Finding Preimages in Full MD5 Faster than Exhaustive Search

Yu Sasaki and Kazumaro Aoki

NTT Information Sharing Platform Laboratories, NTT Corporation
3-9-11 Midori-cho, Musashino-shi, Tokyo, 180-8585 Japan
`sasaki.yu@lab.ntt.co.jp`

Abstract. In this paper, we present the first cryptographic preimage attack on the full MD5 hash function. This attack, with a complexity of $2^{116.9}$, generates a pseudo-preimage of MD5 and, with a complexity of $2^{123.4}$, generates a preimage of MD5. The memory complexity of the attack is $2^{45} \times 11$ words. Our attack is based on splice-and-cut and local-collision techniques that have been applied to step-reduced MD5 and other hash functions. We first generalize and improve these techniques so that they can be more efficiently applied to many hash functions whose message expansions are a permutation of message-word order in each round. We then apply these techniques to MD5 and optimize the attack by considering the details of MD5 structure.

keywords: MD5, splice-and-cut, local collision, hash function, one-way, preimage

1 Introduction

Cryptographic hash functions are important primitives of cryptographic techniques, which generate short-length strings from arbitrary length input messages. There are many applications to make a scheme secure using a hash function: message compression in digital signatures and message authentication, for example. A hash function H should satisfy several security properties such as

– **Preimage resistance:** for given y, x $s.t.$ $H(x) = y$ must be difficult to find,
– **2nd-preimage resistance:** for given x, x' $s.t.$ $H(x) = H(x'), x \neq x'$ must be difficult to find,
– **Collision resistance:** A pair of (x, x') $s.t.$ $H(x) = H(x'), x \neq x'$ must be difficult to find.

For a given n-bit hash value y, if the hash values of 2^n distinct messages are computed, there is a high probability that one of those values will match with y. Therefore, any method that can find a preimage faster than 2^n hash computation is a threat for hash functions. We stress that National Institute of Standards and Technology (NIST) requires preimage resistance with a complexity of 2^n for SHA-3 candidates [15].

MD5 [11] was proposed by Rivest in 1991. It generates 128-bit hash values by iteratively applying a compression function consisting of 64 steps. Though its security is suspect, MD5 is one of the most widely used hash functions in the world. So, a detailed analysis of the preimage resistance of MD5 is required.

Variants of collision attacks on MD5 were proposed by den Boer and Bosselaers in 1993 [5] and by Dobbertin in 1996 [6]. The first collision attack on MD5 was proposed by Wang et al. in 2005 [16]. Since then, several improved collision attacks have been proposed. The most effective attack, proposed by Klima [8], can generate a collision in one minute with a standard PC. Although there have been several powerful collision attacks on MD5, the preimage resistance of MD5 has not been broken yet.

1.1 History of preimage attacks on MD4-family

The history of preimage attacks on MD4-based hash functions is as follows. (In this paper, we omit the unit of complexity, which is the computational complexity of the compression function of the corresponding hash function.)

The first successful preimage attack was the one proposed by Leurent on MD4 at FSE 2008. The attack, with a complexity of $2^{100.5}$, generates a preimage [9]. The first preimage attack on MD5 was presented by De et al. in 2007. It attacked the first 26 steps with a SAT solver [4]. At ACISP 2008, Sasaki and Aoki presented a preimage attack with a complexity of 2^{96} on intermediate 44 steps of MD5 [13]. The paper shows that if the round order of MD5 is modified, intermediate 51 steps can be attacked. At SAC 2008, Aumasson et al. proposed a preimage attack with a complexity of 2^{102} on the first 47 steps of MD5 and an attack on full HAVAL-3 [2]. Also at SAC 2008, Aoki and Sasaki [1] showed an attack with a complexity of 2^{121} on the last 63 steps of MD5, and showed how to find a preimage of full MD5 slightly faster than the preimage resistance complexity 2^{128} by using a clever brute force algorithm. They also show one-block preimage attack on MD4. At CRYPTO 2008, Cannière and Rechberger attacked 49 steps of SHA-0 and 44 steps of SHA-1 [3]. Sasaki and Aoki proposed attacks on intermediate 52 steps of HAS-160 at ICISC 2008 [12] and 3-, 4-, and 5-pass HAVAL at Asiacrypt 2008 [14].

So far, the preimage resistance of full MD4, full HAVAL-3 and full HAVAL-4 were broken. Although these attacks are theoretically very interesting, they are not important from the industrial view point. On the other hand, regarding widely used hash functions such as MD5 or SHA-1, only step-reduced versions are analyzed and no preimage attack on the full specification has been conducted.

1.2 Related Techniques

Here, we explain previous attack techniques related to our work. See Section 3 for details of each technique.

The attacks on full HAVAL-3 and 47-steps MD5 by Aumasson et al. [2], which are later generalized by Sasaki and Aoki [14] use the *local-collision technique*, where the *absorption properties* are used to make a local collision and the *consistency check* is performed in the attack. On the other hand, the attacks on one-block MD4 and 63-steps MD5 by Aoki and Sasaki [1] use the *splice-and-cut technique*, where the attacks are made to be more efficient with the *partial-matching technique* and the *partial-fixing technique.*

1.3 Our results

This paper proposes the first cryptanalytic preimage attack on the full MD5. It finds a pseudo-preimage of full MD5 with a complexity of $2^{116.9}$ and a preimage of full MD5 with a complexity of $2^{123.4}$. The memory complexity of the attack is $2^{45} \times 11$ words.

In this paper, first, we improve several existing techniques with respect to following four points so that they can be more efficiently applied to various hash functions.

1. **Generalization of the local-collision technique**
   As described in a previous paper [14], the local-collision technique can be applied only if the two chosen neutral words are located a certain number of steps away. Since this limitation is too restrictive, the local-collision technique could not be applied to full MD5. Another paper [12] shows a variant of the local-collision technique; however, it is particular to HAS-160, which is the attack target of the paper. In this paper, we generalize the local-collision technique so that the same advantage can be obtained in various situations. Because our new technique no longer forms a local-collision, we call it *initial-structure technique.*
2. **Extension of the absorption properties**
   When we construct the initial structure, the absorption properties must be considered. In this paper, we newly consider *cross absorption properties*, which are extended versions of the absorption properties. This can further increase the situations where the initial structure can be constructed.
3. **Partial-fixing technique for unknown carry behavior**
   The partial-fixing technique partially computes the step function even if a part of the message words and chaining variables are not known. In previous papers, only partial computations whose carried number effects are deterministic were considered. In this paper, we also consider partial computations where an attacker cannot guess the carried number behavior in advance. Then, we propose an efficient attack procedure that does not increase the total attack complexity.
4. **Efficient consistency check method**
   We also solve a problem of an existing technique, where the consistency of the initial structure or the local collision is inefficiently checked, and thus the complexity becomes too high to attack successfully in some situation.

