國立交通大學

資訊科學與工程研究所

碩 士 論 文

ＭＤ４ 和 ＭＤ５ 碰 撞 攻 擊 之 研 究

A Study of the Collision Cryptanalysis against MD4 and MD5

研 究 生：陳冠廷

指導教授：曾文貴　教授

中 華 民 國 九 十 五 年 六 月

MD4 和 MD5 碰撞攻擊之研究
A Study of the Collision Cryptanalysis against MD4 and MD5

研 究 生：陳冠廷　　Student：Guan-Ting Chen

指導教授：曾文貴　　Advisor：Wen-Guey Tzeng

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

MD4 與 MD5 碰撞攻擊之研究

學生：陳冠廷　　　　　　　　　　　　　指導教授：曾文貴 博士

國立交通大學資訊科學與工程研究所

摘要

王小雲等人在密碼會議 EuroCrypt2005 上發表他們對 MD4 和 MD5 的碰撞攻擊演算法。之後許多研究者根據他們的結果並相繼提出他們對於碰撞攻擊演算法的改進。其中大部分研究者著重在如何讓訊息修改演算法更有效率。在本篇論文中，我們改進了訊息修改演算法。對於 MD5，我們提出一些訊息修改的方法，能滿足第一個回合的一個充分條件和第二個回合的五個充分條件。對於 MD4，我們修正了之前 MD4 碰撞攻擊演算法的一些錯誤。同時我們實作了對於 MD4 和 MD5 的碰撞攻擊演算法。我們對於 MD5 的碰撞攻擊演算法的實作平均大約需要 1.75 個小時去找出一組碰撞的訊息。根據我們的實驗，我們的實作能在 12 個小時內找到一組碰撞的訊息之機率為 1。

關鍵字：碰撞攻擊，差分攻擊，雜湊，MD4，MD5，訊息修改，充分條件。

A Study of the Collision Cryptanalysis against MD4 and MD5

Student: Guan-Ting Chen　　　　Advisor: Dr. Wen-Guey Tzeng

Institute of Computer Science and Engineering

National Chiao Tung University

Abstract

In EuroCrypt2005, Wang et al. publish their collision searching algorithms for MD4 and MD5. Many researchers follow their results and publish their improvements on the collision searching algorithms. Many of them focus on how to do the message modification efficiently. In this thesis, we improve the message modification techniques. We use our message modification methods to satisfy 1 sufficient condition in the first iteration and 5 sufficient conditions in the second iteration for MD5. For MD4 collision searching algorithm, we correct the errors in the previous results. We implement the collision searching algorithm for both MD4 and MD5. Our implementation of the MD5 collision searching algorithm takes about 1.75 hours to give a collision pair in average. The successful probability to find the collision pair in 12 hours is 1 according to our experiments.

Keywords: Collision Cryptanalysis, Differential Cryptanalysis, Hash, MD4, MD5, Message Modification, Sufficient Conditions.

誌　　　謝

首先要感謝我的指導老師曾文貴教授，在我碩士班的學習過程中，帶領我深入密碼學的領域，我使獲益良多。另外，我要感謝口試委員，交大資工蔡錫鈞教授與中研院資訊科學研究所呂及人教授，在論文上給我許多建議與指導，讓我的論文更加完善。

除此之外，我要感謝實驗室學長成康的指導，實驗室同學君偉、資工 92 級陳榮傑老師密碼理論實驗室漢璋的幫忙，和你們一起討論和學習是一件愉快的事，室友哲瑩在程式上的一些建議和幫忙，以及所有曾經與我一起在資科系計中工作過和一起在實驗室學習的夥伴在精神方面的鼓勵。

最後，我要感謝我的父母、弟弟、小舅和舅媽，不論在精神或物質上都給我極大的支持，讓我能在無後顧之憂的情況下順利完成學業。得之於人者太多，出之於己者太少，要感謝的人實在太多了，無法一一列舉，就感謝天罷。

Contents

List of Tables

Chapter 1

Introduction

Cryptographic Hash Functions are a kind of important primitive in cryptographic applications. Many cryptographic schemes, such as digital signatures, MACs, authentications, certificates, IBE, and so on, use the hash functions as their internal subroutines to construct the whole cryptographic schemes. Some cryptographic schemes that use hash functions as their internal subroutines are proven to be secure by assuming that their underlying hash functions are truly-random functions. This is called the *random oracle model*. But some studies show that the security of the cryptographic schemes under the random oracle model is not guaranteed when their underlying hash functions are not truly-random functions. The studies don't find the insecurity of the existing cryptographic schemes that are proven to be secure under the random oracle model in the real world. Instead, they give a counter example to show that the cryptographic schemes proven secure under the random oracle model are not secure if their underlying hash functions are substituted with a real hash function such as SHA-1, not a truly random function [CGH98, CGH04]. Goldwasser and Kalai [GK03] find that the security of the Fiat-Shamir transformation that transform secure 3-round public-coin identification schemes into digital signature schemes is not guaranteed although the methodology is proven secure under the random oracle model. Therefore, we can't ignore the real structure of the hash functions and just see them as the black-box of the truly-random functions. A cryptographic scheme that uses an insecure hash function as its internal subroutine may be insecure [LWdW05, Mik04, LdW05]. So many researchers do their best to re-evaluate the security of the existing cryptographic hash algorithms and study how to design a secure ones.

