Improved Preimage Attack on One-block MD4

Jinmin Zhong and Xuejia Lai

Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai 200240, China
jinminzhong@gmail.com,lai-xj@cs.sjtu.edu.cn

Abstract. We propose an improved preimage attack on one-block MD4 with the time complexity $2^{94.98}$ MD4 compression function operations, as compared to 2^{107} in [3]. We research the attack procedure in [3] and formulate the complexity for computing a preimage attack on one-block MD4. We attain the result mainly through the following two aspects with the help of the complexity formula. First, we continue to compute two more steps backward to get two more chaining values for comparison during the meet-in-the-middle attack. Second, we search two more neutral words in one independent chunk, and then propose the multi-neutral-word partial-fixing technique to get more message freedom and skip ten steps for partial-fixing, as compared to previous four steps. We also use the initial structure technique and apply the same idea to improve the pseudo-preimage and preimage attacks on Extended MD4 with $2^{25.2}$ and $2^{12.6}$ improvement factor, as compared to previous attacks in [20], respectively.

Key words: MD4, Extended MD4, meet-in-the-middle, preimage

1 Introduction

A cryptographic hash function takes an input message m of arbitrary length and produces an output h of fixed length, i.e., $h = \text{HASH}(m)$. Cryptographic hash functions have many important applications, such as digital signatures, message authentication codes, and random number generators. A cryptographic hash function should have the following three properties.

- Preimage resistance: given a hash value h, it is hard to find a message m so that $h = \text{HASH}(m)$.
- Second preimage resistance: given a message m_1, it is hard to find another message m_2 and $m_2 \neq m_1$ so that $\text{HASH}(m_2) = \text{HASH}(m_1)$.
- Collision resistance: it is hard to find two messages m_1 and m_2 ($m_1 \neq m_2$) so that $\text{HASH}(m_1) = \text{HASH}(m_2)$.

MD4, which was introduced by Rivest in 1990 [17], is a cryptographic hash function. Its design goals are security, speed, simplicity and compactness, and favoring little-endian architectures, respectively. The design philosophy of the most used cryptographic hash function, such as MD5, SHA-1 and SHA-2, even many of SHA-3 candidates, is original from MD4. Though the collision resistance of MD4 was broken in 1996 [9], MD4 has still been applied in many aspects because of its preimage resistance and speed. First, MD4 is used in ed2k URI scheme and it provides a unique identifier for a file in the popular eDonkey2000 /eMule P2P networks. Second, MD4 is also used in the rsync protocol. Third, MD4 is used in NT LAN Manager version one mainly employed in early versions of Windows NT. Fourth, MD4 is used in the S/KEY one-Time Password System.

1.1 Related Works

We mainly introduce collision attacks on MD4 and preimage attacks on MD4-family here. Merkle showed an unpublished collision attack on the first two rounds of MD4. Den Boer et al. presented a collision attack on the last two rounds of MD4 [8]. Vaudenay et al. also showed a collision attack on the first two rounds of MD4 [22]. The first collision attack on the full rounds of MD4 was presented by Dobbertin in 1996, as well as a collision of the slightly modified variant of Extended MD4, where both lines have the same initial state [9]. The very efficient collision attack on MD4 was published using the innovative method by Wang et al. in Eurocrypt 2005 [24]. Sasaki et al. improved the collision attack on MD4 using the different message difference in FSE 2007 [21]. Yu and Wang presented a new type multi-collision attack on the compression function of MD4 in ICISC 2007 [26].

The result that the first two rounds of MD4 is not one way was proposed by Dobbertin in FSE 1998 [10]. Kuwakado and Tanaka proposed a method to find preimage on reduced MD4 which only consists of the first and third rounds in 1999 [13]. Yu et al. presented a kind of second-preimage attack on MD4 in CANS 2005 [25]. But it is effective only for very long messages with low complexity. De et al. showed a preimage attack on 2 rounds and 7 steps of MD4 using SAT solvers [6]. The first preimage attack on the full MD4 was proposed by Leurent in FSE 2008 [15]. Its pseudo-preimage and preimage are with complexity of 2^{96} and $2^{100.5}$, respectively. In SAC 2008, Aoki and Sasaki presented preimage attacks on one-block MD4 with the complexity of 2^{107} and 63-step MD5 [3].

In SAC 2008, Aumasson et al. showed a preimage attack on 3-pass HAVAL [4]. In CRYPTO 2008, Cannière and Rechberger presented preimage attacks on 49 steps of SHA-0 and 44 steps of SHA-1 [7]. Wang et al. presented a preimage attack on the first 29 steps of RIPEMD in ISPEC 2009 [23]. Sasaki and Aoki showed preimage attacks on 3-, 4-, and 5-pass HAVAL in ASIACRYPT 2008, full MD5 in EUROCRYPT 2009, and the first 33 and intermediate 35 steps of RIPEMD, full Extended MD4, et al. in ACISP 2009 [18,19,20]. Aoki and Sasaki proposed preimage attacks on 52 steps of SHA-0 and 48 steps of SHA-1 in CRYPTO 2009 [2]. Isobe and Shibutani showed preimage attacks on 24 steps of SHA-2 in FSE 2009 [12]. Aoki et al. presented preimage attacks on 43 steps of SHA-256 and 46 steps of SHA-512 in ASIACRYPT 2009 [1].

Recently, Guo et al. presented preimage attack on MD4 with the time complexity $2^{99.7}$ and memory requirements 2^{64} words [11]. Their time complexity is $2^{78.4}$ and memory requirements are 2^{81} words if 2^{128} precomputation is provided. Both of their preimage length is equal or greater than 2^{50} blocks. They also showed second preimage attack on MD4 with the time complexity $2^{99.7}$ and memory requirements 2^{64} words. Suppose 2^{128} precomputation is provided, the time complexity for second preimage is $2^{69.4}$ and memory requirements are 2^{72} words. Both of their second preimage length is equal or greater than 3 blocks.

1.2 Our Results

Table 1. Comparison of preimage attacks against MD4 and Extended MD4

Hash	Attack	Pseudo-preimage	Preimage	Preimage Length	Memory (words)
MD4	[15]	2^{96}	2^{102}	Twenty Blocks	2^{37}
	[15]	2^{96}	$2^{100.5}$	Tens of Blocks	2^{36}
	[11]	2^{72}	$2^{99.7}$	$\geq 2^{50}$ Blocks	2^{64}
	[11]	2^{72}	$2^{78.4}$, need 2^{128} precomputation	$\geq 2^{50}$ Blocks	2^{81}
	[3]		2^{107}	One-block	$2^{21} \times 3$
	Our Results		$2^{94.98}$	One-block	$2^{35} \times 7$
Extended MD4	[20]	2^{229}	$2^{243.5}$	Two-block	$2^{27} \times 11$
	Our Results	$2^{203.8}$	$2^{230.9}$	Two-block	$2^{52} \times 12$

We research the preimage attack procedure on one-block MD4 in [3], analyze in detail how the attack complexity is acquired, and formulate the complexity for computing a preimage attack on one-block MD4, and then get an observation that we can continue to compute two more steps during backward computation to get two more chaining values used for partial-matching additionally. This can improve the complexity with $2^{3.5}$ factor, as compared to the original one in [3].

We find two more new neutral words additionally in the second message chunk. In order to utilize the new neutral words, we propose the multi-neutral-word partial-fixing technique. This can skip ten steps during the partial-matching and get more message freedom used for the meet-in-the-middle attack. The improvement by the method predominates in our results.

We apply the partial-matching and indirect partial-matching techniques simultaneously to match the chaining values during the meet-in-the-middle attack. This slightly improves the complexity with less than 2^1 factor.

We improve the preimage attack on one-block MD4 with the complexity $2^{94.98}$ using the above three methods.

