How to Construct Sufficient Condition in Searching Collisions of MD5

Yu Sasaki[1], Yusuke Naito[1], Jun Yajima[2], Takeshi Shimoyama[2],
Noboru Kunihiro[1] and Kazuo Ohta[1]

[1] The University of Electro-Communications
Chofugaoka 1-5-1, Chofu-shi, Tokyo, 182-8585, Japan
{yu339,tolucky}@ice.uec.ac.jp
[2] FUJITSU LABORATORIES LTD
4-1-1, Kamikodanaka, Nakahara-ku, Kawasaki, 211-8585, Japan

Abstract. In Eurocrypt 2005, Wang et al. presented a collision attak on MD5. In their paper, they intoduced "Sufficient Condition" which would be needed to generate collisions. In this paper, we explain how to construct sufficent conditions of MD5 when a differential path is given. By applying our algorithm to a collision path given by Wang et al, we found that sufficient conditions introduced by them contained some unnecessary conditions. Generally speaking, when a differential path is given, corresponding sets of sufficient conditions is not unique. In our research, we analyzed the differential path found by Wang et al, and we found a different set of sufficient conditions from that of Wang et al. We have generated collisions by using our sifficient conditions.

1 Introduction

MD5 is a hash function proposed by Rivest in 1992 [3]. In Eurocrypt 2005, Wang et al. presented a collision attack on MD5 [5]. This attack found collisions of MD5 with complexity 2^{37}. This attack is very efficient compared to the complexity of an attack based on birthday paradox (2^{64}). However, the message differential used in this attack is given without any reasoning explanation. In their attack, they first construct sufficient conditions which are needed to generate collisions. Then, they propose message modification techniques in order to satisfy these conditions.

Therefore, the procedure to begin a collision attack is as follows.

Step 1: Find message differentials and a differential path which generate collisions with high probability.
Step 2: Construct Sufficient Conditions which guarantees that desirable differential is always calculated.
Step 3: Find message modifications which can satisfy sufficient conditions with high probability.

In this paper, we explain how to work on Step 2, that is, how to construct sufficient conditions when a differential path is given. So far, a few papers have tried to analyze the sufficient condition table given by Wang et al. [2], [7]. However, these papers did not explain the systematic method to construct sufficient conditions. In this paper, we propose an algorithm to construct sufficient conditions. This paper does not mention Step 1 and Step 3. So far, no paper has been published about Step 1. In SCIS 2006, Yajima et al. publishes a paper which explains how to start the analysis on Step 1 [8]. Whereas, some papers have tried to improve Step 3 [1], [4], [6].

2 Discription of MD5

MD5 has the Markle/Damgård structure. MD5 compresses an arbitrary length messge into an 128-bit hash value. An input message is divided into 512-bit message blocks $(M_0, \ldots M_{n-1})$. First, the output of compression function H_1 is calculated by M_0 and H_0. Here H_0 is defined as follows:

$$H_0 = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)$$

The process of calculating H_1 is called the 1st block. Second, H_2 is calculated by M_1 and H_1. This process is called the 2nd block. Similarly, compression function is calculated until the last message M_{n-1} is used. Let H_n be the hash value of the message.

Compression Function

The input of compression funciton is a 512-bit message M_j. At the first, M_j is divided into 32-bit messages $(m_0, \ldots m_{15})$. The compression function consists of 64 steps. Step 1 to 16 are called 1st round, step 17 to 32 are called 2nd round, step 33 to 48 are called 3rd round, and step 49 to 64 are called 4th round. In each step, one of chaining variabls $a_i, d_i, c_i, b_i (1 \leq i \leq 16)$ is updated in this order. The figure of the calculation for each step is shown in Figure 1.