There are many existing hash functions, such as MD2 [Kal92], MD4 [Riv90, Riv92a], MD5 [Riv92b],RIPEMD-128 and RIPEMD-160 [DBP96], HAVAL [ZPS92], SHA-1 [FIP95], SHA-256 and SHA-512 [FIP02], Whirlpool [RB00], and so on. Before these hash functions were proposed, the hash functions were constructed from blockcipher. Generally speaking, the encryption and decryption of the blockcipher is less efficient than the compression of the hash function on the same message (or plaintext). If a large file is transfered from Alice to Bob on 100M ethernet, the computation time of the blockcipher-based hash value of the file will be larger than the transmission time. We want the hash value to be computed as fast as possible to meet the transmission speed of the ethernet. Nowadays, when we download files from the FTP or Web site, these files are usually associate with their hash values or PGP digital signatures, that also use cryptographic hash functions as their internal functions, for file integrity. The efficiency of computing the hash values is very important due to their heavy use, and MD2 was not used widely because of its inefficiency. Besides it, some researchers [RC97, Mul04, KM05] figure out the insecurity of MD2. MD4 is an early-appeared hash function that is used to replace MD2 and designed for modern computers. All MD4 operations are basic arithmetic and Boolean operations on 32-bit words that are suitable in modern computers and able to be computed efficiently. So the MD4 hash value can be computed fast by using modern computers. This type of hash functions, referred as *dedicated hash functions*, is quite different from the traditional design of the blockcipher-based hash functions. Generally speaking, the computational cost of a blockcipher is much larger than that of a dedicated hash functions. After MD4 was published, many new dedicated hash functions, such as MD5, HAVAL, RIPEMD, RIPEMD-160, SHA-1, etc, follow the design of MD4. Among these hash functions, MD5 and SHA-1 are used most widely.

On the other hand, some researches evaluate the security of the existing hash functions. Some focus on the Merkle-Damgård structure that is widely used in many hash functions. Some check whether the compression function is secure. After the publication of MD4 and other their derived dedicated hash functions, many researchers started to analyze them. Recently, Wang et al. give collision searching algorithms for MD4 [WLF$^+$05], MD5 [WY05], and

SHA-1 [WYY05]. The time complexity of their collision searching algorithms for MD4 and MD5 are small enough to run on modern PCs. After the publication of their researches on MD4 and MD5 collisions, some researchers [NSKO05, HPR04, Kli05b, Kli05a, SNKO05, LL05] published their results based on Wang et al.'s results subsequently. Some implementations of the collision searching algorithms are also given [Sta05a, Sta05b].

By our study, it is still possible to improve the previous collision searching algorithms. In this thesis, we give some techniques in speeding up the collision searching algorithms. Although Wang et al.'s [WLF+05, WY05] and their subsequent results [NSKO05, HPR04, Kli05b, Kli05a, SNKO05, LL05] are sufficient enough, there's something that can be improved. Most improvements on the collision searching algorithms on MD4 and MD5 focus on message modification that we will introduce in Chapter 4. We improve the message modification techniques to ensure that more sufficient conditions are fulfilled. According to our research, we find that there's something wrong in Naito et al.'s results [NSKO05] for the MD4 collision searching algorithm. We give some message modification methods to correct these errors. For MD5 collision searching algorithm, we give some message modification methods to satisfy the sufficient condition "$c_{5,32} = d_{5.32}$" in the first iteration and the sufficient conditions "$d_{5,18} = 1$", "$d_{5,32} = a_{5,32}$", "$c_{5,18} = 0$", "$c_{5,32} = d_{5,32}$", and "$d_{6,32} = a_{6,32}$" in the second iteration for MD5. Sasaki et al. also give message modification method for the sufficient condition "$c_{5,32} = d_{5.32}$" in the first iteration [SNKO05]. But their method reduce the set size of the collision pair too much. Then we implement the collision searching algorithm for both MD4 and MD5 and it takes about 1.75 hours to given a MD5 collision pair in average. The successful probability to find the collision pair in 12 hours is 1 according to our experiments.

The Organization of the Thesis. In Chapter 2, we give some introduction of the hash functions, MD4 and MD5. In Chapter 3, we review the collision searching algorithms on MD4 and MD5 given by Wang et al. In Chapter 4, we introduce the previous results on message modification techniques, and give our own improvements from theirs. In Chapter 5, we give an introduction of our implementation on MD5 and analyze it. Finally, We conclude this thesis in Chapter 6.