We stress that our improved techniques are not particular to MD5, but can be generally applied to hash functions whose message expansions are permutations of message-word order in each round.

Secondly, we combine all of our improved techniques and apply them to full MD5. Then, we optimize the attack by considering the details of MD5 structure. A summary of our results and previous results is shown in Table 1.

Table 1. Comparison of preimage attacks on MD5

Paper	Number of attacked steps	Complexity	
		Pseudo-preimage	Preimage
[4]	26	Not given †	
[13]	44 (Steps 3-46)	2^{96} †	
[2]	47	2^{96}	2^{102}
[1]	63	2^{112}	2^{121}
[1]	64 (Full)	$2^{125.7}$ ‡	2^{127} ‡
This paper	64 (Full)	$2^{116.9}$	$2^{123.4}$

† One-block attack.
‡ The brute force attack is used, but the computation order is optimized.

The organization of this paper is as follows. In Section 2, we describe the specification of MD5 and introduce the notation. In Section 3, we briefly describe the related work. In Section 4, we improve several existing techniques. In Section 5, we describe the attack on MD5 in detail and evaluate its complexity. In Section 6, we conclude this paper.

2 Description of MD5

2.1 MD5 specification and its properties

This section describes the specification of MD5. For details, we refer to [11].

MD5 is one of the Merkle-Damgård hash functions, that is, the hash value is computed as follows:

$$\begin{cases} H_0 \leftarrow IV, \\ H_{i+1} \leftarrow \mathrm{md5}(H_i, M_i) \qquad \text{for } i = 0, 1, \ldots, n-1, \end{cases} \tag{1}$$

where IV is the initial value defined in the specification, md5: $\{0,1\}^{128} \times \{0,1\}^{512} \rightarrow \{0,1\}^{128}$ is the compression function of MD5, and H_n is the output of the hash function. Before (1) is applied, the messages string M is processed as follows.

- The messages are padded in 512-bit multiples.
- The padded string includes the length of the message represented by 64 bits. The length string is represented as little endian and is placed at the end of the padding part.

After this process, the message string is divided into 512-bit blocks, M_i ($i = 0, 1, \ldots, n-1$).

The compression function $H_{i+1} \leftarrow \mathrm{md5}(H_i, M_i)$ is computed as follows.

1. M_i is divided into 32-bit message words m_j ($j = 0, 1, \ldots, 15$).

2. The following recurrence is done.
$$\begin{cases} p_0 \leftarrow H_i \\ p_{j+1} \leftarrow R_j(p_j, m_{\pi(j)}) \qquad \text{for } j = 0, 1, \ldots, 63 \end{cases}$$
3. H_{i+1} $(= p_{64} + H_i)$ is output, where "+" denotes 32-bit word-wise addition. In this paper, we similarly use "−" to denote 32-bit word-wise subtraction.

R_j is the step function for Step j. Let Q_j be a 32-bit value that satisfies $p_j = (Q_{j-3}, Q_j, Q_{j-1}, Q_{j-2})$. $R_j(p_j, m_{\pi(j)})$ computes p_{j+1} as follows:

$$\begin{cases} Q_{j+1} \leftarrow Q_j + (Q_{j-3} + \Phi_j(Q_j, Q_{j-1}, Q_{j-2}) + m_{\pi(j)} + k_j) \lll s_j, \\ p_{j+1} \leftarrow (Q_{j-2}, Q_{j+1}, Q_j, Q_{j-1}), \end{cases}$$

where Φ_j, k_j, and $\lll s_j$ are the bitwise Boolean function, constant value, and left rotation defined in the specification, respectively. $\pi(j)$ is a function for MD5 message expansion. Details of Φ_j, s_j, and $\pi(j)$ are shown in Table 2. Note $R_j^{-1}(p_{j+1}, m_{\pi(j)})$ is computed with almost the same complexity as that of R_j.

Table 2. Boolean functions, rotation numbers, and message expansion of MD5

$\Phi_0, \Phi_1, \ldots, \Phi_{15}$	$\Phi_j(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
$\Phi_{16}, \Phi_{17}, \ldots, \Phi_{31}$	$\Phi_j(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$
$\Phi_{32}, \Phi_{33}, \ldots, \Phi_{47}$	$\Phi_j(X, Y, Z) = X \oplus Y \oplus Z$
$\Phi_{48}, \Phi_{49}, \ldots, \Phi_{63}$	$\Phi_j(X, Y, Z) = Y \oplus (X \vee \neg Z)$
$s_0, s_1, \ldots, s_{15}$	7 12 17 22 7 12 17 22 7 12 17 22 7 12 17 22
$s_{16}, s_{17}, \ldots, s_{31}$	5 9 14 20 5 9 14 20 5 9 14 20 5 9 14 20
$s_{32}, s_{33}, \ldots, s_{47}$	4 11 16 23 4 11 16 23 4 11 16 23 4 11 16 23
$s_{48}, s_{49}, \ldots, s_{63}$	6 10 15 21 6 10 15 21 6 10 15 21 6 10 15 21
$\pi(0), \pi(1), \ldots, \pi(15)$	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
$\pi(16), \pi(17), \ldots, \pi(31)$	1 6 11 0 5 10 15 4 9 14 3 8 13 2 7 12
$\pi(32), \pi(33), \ldots, \pi(47)$	5 8 11 14 1 4 7 10 13 0 3 6 9 12 15 2
$\pi(48), \pi(49), \ldots, \pi(63)$	0 7 14 5 12 3 10 1 8 15 6 13 4 11 2 9

3 Related works

3.1 Converting pseudo-preimages to a preimage

For a given hash value H_N and a compression function CF, pseudo-preimage is a pair of $(v, M), v \neq IV$ such that $CF(v, M) = H_N$. First, we describe the generic algorithm for the Merkle-Damgård hash functions with n-bit output, which converts pseudo-preimages to a preimage [10, Fact 9.99]. Assume that there is an algorithm that finds $(H_1, (M_1, M_2, \ldots, M_{N-1}))$ such that $H_{i+1} = CF(H_i, M_i)$ $(i = 1, 2, \ldots, N-1)$ with the complexity of 2^x and H_1 looks random. Prepare a table that includes $2^{n/2-x/2}$ entries of $(H_1, (M_1, M_2, \ldots, M_{N-1}))$. Compute $2^{n/2+x/2}$ $CF(H_0, M_0)$ for random M_0. One of the results will agree with one of the entries in the table with high probability. The required complexity of the attack is about $2^{n/2+1+x/2}$. Therefore, showing how to find (H_1, M_1) from a given hash value within $2^x, x < n-2$ is enough for a theoretical preimage attack.

3.2 Preimage attack on 63 steps of MD5

At SAC 2008, Aoki and Sasaki proposed a preimage attack on 63 steps of MD5 based on the splice-and-cut, partial-matching, and partial-fixing techniques [1].

