

Surveying And Reproducing MD5 Fast Collision Attack Algorithms

Meng Zheng
Electrical Engineering
Delft University of Technology
Delft, Netherlands
M.Zheng-3@student.tudelft.nl

Yuhang Tian
Embedded Systems
Delft University of Technology
Delft, Netherlands
Y.Tian-13@student.tudelft.nl

Abstract—This paper is the survey for exploring the MD5 fast collision based on M. Stevens' paper published in 2012 where he extends the research carried out by Wang et al. and shows that the original structure of differential path cannot be proved as satisfying the "sufficient conditions". Wang's collision method has the limitation of the identical prefix, whereas Stevens breaks this limitation to make the method become chosen-prefix collision attacks, which is the main contribution for him compared to primitive achievements. Our work, in this survey, is mainly 1) to explain the significance of MD5 collision, 2) to reproduce the fast collision attack algorithm for MD5, including the methods of X. Wang, V. Klima and M. Stevens 3) to compare the different implementations of collision above.

Index Terms—MD5, collision attack, cryptography

I. INTRODUCTION

This section will mainly demonstrate some fundamentals about the hash function and give a brief view of the Merkle-Damgard (MD) family, especially MD5.

For a one-way hash function, the collision attack is to find two different messages M and M' that have the identical hash values $H(M) = H(M')$. In terms of the Merkle-Damgard family, like MD5, the most efficient technology to find the collisions for them is *differential cryptanalysis*. The core idea is to investigate and control the differences between two preimages when they propagate through the hash blocks and to generate two identical digests at the end. Following this idea, Xiaoyun Wang et al., published a paper in 2004 and proposed an effective attack for MD5 and achieved the collision attack. The paper points out that using two consecutive blocks with some constraints can generate two 1024-bit messages with identical hash digests - M_1 varies from M_2 ($M_1 \neq M_2$) but $MD5(M_1, IHV_i) = MD5(M_2, IHV_i)$.

MD5 structure has the property that if $MD5(x) = MD5(y)$ then $MD5(x||z) = MD5(y||z)$ where " $||$ " denotes concatenation. According to this structure-property and combining the result provided by Wang, Dan Kaminsky in 2005 created *Stripwire* to mislead integrity checking, and so did Ondrej Mikle. This kind of attacks based on Wang's knowledge is *identical-prefix collision attack*. However, this type of attacks is quite limited as it requires that the initial IHV for those two consecutive blocks ought to be the same.

Later on, in 2007, Marc Stevens made great progress who removed that constraint - no requirement for identical prefix

anymore - and he generated *chosen-prefix collision attacks* for MD5 which was also his main contribution. Without the restriction of the identical prefix, he applied this attack to forgery certificates such as rogue X.509 CA. X.509 was widely adopted to guarantee the security of the HTTPS website. The result reassured that MD5 could not be a secure certifying scheme anymore.

The following report will be generally divided into four sections. Firstly, it will detail the structure of the MD5 algorithm and give demos with both python and java versions. Following that, it will demonstrate the identical-prefix collision attack created by Wang et al. which is a brilliant attacking method different from the original methods based on brute-forcing or birthday-paradox and will show how this affects the secrecy of MD5, and it will present some other modified or even advanced attacks based on the idea from Wang. Then, it will quantitatively compare the performances among them. Finally, it will give a summary that consists of the reflection for the project and suggestions for future work.

II. MD5 MESSAGE-DIGEST ALGORITHM

This section will mainly present the forward computation of the MD5 message-digest algorithm. It will divide the whole algorithm into several parts in order to analyze it. There are already exhaustive researches and sufficient papers relevant to the computing process of MD5, thus only the core concept will be detailed.

The MD5 message-digest algorithm, MD5 algorithm for short, takes the number of N 512 bits messages including padding and an initial 128 bits IHV as inputs to generate a 128 bits fingerprint. The whole algorithm process can be separated into three parts.

Part I - padding: Initially, it uses method 1 padding - append a bit '1' to the message and then append many bits '0' - to make the length of the padded message become 448 (mod 512). After that, for the last remaining 64 bits, it appends the length of the original unpadded message. As a consequence, the message length is now $512N$ where N is an integer.