Chapter 2

Description of The Hash Algorithms

2.1 Hash Functions

A cryptographic hash function F with a k-bit output is a function that accepts an arbitrary length input $M \in \{0,1\}^*$ and outputs a fixed length output $H \in \{0,1\}^k$. We call that H is the hash value of the input message M via the hash function F. Cryptographic hash functions at least need to satisfy the following three basic security requirements:

- *Preimage Resistance*: Given a randomly chosen image $H \in \{0,1\}^k$, there is no probabilistic polynomial time adversary with less than about 2^k computations to give a value M such that $F(M) = H$ with an non-negligible probability with respective to its output size k. If the hash function F satisfy this security requirement, we also say that the hash function F is an one-way hash function.

- *Second Preimage Resistance*: Given a randomly chosen value M, there is no probabilistic polynomial time adversary with less than about 2^k computations to give another M', $M' \neq M$, such that $F(M) = F(M')$ with an non-negligible probability with respective to its output size k.

- *Collision Resistance*: There is no probabilistic polynomial time adversary with less than about $2^{\frac{k}{2}}$ computations to find M and M', $M \neq M'$, such that $F(M) = F(M')$ with an non-negligible probability with respective to its output size k. The value $2^{\frac{k}{2}}$ is the theoretic lower bound of the *birthday attack* suppose the hash function F is balance over its image.

The above three properties are the minimal security requirements for the cryptographic hash functions. There are still other security requirements for hash functions on other specific cryptographic applications. We can find that breaking the second preimage resistance property is more difficult than breaking the collision resistance property. But there is no reduction relation between preimage resistance and second preimage resistance.

Many existing cryptographic hash functions are iterated hash functions built from compression functions. We denote the compression function as f, and its input and output as x and y, respectively. The length of x and y is always fixed, and $|x| > |y|$. But the input length of F is arbitrary, so F need to call the compression function f many times to compute the hash value.

Many existing cryptographic hash functions use the Merkel-Damgård construction [Dam89, Mer89] as their iterative structure. It is that, if we want to compute the hash value H of the message M, we compute as follows:

1. We pad the message M according to its padding algorithm. For brevity of explaining, we also denote the padded message as M. After padding, $|M|$ is an integer multiple of $|x| - |y|$ bits.

2. We divide M into n blocks, each block size is equal to $|x| - |y|$, where $n = \frac{|M|}{|x|-|y|}$. For each block of M, we denote the i^{th} block as M_i. So $M = M_1||M_2||\ldots||M_{n-1}||M_n$.

3. We use the following recursive equation to compute the hash values:

$$y_i = f(y_{i-1}||M_i) \quad \text{for } i = 1 \text{ to } n$$

   where y_0 =IV.

4. Finally $H = y_n$ is the hash value of M via the hash function F.

In the step 3 of the above computation of H, IV, or initial value, is a fixed value chosen by the designer of the hash function. In step 1, we mention that the padding algorithm can be split from the Merkel-Damgård construction of the hash functions, as long as the sender and the receiver knows the padding algorithm. Many padding algorithms, such as RC5-CBC-PAD [BR96], random padding, the padding for ESP [Ken05, Page 15], and so on, used in blockcipher can be used in hash functions. In blockcipher, when ciphertext is decrypted then the padded message must be removed to recover the original plaintext so that the padding algorithms must be invertible. Unlike the blockcipher algorithms, the padding algorithms of the hash functions are not necessary to be invertible because the original message M is a plaintext and are known by both the sender and the receiver. The padded message can be also a checksum of the message M or something else. The checksum is also a hash value, but it is not a cryptographic hash value. The checksum algorithms is much more efficient than the cryptographic ones, but it's not necessary to satisfy the three basic security requirements of the cryptographic hash functions. Its output length is usually shorter than that of the cryptographic hash functions. Nowadays, the string $10^{|x|-|y|-(|M|+1 \mod (|x|-|y|))}$ is padded and then the length of the unpadded message is also appeared in the last block after the padded message in many hash functions that use the Merkle-Damågrd structure, such as MD4, MD5, SHA-1 and so on.

Actually the cryptographic hash functions are similar (but not the same) to the blockciphers in many aspects. For a long message, we partition it into several blocks and then use the blockcipher, such as DES, RC5, and AES, to encrypt each block. There are many *mode of operation*s, such as ECB, CBC, CFB, OFB, and so on, used during the encryption or decryption of the blockcipher for long messages. We can also see the Merkle-Damgård construction as the mode used in cryptographic hash functions.

Someone may use $h_i = f(h_{i-1}, m_{i-1})$ to denote the computation of the compression function instead of $y_i = f(y_{i-1}||M_{i-1})$. We also call h_{i-1} as the initial value, h_i as the hash value and m_{i-1} as the message block of the compression function f in this (the i^{th}) iteration in later description. In the following statements, we will use $h_i = f(h_{i-1}, m_{i-1})$ to denote the computation of the compression function instead of $y_i = f(y_{i-1}||M_{i-1})$ for brevity of explaining.