The *splice-and-cut technique* is a way to apply the meet-in-the-middle attack. The authors consider the first and last steps as consecutive steps, and divide the attack target into two *chunks* of steps so that each chunk includes independent message words from the other chunk. Such message words are called *neutral words*. Then, a pseudo-preimage is computed by the meet-in-the-middle attack.

The *partial-matching technique* enables an attacker to skip several steps of an attack target when searching for chunks. Assume that one of the divided chunks provides the value of p_i, where $p_i = (Q_{i-3}, Q_i, Q_{i-2}, Q_{i-1})$, and the other chunk provides the value of p_{i+3}, where $p_{i+3} = (Q_i, Q_{i+3}, Q_{i+2}, Q_{i+1})$. p_i and p_{i+3} cannot be directly compared; however, a part of the values, that is, 32-bits of Q_i, can be compared immediately. In such a case, one can ignore messages used in steps $i, i+1$, and $i+2$ when the meet-in-the-middle attack is performed.

The *partial-fixing technique* enables an attacker to skip more steps. The idea is to fix a part of the neutral words so that an attacker can partially compute a chunk even if a neutral word for the other chunk appears. This enables the attacker to skip more steps. For example, consider the equation for computing Q_{j-3} in the inversion of the step function $R_j^{-1}(p_{j+1}, m_{\pi(j)})$:

$$Q_{j-3} = ((Q_{j+1} - Q_j) \ggg s_j) - \Phi_j(Q_j, Q_{j-1}, Q_{j-2}) - m_{\pi(j)} - k_j. \qquad (2)$$

When the lower n bits of $Q_{j-1},\ Q_{j-2}$, and $m_{\pi(j)}$ are fixed and other variables are fully fixed, the lower n bits of Q_{j-3} can be computed independently from the higher $32 - n$ bits of $Q_{j-1},\ Q_{j-2}$, and $m_{\pi(j)}$.

3.3 Preimage attack on HAVAL

A combination of the meet-in-the-middle and local collision was first proposed by Aumasson et al. [2]. Sasaki and Aoki further improved this by using the splice-and-cut technique instead of the simple meet-in-the-middle attack [14]. As a result, they succeeded in attacking full HAVAL-3, full HAVAL-4, and step-reduced HAVAL-5, and slightly improved the complexity of the brute force attack on full HAVAL-5.

The *local-collision technique* named by Sasaki and Aoki [14] enables an attacker to skip several steps at the beginning of chunks. The key idea of this technique is to select two neutral words that can form a local collision. Schematic explanation is shown in the left diagram of Figure 1. To achieve this, the selected neutral words must be exactly $(L \cdot n + 1)$ steps away each other, where $n \geq 1$ and L denotes the number of chaining variables, e.g., L=8 for HAVAL and L=4 for MD5. Changes of the neutral words' values must be guaranteed not to give any influence to other chaining variables. To stop the difference propagating through Φ_j, the values of the other chaining variables must be fixed so that input differences are ignored in the output value. Such properties are called *absorption properties*. Finally, changes of neutral-words' values are checked to be offset each other. For example, in the left diagram of Figure 1, we need to check $Q^{\text{1st}}_{j-3} + m^{\text{2nd}} + m^{\text{1st}} \overset{?}{=} Q^{\text{2nd}}_{j+5}$ for a given $(m^{\text{1st}}, Q^{\text{1st}}_{j-3}, m^{\text{2nd}}, Q^{\text{2nd}}_{j+5})$. We call such a checking procedure *consistency check*.

Because a local collision of HAVAL can be formed by only two message words and Φ_j has many absorption properties, the local-collision technique can be effectively applied to HAVAL.

3.4 Preimage attack on HAS-160

An example of a variant of the local-collision technique is shown in Ref. [12]. Differently from Ref. [14], Ref. [12] applies the local-collision technique even if two neutral words are located three steps away, not $(L \cdot n + 1)$ steps away. However, this technique is particular to their attack target HAS-160.

4 Improved techniques

We applied all the previously mentioned techniques to full MD5, but the attempt failed. To attack MD5, further improvements are necessary. In this section, we give some intuition of our improved idea. For the concrete application to MD5, we refer to Section 5.

4.1 Initial structure: generalization of local-collision technique

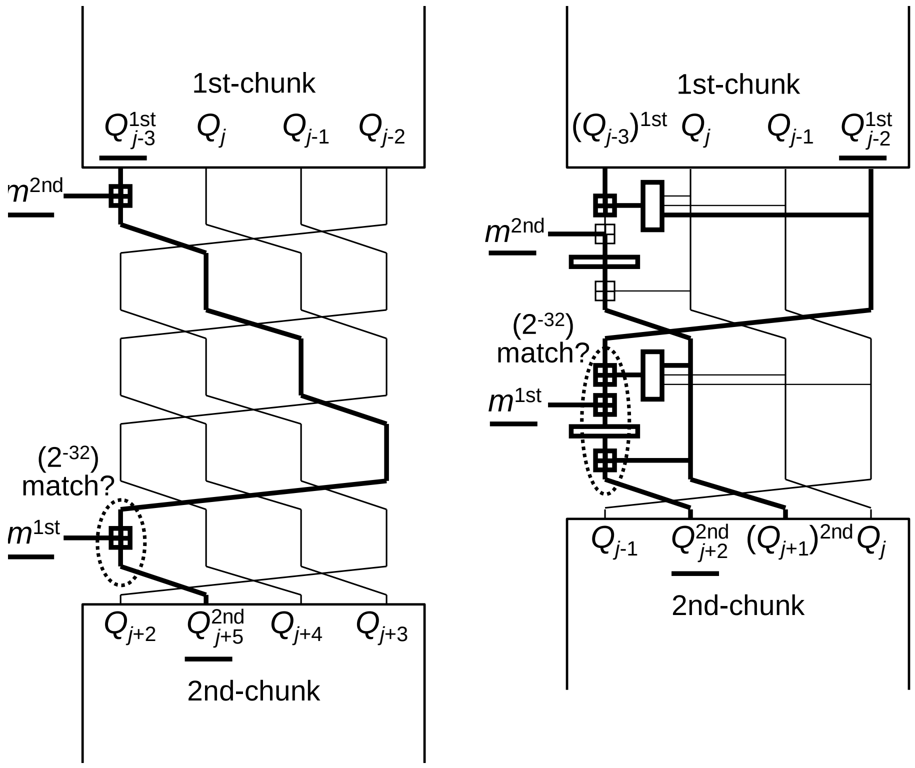
In the previous works, the local-collision technique is applicable if selected neutral words are exactly $(L \cdot n + 1)$ steps away. However, this technique has the following three problems.

1. The limitation of $(L \cdot n + 1)$ steps away is too restrictive to find good chunks.
2. If more than two message words are necessary to form a local collision, the limitation becomes much stronger. In fact, MD5 needs three words.
3. Absorption properties of Φ are necessary to obtain a local collision, however, Φ does not always have such properties.