Part II - segmentation: It will divide the padded message into N 512-bit segments, for each of which it will assign a block that takes a segment and a 128-bit IHV_{in} as inputs and output a 128-bit IHV_{out} . Except for the first block which

IHV is set manually, the input IHV_{in} of a block is given by the output IHV_{out} from the former one. All the blocks are concatenated in serial like Fig. 1.

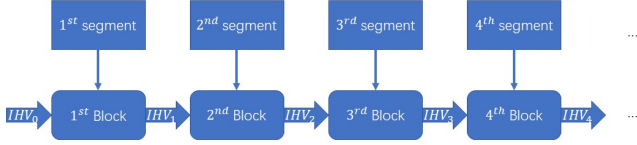


Fig. 1. Blocks Ordering

Part III - computation: For each block, it will continue to divide the input 512-bit segment into 16 words (32 bits). It will run 4 rounds and each round consists of 16 steps; as a result, there are 64 steps in total. Each step includes some bit-wise operations and the operations may vary among steps. One-step structure has been shown in Fig. 2. The one-step process can be used a mathematical formula to represent as:

$$(A', B', C', D') = (D', B, B', C') \quad (1)$$

$$B = RL[f_t(B, C, D) + A + m_i + K_i, s_i] + B \quad (2)$$

where RL is the Left Rotation function, f_t is the compression function varying from each other every 16 steps, m_i is the i^{th} word, K_i is the addition constant, s_i is the rotation constant. More details can be found in paper [2]. If the current block is not the last, it will propagate the output to the next block as the input for the next on. Otherwise, in the final step of a block, it will add the current A', B', C', D' to the very beginning $IHV_{in} = A, B, C, D$ as the output IHV_{out} :

$$IHV_{out} = (A + Q61, B + Q64, C + Q63, D + Q62) \quad (3)$$

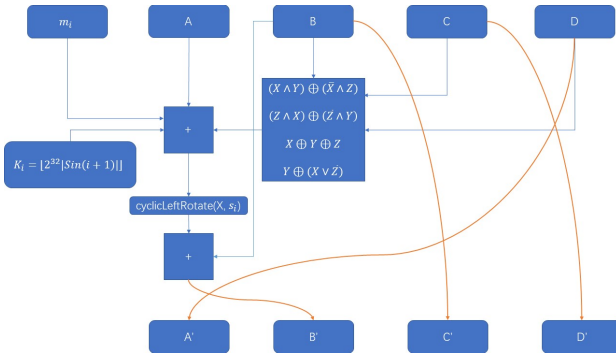


Fig. 2. MD5 Main Structure

For instance, the pre-image "TU Delft" is going to be computed by MD5. Before being fed into the blocks directly, it needs to be encoded and its original length should be calculated before the padding.

$$\begin{aligned} \text{Encode("TU Delft")} &= 54552044 \ 656c6674 \ \text{hex} \\ |\text{Encode("TU Delft")}| &= 40 \ \text{hex} \end{aligned}$$

Then it should be padded as mentioned in part II. It should be noticed that the appending follows the *little-endian*.

```
54552044 656c6674 80000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 40000000 00000000
```

Since the total length after padding is merely 512 bits, it only requires one block to contain. After 64 steps computation shown in part III, the 128 bits digest can be obtained. The code for the whole process is provided: MD5.ipynb (requires Jupyter Notebook) and MD5.java (requires JDK).

```
467764e7 f10c7769 5db73ab3 d414c65d hex
```

The MD5 algorithm invented by Ronald Rivest in 1992. Although it met some threatens in 1996, it had been used for approximately 12 years before Wang et al. found an effective way to find a collision. In the next section, it will show how Wang's elegant approach makes the MD5 algorithm become history.

III. MD5 COLLISION ATTACK

This section will mainly introduce the method created by Wang et al., who published her paper in 2004 where she unprecedentedly found an efficient scheme to build a collision attack for MD5. After that, it will provide the idea of how to build a real collision from shallow to deep.