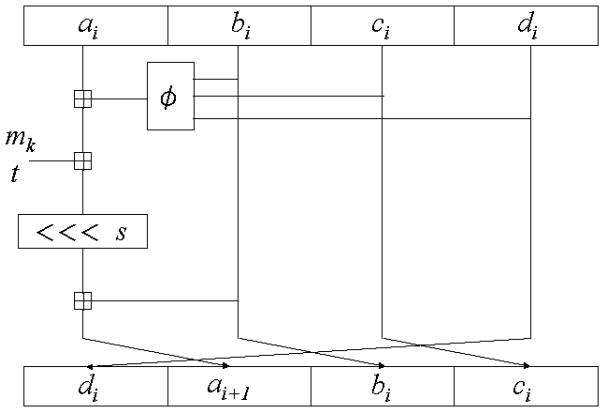

Fig. 1. Structure of the Compuression Function

In Figure 1, s denotes a left cyclic shift number defined in each step. t denotes a constant defined in each step. m denotes a messge, and which m is used is defined in each round. ϕ is a non-linear boolean function defined in each round. Details of ϕ is as follows.

- 1st round: $\phi(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
- 2nd round: $\phi(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$
- 3rd round: $\phi(X, Y, Z) = X \oplus Y \oplus Z$
- 4th round: $\phi(X, Y, Z) = (X \vee \neg Z) \oplus Y$

After 64 steps are calculated, update initial values for the next message block as follows.

$$
\begin{aligned}
aa_0 &\leftarrow a_{16} + a_0 \\
dd_0 &\leftarrow d_{16} + d_0 \\
cc_0 &\leftarrow c_{16} + c_0 \\
bb_0 &\leftarrow b_{16} + b_0
\end{aligned}
$$

At the last, let H_{j+1} be a concatination of these values.

$$H_{j+1} \leftarrow (aa_0 | bb_0 | cc_0 | dd_0)$$

3 Sufficient Condition Construct Algorithm (SC algorithm)

Sufficient condition is constructed in order to guarantee that desirable differential is always calculated. Therefore, differential path search algorithm should be executed before the SC algorithm. In this paper, we assume that diffrentials of chaining variables for all steps are given in advance. If you want to get the detaild information for differential path search algorithm, please refer to [8].

The SC algorithm is going backward from the last step of the compression function. Therefore, if given collision differenatials are 2-block differentials, we start the SC algorithm from step 64 in the 2nd block.

3.1 Notation

To explain the SC algorithm, we need to explain notations. Let x_i be one of chaining variables a_i, b_i, c_i, d_i. Then, $x_{i,j} (1 \leq j \leq 32)$ represents the value of the j-th bit of chaining variable x_i. $x_i[j]$ and $x_i[-j]$ represent differentials of the j-th bit in chaining variable x_i. $x_i[j]$ means that differential of the j-th bit in x_i is 1, that is, $x'_{i,j} - x_{i,j} = 1$. Whereas, $x_i[-j]$ means that differential of the j-th bit in x_i is -1, that is, $x'_{i,j} - x_{i,j} = -1$.

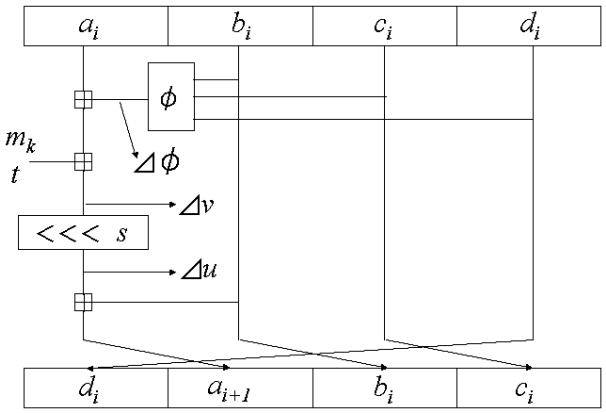

Fig. 2. Notations

In Figure 2, we name the differential after the rotation as Δu, we name the differential before the rotation as Δv, and we name the differential of the output of ϕ as $\Delta\phi$. Furthermore, $\Delta\phi_i$ denotes the $\Delta\phi$ in the i-th step, and $\Delta\phi_{i,j}$ denotes the j-th bit of $\Delta\phi_i$.