The above problems are solved by our new technique. It is visualized in Figure 1.

Previous work fixes the value of Q_j, Q_{j-1}, Q_{j-2} and $Q_{j+4}, Q_{j+3}, Q_{j+2}$ for any value of $m^{\text{1st}}, Q^{\text{1st}}, m^{\text{2nd}}, Q^{\text{2nd}}$. However, we found the essential point is to make the first chunk independent of $(m^{\text{2nd}}, Q^{\text{2nd}})$ and make the second chunk independent of $(m^{\text{1st}}, Q^{\text{1st}})$. Therefore, Q_j, Q_{j-1}, Q_{j-2}, which are included in the first chunk, can be changed depending on the value of $(m^{\text{1st}}, Q^{\text{1st}})$. Similarly, $Q_{j+4}, Q_{j+3}, Q_{j+2}$ can be changed depending on $(m^{\text{2nd}}, Q^{\text{2nd}})$.

Based on this observation, we can construct several new patterns of "local collision". Because these patterns no longer form a local collision, we call this technique *initial structure*. The following is the concept of the initial structure.



1st-chunk
Q^{1st}_{j-3} Q_j Q_{j-1} Q_{j-2}
m^{2nd}
(2^-32) match?
m^{1st}
Q_{j+2} Q^{2nd}_{j+5} Q_{j+4} Q_{j+3}
2nd-chunk
1st-chunk
$(Q_{j-3})^{1st}$ Q_j Q_{j-1} Q^{1st}_{j-2}
m^{2nd}
(2^-32) match?
m^{1st}
Q_{j-1} Q^{2nd}_{j+2} $(Q_{j+1})^{2nd}$ Q_j
2nd-chunk

Left side describes local-collision technique; right side describes our generalization called initial structure. Underlined variables are neutral words. Notation $(Q)^x$ denotes a chaining variable whose value changes depending on the value of neutral words for x-chunk.

Fig. 1. MD5 structures with old and new techniques applied

Initial structure is a few consecutive steps including at least two neutral words named m^{2nd} and m^{1st}, where steps after the initial structure (2nd chunk) can be computed independently of m^{1st} and steps before the initial structure (1st chunk) can be computed independently of m^{2nd}.

In the above concept, if m^{1st} appears in an earlier step than m^{2nd}, the structure can be included in the first and second chunks, hence the attack can be easily performed. We are interested in the case where m^{2nd} appears earlier than m^{1st}.

An example of the initial structure of MD5 consisting of two steps is shown in Figure 1. In this structure, 2^{32} values of $m^{1st}, Q^{1st}, m^{2nd}, Q^{2nd}$ are tried when we compute two chunks. To make the second chunk independent of Q^{1st}, we choose Q_{j-3} to cancel the change of Q^{1st}. Similarly, when we compute the second chunk, we compute Q_{j+1} according to the value of m^{2nd}. In the end, this structure provides 2^{64} free bits for both the first and second chunks, guarantees that the first and second chunks are independent of each other, and succeeds with a probability of 2^{-32} for randomly given $m^{1st}, Q^{1st}, m^{2nd}, Q^{2nd}$.

As the example shown in Figure 1, some initial structure patterns do not use the absorption properties of Φ. This gives a big advantage to an attacker compared to the previous local-collision technique because such structures can be constructed even if Φ does not have absorption properties, e.g., Φ is XOR.

We manually searched for patterns of initial structures that are formed within 5 steps, and found that patterns shown in Table 3 can form the initial structure.

Table 3. Possible patterns of initial structure of MD5

	$m_{\pi(i)}$	$m_{\pi(i+1)}$	$m_{\pi(i+2)}$	$m_{\pi(i+3)}$	$m_{\pi(i+4)}$
Pattern 1	○	○			
Pattern 2	○	○	○		
Pattern 3	○	○			○
Pattern 4	○			○	○

○ denotes a neutral word that is necessary to form initial structure.
Note: Pattern 3 is the local collision usually considered.

4.2 Cross absorption property

The cross absorption property is an extension of the absorption property. By considering the cross absorption property, the number of possible initial structure patterns can be increased.

The absorption properties of MD5 summarized in Ref. [13] focus on how to ignore one of the input variables of $\Phi_j(X, Y, Z)$. This enables us to fix the output of $\Phi_j(X, Y, Z)$ even if one of X, Y, Z changes.

Cross absorption properties enable us to fix the output of $\Phi_j(X, Y, Z)$ even if two of X, Y, Z are changed. To achieve cross absorption properties, we partially fix changing variables so that fixed bits cover all 32 bits. For example, let us consider $\Phi_j(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$, where Y and Z are neutral words. To fix the output, we first fix lower n bits of Y and fix lower n bits of X to 1. Then, we fix higher $32 - n$ bits of Z and fix higher $32 - n$ bits of X to 0. As a result, the output of Φ_j is fixed to Y in lower n bits and Z in higher $32 - n$ bits for any value of lower n bits of Z and higher $32 - n$ bits of Y.

By considering the cross absorption properties, more complicated initial structures shown in Table 4 can be constructed. Pattern 6 is useful because

Table 4. Initial structure with cross absorption properties

	$m_{\pi(i)}$	$m_{\pi(i+1)}$	$m_{\pi(i+2)}$	$m_{\pi(i+3)}$	$m_{\pi(i+4)}$
Pattern 5	○		○		
Pattern 6	○			○	

only two message words are involved and the length of the structure is relatively long (4 steps). See Section 5.2 for the application to MD5.

4.3 Partial-fixing technique for unknown carried number behavior

The previous partial-fixing technique on MD5 [1] enables us to skip six steps at the end of chunks by partially computing the chaining variables. Let us consider the equation $A + B$, where A and B are only partially known. When known part of A and B starts from LSB, we can uniquely compute $A + B$ in the same number of bits, whereas, when known part starts from an intermediate bit x, we cannot uniquely determine intermediate bits of $A + B$ due to the unknown carried number from bit $x-1$ to x. However, by considering both possible carried number patterns, the number of candidates of $A + B$ can be reduced to only two. Consequently, for each addition of values with intermediate known bits, we obtain the correct pairs and the same number of wrong pairs.

A small amount of incorrect pairs can be filtered out with negligible complexity. After we find the corresponding message by a partial matching test, we compute the step function and check the exact carried number value step by step. This computation costs only 1 step, that is, $2^{-6}(= \frac{1}{64})$ MD5 computations, and the number of remaining pairs will be reduced by checking the correctness of carried number assumption and matching test for increased known bits. In the end, when the number of unknown carried numbers is up to 6, we consider all 2^6 possible carried number patterns, and incorrect data is filtered out with a complexity of $1(= 2^6 \cdot 2^{-6})$ MD5 computations, which is a very small extra cost. This enables us to skip eight steps at the end of chunks. See Section 5.3 for the application to MD5.

4.4 Efficient consistency check method

In previous works, the consistency of the initial structure (or local-collision) is checked after the partial matching test of chunk's results is finished. This strategy fails if the number of matched bits is small.