A. Wang Collision Theory

The theorem behind the identical-prefix attack invented by Wang et al. is *differential path*. Taking two messages $M = \{m_0, m_1, \dots, m_{n-1}\}$ and $M' = \{m'_0, m'_1, \dots, m'_{n-1}\}$ ($\exists i \in [0, k)$ s.t. $m_i \neq m'_i$ and $\forall i \in [k, n)$ s.t. $m_i = m'_i$) and identical prefix $IHV_0 = IHV'_0$ as the initial inputs, the initial difference between IHV s is 0, denoted as $\Delta IHV_0 = 0$. When these two different messages propagate through two different paths consisting of MD5 blocks, we record the intermediate IHV s of these two paths as:

$$\begin{aligned} \Delta IHV_0 &\xrightarrow{m_0, m'_0} \Delta IHV_1 \xrightarrow{m_1, m'_1} \Delta IHV_2 \rightarrow \\ &\dots \rightarrow \Delta IHV_{n-1} \xrightarrow{m_{n-1}, m'_{n-1}} \Delta IHV_n \end{aligned}$$

If at a position where the intermediate IHV s become no differences again that $\Delta IHV_k = 0$ ($k \neq 0$), from then on, all the remaining blocks' outputs will be identical $\Delta IHV_i = 0$ ($i \geq k$) due to the property of MD5 mentioned in section I. If any two non-identical messages found can generate this phenomenon, then the *differential path collision* is found. Based on this differential path idea, Wang uses two consecutive blocks, fabulously, the latter of which can eliminate the differences caused by the former one, like Fig. 3 shown.

In order to use the latter block to cancel the variations caused by the former one, there are some restrictions for the two 1024-bit messages also with some sufficient conditions for the intermediate IV s - the IV represents the temporary output

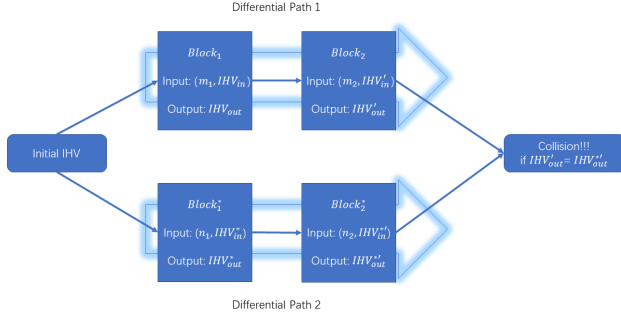


Fig. 3. Differential Paths

for each step. 1) For two 512-bit messages of the first block, $\delta m_4 = 2^{31}$, $\delta m_{11} = 2^{15}$, $\delta m_{14} = 2^{31}$ and other $\delta m_i = 0$.

$$\delta m_{14} = 2^{31}, \delta m_{11} = 2^{15}, \delta m_{14} = 2^{31}, \delta m_{i \in \text{other}} = 0 \quad (4)$$

2) For the second block two 512-bit messages, $\delta m_4 = 2^{31}$, $\delta m_{11} = 2^{-15}$, $\delta m_{14} = 2^{31}$ and other $\delta m_i = 0$.

$$\delta m_{14} = 2^{31}, \delta m_{11} = -2^{15}, \delta m_{14} = 2^{31}, \delta m_{i \in \text{other}} = 0 \quad (5)$$

3) For the output IHV_1 and IHV'_1 of the first two blocks on different paths - $\delta IHV_1 = \{\delta IV_{1A}, \delta IV_{1B}, \delta IV_{1C}, \delta IV_{1D}\}$ represents their differences, it requires that $\delta IV_{1A} = 2^{31}$, $\delta IV_{1B} = 2^{31} + 2^{25}$, $\delta IV_{1C} = 2^{31} + 2^{25}$ and $\delta IV_{1D} = 2^{31} + 2^{25}$.

$$\delta IHV_1 = \{2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25}\} \quad (6)$$

4*) For the output IHV_2 and IHV'_2 of the second two blocks, $\delta IV_{2A} = 0$, $\delta IV_{2B} = 0$, $\delta IV_{2C} = 0$ and $\delta IV_{2D} = 0$.

$$\delta IHV_2 = \{0, 0, 0, 0\} \quad (7)$$

These four requirements are preliminary to generate the cancellation and get the *differential path collision* at the end. The more specific conditions for each step's intermediate IV has been shown in paper [2].