3.2 Calculation for $\Delta\phi$

We assume that differentials of all chaining variables are given in advance. Therefore, we assume that Δa_{i+1}, $\Delta a_i, \Delta b_i, \Delta c_i$ and Δd_i in Figure 2 are given in advance. To control the behaviour of the boolean function ϕ, we first need to calculate ideal $\Delta\phi$ in the following way:

Step 1: Calculate $\Delta u = \Delta a_{i+1} - \Delta b_i$.
Step 2: Compute all possible Δv s.t. $\Delta u = (\Delta v \lll s)$.
Step 3: Calculate $\Delta\phi = \Delta v - \Delta m_k - \Delta a_i$.

In step 2, we cannot get all possible Δv by calculateing $\Delta v = (\Delta u \ggg s)$ because of the relationship between the effect of carry and rotation. Therefore, we should try all possible values of u, and calculate a value of $((u + \Delta u) \ggg s) - (u \ggg s)$ as a value of Δv. The value of u is a 32-bit number. Therefore, we need to repeat this calculation 2^{32} times for each step. This calculation is executed by a computer. The algorithm of this computation is as follows.

```
for (u=0x00000000 to 0xffffffff){
   v = u >>> s;
   u'= u + (delta u);
   v'= u'>>> s;
   diff = v' - v;
   counter[diff]++;
}
```

As a result of this computation, we will get at most 4 kinds of Δv. In order to raise the probability that a collision search algorithm succeeds in this step, we choose the most appeared value as a Δv. However, if sufficient conditions cannot be constructed for such Δv, it is possible to choose the differential which has smaller probability. The influence of the decrement of the probability by choosing such differential in the first round is ignoable. However, if it is in the second round or later, the influence of the decrement of the probability may be critical. In this case, choosing another differential path may be required.

3.3 Constructing Sufficient Conditions

Sufficient conditions can be classified in 2 types. One is conditions for controling the length of carry in chaining variabls, the other is conditions for controling the differentials of the output of the non-linear function ϕ. In this section, we explain how to construct each condition.

Conditions for Controling the Length of the Carry These conditions are constructed in order to control the length of the carry in chaining variables. For example, we assume that a differential of a_2 is 2^6. In this case, if $a_{2,7}$ is 0, $a_{2,7}$ changes from 0 to 1, and no carry occurs by this differential. On the other hand, if $a_{2,7}$ is 1, the carry is transmitted to upper bits of a_2, and several bits may be changed by this differential. If the length of the carry is different, the result of moduler integer addition is not changed but the output of the non-linear function ϕ becomes different. Therefore, if we want to stop the differential from expanding, we construct sufficient conditions which prevent carry, in this case $a_{2,7} = 0$. On the other hand, if we want to expand the effect of the differential, we construct the sufficinent condition $a_{2,7} = 1$. Generally speaking, if the number of bits which will be changed by a differential is increased, the number of sufficient conditions where we need to construct is also increased. Therefore, we first construct conditions which stop the differenatial from expanding. Then, if these conditions are contradicted when we construct other conditions, we choose the other conditions which expands the effect of the differential.

Conditions for Controling the Output of the Non-linear Function ϕ The output differential of the non-linear function ϕ can be controled by constructing appropreate sufficient conditions. To explain the basic idea of how to construct conditions, we give a small example. We assume following situations.

- The step is 64. (In this step, input chaining variables for ϕ are c_{16}, d_{16} and a_{16}.)
- The value of $\Delta\phi_{64}$ is $\pm 2^{31}$.
- Differentials of c_{16} is $c_{16}[-26, \pm 32]$.
- Differentials of d_{16} is $d_{16}[-26, \pm 32]$.
- Differentials of a_{16} is $a_{16}[\pm 32]$.