We use the initial structure technique to get one more neutral word and apply the same idea to improve the preimage attack on Extended MD4 in [20]. This can get the improvement with $2^{25.2}$ and $2^{12.6}$ factor, as compared to the previous pseudo-preimage and preimage attacks on Extended MD4 in [20], respectively. Meanwhile, we provide a corrected version of the swapping function of the compression function of Extended MD4 in [20].

A summary of our results and previously published results is provided in Table 1.

2 Specification of MD4 and Extended MD4

2.1 Specification of MD4

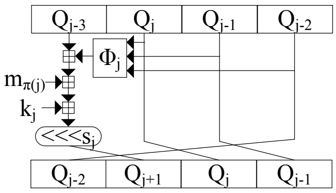

Fig. 1. The state update function of MD4.

The MD4 algorithm takes an input message whose length can be arbitrary and produces an output 128-bit hash value. First of all, the input message is padded, i.e., the message is appended a single 1 and some 0 so that the length of the padded message becomes congruent to 448, modulo 512. Then, the length of the original message before the padded bits were added is appended to the already padded message. The MD4 algorithm uses the Merkle-Damgård iterated structure [5,16]. The compression function of MD4 has three rounds and every round has 16 steps. Every step uses equation (1) to update the chaining variable. $\pi(j)$ is a function of MD4 message expansion. ϕ_j, s_j, and k_j are bitwise Boolean function, const value, and left rotation, respectively. Equation (2) is contrary for equation (1). $\pi(j)$ and ϕ_j are shown in Table 2. s_j and k_j are shown in Table 3. IV is the initial value defined in the specification. Equation (3) represents a run procedure of compression function of MD4. R_j is the state update function of MD4 as shown in Fig.1. Here, p_j is an internal state and p_j=(Q_{j-3}, Q_j, Q_{j-1}, Q_{j-2}).

$$Q_{j+1} = (Q_{j-3} + \phi_j(Q_j, Q_{j-1}, Q_{j-2}) + m_{\pi(j)} + k_j) \lll s_j, \quad 0 \le j \le 47. \quad (1)$$

$$Q_{j-3} = (Q_{j+1} \ggg s_j) - \phi_j(Q_j, Q_{j-1}, Q_{j-2}) - m_{\pi(j)} - k_j, \quad 0 \le j \le 47. \quad (2)$$

$$IV = (Q_{-3}||Q_0||Q_{-1}||Q_{-2}) = (0x67452301||0xefcdab89||0x98badcfe||0x10325476).$$

$$\begin{cases} p_0 = H_i, \\ p_{j+1} = R_j(p_j, m_{\pi(j)}), for\ j = 0, 9, ..., 47, \\ H_{i+1} = p_{48} + p_0. \end{cases} \quad (3)$$

Notation $\overset{b-a}{m_i}$ (or $\overset{b-a}{Q_i}$) denotes the bits between the b^{th} and a^{th} bits of m_i (or Q_i), and $\overset{all}{m_i}$ (or $\overset{all}{Q_i}$) denotes the bits between the 31^{st} and 0^{th} bits of m_i (or Q_i). Note that we start counting from the least significant bit.

Table 2. Message expansion and Boolean functions of MD4

j	$\pi(j)$	ϕ_j
$0\ 1 \cdots\ 15$	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	$(X \wedge Y) \vee (\neg X \wedge Z)$
$16\ 17 \cdots\ 31$	0,4,8,12,1,5,9,13,2,6,10,14,3,7,11,15	$(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$
$32\ 33 \cdots\ 47$	0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15	$X \oplus Y \oplus Z$

Table 3. Number of rotation and Magic constants of MD4

j	s_j	k_j
$0\ 1 \cdots\ 15$	3,7,11,19,3,7,11,19,3,7,11,19,3,7,11,19	$K_0 = 0$
$16\ 17 \cdots\ 31$	3,5,9,13,3,5,9,13,3,5,9,13,3,5,9,13	$K_1 = 0x5a827999$
$32\ 33 \cdots\ 47$	3,9,11,15,3,9,11,15,3,9,11,15,3,9,11,15	$K_2 = 0x6ed9eba1$

2.2 Specification of Extended MD4

Rivest also proposed an extension version of MD4 with 256-bit hash values, which is called Extended MD4 in [9], in order to satisfy higher security [17]. The compression function of Extended MD4 is made up of two copies of compress function of MD4 running in parallel. The first copy is the standard compression function of MD4. The second copy differs only on the initial state and the magic constants.

The initial state of the second copy is:

$$IV^{'} = (0x33221100||0x77665544||0xbbaa9988||0xffeeddcc).$$

The magic constants of the second copy is:

$$K_0^{'} = 0, K_1^{'} = 0x50a28be6, K_2^{'} = 0x5c4dd124.$$

The values of the A registers in the two copies are exchanged at the end of the compression function. The final hash value is produced by concatenating the results of the two copies. In order to explain the above exchange process, we suppose that the output value of left copy of the compression function of Extended MD4 is $(A_0^L + A_{48}^L,\ B_0^L + B_{48}^L,\ C_0^L + C_{48}^L,\ D_0^L + D_{48}^L)$ and the output value of right copy is $(A_0^R + A_{48}^R,\ B_0^R + B_{48}^R,\ C_0^R + C_{48}^R,\ D_0^R + D_{48}^R)$. Then, after exchanged, the result of the compression function of Extended MD4 is $(A_0^R + A_{48}^R,\ B_0^L + B_{48}^L,\ C_0^L + C_{48}^L,\ D_0^L + D_{48}^L,\ A_0^L + A_{48}^L,\ B_0^R + B_{48}^R,\ C_0^R + C_{48}^R,\ D_0^R + D_{48}^R)$.

In [20], the description about the exchange process in the two copies of the compression function of Extended MD4, that the values of Q_{16}, Q_{32} and Q_{48} in the two copies are exchanged, does not correspond to the one in [17]. But this hardly influence the complexity.

The swapping function of the compression function of Extended MD4, which is used for interaction between two copies, is different from those of RIPEMD-256 and RIPEMD-320.

3 Related Techniques

Here, we introduce some related techniques.

3.1 Meet-in-the-Middle Attack

The meet-in-the-middle attack is a type of birthday attack and makes use of a space-time tradeoff. The unbalanced meet-in-the-middle attack was proposed first in [14]. The compression function computes forward to the given step and gets a set of results, and then the compression function computes backward and gets another set of results. The two sets of results are compared to search a intersection. The two computation procedures must be independent on each other so that the birthday attack rule can be applied.

3.2 Converting Pseudo-Preimage Attack into Preimage Attack

The method of converting a pseudo-preimage attack into a preimage attack was proposed first in [14]. Because the initial chaining value of pseudo-preimage is not the fixed IV, we hash a message block for connecting the fixed IV with the initial chaining value. Searching the proper message block accords with the birthday attack rule. If it takes 2^k complexity to produce a pseudo-preimage in a n-bit iterated hash function, then the total complexity to produce a preimage is $2^{1+\frac{n+k}{2}}$.

3.3 Splice-and-Cut Technique

The splice-and-cut technique is proposed first in [3]. During a preimage attack, the hash value is given, and thus it is a constant. Therefore, the final output state subtracted from the hash value is the initial internal state in the Davies-Meyer mode. Thus we can regard the first and last steps as consecutive steps, and then any step can be considered a starting step in the meet-in-the-middle attack.

3.4 Initial Structure Technique

The initial structure technique was proposed first in [19]. The neutral words for the two message chunks may mix in the beginning of the attack point so as to be difficult to find two independent message chunks during the meet-in-the-middle attack. This technique is used to solve the problem and helps to search better two independent message chunks for a more successful meet-in-the-middle attack.

3.5 Partial-Matching, Partial-Fixing and Indirect-Partial-Matching Techniques

The partial-matching and partial-fixing techniques are also presented first in [3]. The state update function does not update all of chaining variables of an internal state. The partial-matching technique makes use of the property so that several steps between two independent message chunks can be skipped.