The conditions mentioned above can increase the probability of finding a collision successfully at the end. Compared with the brute-force attack which result can only be verified at the final stage, the sufficient conditions between the steps can determine the running program whether it needs to restart searching halfway when any of the conditions are not fulfilled.

B. Wang Collision Implementation

Starting from this paragraph, it will demonstrate more details about how to implement a collision algorithm from a draft intuitive version to a better and complete one.

By observing the block structure of MD5 in Fig. 2 along with its one-step computation function Eq. 2, we can find that some parameters like addition constant, rotation constant and compress function are fixed for each step, and the input word m_i acts like a variable which determines the output $\{A, B, C, D\}$. Furthermore, most of the operations are added to B whereas A, C and D are only changed the positions after each step. That is the reason why only B has a condition table.

Since we have known all the restrictions/conditions on Q_t (the t^{th} step B , here $t = i$), intuitively, we can randomly generate a m_i corresponding for a step and make a loop until it finds a suitable m_i that generates the B satisfying Q_t for this step. It looks feasible, but beginning from the second block, there are two inputs which are random word m_i and intermediate value IV_{i-1} generated from the former random m_{i-1} .

In order to avoid introducing two randomnesses at one time, we can use the "inverse way". As the conditions of Q_t s are predefined, we can use *pseudo number generator* to generate the Q_t which suffices the conditions, and use this formula to reversely compute the word m_i :

$$m_i = RR(Q_{i+1} - Q_i, s[i]) - f_t(Q_i, Q_{i-1}, Q_{i-2}) - Q_{i-3} - K[i] \quad (8)$$

where RR is the Right Rotation function, Q_t is the intermediate output value of the t^{th} step. Eq. 8 reveals that the 16 words (m_0, m_2, \dots, m_{15}) can be determined only by the initial seventeen intermediate output (Q_0, Q_1, \dots, Q_{16}). For each m_t , it needs 5 Q s to decide. For instance, calculating m_6 needs foreknowledge about Q_7, Q_6, Q_5, Q_4 and Q_3 . Therefore, we can randomly generate satisfying Q_t , $3 \leq t \leq 16$ (In terms of Wang's version, there is no restrictions for Q_1 and Q_2) and reversely compute the corresponding m_t , then use the known Q_t and calculated m_t to forward compute the remaining Q_t s. The pseudo codes for the two blocks are provided below Alg. 1.

Algorithm 1 Wang Collision Algorithm

Input: 128-bit Initial $IHV_0 = \{A, B, C, D\}$

Output: 128-bit Digest $IHV_2 = \{\hat{A}, \hat{B}, \hat{C}, \hat{D}\}$

```

1: while Block 1 conditions fulfilled do
2:   while conditions fulfilled do
3:     generate  $Q[3], Q[4], \dots, Q[16]$ 
4:     testing several of them each time, and for each
       time,
5:       if test not passed then
6:         continue
7:       end if
8:   end while
9:   calculate  $x[6], x[7], \dots, x[15]$  by Eq. 8
10:  for  $i \leftarrow 0$  to  $ffff_{hex}$  do
11:    generate  $Q[17]$ 
12:    calculate  $Q[18], Q[19]$  by Eq. 2
13:    if test passed then
14:      break
15:    end if
16:  end for
17:  calculate  $x[1]$  by Eq. 8
18:   $Q[20] \leftarrow Q[19] \& 80000000_{hex}$ 
19:  while  $Q[20] \neq Q[19] \& 80000000_{hex}$  do
20:    calculate  $x[0], x[1], x[2], x[3], x[4], x[5], Q[1], Q[2]$ 
21:    calculate  $Q[21], Q[22], \dots, Q[64]$ , for each time of
       calculation,
22:    if test not passed then
23:      continue