Under above situations, we first need to set $\Delta\phi_{64,26} = 0$. The value of $\Delta\phi_{64,26}$ is calculated depending on $c_{16,26}, d_{16,26}$ and $a_{16,26}$. Therefore, we consider all possible input values and find conditions of input variables whose output differential is 0. Table 1 shows the value of $\Delta\phi_{64,26}$ for all inputs. In Table 1, x, y, z represents the value of $c_{16,26}, d_{16,26}$,and $a_{16,26}$ respectively. The function ϕ is $\phi(x, y, z) = (x \vee \neg z) \oplus y$ since step 64 is in the 4th round. ϕ' is an output of ϕ after message diffrentials are added. Since a_{16} does not have differential in the 26-th bit, the value of z keeps unchanged. On the other hand, since c_{16} and d_{16} have differentials in the 26-th bit, the value of x and y are changed.

Table 1. Constructing Conditions for "$\Delta\phi_{64,26} = 0$"

x, y, z	$\phi(x, y, z)$	$\phi' = \phi(\neg x, \neg y, z)$	$\Delta\phi(= \phi' - \phi)$
0,0,0	1	0	-1
0,0,1	0	0	0
0,1,0	0	1	1
0,1,1	1	1	0
1,0,0	1	0	-1
1,0,1	1	1	0
1,1,0	0	1	1
1,1,1	0	0	0

From the Table 1, we can get to know that a condition which sets $\Delta\phi_{64,26} = 0$ is "$z = 1$", that is, "$a_{16,26} = 1$".

Next, we need to set $\Delta\phi_{64,32} = \pm 1$. Similar to above procedure, we make the table of $\Delta\phi_{64,32}$ for all possible inputs, and extract a sufficient condition. Table 2 shows the table of $\Delta\phi_{64,32}$ for all possible inputs.

Table 2. Constructing Conditions for "$\Delta\phi_{64,32} = \pm 1$"

x, y, z	$\phi(x, y, z)$	$\phi' = \phi(\neg x, \neg y, \neg z)$	$\Delta\phi(= \phi' - \phi)$
0,0,0	1	0	-1
0,0,1	0	0	0
0,1,0	0	1	1
0,1,1	1	1	0
1,0,0	1	1	0
1,0,1	1	0	-1
1,1,0	0	0	0
1,1,1	0	1	1

From the Table 2, we can get to know that a condition which sets the value of $\Delta\phi_{64,32} = \pm 1$ is "$x = z$", that is, "$c_{16,32} = a_{16,32}$".

3.4 Various Techniques to Avoid Contradiction

There are various techniques in order to avoid contradiction of the sufficient condition.

Extending Carry We sometimes need to expand the effect of the differential by using a long carry. This situation occurs when $\Delta\phi_{i,j} \neq 0$ but j-th bit of all input chaining variables to ϕ_i are 0. As long as differentials of all input chaining variables are 0, it is impossible to make $\Delta\phi \neq 0$. Therefore, we extend the carry in the nearest lower bit until j-th bit. We give a small example to explain this.

Now, we assume the value of $\Delta\phi_i$ is 2^{-19}, and bit differentials of x_i, y_i, z_i which are input variables for the ϕ_i are $x_i[5], y_i[-16], z_i[10, 11, -12]$. Since a bit differential which is nearest to 2^{-19} is $y_i[-16]$, we expand the effect of $y_i[-16]$ until the 20th bit by using the carry. To archive this, we construct sufficient conditions $y_{i,16} = 0, y_{i,17} = 0, y_{i,18} = 0, y_{i,19} = 0, y_{i,20} = 1$.

Changing $\Delta\phi$ The value of $\Delta\phi$ can be changed by transforming expression. For example, $\Delta\phi = 2^{20}$ can be changed to $\Delta\phi = -2^{20} + 2^{21}$, or $\Delta\phi = -2^{20} - 2^{21} + 2^{22}$ and so on. This transformation is useful in the following situation.