For example, we consider the attack procedure for the left structure in Figure 1. We compute chunks for 2^{64} values of (m^{1st}, Q^{1st}) and (m^{2nd}, Q^{2nd}). Assume the partial matching test works for small numbers of bits, e.g., only 12 bits. Ideally, we should obtain $2^{84}(= 2^{128} \cdot 2^{-32} \cdot 2^{-12})$ pairs where the partial 12 bits are matched and the initial structure is properly satisfied with a complexity of 2^{84}. Therefore, by repeating the above procedure 2^{32} times, we expect to obtain a pair where all 128 bits are matched with a complexity of $2^{116}(= 2^{84} \cdot 2^{32})$. However, the previous method computes a few steps for $2^{116}(= 2^{128} \cdot 2^{-12})$ pairs after the 12-bit matching test, and then, checks the full-bit match test and finally checks the consistency of the initial structure, which is satisfied with a probability of 2^{-32}. Computing a few steps for 2^{116} pairs costs roughly 2^{116} step function computation, and repeating this procedure 2^{32} times requires 2^{148}, which is worse than the brute force attack.

We solve this problem by performing the consistency check together with the partial matching test. This can be performed with a small amount of extra computation and memory. Again, we consider the attack procedure for the left structure in Figure 1. When we compute the first chunk by trying all (m^{1st}, Q^{1st}), we additionally store the value of $m^{1st} + Q^{1st}$ in a table. Then, when we compute the second chunk by trying all (m^{2nd}, Q^{2nd}), we compute $Q^{2nd} - m^{2nd}$ and compare it with $m^{1st} + Q^{1st}$ stored in the table. By this effort, the previous example examines a 44-bit matching test instead of a 12-bit matching test for 2^{128} pairs, and thus, the complexity becomes 2^{84}. After 2^{32} repetation of this procedure, we will find a 128-bit matched pair with a complexity of 2^{116}.

5 Preimage attacks on full MD5

5.1 Selected initial structure and chunks

When we searched for good chunks, we assumed that the partial-matching and partial-fixing techniques could enable us to skip a maximum of 11 steps. Under this assumption, we considered all possible patterns and positions of the initial structure. As a result, we found that the pattern 6 in Table 4 for $i = 14$ skipping steps 43-50 is the only useful pattern. This chunk separation is shown in Figure 2.

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	
Step	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
index	1	⑥	11	0	5	10	15	4	9	⑭	3	8	13	2	7	12
structure			second chunk													
Step	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
index	5	8	11	⑭	1	4	7	10	13	0	3	⑥	9	12	15	2
	second chunk											skip				
Step	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
index	0	7	⑭	5	12	3	10	1	8	15	⑥	13	4	11	2	9
	skip			first chunk												

Fig. 2. Selected chunks for full-round MD5

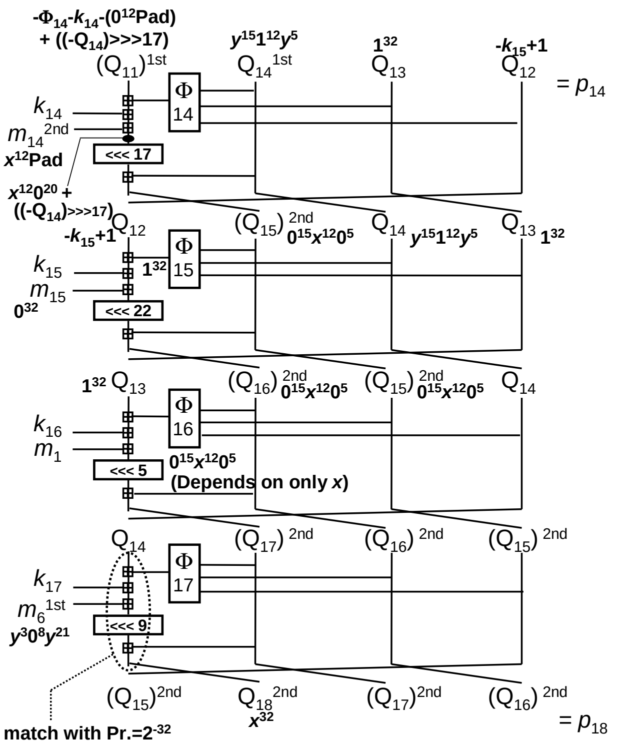
5.2 Details of initial structure for full MD5

Construction of the initial structure is complicated. We need to consider the rotation number s_j and constant k_j in each step. First, we show how to fix message words and chaining variables inside the initial structure in Figure 3 and then explain how computations in the initial structure behave. We have confirmed that the numbers and positions of fixed bits are optimal when both of the initial structure and partial-fixing techniques are considered.

Numbers written in a small bold font near variables denote the value of each variable. To denote bit information of variables, we use notation a^b, which means the one-bit value a continues for b bits. For example, 0^{32} means all 32 bits are 0, and $0^{15}x^{12}0^5$ means that the first 5 bits[1] are fixed to 0, the next 12 bits are free-bits for the second chunk, and the last 15 bits are fixed to 0.

To construct the initial structure, we firstly choose m_6 and Q_{14} as neutral words for the first chunk and m_{14} and Q_{18} as neutral words for the second chunk so that both chunks can produce 2^{64} items whereas the consistency of the initial structure checked in the dotted circle is satisfied with a probability

[1] In this paper, LSB is the first bit (= 0th bit), and MSB is the last bit (= 31st bit).



The lower 20 bits of m_{14} are fixed to satisfy the message padding. For example, all bits are fixed to 1.

Fig. 3. Initial structure for full MD5

of 2^{-32}. In Figure 3, we use notation $^{\text{1st}}$ and $^{\text{2nd}}$ to denote neutral words for the first and second chunks, respectively. Let x and y represent a free bit in the neutral words for the second and first chunks, respectively. Here, free bit means the unfixed bits of neutral words where we try all values when we perform the meet-in-the-middle attack. We secondly fix values of variables to guarantee that p_{14} can be computed independently of the value of 'x's and p_{18} can be computed independently of the value of 'y's. We also choose variables that are computed depending on the value of neutral words for each chunk. In Figure 3, we indicate such variables with notation $(\ )^{\text{1st}}$ and $(\ )^{\text{2nd}}$.

In Remarks of this section, we will explain Q_{11} can be computed independently of 'x's of m_{14}. Therefore, p_{14} is independent of 'x's. Now, we explain why p_{18} is guaranteed to be independent of 'y's by fixing values as shown in Figure 3.

Values of 'y's in m_6 only give impact to the data line where the consistency is checked in step 17 with a probability of 2^{-32}. Therefore, p_{18} is independent of 'y's in m_6. The remaining work is to guarantee that values of 'y's in Q_{14} do not impact other data lines in steps 14, 15, and 16.