```

```

24:     end if
25: end while
26: calculate  $IHV_1$  by Eq. 3
27: end while
28:
29: while Block 2 conditions fulfilled do
30:     while conditions fulfilled do
31:         generate Q[1] and Q[2]
32:         if test not passed then
33:             continue
34:         end if
35:         calculate xx[1] by Eq. 8
36:         ... (the same for Q[3], Q[4], ..., Q[12])
37:         generate Q[13]
38:         if test not passed then
39:             continue
40:         end if
41:         calculate xx[13] by Eq. 8
42:     end while
43:     for  $i \leftarrow 0$  to  $10000_{hex}$  do
44:         generate Q[15]
45:         if test not passed then
46:             continue
47:         end if
48:         calculate xx[14] by Eq. 8
49:         generate Q[16]
50:         calculate xx[15] by Eq. 8
51:         calculate Q[17] by Eq. 2
52:         if test not passed then
53:             modify xx[1]
54:             update Q[2]
55:             recalculate Q[17] by Eq. 2
56:             if Q[2] condition not fulfilled then
57:                 break
58:             end if
59:             recalculate xx[2],xx[3],xx[4],xx[5] by Eq. 8
60:         end if
61:         ... (the same for Q[18] and Q[19] calculate Q[20],
        Q[21], ..., Q[64], for each time of calculation,
62:         if test not passed then
63:             continue
64:         end if
65:         break
66:     end for
67:     calculate  $IHV_2$  by Eq. 3
68: end while

```

The original code of Wang's version cannot be found through the internet, but there exist some duplicates made by other researchers. The above pseudo-code is made according to S. Thomsen's wangmd5.c who claims that the idea is retrieved from Wang. We provide WangMD5.cpp and WangMD5.java versions. It approximately costs 20 to 30 minutes to find a pair of messages which outputs the identical digests.

C. Implementation Along with Tunnels

Other than improving the structure, in 2006, V. Klima generates a new idea of improving the previous method of finding collisions. He substitutes the multi-message modification in the original code by tunnels which exponentially increases the finding speed of collision. Using his method, it costs an average of fewer than 2 minutes to find a collision pair.

In the previous section, multi-message modification is based on that if $M - M'$ fulfills the specific constraints C , then $MD5(M) - MD5(M')$ fulfills the corresponding constraints C^* , so the "sufficient conditions" are designed for this purpose. However, in terms of tunnelling, it creates multiple subsets of sufficient conditions to segment the original testing process, and at the same time, it guarantees that if message M fulfills those conditions, then the transformed message can also fulfill. Instead of using multi-message modification, tunnels can significantly reduce the searching time and increase the probability of meeting the specific conditions.

The pseudo code for the implementation can be found below Alg. 2. In block 1, tunnels $Q_{10}, Q_{20}, Q_{13}, Q_{14}, Q_4$ and Q_9 are employed, while in block 2, tunnels $Q_{16}, Q_{1,2}, Q_4$ and Q_9 .

Algorithm 2 Klima Collision Algorithm

Input: 128-bit Initial $IHV_0 = \{A, B, C, D\}$

Output: 128-bit Digest $IHV_2 = \{\hat{A}, \hat{B}, \hat{C}, \hat{D}\}$

```

1: while block 1 conditions fulfilled do
2:     generate Q[1], Q[3], ..., Q[17] fulfilling conditions
3:     calculate x[0], x[1], ..., x[15] by Eq. 8
4:     calculate Q[2], Q[18], ..., Q[24] by Eq. 2
5:     for  $itr_{Q_{10}} \leftarrow 0$  to  $2^3$  do
6:         Tunnel  $Q_{10}$ 
7:         for  $itr_{Q_{20}} \leftarrow 0$  to  $2^6$  do
8:             Tunnel  $Q_{20}$ 
9:             for  $itr_{Q_{13}} \leftarrow 0$  to  $2^{12}$  do
10:                 Tunnel  $Q_{13}$ 
11:                 for  $itr_{Q_{14}} \leftarrow 0$  to  $2^9$  do
12:                     Tunnel  $Q_{14}$ 
13:                     for  $itr_{Q_4} \leftarrow 0$  to  $2^0$  do
14:                         Tunnel  $Q_4$ 
15:                         for  $itr_{Q_9} \leftarrow 0$  to  $2^3$  do
16:                             Tunnel  $Q_9$ 
17:                             calculate Q[25], Q[26], ...,
                                Q[64] by Eq. 2
18:                             calculate  $IHV_1$  by Eq. 3
19:                         end for
20:                     end for
21:                 end for
22:             end for
23:         end for
24:     end for
25: end while
26:
27: while block 2 conditions fulfilled do
28:     generate Q[1], Q[3], ..., Q[14] fulfilling conditions
29:     for  $itr_{Q_{16}} \leftarrow 0$  to  $2^{25}$  do