2.2 The Compression Function of MD4

In this section, we denote the input of the compression function of the message digest algorithm MD4 in this iteration as (h, m), where h and m are the initial value and message block of the compression function, respectively. In MD4 spec, h consists of 4 words, i.e., $h = a_0||b_0||c_0||d_0$, and a_0, b_0, c_0 and d_0 are all 32-bit words. The message block m consists of 16 words, we denote $m = m_0||m_1||\ldots||m_{15}$, where m_i are all 32-bit words for all $0 \leq i \leq 15$. There are some internal values used during the computation of the compression function of MD4. In this thesis, we will call these internal values as chaining values, shorten as CV. In MD4, there are four registers that are used to store the CVs, and we denote them as a, b, c, and d. Then given the input (m, h), we set $(a, b, c, d) = (a_0, b_0, c_0, d_0)$, and the output of the compression function is computed as follows:

$$\Sigma_i = a + \phi_i(b, c, d) + w_i + t_i \qquad a = \Sigma_i \lll s_i \qquad \text{if } i \mod 4 = 1$$

$$\Sigma_i = d + \phi_i(a,b,c) + w_i + t_i \qquad d = \Sigma_i \lll s_i \qquad \text{if } i \mod 4 = 2$$

$$\Sigma_i = c + \phi_i(d,a,b) + w_i + t_i \qquad c = \Sigma_i \lll s_i \qquad \text{if } i \mod 4 = 3$$

$$\Sigma_i = b + \phi_i(c,d,a) + w_i + t_i \qquad b = \Sigma_i \lll s_i \qquad \text{if } i \mod 4 = 0$$

for $i = 1$ to 48.

In the above equations, s_i's are step-dependent constants, w_i's are chosen from m_i for $0 \leq i \leq 15$ according to its step i, t_i's are round-dependent constants, and ϕ_i's are round-dependent Boolean functions. All variables used in the above steps are all 32-bit words, and the addition operation is under the group $Z_{2^{32}}$, and the operation "$\lll$" is the cyclic left rotation on 32-bit words.

We divide all the 48 steps of the computation of the compression function into 3 rounds, and each round is of 16-step. The round-dependent Boolean function is defined as follows:

$$\begin{aligned}
&\phi_i(X,Y,Z) &&= \\
&FF(X,Y,Z) &&= (X \wedge Y) \vee (\overline{X} \wedge Z) && \text{for } 1 \leq i \leq 16 \\
&GG(X,Y,Z) &&= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) && \text{for } 17 \leq i \leq 32 \\
&HH(X,Y,Z) &&= X \oplus Y \oplus Z && \text{for } 33 \leq i \leq 48
\end{aligned}$$

Finally, the hash value of the compression function f is $(a_0 + a)||(b_0 + b)||(c_0 + c)||(d_0 + d)$, where the addition is also under the group $Z_{2^{32}}$.

Later, we will denote a_i to the i^{th} update of the register a, i.e., we denote the j^{th} update of the chaining value as $a_{\lceil \frac{j}{4} \rceil}$ if j mod $4 = 1$, and b_i, c_i, and

d_i are defined similarly.

2.3 The Compression Function of MD5

MD4 and MD5 are both the dedicated hash functions and MD5 is the strengthened version of MD4 with a little modification on MD4 algorithm. MD5 is used to replace MD4 and now is still used widely. So the compression function of MD5 is similar with that of MD4 but for some exceptions, and we will mention them later in this section. In this section, we also denote the input of the compression function of the message digest algorithm MD5 in this iteration as (h, m), where h and m are the initial value and message block of the compression function, respectively. Like MD4, in MD5 spec, h also consists of 4 words, i.e., $h = a_0||b_0||c_0||d_0$, and a_0, b_0, c_0 and d_0 are all 32-bit words. The message block m also consists of 16 words, we denote $m = m_0||m_1||\ldots||m_{15}$, where m_i are all 32-bit words for all $0 \leq i \leq 15$. There are four registers that are used to store the CV, and we denote them as a, b, c, and d. Then given the input (m, h), we set $(a, b, c, d) = (a_0, b_0, c_0, d_0)$, and the output of the compression function is computed as follows:

$$\begin{aligned}
&\Sigma_i = a + \phi_i(b, c, d) + w_i + t_i && a = b + \Sigma_i \lll s_i && \text{if } i \mod 4 = 1\\
&\Sigma_i = d + \phi_i(a, b, c) + w_i + t_i && d = a + \Sigma_i \lll s_i && \text{if } i \mod 4 = 2\\
&\Sigma_i = c + \phi_i(d, a, b) + w_i + t_i && c = d + \Sigma_i \lll s_i && \text{if } i \mod 4 = 3\\
&\Sigma_i = b + \phi_i(c, d, a) + w_i + t_i && b = c + \Sigma_i \lll s_i && \text{if } i \mod 4 = 0
\end{aligned}$$

for $i = 1$ to 64.