1. In step 14, values of y in Q_{14} can impact the value of Q_{15} through Φ_{14} and through the direct addition from Q_{14} to Q_{15}. To prevent these impacts, we

choose the value of Q_{11} so that the sum of Q_{11}, output of Φ_{14}, $Q_{14} \ggg s_{14}$, and fixed part (lower 20 bits) of m_{14} are always the same value. Therefore, every time we choose Q_{14}, we compute Q_{11} as follows.

$$Q_{11} = -\Phi_{14}(Q_{14}, Q_{13}, Q_{12}) - k_{14} - (m_{14} \wedge \texttt{0xfffff}) + ((-Q_{14}) \ggg s_{14}), \quad (3)$$

where we also cancel the addition of k_{14} for simplicity. This cancellation may fail because of the relationship of addition and rotation. This problem is solved in the Remarks of this section.

2. In step 15, we arrange the values of Q_{15}, Q_{14}, and Q_{13} so that changes of 'y's in Q_{14} is absorbed in the computation of Φ_{15}. Because two input variables Q_{15} and Q_{14} have free-bits, we use the cross absorption property introduced in Section 4.2. Remember $\Phi_{15} = (Q_{15} \wedge Q_{14}) \vee (\neg Q_{15} \wedge Q_{13})$. Because the values of Q_{15} and Q_{13} are 0 and 1, respectively, in bit positions 0-4 and 17-31, the value of Φ_{15} becomes 1. In bit positions 5-16, because the values of Q_{14} and Q_{13} are 1, the value of Φ_{15} becomes 1. Therefore, regardless of the value of 'y's in Q_{14}, the output of Φ_{15} is fixed to 1^{32}.
3. In step 16, the Boolean function is $\Phi_{16} = (Q_{16} \wedge Q_{14}) \vee (Q_{15} \wedge \neg Q_{14})$. If Q_{16} can be fixed to the same value as Q_{15}, Q_{14} is absorbed in the computation of Φ_{16}. This is achieved by setting $Q_{12} + \Phi_{15} + k_{15} + m_{15} = 0$ since $Q_{16} = Q_{15} + (Q_{12} + \Phi_{15} + k_{15} + m_{15}) \lll 22$. Remember, m_{15} is involved in the message padding part. To guarantee that the length of the preimage will be at most $2^{32} - 1$ bits, we fix m_{15} to 0. We know that $\Phi_{15} = \texttt{0xffffffff} = -1$. Therefore, fixing $Q_{12} = -k_{15} + 1$ can achieve the desired condition.

Finally, p_{18} is guaranteed to be independent of 'y's, and the initial structure is properly constructed for any selection of 'x's and 'y's.

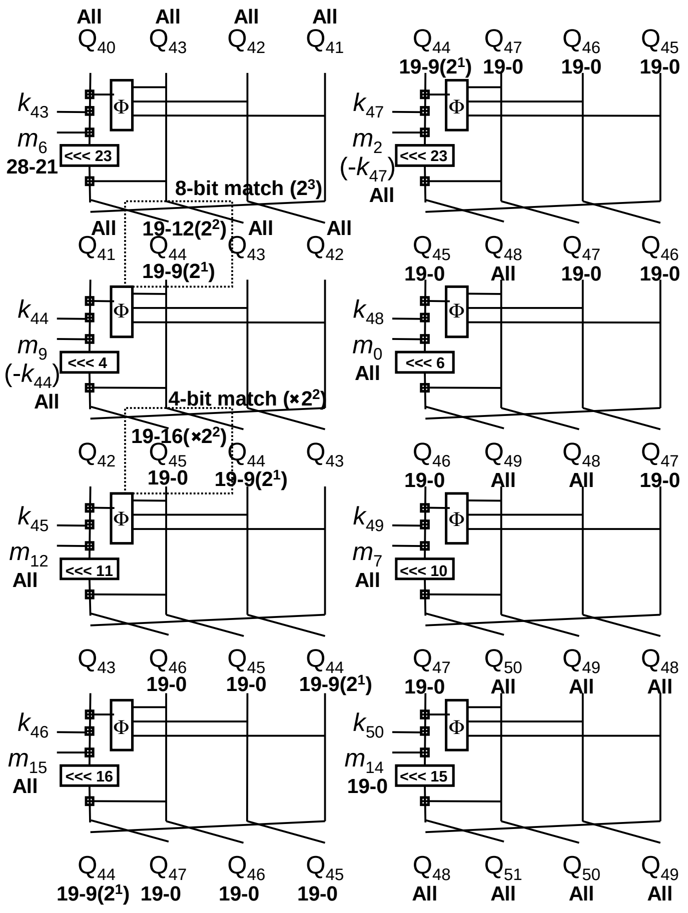
Remarks. Computation for step 14 performed by equation (3) may fail and the probability of this depends on the values of chaining variables and the message word. We experimentally confirmed that for all 2^{32} possible patterns of unfixed bits in (m_{14}, Q_{14}), the choice of Q_{14} does not impact m_{14} with high probability. Specifically, for any (m_{14}, Q_{14}), the following equation holds.

$$Q_{11} + \Phi_{14} + k_{14} + (m_{14} \wedge \texttt{0xfffff}) = ((-Q_{14}) \ggg s_{14}) \quad Pr. = 1, \quad (4)$$

$$((m_{14} + ((-Q_{14}) \ggg s_{14})) \lll s_{14}) + Q_{14} = (m_{14} \lll s_{14}) \; Pr. = 1 - 2^{-17}. (5)$$

5.3 Details of partial-fixing for skipping 8 steps

As is explained in Section 4.3, meet-in-the-middle for skipping 8 steps will need to deal with unknown carried number behavior. The number of bits matched and number of unknown carried numbers depend on the number of rotations in each step. For the chunk we chose, we can apply 12-bit matching including 5 unknown carried numbers. We explain how the partial computation is performed step by step. The schematic explanation is in Figure 4. We use a notation $X^{b_2 - b_1}$ to denote that values of bit positions b_1 to b_2 of a variable X are known.



Figures written in a small bold font denote the known bits of each variable.