If we can fix some bits of a neutral word in order to be able to continue to compute, such technique is called the partial-fixing technique. The partial-fixing technique is the partial-matching technique in nature.

The indirect-partial-matching technique, which is proposed first in [1], is an extension of the partial-matching technique. If the computation of a matching point can be decomposed the independent computation of the functions of the neutral words, the indirect-partial-matching technique can be applied.

Table 4. Message word distribution for one-block MD4 in [3]

Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
index	0	1	2	③	4	5	6	⑦	⑧	9	10	11	12	13	14	15
	⟵ first chunk ⟵								⟶ second chunk ⟶							
Step	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
index	0	4	⑧	12	1	5	9	13	2	6	10	14	③	⑦	11	15
	⟶ second chunk ⟶												skip			
Step	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
index	0	⑧	4	12	2	10	6	14	1	9	5	13	③	11	⑦	15
	skip		⟵ first chunk ⟵													

4 How to Improve the Preimage Attack on One-block MD4

The message word distribution for one-block MD4 in [3] is shown in Table 4. In Table 4, the neutral word for the first chunk is m_8 ($\overset{31-11}{m_8}$ are free). The neutral words for the second chunk are m_3 and m_7 ($\overset{all}{m_7}$ are free). But the value of m_3

Table 5. Our Message word distribution for one-block MD4

Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
index	0	1	2	③	4	5	6	⑦	⑧	9	⑩	11	⑫	13	14	15
	⟵ first chunk ⟵								⟶ second chunk ⟶							
Step	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
index	0	4	⑧	⑫	1	5	9	13	2	6	⑩	14	③	⑦	11	15
	⟶ second chunk ⟶												skip			
Step	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
index	0	⑧	4	⑫	2	⑩	6	14	1	9	5	13	③	11	⑦	15
	skip						⟵ first chunk ⟵									

completely depends on that of m_7 in order to form a local collision in the first round. Therefore, the neutral word for the second chunk is only m_7 in nature. The general procedure of preimage attack on one-block MD4 in [3] is as follows. We compute forward using all possible values of m_8 (thereinto, $\overset{31-11}{m_8}$ are used to compute as the free bits, and $\overset{10-0}{m_8}$ are fixed for partial-fixing) as a variable, and store the values of $(\overset{all}{Q_{27}}, \overset{all}{Q_{28}}, \overset{all}{m_8})$ into the table L, then 2^{21} pairs are produced. Then, we compute backward using m_7 ($\overset{all}{m_7}$ are free) as a variable and $\overset{7-0}{m_8}$ as known values, and we get the values of $(\overset{10-0}{Q_{27}}, \overset{10-0}{Q_{28}})$ which are used to match with the corresponding values in the table L.

Here, we introduce two propositions. The key idea of our improvements is from the following two propositions.

Proposition 1. *If any four successive chaining variables $(Q_i, Q_{i+1}, Q_{i+2}, Q_{i+3})$ for the forward and backward computation are matched successfully during the meet-in-the-middle attack on MD4, the chaining variables for the forward computation succeed in connecting to the ones for backward computation.*

Proof. According to equation (1), the value of Q_{i+4} can be fixed on by the values of Q_i, Q_{i+1}, Q_{i+2}, and Q_{i+3} during the forward computation. Similarly, the values of Q_{i+5}, Q_{i+6}, $\cdots$, Q_{48} can be fixed on.

According to equation (2), the value of Q_{i-1} can be fixed on by the values of Q_i, Q_{i+1}, Q_{i+2}, and Q_{i+3} during the backward computation. Similarly, the values of Q_{i-2}, Q_{i-3}, $\cdots$, Q_0 can be fixed on.

Proposition 2. *For one-block MD4, let f_1 be the number of free bits of neutral words for the second chunk, b_1 be the number of free bits of neutral words for the first chunk. Let f be the total number of all free bits of neutral words, then*

$$f = f_1 + b_1. \tag{4}$$

Let n_1 be the number of the steps in the first chunk, n_2 be the number of the steps in the second chunk. Let t_1 be number of the matched bits for the first *matching, t_i be number of the matched bits for the i^{th} matching, a_i be number of the computing steps between the end of the $(i-1)^{th}$ matching phase and the start of the i^{th} matching phase. The time complexity of the first attack procedure (before repeated) is 2^r, then*

$$2^r = 2^{f_1} \times \frac{n_1}{48} + 2^{b_1} \times \frac{n_2}{48} + 2^{f_1+b_1-t_1} \times \frac{a_1}{48} + \cdots + 2^{f_1+b_1-\sum_{j=1}^{i} t_j} \times \frac{a_i}{48} + \cdots . \quad (5)$$

The computation in the right side of equation (5) ends with the condition that any four successive chaining values (Q_i, Q_{i+1}, Q_{i+2}, Q_{i+3}) are compared, or the quantity of $2^{f_1+b_1-\sum_{j=1}^{i} t_j} \times \frac{a_i}{48}$, compared to the complexity of other steps, is negligible. The total time complexity to produce a preimage for one-block MD4 is

$$C_{preimage\_complexity} = 2^{128-f+r}. \quad (6)$$

Proof. According to the procedure in the meet-in-the-middle attack, we know the first attack procedure is made up of the computation of the first chunk (i.e. backward computation), the computation of the second chunk (i.e. forward computation), and the computation of the matching procedure. If any four successive chaining variables (Q_i, Q_{i+1}, Q_{i+2}, Q_{i+3}) are matched entirety, the meet-in-the-middle attack succeeds, otherwise, it fails and needs to repeat the attack procedure. Furthermore, if the quantity of $2^{f_1+b_1-\sum_{j=1}^{i} t_j} \times \frac{a_i}{48}$, compared to the complexity of other steps, is negligible, the quantity is not needed to add to the time complexity. Therefore, the time complexity of the first attack procedure (before repeated) is shown by equation (5). The attack is statistical. Because the number of the bits of the chaining variables (Q_i, Q_{i+1}, Q_{i+2}, Q_{i+3}) is 128, and the number of total free bits is f, the first attack procedure needs to repeat 2^{128-f} times. Thus the total time complexity to produce a preimage for one-block MD4 is shown by equation (6).

We should make $-f + r$ as little as possible according to Proposition 2, moreover we can compute the optimal complexity by the program, as shown in Appendix A, according to Proposition 2.

We explain how to improve the complexity through the following three ways. The following three ways can be used independently, and also be combined to improve the complexity.

4.1 Improvement One

According to Proposition 1, we can try to store the partial values of (Q_{25}, Q_{26}, Q_{27}, Q_{28}) for comparing, instead of those of (Q_{27}, Q_{28}), in order to improve the complexity. There is a statement in [3], that *"In MD4, up to four steps can be additionally skipped by the partial-fixing technique "*. But we discover that six steps can be additionally skipped by the partial-fixing technique, i.e., we can compute the partial values of Q_{30}, Q_{29}, Q_{28}, Q_{27}, Q_{26} and Q_{25} using the partial value of m_8 by the partial-fixing technique. In [3], they only compute the partial values of Q_{30}, Q_{29}, Q_{28} and Q_{27} using the partial value of m_8. We conjecture that the reason why the partial values of Q_{26} and Q_{25} are not continued to compute in [3] is that the neutral word m_7 is met when Q_{26} is computed. Though m_7 is a neutral word for the second chunk, the value of m_7 is known during the backward computation, so we can still compute Q_{26} using m_7. We combine the above observation with the idea from Proposition 2 to improve the complexity. We compute forward using all possible values of m_8 (thereinto, $\overset{31-8}{m_8}$ are free and $\overset{7-0}{m_8}$ are fixed for partial-fixing) as a variable, and store the values of ($\overset{all}{Q_{25}}$, $\overset{all}{Q_{26}}$, $\overset{all}{Q_{27}}$, $\overset{all}{Q_{28}}$, $\overset{all}{m_8}$) into the table L, where 2^{24} pairs produced are saved. Then, we compute two more steps backward to get the values of $\overset{2-0}{Q_{25}}$ and $\overset{2-0}{Q_{26}}$, thus we can compare ($\overset{2-0}{Q_{25}}$, $\overset{2-0}{Q_{26}}$, $\overset{7-0}{Q_{27}}$, $\overset{7-0}{Q_{28}}$) with the corresponding values in the table L. This can improve the complexity with $2^{3.5}$ factor compared to the original one in [3].