In the above equations, s_i's are step-dependent constants, w_i's are chosen from m_i for $0 \leq i \leq 15$ according to its step i, and ϕ_i's are round-dependent Boolean functions. All variables used in the above steps are all 32-bit words, the addition operation is under the group $Z_{2^{32}}$, and the operation "$\lll$" is the cyclic left rotation on 32-bit words. But note here, unlike MD4, t_i's are step-dependent constants, not round-dependent constants. So there are 64 different t_i's for $i = 1$ to 64 in the whole MD5 compression function. In MD4, there are only 3 different t_i's for $i = 1$ to 48 in the whole compression function. There are total 64 steps, not 48 steps, in the compression function of MD5. In each step computation, there is a little difference between MD4 and MD5. The Σ_i's computation of MD4 and of MD5 are the same. But when updating the register of the chaining value, MD5 will add the last one updated chaining value and the addition is also under the group $Z_{2^{32}}$. Here we divide all the 64 steps of the computation of the compression function into 4 rounds, and each round is also of 16-step like MD4. The round-dependent Boolean function is defined as follows:

$$
\begin{array}{lll}
\phi_i(X,Y,Z) & = & \\
FF(X,Y,Z) & = (X \wedge Y) \vee (\overline{X} \wedge Z) & \text{for } 1 \leq i \leq 16 \\
GG(X,Y,Z) & = (X \wedge Z) \vee (Y \wedge \overline{Z}) & \text{for } 17 \leq i \leq 32 \\
HH(X,Y,Z) & = X \oplus Y \oplus Z & \text{for } 33 \leq i \leq 48 \\
II(X,Y,Z) & = Y \oplus (x \vee \overline{Z}) & \text{for } 49 \leq i \leq 64
\end{array}
$$

Here we can observe that the round-dependent Boolean function ϕ_i of MD5 are not all the same as that of MD4.

Finally, the hash value of the compression function f is $(a_0 + a)||(b_0 + b)||(c_0 + c)||(d_0 + d)$, where the addition is also under the group $Z_{2^{32}}$. We stress that without final step (add the initial value a_0, b_0, c_0, and d_0 to the registers a, b, c, and d), the whole computation of the MD4 and MD5 are not one-way given the message block m.

Like that of MD4, we will denote a_i to the i^{th} update of the register a, and b_i, c_i, and d_i are defined similarly.

Chapter 3

Review of Wang et al.'s Attack on MD4 and MD5

3.1 Collision Differentials

The attacks given by Wang et al. on MD4 and MD5 are differential attacks whose differential is under the addition operation of the group $Z_{2^{32}}$. They focus on the flaw of the compression function, not the whole hash function. Both MD4 and MD5 use the Merkel-Damgård construction as their iterative hash structure, the block that represent the original length of the message is always padded as the last block of the input of the hash function. So we can find that Wang et al. want to find a equal-length collision pair such that all the padded strings are the same. According to the Merkel-Damgård construction, the hash value of the original message are the same implies that the hash value of the padded message are also the same. In other words, Wang et al. give messages (M, M') where $|M| = |M'|$ and then the same padded message $\hat{M}$ will be padded after M and M'. Here we denote the padded message of M and M' as $\tilde{M} = M||\hat{M}$ and $\tilde{M}' = M'||\hat{M}$, respectively.

We set $n = \frac{|M|}{r} = \frac{|M'|}{r}$ and $\tilde{n} = \frac{|\tilde{M}|}{r} = \frac{|\tilde{M}'|}{r} = n + 2$ where r is the size of the message block in each iteration of the compression function, and the length of M or M' is always an integer multiple of r bits because Wang et al. focus on the compression functions. According to the description of the Merkel-Damgård construction described in the previous chapter, the hash value in the n^{th} iteration of the computations of the original messages M and M' are $h_n = (h_{n-1}||M_n)$ and $h'_n = (h'_{n-1}||M'_n)$, respectively. The messages given by Wang et al. will let $h_n = h'_n$, and the following message of $\tilde{M}$ and $\tilde{M}'$ are the same, so the final hash values will be the same.

They want to find a collision pair (M, M'), M and M' are both of message block size (512-bit), of MD4 such that:

$$\Delta h_0 = 0 \overset{\Delta M}{\rightarrow} \Delta h_1 = 0$$

where $\Delta M = M' - M = \Delta m_0||\Delta m_1||\ldots||\Delta m_{15} = (m'_0 - m_0)||(m'_1 - m_1)||\ldots||(m'_{15} - m_{15})$, and

$$\Delta m_1 = 2^{31}, \quad \Delta m_2 = 2^{31} - 2^{28}, \quad \Delta m_{12} = -2^{16},$$
$$\text{and } \Delta m_i = 0 \qquad \text{for } 0 \leq i \leq 15 \ \wedge \ i \neq 1\ ,2\ ,12.$$

