysis of the hash functions MD4 and RIPEMD. In *Proc. the 24th Int. Conf. Theory and Applications of Cryptographic Techniques* (*EUROCRYPT*), May 2005, pp.1-18.

[23] Wang G L, Wang M Q. Cryptanalysis of reduced RIPEMD-128. *Journal of Software*, 2008, 19(9): 2442-2448.

[24] Wang X Y, Yu H B. How to break MD5 and other hash functions. In *Proc. the 24th Int. Conf. Theory and Applications of Cryptographic Techniques* (*EUROCRYPT*), May 2005, pp.19-35.

[25] Wang X Y, Yu H B, Yin L. Efficient collision search attacks on SHA-0. In *Proc. the 25th Int. Cryptology Conf.* (*CRYPTO*), Aug. 2005, pp.1-16.

[26] Wang X Y, Yin Y L, Yu H B. Finding collisions on the full SHA-1. In *Proc. the 25th Int. Cryptology Conf.* (*CRYPTO*), Aug. 2005, pp.17-36.

[27] Wang X Y, Feng D G, Yu X Y. An attack on hash function HAVAL-128. *Science in China Ser. F*: *Information Sciences*, 2005, 48(5): 545-556.

[28] Yu H B, Wang X Y, Yun A, Park S. Cryptanalysis of the full HAVAL with 4 and 5 passes. In *Proc. the 13th Int. Workshop on Fast Software Encryption*, Mar. 2006, pp.89-110.

[29] Yu H B, Wang X Y. Cryptanalysis of the compression function of SIMD. In *Proc. the 16th Australasian Conf. Information Security and Privacy*, Jul. 2011, pp.157-171.

[30] Yu H B, Chen J Z, Jia K T, Wang X Y. Near-collision attack on the step-reduced compression function of Skein-256. *IACR Cryptology ePrint Archive*, Report 2011/148, 2011, http://eprint.iacr.org/.

[31] Biham E, Shamir A. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 1991, 4(1): 3-72.

[32] Yu H B, Wang G L, Zhang G Y, Wang X Y. The second-preimage attack on MD4. In *Proc. the 4th Int. Conf. Cryptology and Network Security* (*CRYPTO*), Dec. 2005, pp.1-12.

[33] De Cannière C, Rechberger C. Finding SHA-1 characteristics: General results and applications. In *Proc. the 12th Int. Conf. Theory and Application of Cryptology and Information Security* (*ASIACRYPT*), Dec. 2006, pp.1-20.

[34] De Cannière C, Mendel F, Rechberger C. Collisions for 70-Step SHA-1: On the full cost of collision search. In *Proc. the 14th Int. Workshop. Selected Area in Cryptolography*, Aug. 2007, pp.56-73.

[35] Knudsen L R, Mathiassen J E. Preimage and collision attacks on MD2. In *Proc. the 12th Int. Conf. Fast Software Encryption*, Feb. 2005, pp.255-267.

[36] Muller F. The MD2 Hash function is not one-way. In *Proc. the 10th Int. Conf. Theory and Application of Cryptology and Information Security* (*ASIACRYPT*), Dec. 2004, pp.214-229.

[37] Aoki K, Sasaki Y. Preimage attacks on one-block MD4, 63-step MD5 and more. In *Proc. the 15th Int. Workshop. Selected Area in Cryptolography*, Aug. 2008, pp.103-119.

[38] De D, Kumarasubramanian A, Venkatesan R. Inversion attacks on secure Hash functions using SAT solvers. In *Proc. the 10th Int. Conf. Theory and Applications of Satisfiability Testing*, May 2007, pp.377-382.

[39] Guo J, Ling S, Rechberger C, Wang H. Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In *Proc. the 16th Int.*

Conf. Theory and Application of Cryptology and Information Security (*ASIACRYPT*), Dec. 2010, pp.56-75.

[40] Leurent G. MD4 is not one-way. In *Proc. the 15th Int. Conf. Fast Software Encryption*, Feb. 2008, pp.412-428.