```

```

30: Tunnel  $Q_{16}$ 
31: for  $itr_{Q1,Q2} \leftarrow 0$  to  $2^{Q1Q2_{strength}}$  do
32:   Tunnel  $Q_{1,2}$ 
33:   for  $itr_{Q4} \leftarrow 0$  to  $2^6$  do
34:     Tunnel  $Q_4$ 
35:     for  $itr_{Q9} \leftarrow 0$  to  $2^8$  do
36:       Tunnel  $Q_9$ 
37:       calculate  $Q[25], Q[26], \dots, Q[64]$  by
Eq. 2
38:       calculate  $IHV_2$  by Eq. 3
39:     end for
40:   end for
41: end for
42: end for
43: end while

```

We provide java version source code KlimaMD5.java and jar application KlimaMD5.jar for the testing.

D. Meng's Part

Please start from here and give an appropriate name for your subsection. You are welcome to use and reference the equations motioned in the previous sections.

E. Identical Prefix Attack Applications

Thereafter, people can apply those methods to find as many collisions. In the same year after Wang et al. published their paper, D. Kaminsky preemptively applied their scheme to prove that MD5 Collision adversely impacted file-oriented system auditors, like Tripwire, because Tripwire was usually configured to MD5 hashes by the users. However, in his conclusion, he claimed that the influence was not catastrophic even though the "functionally identical" of MD5 was broken.

Similarly, O. Mikle in 2004, used the knowledge of MD5 Collision trying to attack digital signature with GPG. He claimed that since the scheme was not about the second pre-image attack, the harms of collision would not be serious.

According to the above declarations, Wang's method has two shortages. The first one is that the collision messages are randomly generated which are always meaningless ones. The second one is that the colliding messages must be generated simultaneously along differential paths.

F. Chosen Prefix Attack

The chosen prefix attack is proposed by M. Stevens and defined as the following: for any two chosen message prefixes P and P' , suffixes S and S' can be constructed such that the concatenated values $P||S$ and $P'||S'$ form a chosen-prefix collision for MD5. For the perspective of identical prefix attack, it requires the initial IHV_0 - it can be the digest of the previous block - should be the same for two differential paths, which means that all the block messages before two consecutive collision blocks should be the same (this is not rigours explanation, since different messages may generate the identical digests, but it is infeasible). In contrast, the Chosen

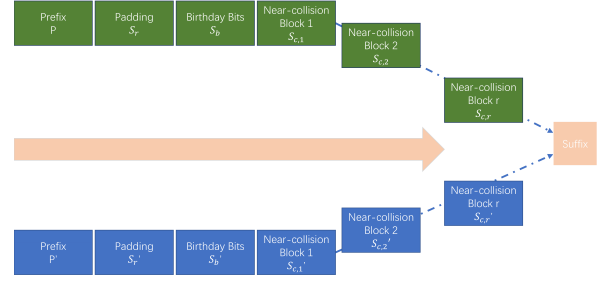


Fig. 4. Chosen-prefix Attack Structure

Prefix Attack does not have such requirements. The structure of the chosen prefix attack has been shown in Fig. 4.

The details of the chosen-prefix collision implementation will not be presented in this paper, as it is not a fast collision method, but it extends the traditional identical-prefix and further damages the secrecy of MD5. Besides, M. Stevens provides a method that only employs one single block to generate the collision.