4.2 Improvement Two

We know that the neutral word m_7 for the second chunk has 32 free bits, while the neutral word m_8 for the first chunk only has 24 free bits, in Table 4. The number of free bits of m_8, which is less than that of m_7, is a big bottleneck for improve the complexity according to Proposition 2. We should manage to increase the number of the free bits of the neutral words for the first chunk according to Proposition 2 if we want to improve the complexity about preimage attack on one-block MD4 further. Based on the above idea, we try to search new neutral words for the first chunk. There are only six message words, i.e., (m_3, m_7, m_{11}, m_{15}, m_0, m_8), in the skip interval in Table 4, where m_3, m_7 and m_8 have been used as neutral words. Thus, we test other message words, i.e., (m_{11}, m_{15}, m_0), and find they can not be used as neutral words. Therefore, we have no choice but to extend the skip interval. We add m_4, which is next to m_8 in the third round, to the skip interval. Then we test m_4, and find that it cannot be used as a neutral word for the first chunk. Anyway, we continue to test m_{12}, which is next two steps to m_8 in the third round. Luckily, m_{12} can be used as a neutral word for the first chunk. But we meet another problem that m_{12} is next two steps to m_8 in the third round, and how do we overcome the two steps? We try to use the partial-fixing technique to deal with it, i.e., some bits of m_{12} are fixed and the other bits of m_{12} are used as free bits so that we can continue to compute backward with the fixed bits of m_{12} and get the free bits of m_{12} for improving the meet-in-the-middle attack. As a result, it is feasible. In the same way, m_{10} can also be used as a neutral word for the first chunk. Therefore, the sum of the free bits which m_8, m_{12} and m_{10} can provide is much more than the one which m_8 can do. Because m_8, m_{12} and m_{10} are used as neutral words for the first chunk, moreover, all of them are used by the partial-fixing. We call the technique *multi-neutral-word partial-fixing*. The *multi-neutral-word partial-fixing technique* is a variation of the partial-fixing technique, where multiple neutral words in one message chunk are partially fixed to skip more steps for the matching-part in the meet-in-the-middle attack and get more message free bits to improve the meet-in-the-middle attack. We can additionally skip ten steps for the matching-part for the preimage attack on one-block MD4 (even eleven steps for the matching-part for the preimage attack on Extended MD4) using the multi-neutral-word partial-fixing technique, instead of four steps claimed in [3]. We may find more neutral words and better message chunks using the multi-neutral-word partial-fixing technique. These results are shown in Tables 5 and 6.

4.3 Improvement Three

We apply the partial-matching and indirect-partial-matching techniques simultaneously to compare the chaining variables during the meet-in-the-middle attack. Though this can only slightly improve the complexity, we first show that the two techniques can be applied simultaneously.

$$Q_{29} = (Q_{25} + \phi_{28}(Q_{28}, Q_{27}, Q_{26}) + t_2 + m_3) \lll 3. \tag{7}$$

We cannot compute Q_{29} during the forward computation according to equation (7) as m_3 is an unknown value. But we can compute $X{=}Q_{25}{+}\phi_{28}(Q_{28}, Q_{27}, Q_{26}){+}t_2$. Then we can save $(Q_{26}, Q_{27}, Q_{28}, X)$, instead of $(Q_{25}, Q_{26}, Q_{27}, Q_{28})$, in the table L. The procedures are shown in Steps 3b and 3c in Section 5.1. Note that we can not store $(Q_{25}, Q_{26}, Q_{27}, Q_{28}, X)$ into the table L for comparing. It is because X is derived from Q_{25}, Q_{26}, Q_{27} and Q_{28}, and the entropy of X exists those of Q_{25}, Q_{26}, Q_{27} and Q_{28}.

The following equation (8) is from Step 4i in Section 5.1.

$$\overset{13-0}{Q_{29}} = (\overset{22-0}{Q_{33}} \ggg 3) - \phi_{32}(\overset{22-0}{Q_{32}}, \overset{22-0}{Q_{31}}, \overset{13-0}{Q_{30}}) - \overset{all}{m_0} - \overset{all}{t_3} \tag{8}$$

Some bits of Q_{29} can be computed according to equation (8) and m_3 is a known value during the backward computation. Let $\overset{10-0}{X} = (\overset{13-0}{Q_{29}} \ggg 3) - \overset{all}{m_3}$ such that we can compare $(\overset{8-0}{Q_{26}}, \overset{9-0}{Q_{27}}, \overset{9-0}{Q_{28}}, \overset{10-0}{X})$, instead of $(\overset{8-0}{Q_{25}}, \overset{8-0}{Q_{26}}, \overset{9-0}{Q_{27}}, \overset{9-0}{Q_{28}})$, with $(\overset{all}{Q_{26}}, \overset{all}{Q_{27}}, \overset{all}{Q_{28}}, \overset{all}{X})$ in the table L. The procedures are shown in Steps 4i $\sim$ 4n in Section 5.1.

The method can provide two advantages. One advantage is that we can compare more known bits of X ($\overset{10-0}{X}$ are known) than those of Q_{25} ($\overset{8-0}{Q_{25}}$ are known). The other is that we can take less cost as X is computed and compared more earlier than Q_{25} during the backward computation.

5 Preimage Attack on One-block MD4

5.1 Attack Procedure of One-block MD4

Assume that the given hash value is H.