[41] Zhong J M, Lai X J. Improved preimage attack on one-block MD4. *Journal of Systems and Software*, 2012, 85(4): 981-994.

[42] Sasaki Y, Aoki K. Finding preimages in full MD5 faster than exhaustive search. In *Proc. the 28th Int. Conf. Theory and Applications of Gryptolographic Techniques* (*EUROCRYPT*), Apr. 2009, pp.134-152.

[43] Sasaki Y, Aoki K. Preimage attacks on 3, 4, and 5-pass HAVAL. In *Proc. the 14th Int. Conf. Theory and Application of Cryptology and Information Security* (*ASIACRYPT*), Dec. 2008, pp.253-271.

[44] Sasaki Y, Aoki K. Meet-in-the-middle preimage attacks on double-branch hash functions: Application to RIPEMD and others. In *Proc. the 14th Australasian Conf. Information Security and Privacy*, Jul. 2009, pp.214-231.

[45] Wang G L, Wang S H. Preimage attack on Hash function RIPEMD. In *Proc. the 5th Int. Conf. Information Security Practice and Experience*, Apr. 2009, pp.274-284.

[46] Ohtahara C, Sasaki Y, Shimoyama T. Preimage attacks on step-reduced RIPEMD-128 and RIPEMD-160. In *Proc. the 6th Int. Conf. Information Security and Cryptology* (*INSCRYPT*), Oct. 2010, pp.169-186.

[47] Aoki K, Sasaki Y. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In *Proc. the 29th Int. Cryptology Conf.* (*CRYPTO*), Aug. 2009, pp.70-89.

[48] Aoki K, Guo J, Matusiewicz K, Sasaki Y, Wang L. Preimages for step reduced SHA-2. In *Proc. the 15th Int. Conf. Theory and Application of Cryptology and Information Security* (*ASIACRYPT*), Dec. 2009, pp.578-597.

[49] Mendel F, Pramstaller N, Rechberger C. A (Second) preimage attack on the GOST Hash function. In *Proc. the 15th Int. Conf. Fast Software Encryption*, Feb. 2008, pp.224-234.

[50] Khovratovich D, Rechberger C, Savelieva A. Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. In *Proc. the 19th Int. Conf. Fast Software Encryption*, Mar. 2012, pp.244-263.

[51] Sasaki Y. Meet-in-the-middle preimage attacks on AES hashing modes and an application to Whirlpool. In *Proc. the 18thInt. Conf. Fast Software Encryption*, Feb. 2011, pp.378-396.

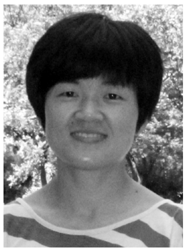
[52] Wu S, Feng D G, Wu W L, Guo J, Dong L, Zou J. (Pseudo) preimage attack on reduced-round Grøstl hash function and others. In *Proc. the 19th Int. Conf. Fast Software Encryption*, Mar. 2012, pp.127-145.

[53] Aoki K, Sasaki Y. Preimage attacks on one-block MD4, 63-step MD5 and more. In *Proc. the 15th Int. Workshop. Selected Area in Cryptology*, Aug. 2008, pp.103-119.

[54] Diffie W, Hellman M E. Exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 1977, 10(6): 74-84.

[55] Li J, Isobe T, Shibutani K. Converting meet-in-the-middle preimage attack into pseudo collision attack: Application to SHA-2. In *Proc. the 19th Int. Conf. Fast Software Encryption*, Mar. 2012, pp.264-286.

[56] Wang L, Sasaki Y, Komatsubara W, Ohta K, Sakiyama K. (Second) preimage attacks on step-reduced RIPEMD/RIPEMD-128 with a new local-collision approach. In *Proc. the 11th Int. Conf. Topics in Cryptology*, Feb. 2011, pp.197-212.



Gao-Li Wang is currently an associate professor in School of Computer Science and Technology, Donghua University, Shanghai. She got her B.S. degree in fundamental mathematics in 2003, and Ph.D. degree in cryptography in 2008, both from Shandong University of China, Jinan. Her research interests include cryptanalysis and design of hash functions and block ciphers.