Fig. 4. Partial-matching for 8 steps

Inverse computation for Steps 50-48. This is exactly the same as the partial-fixing technique used in Ref. [1]. In details, the equation for computing Q_{47} in $R_{50}^{-1}(p_{51}, m_{50})$ is as follows.

$$Q_{47} = ((Q_{51}^{31-0} - Q_{50}^{31-0}) \ggg s_{50}) - \Phi_{50}(Q_{50}^{31-0}, Q_{49}^{31-0}, Q_{48}^{31-0}) - m_{\pi(50)}^{19-0} - k_{50}. \tag{6}$$

k_j is constant, hence k_j is known value. Because the lower 20 bits (positions 0 to 19) of $m_{14}(= m_{\pi(50)})$ are fixed and known, we can uniquely obtain the lower 20 bits of Q_{47} independently of the upper 12 bits of m_{14}. Similarly, the lower 20 bits of Q_{46} and Q_{45} can be uniquely computed as follows:

$$Q_{46} = ((Q_{50}^{31-0} - Q_{49}^{31-0}) \ggg s_{49}) - \Phi_{49}(Q_{49}^{31-0}, Q_{48}^{31-0}, Q_{47}^{19-0}) - m_{\pi(49)}^{31-0} - k_{49}, \tag{7}$$

$$Q_{45} = ((Q_{49}^{31-0} - Q_{48}^{31-0}) \ggg s_{48}) - \Phi_{48}(Q_{48}^{31-0}, Q_{47}^{19-0}, Q_{46}^{19-0}) - m_{\pi(48)}^{31-0} - k_{48}. \tag{8}$$

Inverse computation for Step 47. Equation for Q_{44} is as follows:

$$Q_{44} = ((Q_{48}^{31-0} - Q_{47}^{19-0}) \ggg s_{47}) - \Phi_{47}(Q_{47}^{19-0}, Q_{46}^{19-0}, Q_{45}^{19-0}) - m_{\pi(47)}^{31-0} - k_{47}. \quad (9)$$

We can uniquely compute the lower 20 bits of $Q_{48} - Q_{47}$. Let the value after the right rotation by $23(= s_{47})$ bits be u, and then, we uniquely obtain u^{28-9}. We can also compute the lower 20 bits of the output of Φ_{47}. Set the value of $m_2(= m_{\pi(47)})$ to $-k_{47}$ in advance. Then, the equation (9) becomes $u^{28-9} - \Phi_{47}^{19-0}$. By considering two possible carried number patterns from bit position 8 to 9, we can obtain two candidates of Q_{44}^{19-9}.

Forward computation for Step 43. $m_6(= m_{\pi(43)})$ in bit positions 21-28 are fixed. Then, the equation for Q_{44} in $R_{43}(p_{43}, m_{\pi(43)})$ is as follows.

$$Q_{44} = Q_{43}^{31-0} + (Q_{40}^{31-0} + \Phi_{43}(Q_{43}^{31-0}, Q_{42}^{31-0}, Q_{41}^{31-0}) + m_{\pi(43)}^{28-21} + k_{43}) \lll s_{43}. \quad (10)$$

By considering two possible carried number patterns from bit 20 to 21, we obtain two candidates of bit positions 21-28 of $m_6 + (Q_{40} + \Phi_{43} + k_{43})$. Let the value after the left rotation by $23(= s_{43})$ bits be v, and thus, we obtain two candidates of v^{19-12}. Finally, by considering two carried number patterns from bit 11 to 12 in the addition of Q_{43}, we obtain two candidates of Q_{44}^{19-12} for each v^{19-12}. (In total, we obtain 2^2 candidates of Q_{44}^{19-12} for each p_{43}.)

Forward computation for Step 44. The equation for Q_{45} is as follows.

$$Q_{45} = Q_{44}^{19-12} + (Q_{41}^{31-0} + \Phi_{44}(Q_{44}^{19-12}, Q_{43}^{31-0}, Q_{42}^{31-0}) + m_{\pi(44)}^{31-0} + k_{44}) \lll s_{44}. \quad (11)$$

We set $m_9(= m_{\pi(44)})$ to $-k_{44}$ to ignore the addition of these values. Bits 12-19 of Φ_{44} can be computed. Then, we obtain two candidates of $(Q_{41} + \Phi_{44})^{19-12}$. After the left rotation by $4(= s_{44})$ bits, known bits are moved to 16-23. Finally, after the addition of Q_{44}, we obtain two candidates of Q_{45}^{19-16} for each $(Q_{41} + \Phi_{44})^{19-12}$. (In total, we obtain 2^2 candidates of Q_{45}^{19-16} for each Q_{44}^{19-12}.)

As a result, by comparing forward and backward computation results, we can compare Q_{44}^{19-12} in total 8 bits with 3 unknown carried numbers and Q_{45}^{19-16} in total 4 bits with 2 unknown carried numbers. Therefore, our attack overall performs 12-bit match with 5 unknown carried numbers.

5.4 Attack procedure

The attack procedure for a given hash value H_n is as follows:

1. Set chaining variables in the initial structure as shown in Figure 3.
2. Set m_2, m_9, m_{15}, and part of m_6 and m_{14} as shown in Figures 3 and 4. Set other message words to randomly chosen values but satisfy the padding.
3. For all possible values of bit positions 0-20 and 29-31 of m_6 and bit positions 0-4 and 17-31 of Q_{14}, in total 44 free-bits,

(a) Compute Q_{11} by equation (3),
(b) Compute $Q_{14}+m_6$ for efficient consistency check. Let this value be C^{1st}.
(c) Do the following.

$$\begin{cases} p_j \leftarrow R_j^{-1}(p_{j+1}, m_{\pi(j)}) & \text{for } j = 13, 12, \ldots, 0, \\ p_{64} \leftarrow H_n - p_0, & \\ p_j \leftarrow R_j^{-1}(p_{j+1}, m_{\pi(j)}) & \text{for } j = 63, 62, \ldots, 51, \end{cases}$$

(d) Compute $Q_{47}^{19-0}, Q_{46}^{19-0}$, and Q_{45}^{19-0} by equations (6), (7), and (8).
(e) Compute two candidates of Q_{44}^{19-12} by equation (9).
(f) Make a table of $(m_6, Q_{14}, C^{\text{1st}}, p_{51}, Q_{47}, Q_{46}, Q_{45}, Q_{44})$.

4. For all possible values of bit positions 20-31 of m_{14} and all bits of Q_{18}, in total 44 free-bits,
(a) Compute Q_{15}, Q_{16}, and Q_{17} as shown in Figure 3.
(b) Compute $((Q_{18} - Q_{17}) \ggg s_{17}) - \Phi_{17} - k_{17}$ for the efficient consistency check. Let this value be C^{2nd}.
(c) Compute $p_{j+1} \leftarrow R_j(p_j, m_{\pi(j)})$ for $j = 18, 19, \ldots, 42$.
(d) i. Compute 2^2 candidates of Q_{44}^{19-12} for each p_{43} by equation (10), and Q_{45}^{19-16} for each Q_{44}^{19-12} by equation (11). In total, for each p_{43}, we obtain 2^4 candidates of $(Q_{44}^{19-12}, Q_{45}^{19-16})$.
ii. Check whether bits 12-19 of Q_{44} and bits 16-19 Q_{45} in total 12 bits are matched with those in the table and C^{2nd} is matched with C^{1st} in the table.
iii. If matched, compute $R_{43}(p_{43}, m_6)$ by corresponding m_6 and check whether bits 9-11 of Q_{44} are matched and the carried number assumption of Q_{44} is correct.
iv. If matched, compute $R_{44}(p_{44}, m_9)$ and check whether bits 0-15 of Q_{45} are matched and the carried number assumption of Q_{45} is correct.
v. Similarly, compute Q_{46} to Q_{51} and check the matching. If all bits are matched, the corresponding (p_0, M) is a pseudo-preimage.