1. Set $p_{48} = H - IV(p_0)$.
   Set C_1, C_2, C_3 and C_4 random values.
   Set the MSB of m_{13} one, the others bits of m_{13} random value, $m_{14} = 447$ and $m_{15} = 0$ to satisfy the padding rule for one-block message.
   Set $\overset{14-0}{m_8}$, $\overset{22-0}{m_{10}}$, $\overset{22-0}{m_{12}}$ and $m_i (i \in \{9, 11\})$ random values.
   Compute $p_{47} = R_{47}^{-1}(p_{48}, m_{15})$. (Note p_{48} and m_{15} are known.)
2. Compute the following equation,
   (a) $m_0 = (C_4 \ggg s_0) - Q_{-3} - \phi_0(Q_0, Q_{-1}, Q_{-2})$
   (b) $m_1 = (C_3 \ggg s_1) - Q_{-2} - \phi_1(C_4, Q_0, Q_{-1})$
   (c) $m_2 = (C_3 \ggg s_2) - Q_{-1} - \phi_2(C_3, C_4, Q_0)$
   (d) $m_4 = (0 \ggg s_4) - C_4 - C_3$
   (e) $m_5 = 1 - C_3 - C_3 = 1 - 2C_3$
   (f) $m_6 = (C_2 \ggg s_6) - C_3$
3. For all possible values of m_8, m_{10} and m_{12} (thereinto, $\overset{31-15}{m_8}$, $\overset{31-23}{m_{10}}$, and $\overset{31-23}{m_{12}}$ are free, so there are 35 (= 17 + 9 + 9) free bits in total), do the following,
   (a) $p_{j+1} = R_j(p_j, m_{\pi(j)})$, for $j = 8, 9, ..., 27$.
   (b) Let $X = Q_{25} + \phi_{28}(Q_{28}, Q_{27}, Q_{26}) + t_2$ (Note $Q_{29} = (X + m_3) \lll 3$)
   (c) make the table L which saves (m_8, m_{10}, m_{12}, Q_{26}, Q_{27}, Q_{28}, X), and the size of the table L is $2^{35} \times 7$ words.
4. For all possible values m_7 ($\overset{all}{m_7}$, 32 free bits in total), do the following,
   (a) $* = (C_1 \ggg s_7) - \phi_7(C_2, 1, 0) - m_7$
   (b) $m_3 = (* \ggg s_3) - Q_0 - \phi_3(C_3, C_3, C_4)$
   (c) $p_j = R_j^{-1}(p_{j+1}, m_{\pi(j)})$, for $j = 46, 45, ..., 38$.
   (d) $\overset{22-0}{Q_{34}} = (\overset{all}{Q_{38}} \ggg 9) - \phi_{37}(\overset{all}{Q_{37}}, \overset{all}{Q_{36}}, \overset{all}{Q_{35}}) - \overset{22-0}{m_{10}} - \overset{all}{t_3}$
   (e) $\overset{22-0}{Q_{33}} = (\overset{all}{Q_{37}} \ggg 3) - \phi_{36}(\overset{all}{Q_{36}}, \overset{all}{Q_{35}}, \overset{22-0}{Q_{34}}) - \overset{all}{m_2} - \overset{all}{t_3}$
   (f) $\overset{22-0}{Q_{32}} = (\overset{all}{Q_{36}} \ggg 15) - \phi_{35}(\overset{all}{Q_{35}}, \overset{22-0}{Q_{34}}, \overset{22-0}{Q_{33}}) - \overset{22-0}{m_{12}} - \overset{all}{t_3}$
   (g) $\overset{22-0}{Q_{31}} = (\overset{all}{Q_{35}} \ggg 11) - \phi_{34}(\overset{22-0}{Q_{34}}, \overset{22-0}{Q_{33}}, \overset{22-0}{Q_{32}}) - \overset{all}{m_4} - \overset{all}{t_3}$
   (h) $\overset{13-0}{Q_{30}} = (\overset{22-0}{Q_{34}} \ggg 9) - \phi_{33}(\overset{22-0}{Q_{33}}, \overset{22-0}{Q_{32}}, \overset{22-0}{Q_{31}}) - \overset{14-0}{m_8} - \overset{all}{t_3}$
   (i) $\overset{13-0}{Q_{29}} = (\overset{22-0}{Q_{33}} \ggg 3) - \phi_{32}(\overset{22-0}{Q_{32}}, \overset{22-0}{Q_{31}}, \overset{13-0}{Q_{30}}) - \overset{all}{m_0} - \overset{all}{t_3}$
   (j) $\overset{10-0}{X} = (\overset{13-0}{Q_{29}} \ggg 3) - \overset{all}{m_3}$
   (k) $\overset{9-0}{Q_{28}} = (\overset{22-0}{Q_{32}} \ggg 13) - \phi_{31}(\overset{22-0}{Q_{31}}, \overset{13-0}{Q_{30}}, \overset{13-0}{Q_{29}}) - \overset{all}{m_{15}} - \overset{all}{t_2}$
   (l) $\overset{9-0}{Q_{27}} = (\overset{22-0}{Q_{31}} \ggg 9) - \phi_{30}(\overset{13-0}{Q_{30}}, \overset{13-0}{Q_{29}}, \overset{9-0}{Q_{28}}) - \overset{all}{m_{11}} - \overset{all}{t_2}$
   (m) $\overset{8-0}{Q_{26}} = (\overset{13-0}{Q_{30}} \ggg 5) - \phi_{29}(\overset{13-0}{Q_{29}}, \overset{9-0}{Q_{28}}, \overset{9-0}{Q_{27}}) - \overset{all}{m_7} - \overset{all}{t_2}$
   (n) Compare ($\overset{8-0}{Q_{26}}$, $\overset{9-0}{Q_{27}}$, $\overset{9-0}{Q_{28}}$, $\overset{10-0}{X}$) with ($\overset{all}{Q_{26}}, \overset{all}{Q_{27}}, \overset{all}{Q_{28}}, \overset{all}{X}$) in the table L, and then 40 (= 9 + 10 + 10 + 11) bits take part in comparison, so 2^{27} (= $2^{32} \times 2^{35} \times 2^{-40}$) pairs remain.
   (o) If matched, compute all of bits of Q_{34} with the corresponding m_{10} in the table L, Q_{33}, all of bits of Q_{32} with the corresponding m_{12} in the table L, Q_{31}, Q_{30} with the corresponding m_8 in the table L, and then

compute all of bits of Q_{29} and X using the remained pairs, respectively. Compare $\overset{31-11}{X}$ with X in the table L, i.e., compare the remained 21 bits of X, and 2^6 $(= 2^{27} \times 2^{-21})$ pairs remain.

(p) Compute Q_{28}, Q_{27}, and Q_{26}, and then compare them with Q_{28}, Q_{27} and Q_{26} in the table L using the remained pairs. If all of bits are matched, $m_0, m_1, ..., m_{15}$ as a preimage returns, otherwise, repeat the attack procedure.

5.2 Complexity Analysis for Preimage Attack on One-block MD4

(We use CF to represent compression function operations.)

1. The cost in Step 1 and Steps 2a∼2f compared to that of other steps is negligible.
2. The cost in Steps 3a and 3b is approximately $2^{35} \times \frac{20+1}{48}$ CF. The memory requirements in Step 3c is $2^{35} \times 7$ words.
3. The cost in Steps 4a, 4b, and 4c is $2^{32} \times \frac{1}{48}$ CF, $2^{32} \times \frac{1}{48}$ CF and $2^{32} \times \frac{9}{48}$ CF, respectively. The total cost in Steps $4d$∼$4m$ is $2^{32} \times \frac{10}{48}$ CF.
4. The cost in Step 4n is negligible. After comparison in Step 4n, 2^{26} pairs remain.
5. In Step 4o, the cost of computing Q_{34}, Q_{33}, Q_{32}, Q_{31}, Q_{30}, Q_{29}, and X is $2^{27} \times \frac{7}{48}$ CF. The cost in Step 4p is negligible.
6. The complexity is approximately $2^{33.98}$ $(\approx 2^{35} \times \frac{20+1}{48} + 2^{32} \times \frac{21}{48} + 2^{27} \times \frac{7}{48})$ CF from Step 1 to Step 4m. 2^{67} $(= 2^{35} \times 2^{32})$ pairs are produced in Steps 3 and 4 in total. Therefore, it needs to repeat 2^{61} $(= 2^{128-67})$ times. The total complexity to produce a preimage of MD4 is $2^{94.98}$ $(= 2^{33.98} \times 2^{61})$ CF. The memory requirements are $2^{35} \times 7$ words, i.e., the one in Step 3c. The program codes, which are used to compute the optimal complexity of preimage on one-block MD4, are in Appendix A.

6 Preimage Attack on Extended MD4

We still use the message word distribution shown in Table 5 as the one for preimage on two-block Extended MD4. In order to facilitate comparison, we also display the message word distribution shown in Table 7, which is used for both preimage attack on two-block Extended MD4 in [20] and preimage attack on two-block MD4 in [3].

For the preimage attack on two-block Extended MD4, we use the message words (m_3, m_7) as the neutral words for the second chunk and (m_8, m_{10}, m_{12}) as the neutral words for the first chunk, while the neutral word for the second chunk is m_7 and the neutral word for the first chunk is m_8 in [20].