G. Chosen Prefix Attack Application

The chosen prefix attack can be abused in many scenarios. For instance, inserting two images as suffixes into two different documents. The messages before the colour images act like the prefixes. The MD5 of the message in one document varies from another, but after patching with the colour images, they become the same. In other words, in this situation, $P||S$ and $P'||S'$, P is the actual message containing the correct information whereas P' is the forgery message containing the misleading information. People may point out that the colour image S is different from the image S' , but in reality, it quite difficult to distinguish them using human's vision because the attacker can hide the differences in the pixels if he elaborates them carefully. The image can be a barcode or a logo that usually appear ubiquitously.

If readers are interested in such an application, the website built by M. Stevens provides the way to forgery the X.509 certificates.

IV. ATTACKS COMPARISON

TABLE I
COMPARISON OF MD5 COLLISION SCHEMES

Name	Searching Time	Feature
Wang Modified Version	20 minutes	multi-message modification
Klima Version	1.5 minutes	Tunnels, PoV
Stevens CPC Version	1 hours	prefixes can be chosen

The above testing results are based on CPU AMD 4900H. According to Tab. I, the fast algorithm to find a collision pair of MD5 is the one provided by Klima equipped with tunnels. It should be mentioned that Klima's method also owns the least variance. It sometimes spends a half-hour to obtain the

result using Wang's scheme, but sometimes costs less than ten minutes if lucky; whereas Klima's version often costs less than 2 minutes. Steven's versions are fancy that can choose the prefix or even use a single block to obtain the collision, but the trade-off is spending more time.

V. CONCLUSION AND FUTURE WORK

This report firstly presents how the MD5 message-digest algorithm works and provides the demo for the readers to test. Then, it mainly provides three methods for finding the MD5 collision. The initial one is provided by Wang et al. which is also the first version to fast search the collision pair. The second one is created by Klima who implements the tunnels using PoV to accelerate the searching speed and increase the collision probability. The third one is the chosen-prefix attack invented by Steven. Not only did he propose the idea of a chosen-prefix attack, but he also found the conditions in Wang's first version were insufficient. However, in this paper, we only present his main contribution, Chosen Prefix Collision Attack. Finally, it compares the performances of different methods.

In this paper, we only reproduce the fast collision algorithm of MD5 and do not do much work for other advanced attacks like the chosen-prefix one and their applications. In addition, the idea of MD5 collision can be mimicked to SHA-1 as well. These can be suggestions for future work.

VI. REFERENCES

REFERENCES

- [1] Stevens, M., Lenstra, A. K., & De Weger, B. (2012). *Chosen-prefix collisions for MD5 and applications*. International Journal of Applied Cryptography, 2(4), 322-359.
- [2] Stevens, M. (2012). *Attacks on hash functions and applications*. Mathematical Institute, Faculty of Science, Leiden University, 3.
- [3] Stevens, M. (2006). *Fast Collision Attack on MD5*. IACR Cryptol. ePrint Arch., 2006, 104.
- [4] Wang, X., & Yu, H. (2005, May). *How to break MD5 and other hash functions*. In Annual international conference on the theory and applications of cryptographic techniques (pp. 19-35). Springer, Berlin, Heidelberg.
- [5] Klima, V. (2006). *Tunnels in Hash Functions: MD5 Collisions Within a Minute*. IACR Cryptol. ePrint Arch., 2006, 105.
- [6] Klima, V. (2005). *Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications*. IACR Cryptol. ePrint Arch., 2005, 102.
- [7] Hawkes, P., Paddon, M., & Rose, G. G. (2004). *Musings on the Wang et al. MD5 Collision*. IACR Cryptol. ePrint Arch., 2004, 264.
- [8] Smart, N. P., & Smart, N. P. (2016). *Cryptography made simple*. Springer.
- [9] Mikle, O. (2004). *Practical Attacks on Digital Signatures Using MD5 Message Digest*. IACR Cryptol. ePrint Arch., 2004, 356.
- [10] Kaminsky, D. (2005). *MD5 to be considered harmful someday*. In Aggressive Network Self-Defense (pp. 323-337). Syngress.
- [11] Kashyap, N. D. (2006). *A meaningful MD5 hash collision attack*.