– Input chaining variables for ϕ_i are x, y and z.
– $\phi_i(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$
– The value of $\Delta\phi_i$ is 2^{14}.
– Differentials of x is 0.
– Differentials of y is 0.
– Differentials of z is $z[-15, -16, -17. - 18, -19, 20]$.

In this case, it is impossible to make $\Delta\phi_{i,15} = 1$ by setting sufficient conditions. However, it is possible to make $\Delta\phi_{i,15} = -1$. Therefore, we replace the value of $\Delta\phi_i = 2^{14}$ with $\Delta\phi_i = -2^{14} - 2^{15} - 2^{16} - 2^{17} - 2^{18} + 2^{19}$, and construct sufficient conditions $x_{15} = 0, \ldots, x_{20} = 0$ in order to make the ideal $\Delta\phi_i$.

3.5 Entire SC Algorithm

In this algorithm, whenever a new condition is constructed, we remember which step it is constructed. If a conditon is contradicted to another condition, we keep the newer condition, and then go back to the step where the older condition has been constructed. This concept is shown in Figure 3.

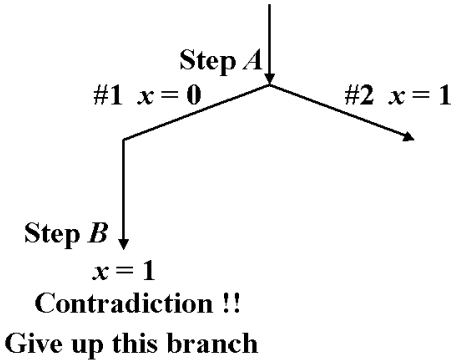

Fig. 3. Concept of SC Algorithm

In Figure 3, we assume that we have two choices in step A, and we choose condition #1 at the first. Therefore, we keep condition #2 as a choice. However, if the chosen condition #1 becomes contradicted to other condition in step B, the algorithm goes back to step A, and chooses condition #2 in this time. Then the algorithm restarts from step A.

In the rest of this section, we explain details of the SC algorithm

Intialization Part In all steps, calculate all possible $\Delta\phi$, then choose one which has the highest probability as $\Delta\phi$. Remember other candidates as choices (Discussed in 3.2).

Main Part The SC algorithm goes backward from the last step of the last block to the 1st step of the 1st block. In each step, we construct sufficent conditions in order to stop the carry of chaining variables. Then we remember the condition which extends the carry as choices (Discussed in 3.3.1). After that, we construct conditions to control $\Delta\phi$. These conditions are constructed from lower bit, that is, from the 1st bit to the 32nd bit. In each bit, construct sufficient conditions in order to control the value of $\Delta\phi$ (Discussed in 3.3.2). If there are more than 1 choises of sufficient conditions for $\Delta\phi$, either of them can be chosen, then we remenber other conditions as choices. If conditions controling $\Delta\phi$ are contradicted or cannot be constructed, we have some choices. In order to solve problems, we first try the process 1. If problems are not solved, we try the process 2 next. Then try the process 3, and finally, we try the process 4.

1. If $\Delta\phi \neq 0$ and all inputs to ϕ do not have a differential, we try to extend the nearest carry until this bit (Discussed in 3.4.1).
2. If $\Delta\phi$ can be changed, we try to change it (Discussed in 3.4.2).
3. If conditions controling $\Delta\phi$ are contradicted to other condition, we keep the new condition and go back to the step where the old condition has been constructed. Then choose other choices for the old condition (See Figure 3).
4. If all choices for old conditions are tried in process 3 and problems are still not solved, it is impossible to construct sufficient conditions for given $\Delta\phi$. Therefore, we choose another $\Delta\phi$ which we got in the Initialization part.

If the process 4 failed, that means no sufficient condition exists for the given differetial path.

4 Unnecessary Sufficient Conditions for the differential path of Wang et al.

To check the availability of the SC algirhtm, we apply this to the differential path given by Wang et al. Through this work, we found that a sufficient condition table given by them contained unnecessary conditions.