5.5 Complexity evaluation

Let the complexity of 1 step function be $\frac{1}{64}$ MD5 compression function.

Steps 1 and 2: Negligible.
Step 3a: The complexity is $2^{44} \cdot \frac{1}{64}$.
Step 3b: The complexity is much less than $2^{44} \cdot \frac{1}{64}$.
Step 3c: The complexity is $2^{44} \cdot \frac{27}{64}$.
Step 3d: The complexity is $2^{44} \cdot \frac{3}{64}$.
Steps 3e, 3f: The complexity is $2^{44} \cdot 2^1 \cdot \frac{1}{64}$ and provides 2^{45} items in the table.
Step 4a: The complexity is $2^{44} \cdot \frac{3}{64}$.
Step 4b: The complexity is $2^{44} \cdot \frac{1}{64}$.
Step 4c: The complexity is $2^{44} \cdot \frac{25}{64}$.
Step 4(d)i: The complexity is $2^{44} \cdot 2^2 \cdot \frac{1}{64} + 2^{44} \cdot 2^{2+2} \cdot \frac{1}{64}$, and provides 2^{48} candidates.

Step 4(d)ii: Comparison can be performed with negligible cost by the standard meet-in-the-middle method. The number of remaining pairs is $2^{49} (= 2^{45} \cdot 2^{48} \cdot 2^{-12} \cdot 2^{-32})$.

Step 4(d)iii: The complexity is $2^{43} (= 2^{49} \cdot \frac{1}{64})$. The number of remaining pairs is $2^{44} (= 2^{49} \cdot 2^{-3} \cdot 2^{-2})$.

Step 4(d)iv: The complexity is $2^{38} (= 2^{44} \cdot \frac{1}{64})$. The number of remaining pairs is $2^{26} (= 2^{44} \cdot 2^{-16} \cdot 2^{-2})$.

Step 4(d)v: The complexity is negligible compared to those of the other steps.

The sum of the above complexity is $2^{44} \cdot \frac{116}{64} \approx 2^{44.86}$. This means that we can obtain 2^{44} pairs where 12 bits are matched with a complexity of $2^{44.86}$. Therefore, by repeating the above procedure 2^{72} times, we expect to obtain a pseudo-preimage. Finally, the complexity of finding a pseudo-preimage of MD5 is $2^{116.86} (= 2^{44.86} \cdot 2^{72})$, and this is converted to a preimage attack with a complexity of $2^{123.43} \approx 2^{123.4}$ with the conversion algorithm explained in Section 3.1.

In the attack procedure, the dominant memory complexity is for Step 3f, which requires 2^{45} $(m_6, Q_{14}, C^{1\text{st}}, p_{51}, Q_{47}, Q_{46}, Q_{45}, Q_{44})$s to be stored. Therefore the memory complexity of our attack is at most $2^{45} \times 11$ words.

Remarks

Because the value of m_{14}, which is the lower 32-bits of the message length string, is not fixed in our attack, we cannot fix the length of preimage in advance. Therefore, when we convert pseudo-preimages to a preimage, the required message length is different for each pseudo-preimage. This problem is solved by using *expandable message* described in [7]. Note the cost for constructing an expandable message is negligible compared to the complexity of the preimage attack.

6 Conclusion

This paper shows a preimage attack on full MD5. Compared to the previous preimage attacks, we developed several new techniques: the initial structure, which is a generalization of the previous local-collision technique, the cross absorption properties, the partial-fixing technique for unknown carried number behavior, and the efficient consistency check method for the initial structure. By combining these techniques, our attack with a complexity of $2^{116.9}$ finds a pseudo-preimage of full MD5, and with a complexity of $2^{123.4}$ finds a preimage of full MD5. The memory complexity of the attack is $2^{45} \times 11$ words.

References

1. Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In *Workshop Records of SAC 2008*, pages 82–98, 2008.
2. Jean-Philippe Aumasson, Willi Meier, and Florian Mendel. Preimage attacks on 3-pass HAVAL and step-reduced MD5. In *Workshop Records of SAC 2008*, pages 99–114, 2008.

3. Christophe De Cannière and Christian Rechberger. Preimages for reduced SHA-0 and SHA-1. In David Wagner, editor, *Advances in Cryptology — CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 179–202, Berlin, Heidelberg, New York, 2008. Springer-Verlag. (slides on preliminary results were appeared at ESC 2008 seminar `http://wiki.uni.lu/esc/`).
4. Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion attacks on secure hash functions using SAT solvers. In Joao Marques-Silva and Karem Sakallah, editors, *Theory and Applications of Satisfiability Testing — SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 377–382. Springer-Verlag, Berlin, Heidelberg, New York, 2007.
5. Bert den Boer and Antoon Bosselaers. Collisions for the compression function of MD5. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT 1993*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer-Verlag, Berlin, Heidelberg, New York, 1994.
6. Hans Dobbertin. The status of MD5 after a recent attack. *CryptoBytes The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.*, 2(2):SUMMER, 1996.
7. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer-Verlag, Berlin, Heidelberg, New York, 2005.
8. Vlastimil Klima. Tunnels in hash functions: MD5 collisions within a minute. In *IACR Cryptology ePrint Archive: Report 2006/105*, 2006. `http://eprint.iacr.org/2006/105.pdf`.
9. Gaëtan Leurent. MD4 is not one-way. In Kaisa Nyberg, editor, *Fast Software Encryption (FSE 2008)*, volume 5086 of *Lecture Notes in Computer Science*, pages 412–428, Berlin, Heidelberg, New York, 2008.
10. Alfred John Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography.* CRC Press, 1997.
11. Ronald L. Rivest. *Request for Comments 1321: The MD5 Message Digest Algorithm.* The Internet Engineering Task Force, 1992. (`http://www.ietf.org/rfc/rfc1321.txt`).
12. Yu Sasaki and Kazumaro Aoki. A preimage attack for 52-steps HAS-160. In *Preproceedings of Information Security and Cryptology ICISC 2008*, 2008.
13. Yu Sasaki and Kazumaro Aoki. Preimage attack on step-reduced MD5. In Yi Mu and Willy Susilo, editors, *Information Security and Privacy, 13th Australasian Conference, ACISP 2008*, volume 5107 of *Lecture Notes in Computer Science*, pages 282–296, Berlin, Heidelberg, New York, 2008. Springer-Verlag.
14. Yu Sasaki and Kazumaro Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In Josef Pawel Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271, Berlin, Heidelberg, New York, 2008. Springer-Verlag.
15. U.S. Department of Commerce, National Institute of Standards and Technology. *Federal Register /Vol. 72, No. 212/Friday, November 2, 2007/Notices*, 2007. (`http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`).
16. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, Berlin, Heidelberg, New York, 2005.