We continue to analyze the message word distribution in Table 5 in order to improve the complexity further. We check if it is possible to put m_6 in round 3 into the skip interval and make it be a neutral word for the first chunk. However, m_6 in round 1 is in the first chunk. If we would regard m_6 as a neutral word for

Table 6. Our Message word distribution for two-block Extended MD4

Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
index	0	1	2	③	4	5	⑥	⑦	⑧	9	⑩	11	⑫	13	14	15
	⟵ first chunk ⟵						initial structure		⟶ second chunk ⟶							
Step	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
index	0	4	⑧	⑫	1	5	9	13	2	⑥	⑩	14	③	⑦	11	15
	⟶ second chunk ⟶												skip			
Step	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
index	0	⑧	4	⑫	2	⑩	⑥	14	1	9	5	13	③	11	⑦	15
	skip							⟵ first chunk ⟵								

Table 7. Message word distribution for two-block Extended MD4 in [20]

Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
index	0	1	2	3	4	5	6	⑦	⑧	9	10	11	12	13	14	15
	⟵ first chunk ⟵								⟶ second chunk ⟶							
Step	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
index	0	4	⑧	12	1	5	9	13	2	6	10	14	3	⑦	11	15
	⟶ second chunk ⟶													skip		
Step	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
index	0	⑧	4	12	2	10	6	14	1	9	5	13	3	11	⑦	15
	skip		⟵ first chunk ⟵													

the first chunk, we must try to put m_6 in round 1 into the second chunk. We find that m_6 in round 1 is next to m_7, which is a neutral word for the second chunk. Hence, we can move the splitting point one step leftward and apply a 2-step initial structure to m_6 and m_7 in round 1, as shown in Fig. 2, and then m_6 is changed with m_7 in round 1 so that m_6 can be a neutral word for the first chunk, and m_7 remains unchanged and is still a neutral word for the second chunk, as shown in Table 6. Our preimage attack procedure on Extended MD4 is similar to that on one-block MD4. The interaction between the copies of Extended MD4 is simple and easy to deal with according to the description in Section 2.2.

Finally, we improve the results about the pseudo-preimage and preimage attacks on Extended MD4 at the complexity $2^{203.8}$ and $2^{230.9}$, respectively. The detailed attack procedure is presented in next section.

6.1 Attack Procedure of Extended MD4

Assume that the given hash value is H, and $H_{exchange}$ is the value of H exchanged, after the values of the A registers in the two copies are exchanged according to Section 2.2.

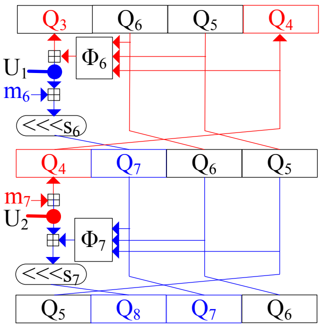

Fig. 2. The initial structure of Extended MD4.

1. Set the MSB of m_{13} one, the others bits of m_{13} random value, $m_{14} = 1024 - 65$ and $m_{15} = 0$ to satisfy the padding rule for two blocks messages.
   Set $\overset{9-0}{m_3}$ random value.
   Set $\overset{25-0}{m_6}$ random value.
   Set $\overset{10-0}{m_8}$ random value.
   Set $\overset{19-0}{m_{10}}$ random value.
   Set $\overset{18-0}{m_{12}}$ random value.
   Set $m_i (i \in \{0, 1, 2, 4, 5, 9, 11\})$ random values.
   Set U_1, U_2, Q_5 and Q_6 random values.
2. For all possible values of m_6, m_8, m_{10}, and m_{12}, ($\overset{31-26}{m_6}$, $\overset{31-11}{m_8}$, $\overset{31-20}{m_{10}}$, and $\overset{31-19}{m_{12}}$ are free, and there are 52 $(= 6 + 21 + 12 + 13)$ free bits in total), do the following,
   (a) $Q_7 = (U_1 + m_6) \lll s_6$
   (b) $Q_8 = (U_2 + \phi_7(Q_7, Q_6, Q_5)) \lll s_7$
   (c) $p_{j+1} = R_j(p_j, m_{\pi(j)})$, for $j = 8, 9, ..., 27$.
   (d) Let $X = Q_{25} + \phi_{28}(Q_{28}, Q_{27}, Q_{26}) + t_2$ (Note $Q_{29} = (X + m_3) \lll 3$)
   (e) Make a table L which saves (m_6, m_8, m_{10}, m_{12}, Q_{26}, Q_{27}, Q_{28}, X, Q'_{26}, Q'_{27}, Q'_{28}, X'), and the size of the table L is $2^{52} \times 12$ words.
3. For all possible values m_3 and m_7 ($\overset{31-10}{m_3}$ and $\overset{31-0}{m_7}$ are free, and there are 54 $(= 22 + 32)$ free bits in total), do the following,
   (a) $Q_4 = U_2 - m_7$
   (b) $Q_3 = U_1 - \phi_6(Q_6, Q_5, Q_4)$
   (c) $p_j = R_j^{-1}(p_{j+1}, m_{\pi(j)})$, for $j = 5, 4, ..., 0$.
   (d) $p_{48} = H_{exchange} - p_0$.
   (e) $p_j = R_j^{-1}(p_{j+1}, m_{\pi(j)})$, for $j = 47, 46, ..., 39$.
   (f) $\overset{25-0}{Q_{35}} = (\overset{all}{Q_{39}} \ggg 11) - \phi_{38}(\overset{all}{Q_{38}}, \overset{all}{Q_{37}}, \overset{all}{Q_{36}}) - \overset{25-0}{m_6} - \overset{all}{t_3}$
   (g) $\overset{19-0}{Q_{34}} = (\overset{all}{Q_{38}} \ggg 9) - \phi_{37}(\overset{all}{Q_{37}}, \overset{all}{Q_{36}}, \overset{25-0}{Q_{35}}) - \overset{19-0}{m_{10}} - \overset{all}{t_3}$
   (h) $\overset{19-0}{Q_{33}} = (\overset{all}{Q_{37}} \ggg 3) - \phi_{36}(\overset{all}{Q_{36}}, \overset{25-0}{Q_{35}}, \overset{19-0}{Q_{34}}) - \overset{all}{m_2} - \overset{all}{t_3}$

(i) $\overset{18-0}{Q_{32}} = (\overset{all}{Q_{36}} \ggg 15) - \phi_{35}(\overset{25-0}{Q_{35}}, \overset{19-0}{Q_{34}}, \overset{19-0}{Q_{33}}) - \overset{18-0}{m_{12}} - \overset{all}{t_3}$

(j) $\overset{14-0}{Q_{31}} = (\overset{25-0}{Q_{35}} \ggg 11) - \phi_{34}(\overset{19-0}{Q_{34}}, \overset{19-0}{Q_{33}}, \overset{18-0}{Q_{32}}) - \overset{all}{m_4} - \overset{all}{t_3}$

(k) $\overset{10-0}{Q_{30}} = (\overset{19-0}{Q_{34}} \ggg 9) - \phi_{33}(\overset{19-0}{Q_{33}}, \overset{18-0}{Q_{32}}, \overset{14-0}{Q_{31}}) - \overset{10-0}{m_8} - \overset{all}{t_3}$

(l) $\overset{10-0}{Q_{29}} = (\overset{19-0}{Q_{33}} \ggg 3) - \phi_{32}(\overset{18-0}{Q_{32}}, \overset{14-0}{Q_{31}}, \overset{10-0}{Q_{30}}) - \overset{all}{m_0} - \overset{all}{t_3}$

(m) $\overset{7-0}{X} = (\overset{10-0}{Q_{29}} \ggg 3) - \overset{all}{m_3}$

(n) $\overset{5-0}{Q_{28}} = (\overset{18-0}{Q_{32}} \ggg 13) - \phi_{31}(\overset{14-0}{Q_{31}}, \overset{10-0}{Q_{30}}, \overset{10-0}{Q_{29}}) - \overset{all}{m_{15}} - \overset{all}{t_2}$