They want to find a two-block message pair (M, M'), where $M = M_1||M_2$ and $M' = M'_1||M'_2$, to give a collision of MD5. The differential path between the compression function is shown as follows:

$$\Delta h_0 = 0 \overset{\Delta M_1}{\rightarrow} \Delta h_1 \overset{\Delta M_2}{\rightarrow} \Delta h_2 = 0$$

where, $\Delta M_1 = M_1' - M_1 = \Delta m_{1,0}||\Delta m_{1,1}||\ldots||\Delta m_{1,15} = (m_{1,0}' - m_{1,0})||(m_{1,1}' - m_{1,1})||\ldots||(m_{1,15}' - m_{1,15})$

$$\Delta m_{1,4} = 2^{31}, \quad \Delta m_{1,11} = 2^{15}, \quad \Delta m_{1,14} = 2^{31},$$
$$\text{and } \Delta m_{1,i} = 0 \qquad \text{for } 0 \le i \le 15 \ \wedge \ i \ne 4\ ,11\ ,14.$$

and $\Delta M_2 = M_2' - M_2 = \Delta m_{2,0}||\Delta m_{2,1}||\ldots||\Delta m_{2,15} = (m_{2,0}' - m_{2,0})||(m_{2,1}' - m_{2,1})||\ldots||(m_{2,15}' - m_{2,15})$

$$\Delta m_{2,4} = 2^{31}, \quad \Delta m_{2,11} = -2^{15}, \quad \Delta m_{2,14} = 2^{31},$$
$$\text{and } \Delta m_{2,i} = 0 \qquad \text{for } 0 \le i \le 15 \ \wedge \ i \ne 4\ ,11\ ,14.$$

The differential value $\Delta h_1 = (2^{31}, 2^{31}+2^{25}, 2^{31}+2^{25}, 2^{31}+2^{25})$. In all the above equations in this section, all of the message block differentials are consisted from its element word differentials.

Wang et al. give a differential path of the chaining values during the computation of the compression function in [WLF+05, Table 5] and [WY05, Table 3 & 5] of MD4 and MD5, respectively. With the differential path of the chaining values, two message M and M' with the differential described above will collide if their differential of the chaining values are the same as Wang et al.'s. But note here, any collision message pair is not always satisfy the differential of the message and of the internal chaining value given by Wang et al. Like the differential cryptanalysis against DES, we always use the differential path (In DES, someone may call that the input differential and the output differential for each round sub-key encryption) that we think that is the most useful for attacking. Wang et al. give a differential path that can be used to find a collision pair fast by them.

3.2 Sufficient Conditions

Given the differential path, it is not practical to find the collision pairs. So Wang et al. also give the sufficient conditions [WLF$^+$05, Table 6] and [WY05, Table 4 & 6] of MD4 and MD5 respectively according to their differential path. It is that if a randomly chosen message M satisfy all the sufficient conditions, then the differential of chaining values of M and $M' = M + \Delta M$ during the computation of the compression function will be equal to that given by Wang et al. In other words, the sufficient conditions of the message M are to guarantee the differential path of the message pair M and M' to be equivalent with Wang et al.'s. With the sufficient conditions, we can focus on the message M itself, not the collision pair (M, M') instead. So we now want to find a message M that satisfy all the sufficient conditions given by Wang et al. If the message M satisfy all the sufficient conditions, then $M' = M + \Delta M$ and M will collide.

Note here that the sufficient conditions can be computed by anyone who knows the differential path. The sufficient conditions given by Wang et al. are also computed according to the differential path given by them. Some papers post on ePrint to figure out that the sufficient conditions given by Wang et al. are not sufficient and can be relaxed [YS05, LL05, SNY$^+$06].

They find that some message M that satisfy all the sufficient conditions given by Wang et al., but $M' = M + \Delta M$ and M don't collide. That is because the differential path of the chaining value during the computation of the compression function on M and M' is not the same as Wang et al.'s. So the probability that M and M' will collide is not always 1 under this situation.

3.3 Message Modification

Given the sufficient conditions by Wang et al., the probability of a randomly chosen massage M that will satisfy all the sufficient conditions is very small. So we want the message M to satisfy more sufficient conditions to decrease the probability that M will satisfy all the sufficient conditions. Thus given a randomly chosen message M, we need to modify it to satisfy some sufficient conditions. We introduce the message modification techniques here to modify the randomly chosen message M to decrease the probability that M satisfies all the sufficient conditions.

Single-Message Modification. In MD4 and MD5, each word of the message block is used exactly once during the computation of the compression function in the first round. So we can modify the message to let it satisfy all the sufficient conditions in the first round easily. But there are also many sufficient conditions besides the first round. If we don't care these sufficient conditions, the collision searching algorithm is not efficient enough and may cost much more time to find a collision pair. So we also need to modify the message M to let it satisfy more sufficient conditions. If we want the message to satisfy the sufficient conditions not only in the first round, but outside the first round, it need some tricky techniques, Multi-Message Modification, to achieve this.

Multi-Message Modification. Because some message words are used more than once during the computation after the first round, we can't modify the message word to satisfy the sufficient condition in more than two chaining values in the same time trivially. So we need to think deeper about the compression function and the sufficient conditions and give some definitive modification methods to let the message to satisfy as many sufficient conditions as possible. More sufficient conditions the message satisfies, less the time-complexity of the collision searching algorithms of the hash function MD4 or MD5 is.