4.1 Unnecessary Conditions

So far, some papers have tried to analyze the mehtod of Wang et al, and pointed out mistakes of the sufficient condition table given by Wang et al. [2], [7]. Especially, [2] pointed out the lack of conditions in the 4th round, and claimed that the complexity estimation based on the sufficient condition table given by Wang et al. should be corrected. As a result of applying our SC algorithm to the differential path of Wang et al, we also found the same mistakes.

Furthermore, we newly found unnecessary conditons. An unnecessary condition for the 2nd block is "$b_{16,26} = 1$", and an unnecessary condition for the 1st block is "$c_{3,31} = 0$". Removing "$b_{16,26} = 1$" is important since this condition is related to the estimation of the complexity. By removing "$b_{16,26} = 1$", the complexity of collision search algorithm for the 2nd block becomes $1/2$.

4.2 Unnecessary Carry

Moreover, we found that the length of carry of a_2 for the 1st block could be shorten. In [5], they transmitted carry on the 7-th bit of a_2 until the 23rd bit, that is, the output of a_2 is $a_2[7, 8, \ldots 22, -23]$. However, bit differentials on $a_{2,21}, a_{2,22}$ and $a_{2,23}$ are not used so that we can stop the carry at the 20th step, that is, the output of a_2 becomes $a_2[7, 8, \ldots 19, -20]$. To archive this, we first change the condition on $a_{2,20}$ from "$a_{2,20} = 0$" to "$a_{2,20} = 1$" Then, we can remove sufficient conditions for $a_{2,21}, a_{2,22}$,$a_{2,23}$ and conditions controling output of the non-linear function ϕ on these bits. As a result, we can remove following conditions from the condition table of Wang et al. "$b_{1,21} = c_{1,21}$", "$b_{1,22} = c_{1,22}$", "$b_{1,23} = c_{1,23}$", "$a_{2,21} = 0$", "$a_{2,22} = 0$", "$a_{2,23} = 1$", "$d_{2,21} = 1$", "$d_{2,22} = 1$", "$d_{2,23} = 1$", "$c_{2,22} = 1$", "$c_{2,23} = 1$".

4.3 Merit of Removing Unnecessary Conditions

So far, we pointed out 12 unnecessary conditions in the 1st round, and removed them. However, the complexity of a collision search algorithm is not reduced and computing time of the algorithm is not changed, since these conditions could be corrected with very high probability by single-message modification written in [5]. However, they are still worth removing. In ISEC Nov 2005, an efficient collision search algorithm on MD5 was presented [4]. In this attack, multi-message modification using extra condition was proposed. If unnecessary conditions in the 1st block are removed, we can set extra conditions on those bits. It may result in making sufficient conditions in the 2nd round be correctable. Therefore, we should remove unnecessary conditions as soon as we find them.

Table 3. Generated Collision Messages by Using Our Sufficient Conditions

M_0	0x8b075d00f54501bce81f9cab86312f9d3a8bdca58446d56583e9e8365f99ddba 069badd582343c027f16e96793f95b7bdcdbe711c0dc183a6966bb7243c35a00
M_1	0x1e04541e498038f69d530565fafaf248484f373d4e37823ed76b4922c0b60954 fa3f9f9189df3b0e6307e1fad5ddcc4ff36210cafacc0b54f767ebf2b9391100
M'_0	0x8b075d00f54501bce81f9cab86312f9dba8bdca58446d56583e9e8365f99ddba 069badd582343c027f16e96793f9db7bdcdbe711c0dc183ae966bb7243c35a00
M'_1	0x1e04541e498038f69d530565fafaf248c84f373d4e37823ed76b4922c0b60954 fa3f9f9189df3b0e6307e1fad5dd4c4ff36210cafacc0b547767ebf2b9391100
Hash value	0x2e3757de930d2d15b1c77fa1ee924471

5 Different Sufficient Conditions for the differential path of Wang et al.

In our analysis, we found that the number of sets of sufficient conditions which guarantee that given path is always calculated is not unique. There may exists more than 1. To explain this, we give a small exmaple. We assume following situations.