(o) $\overset{5-0}{Q_{27}} = (\overset{14-0}{Q_{31}} \ggg 9) - \phi_{30}(\overset{10-0}{Q_{30}}, \overset{10-0}{Q_{29}}, \overset{5-0}{Q_{28}}) - \overset{all}{m_{11}} - \overset{all}{t_2}$

(p) $\overset{5-0}{Q_{26}} = (\overset{10-0}{Q_{30}} \ggg 5) - \phi_{29}(\overset{10-0}{Q_{29}}, \overset{5-0}{Q_{28}}, \overset{5-0}{Q_{27}}) - \overset{all}{m_7} - \overset{all}{t_2}$

(q) Compare $(\overset{5-0}{Q_{26}}, \overset{5-0}{Q_{27}}, \overset{5-0}{Q_{28}}, \overset{7-0}{X}, \overset{5-0}{Q'_{26}}, \overset{5-0}{Q'_{27}}, \overset{5-0}{Q'_{28}}, \overset{7-0}{X'})$ with $(\overset{all}{Q_{26}}, \overset{all}{Q_{27}}, \overset{all}{Q_{28}}, \overset{all}{X}, \overset{all}{Q'_{26}}, \overset{all}{Q'_{27}}, \overset{all}{Q'_{28}}, \overset{all}{X'})$ in the table L, and then 52 ($= 2 \times (6+6+6+8)$) bits take part in comparison, so 2^{54} ($= 2^{52} \times 2^{54} \times 2^{-52}$) pairs remain.

(r) If matched, compute all of bits of Q_{35} with the corresponding m_6 in the table L, all of bits of Q_{34} with the corresponding m_{10} in the table L, Q_{33}, all of bits of Q_{32} with the corresponding m_{12} in the table L, respectively, and then compute Q_{31}, Q_{30} with the corresponding m_8 in the table L, and then compute all of bits of Q_{29} and X using the remained pairs, respectively. Compare $\overset{31-8}{X}$ and $\overset{31-8}{X'}$ with X and X' in the table L, respectively, i.e., compare the remained 24 bits of X and X', and 2^4 ($= 2^{52} \times 2^{-24} \times 2^{-24}$) pairs remain.

(s) Compute Q_{28}, Q_{27}, and Q_{26}, and then compare them with Q_{28}, Q_{27} and Q_{26} in the table L using the remained pairs. If all of bits are matched, $m_0, m_1, ..., m_{15}$ as a preimage returns, otherwise, repeat the attack procedure.

6.2 Complexity Analysis for Preimage Attack on Two-block Extended MD4

(We use CF to represent compression function operations of Extended MD4 in this subsection.)

1. The cost in Step 1 compared to that of other steps is negligible.
2. The cost in Steps 2a, 2b, 2c and 2d is approximately $2^{52} \times \frac{1+1+20+1}{48}$ CF. The cost in Step 2e compared to that of other steps is negligible. The memory requirements in Step 2e is $2^{52} \times 12$ words.
3. The cost in Steps 3a and 3d is approximately $2^{54} \times \frac{1}{48}$ CF in all. The cost in Steps 3b, 3c and 3d is approximately $2^{54} \times \frac{1+6+9}{48}$ CF. The total cost in Steps $3f \sim 3p$ is $2^{54} \times \frac{11}{48}$ CF.
4. The cost in Step 3q is negligible. After comparison in Step 3q, 2^{54} pairs remain.
5. In Step 3r, the cost of computing Q_{35}, Q_{34}, Q_{33}, Q_{32}, Q_{31}, Q_{30}, Q_{29}, and X is $2^{54} \times \frac{8}{48}$ CF. The cost in Step 3s is negligible.

6. The complexity is approximately $2^{53.8}$ ($\approx 2^{52} \times \frac{23}{48} + 2^{54} \times \frac{1+16+11+8}{48}$) CF from Step 1 to Step 3r. 2^{106} ($= 2^{52} \times 2^{54}$) pairs are produced in Steps 2 and 3 in total. Therefore, it needs to repeat 2^{150} ($= 2^{256-106}$) times. The total complexity to produce a pseudo-preimage of Extended MD4 is $2^{203.8}$ ($= 2^{53.8} \times 2^{150}$) CF. The complexity to produce a preimage of Extended MD4 is $2^{230.9}$ ($= 2^{\frac{203.8+256}{2}+1}$) CF. The memory requirements are $2^{52} \times 12$ words, i.e., the one in Step 2e. The program codes for computing the optimal complexity of preimage attack on Extended MD4 are in Appendix B.

7 Conclusion

This paper shows the improved preimage attacks on one-block MD4 and two-block Extended MD4. We think that the idea used in the paper can be tried to improve the preimage attacks on other hash functions in the MD4-family.

Acknowledgments. We would like to thank Yu Sasaki, Lei Wang, and Kazumaro Aoki for the very helpful comments and suggestions.

References

1. Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for Step-Reduced SHA-2. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009, Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 578–597. Springer, 2009.
2. Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009, Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009.
3. Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In Roberto Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography 2008, Proceedings*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2009.
4. Jean-Philippe Aumasson, Willi Meier, and Florian Mendel. Preimage Attacks on 3-Pass HAVAL and Step-Reduced MD5. In Roberto Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography 2008, Proceedings*, volume 5381 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2009.
5. Ivan Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1990.
6. Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion attacks on secure hash functions using SAT solvers. In João Marques-Silva and Karem Sakallah, editors, *Theory and Applications of Satisfiability Testing C SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 377–382. Springer, 2007.
7. Christophe De Cannière and Christian Rechberger. Preimages for Reduced SHA-0 and SHA-1. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008, Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 179–202. Springer, 2008.

8. Bert den Boer and Antoon Bosselaers. An Attack on the Last Two Rounds of MD4. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 1992.
9. Hans Dobbertin. Cryptanalysis of MD4. *Journal of Cryptology*, 11(4):253–271, 1998.
10. Hans Dobbertin. The First Two Rounds of MD4 are Not One-Way. In Serge Vaudenay, editor, *Fast Software Encryption '98, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 284–292. Springer, 1998.
11. J. Guo, S. Ling, C. Rechberger, and H. Wang. Advanced meet-in-the-middle preimage attacks: First results on full tiger, and improved results on md4 and sha-2. Cryptology ePrint Archive, Report 2010/016, 2010. `http://eprint.iacr.org/`, accepted by ASIACRYPT 2010.
12. Takanori Isobe and Kyoji Shibutani. Preimage Attacks on Reduced Tiger and SHA-2. In Orr Dunkelman, editor, *Fast Software Encryption 2009, Proceedings*, volume 5665 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 2009.
13. H. Kuwakado and H. Tanaka. New algorithm for finding preimages in a reduced version of the MD4 compression function. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 83(1):97–100, 2000.
14. Xuejia Lai and James L. Massey. Hash Functions Based on Block Ciphers. In Rainer A. Rueppel, editor, *Advances in Cryptology – EUROCRYPT '92, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1993.
15. Gaëtan Leurent. MD4 is Not One-Way. In Kaisa Nyberg, editor, *Fast Software Encryption 2008, Proceedings*, volume 5086 of *Lecture Notes in Computer Science*, pages 412–428. Springer, 2008.
16. Ralph C. Merkle. One Way Hash Functions and DES. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1990.
17. Ronald L. Rivest. The MD4 Message Digest Algorithm. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO '90, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer, 1991.
18. Yu Sasaki and Kazumaro Aoki. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008, Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271. Springer, 2008.
19. Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009, Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009.
20. 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 Juan González Nieto, editors, *Australasian Conference on Information Security and Privacy (ACISP) 2009, Proceedings*, volume 5594 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2009.
21. Yu Sasaki, Lei Wang, Kazuo Ohta, and Noboru Kunihiro. New Message Difference for MD4. In Alex Biryukov, editor, *Fast Software Encryption 2007, Proceedings*, volume 4593 of *Lecture Notes in Computer Science*, pages 329–348. Springer, 2007.
22. Serge Vaudenay. On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In Bart Preneel, editor, *Fast Software Encryption 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 1995.

23. Gaoli Wang and Shaohui Wang. Preimage attack on hash function RIPEMD. In Feng Bao, Hui Li, and Guilin Wang, editors, *Information Security Practice and Experience*, volume 5451 of *Lecture Notes in Computer Science*, pages 274–284. Springer, 2009.
24. 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, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2005.
25. Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The Second-Preimage Attack on MD4. In Yvo Desmedt, Huaxiong Wang, Yi Mu, and Yongqing Li, editors, *Cryptology and Network Security (CANS) 2005, Proceedings*, volume 3810 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.
26. Hongbo Yu and Xiaoyun Wang. Multi-collision Attack on the Compression Functions of MD4 and 3-Pass HAVAL. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Cryptology – ICISC 2007, Proceedings*, volume 4817 of *Lecture Notes in Computer Science*, pages 206–226. Springer, 2007.

A The Program Code for Preimage Attack on One-block MD4

Listing 1.1. The program for computing the complexity of preimage attack on one-block MD4

```
/* The program is used to computer the time complexity of preimage
attack on one-block MD4
*/

#include <iostream>
#include <cmath>
using namespace std;

const int si [9]={5,9,13,3,9,11,15,3,9};
const int n[9]={7,11,15,0, 8,4,12,2,10};

int Get_Fixed_Bit(int Q[], int i, int m[]);

int main(int argc, char* argv[]) {

    int Q[40], m[16];
    int i, j, X;
    int first_free_bits , second_free_bits , total_free_bits ,
        match_bits, remain_bits;
    double min=pow(2.0, 10.0), a, r, h;
    int m8, m10, m12;

    for(m10=18; m10<31; m10++)
        for(m12=18; m12<31; m12++)
            for(m8=10; m8<31; m8++)
```

```
        {
            for (i=0; i<40; i++)
            {
                Q[i]=0;
            }
            for (i=0; i<16; i++)
            {
                m[i]=31;
            }
            m[10]=m10;
            m[12]=m12;
            m[8] =m8;

            Q[38]=31;
            Q[37]=31;
            Q[36]=31;
            Q[35]=31;
            for (i=34; i > 25; i--)
            {
                Q[i]=Get_Fixed_Bit(Q, i, m);
            }

            X=Q[29]-3;
            match_bits = X + Q[28] + Q[27] + Q[26] + 4;

             second_free_bits =32 * 3 - (m[10]+m[12]+m[8]+3);
              first_free_bits   =32;

             total_free_bits  =  first_free_bits  + second_free_bits;
            remain_bits =  total_free_bits  - match_bits;

            a=pow(2.0,  first_free_bits ) * 21.0/48.0;
            a=a + pow(2.0, second_free_bits) * 21.0/48.0;
            i=29;
            j=0;

            while((remain_bits > 10) && (j < 4 ))
            {
                if (0==j)
                {
                    a=a + pow(2.0, remain_bits) * 7.0/48.0;
                    remain_bits=remain_bits - (31-Q[i]) - 3;
                }
                else
                {
                    a=a + pow(2.0, remain_bits) * 1.0/48.0;
                    remain_bits=remain_bits - (31-Q[i-j]);
                }
                j++;
            }
```

```
            r=log(a)/log(2);
            h=128 - total_free_bits + r;
            if (min > h)
                min=h;

        }
        cout<<"preimage="<<min<<endl;
        return 0;
}

int Get_Fixed_Bit(int Q[], int i, int m[]) {
    int k, high=0, temp=0;

    k=i+4;
    if(31==Q[k])
    {
        high=31;
    }
    else
    {
        high=Q[k]-si[k-30];
    }
    if(high<=0)
        return 0;

    if(m[ n[k-30] ] < 31 && m[ n[k-30] ] < high)
    {
        high=m[ n[k-30] ];
    }
    temp = Q[i+3] > Q[i+2] ? Q[i+2]:Q[i+3];
    temp = temp > Q[i+1] ? Q[i+1]:temp;
    high = high > temp ? temp:high;

    return high;
}
```

B The Program Code for Preimage Attack on Extended MD4

Listing 1.2. The program for computing the complexity of pseudo-preimage and preimage attacks on Extended MD4

```

/* The program is used to computer the time complexity of
pseudo-preimage and preimage attacks on two-block Extended_MD4.
*/

```

```
#include <iostream>
#include <cmath>
using namespace std;

const int si[10]={5, 9, 13, 3, 9, 11, 15, 3, 9, 11};
const int n[10]={7, 11, 15, 0, 8, 4, 12, 2, 10, 6};

int Get_Fixed_Bit(int Q[], int i, int m[]);

int main(int argc, char* argv[]) {
    int Q[40], m[16], temp=0;
    int i, j, x;
    int first_free_bits , second_free_bits , total_free_bits ,
        match_bits, remain_bits;
    double min=pow(2.0, 10.0), a, r, h;
    int m6, m8, m10, m12;

    for(m6=25; m6<31; m6++)
        for(m10=18; m10<31; m10++)
            for(m12=18; m12<32; m12++)
                for(m8=10; m8<32; m8++)
                {
                    for(i=0; i<40; i++)
                    {
                        Q[i]=0;
                    }
                    for(i=0; i<16; i++)
                    {
                        m[i]=31;
                    }
                    m[6]=m6;
                    m[10]=m10;
                    m[12]=m12;
                    m[8] =m8;

                    Q[39]=31;
                    Q[38]=31;
                    Q[37]=31;
                    Q[36]=31;
                    for(i=35; i > 25; i--)
                    {
                        Q[i]=Get_Fixed_Bit( Q, i, m);
                    }

                    for( first_free_bits =35; first_free_bits <=64;
     first_free_bits ++)
                    {
                        x=Q[29]-3;
                        match_bits = 2 * ( x + Q[28] + Q[27] + Q[26]
     + 4);
```

```
                        second_free_bits =32 * 4 - (m[6]+m[10]+m[12]+m[8]+4);
                        total_free_bits = first_free_bits + second_free_bits;
                        a=pow(2.0, first_free_bits ) * 28.0/48.0;
                        a=a + pow(2.0, second_free_bits) * 23.0/48.0;
                        remain_bits= total_free_bits - match_bits;
                        i=29;
                        j=0;
                        while((remain_bits > 10) && (j < 4 ))
                        {
                            if(0==j)
                            {
                                a=a + pow(2.0, remain_bits) * (7.0+1)/48.0;
                                remain_bits=remain_bits - 2 * ((31-Q[i]) - 3);
                            }
                            else
                            {
                                a=a + pow(2.0, remain_bits) * 1.0/48.0;
                                remain_bits=remain_bits - 2 * (31-Q[i-j]);
                            }
                            j++;
                        }
                        r=log(a)/log(2);
                        h=256 - total_free_bits + r;
                        if(min > h)
                            min=h;

                    }

                }
                cout<<"pseudo_preimage="<<min<<"; preimage="<<(min+256.0)/2.0 + 1<<endl;

                return 0;
}

int Get_Fixed_Bit(int Q[], int i, int m[]) {
    int k, high=0, temp=0;

    k=i+4;
    if(31==Q[k])
    {
        high=31;
    }
    else
```

```
    {
        high=Q[k]-si[k-30];
    }
    if(0==high)
        return 0;
    if(m[ n[k-30] ] < 31 && m[ n[k-30] ] < high)
    {
        high=m[ n[k-30] ];
    }
    temp = Q[i+3] > Q[i+2] ? Q[i+2]:Q[i+3];
    temp = temp > Q[i+1] ? Q[i+1]:temp;
    high = high > temp ? temp:high;
    Q[i] = high;

    return high;
}
```