- Input chaining variables for ϕ_i are x, y and z.
- $\phi_i(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$
- The value of $\Delta\phi_i$ is 0.
- Differentials of x is $[\pm 32]$.
- Differentials of y is $[\pm 32]$.
- Differentials of z is 0.

In this case, we need to construct sufficient conditions which set $\Delta\phi_{i,32} = 0$. Table 4 is the table of $\Delta\phi_{i,32}$ for all possible inputs.

Table 4. Constructing Conditions for "$\Delta\phi_{i,32} = 0$"

x, y, z	$\phi(x, y, z)$	$\phi' = \phi(\neg x, \neg y, z)$	$\Delta\phi(= \phi' - \phi)$
0,0,0	0	1	1
0,0,1	1	1	0
0,1,0	0	0	0
0,1,1	1	0	-1
1,0,0	0	0	0
1,0,1	0	1	1
1,1,0	1	0	-1
1,1,1	1	1	0

From the Table 4, it can be said that required conditions are "$x = 0, y \neq z$" or "$x = 1, y = z$". This is an example of the situation that we have some choices to construct conditions. This fact indicates that it may be possible to construct sufficeint conditions which guarantee the differential given by Wang et al, but different from their condition table. In our research, we found such conditions for their differential, and try them instead of their sufficient conditions. In this computational experiment, conditions we changed are $d_{2,32}, a_{2,32}$, $a_{2,20}$ and $b_{1,32}$. We also removed unnecessary conditions where we explained in section 4. As a result, we could find a collision. We show a collision which was generated by using our sufficient conditions in Table 3.

6 Conclusion

In this paper, we proposed an algorithm for constructing sufficient conditions when a differential path is given. Then we applied this algorithm to the differential path given by Wang et al. [5]. As a result of this work, we found that 13 conditions can be removed from the table of the sufficient condition of Wang et al. One of the removed 13 conditions is a condition in the 4th round of the 2nd block. Therefore, the complexity of collision search algorithm for the 2nd block whose estimation of the complexity is based on the sufficient condition given by Want et al. becomes 1/2. Since [2] pointed out that [5] was lacked a condition in the 4th round of the 2nd block, by considering [2] and our result, total complexity becomes the same with [5].

We also searched a set of sufficient conditions which guarantees the differential path given by Wang et al. but different from their condition table. We changed 4 conditions in this work. At the last, we succeeded in generating a collision by using our sufficient conditions.

References

1. Vlastimil Klima: Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications, Cryptology ePrint Archive 102, April 2005, http://eprint.iacr.org/2005/102.pdf
2. Jie Liang, Xuejia Lai: Improved Collision Attack on Hash Function MD5, Cryptology ePrint Archive 425, November 2005, http://eprint.iacr.org/2005/425.pdf
3. Ronald Rivest: The MD5 Message Digest Algorithm, CRYPTO'90 Proceedings, 1992, http://theory.lcs.mit.edu/~rivest/Rivest-MD5.txt
4. Yu Sasaki, Yusuke Naito, Noboru Kunihiro, Kazuo Ohta: Improved Collision Attack on MD5, ISEC2005-104, pp. 35-42, 2005.
5. Xiaoyun Wang, Hongbo Yu: How to break MD5 and Other Hash Functions, Advances in EUROCRYPT2005, LNCS 3494, pp. 19-35, Springer-Verlag, 2005.
6. Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yu: Finding Collisions in the Full SHA-1, Crypto 2005, LNCS 3621, pp. 17-36, 2005
7. Jun Yajima, Takeshi Shimoyama: On the collision search and the sufficient conditions of MD5, ISEC 2005-78, pp.15-22, 200,
8. Jun Yajima, Takeshi Shimoyama, Yu Sasaki, Yusuke Naito, Noboru Kunihiro, Kazuo Ohta: How to construct a differential path of MD5 for collision search, SCIS 2006, 2006.