Many recents researches focus on how to let the chaining values of the message M during the computation of the compression function to satisfy more sufficient conditions in the same time. They give many multi-message modifications techniques to achieve the goal. We will introduce these techniques deeper in Chapter 4.

3.4 Collision Searching Algorithm

With the introduction in the previous section of this chapter, we will summarize the collision searching algorithm as below:

MD4 Collision Searching Algorithm:

1. We randomly choose a message block M.
2. We modify the message M by the message modification techniques.
3. For all the sufficient conditions that can't be satisfied by the message modification techniques, we check if the chaining values during the computation of the compression function on M and these sufficient conditions are equivalent. If they are not equivalent, we randomly choose m_{14} and m_{15} of M and go back to step 2.
4. Finally, if the message M satisfy all the sufficient conditions, $M' = M + \Delta M$ and M collide.

MD5 Collision Searching Algorithm:

1. We randomly choose a message block M_1.
2. We modify the message M_1 by the message modification techniques.
3. For all the sufficient conditions that can't be satisfied by the message modification techniques, we check if the chaining value during the com-

putation of the compression function on M_1 and these sufficient conditions are equivalent. If they are not equivalent, we randomly choose $m_{1,14}$ and $m_{1,15}$ of M_1 and go back to step 2.

4. We randomly choose a message block M_2.

5. We modify the message M_2 by the message modification techniques.

6. For all the sufficient conditions that can't be satisfied by the message modification techniques, check if the chaining value during the computation of the compression function on M_2 and these sufficient conditions are equivalent. If they are not equivalent, we randomly choose $m_{2,14}$ and $m_{2,15}$ of M_2 and go back to step 5.

7. Finally, if the message $M = M_1||M_2$ satisfy all the sufficient conditions, $M' = M_1'||M_2' = (M_1 + \Delta M_1)||(M_2 + \Delta M_2)$ and M collide.

Chapter 4

Some Improvements on Message Modification

After Wang et al. published the papers of the collision cryptanalysis against MD4 and MD5 [WLF$^+$05, WY05], many cryptographic researchers study how to improve the performance of the collision searching algorithms. Many researches focus on how to improve the multi-message modification techniques. They use the correct sufficient conditions derived from the same differential path given by Wang et al. In this chapter, we introduce their results so far and give our own improvements. We will use the successful probability instead of the computation numbers of the hash compression function to measure and compare the time complexity of the below multi-message modification results. The successful probability is computed according to the number of unsatisfied sufficient conditions by using the multi-message modification techniques. In other words, the more sufficient conditions are fulfilled by the multi-message modification, the more efficient this multi-message modification algorithm is.

4.1 Previous Results

Many results on finding the collisions of MD4 and MD5, given by Wang et al., are ambiguous. Only a rough discovery is given. The reason for such discovery is not introduced. In the message modification techniques, they also just give a very rough method. If someone follows the multi-message modification methods given by them to find the collision of MD4, he can't find any collision at all. This is found by Naito et al. in their research of the collision searching algorithm on MD4 [NSKO05]. They show that some extra conditions given by Wang et al. may be broken by their type 1 multi-message modification. The *extra conditions* introduced in this chapter are not sufficient conditions, instead they are used to improve the efficiency of the multi-message modification methods outside the first round. Any randomly generated message block M is not necessary to satisfy the extra conditions. The sufficient conditions is to guarantee the differential path of M and M' to hold. The extra conditions are used to guarantee the modification on message words, that are after the first round, doesn't affect too many chaining values that are in the first round. If all except a very small size sufficient conditions are fulfilled by the message block M, then the probability that M and $M' = M + \Delta M$ collide is negligible. So the priority of the sufficient conditions are higher than that of extra conditions, i.e., any extra conditions must not contradict with the sufficient conditions. Below we show the two types of multi-message modifications.

Type 1 Muiti-Message Modification. Here we use the multi-message modification of the sufficient condition "$a_{5,19} = c_{4,19}$" given by Naito et al. [NSKO05, Table 9] as the example to introduce the type 1 multi-message modification. According to the MD4 algorithms, $a_5 = (a_4 + GG(b_4, c_4, d_4) + m_0 + t_{17}) \lll 3$ and m_0 is used to compute the chaining value a_5. We can find that the complement of $m_{0,16}$ will always cause the complement of $a_{5,19}$ to occur because the left shift number is 3. The complement of a_5 implies that "$a_{5,19} \neq c_{4,14}$" becomes "$a_{5,19} = c_{4,14}$" as long as $c_{4,14}$ is unchange. In other words, if some bits of the chaining values don't satisfy their corresponding sufficient conditions, we only need to cause the complement of these bits to occur using anything we can do. But the complement of $m_{0,16}$ will cause the bit $a_{1,19}$ ($a_1 = (a_0+FF(b_0, c_0, d_0)+m_0+t_1) \lll 3$) in the first round to change and $a_{1,i}$ may change due to the carry or the borrow on $a_{1,19}$ for $20 \leq i \leq 32$. To prevent carry or borrow on $a_{1,19}$, we need to consider the direction of the change on $a_{1,19}$. If $a_{1,19} = 1$, then we modify $a_{1,19}$ to 0, otherwise we modify $a_{1,19}$ to 1. Since m_0 is updated by $m_0 = (a_1 \ggg 3) - a_0 - FF(b_0, c_0, d_0) - t_1$, the complement of $m_{0,16}$ occurs. Then the subsequent chaining values d_1 c_1 b_1 and a_2 are computed as follows:

$$d_1 = (d_0 + FF(a_1, b_0, c_0) + m_1 + t_2) \lll 7$$

$$c_1 = (c_0 + FF(d_1, a_1, b_0) + m_2 + t_3) \lll 11$$

$$b_1 = (b_0 + FF(c_1, d_1, a_1) + m_3 + t_4) \lll 19$$

$$a_2 = (a_1 + FF(b_1, c_1, d_1) + m_4 + t_5) \lll 3$$

They all use a_1 to compute themselves and may change because a_1 changes. We need to do something to prevent them from changing because any change of them will result in that the change of all subsequent chaining values. So we need to update the message words m_1, m_2, m_3, and m_4 to cancel the change from a_1 to its subsequent chaining values as follows:

$$m_1 = (d_1 \ggg 7) - d_0 + FF(a_1, b_0, c_0) - t_2$$

$$m_2 = (c_1 \ggg 11) - c_0 + FF(d_1, a_1, b_0) - t_3$$

$$m_3 = (b_1 \ggg 19) - b_0 + FF(c_1, d_1, a_1) - t_4$$

$$m_4 = (a_2 \ggg 3) - a_1 + FF(b_1, c_1, d_1) - t_5$$

We summary the type 1 multi-message modification techniques here. Consider the sufficient condition of the j^{th} bit of the chaining value x in the i^{th} step computation of the compression function. If is is not what we want, we modify it to satisfy the sufficient condition as follows:

1. We find which message word in the message block m is used to compute the chaining values in the i^{th} step. We assume it is m_l for $0 \leq l \leq 15$.

2. We find which chaining value in the first round that use m_l to compute itself, suppose it is y and is in the i'^{th} step.

3. We complement the bit $y_{j-s_i+s_{i'} \mod 32}$ and update the 5 message words that is used to compute the chaining values in the r^{th} to the $(r+4)^{\text{th}}$ steps.

Type 2 Multi-Message Modification. In type 1 multi-message modification, there exist some cases that the techniques may not work. The first case is that $y_{j-s_i+s_{i'} \bmod 32}$ itself is a sufficient condition. The second case is that y is in the r^{th} step and $r \geq 13$. The reason that $r \geq 13$ is not allowed is that some of the chaining values in the r^{th} to the $(r+4)^{\text{th}}$ steps are in the second round. The message used to compute the chaining values in the second round are also used to compute a chaining value in the first round. In this case, the multi-message modification methods are not as easy as those described in the type 1 multi-message modification. There are also some other cases that cause the type 1 multi-message modification to fail. One among them is that the type 1 multi-message modification breaks the extra condition used for other sufficient conditions. The multi-message modification techniques here under this situation are much more complex. We need to introduce the extra condition to help us to guarantee that the modification on the corresponding chaining value doesn't affect its subsequent chaining values too much.

Here we introduce the multi-message modification of the sufficient condition "$c_{5,26} = d_{5,26}$" given by Naito et al. [NSKO05, Table 12] to explain the type 2 multi-message modification. As described in the type 1 multi-message modification, we find that we must consider the bit $a_{3,26-9+3 \bmod 32} = a_{3,20}$. However, we find that $a_{3,20}$ itself is a sufficient condition, so the type 1 multi-message modification doesn't work. So we follow the type 2 multi-message modification given by Naito et al. as follows:

1. We add the extra condition "$d_{2,17} = 0$".

2. We change the bit $d_{2,17}$ from 0 to 1.

3. We update the message word m_5 by $m_5 = (d_2 \ggg 7) - d_1 - FF(a_2, b_1, c_1) - t_6 = ((d_2^{\text{old}} + 2^{16}) \ggg 7) - d_1 - FF(a_2, b_1, c_1) - t_6 = m_5^{\text{old}} + 2^9$.

4. We add the extra condition "$a_{2,17} = b_{1,17}$".

5. We compute the chaining value c_2 by $c_2 = (c_1 + FF(d_2, a_2, b_1) + m_6 + t_7) \lll 11$ but $d_{2,17}$ changes from 0 to 1. However, the extra condition "$a_{2,17} = b_{1,17}$" guarantees that $FF(d_{2,17}, a_{2,17}, b_{1,17}) = FF(0 \to 1, a_{2,17}, b_{1,17}) = b_{1,17} \to a_{2,17}$" unchanges. That implies that c_2 unchange.

6. We add the extra condition "$c_{2,17} = 0$".

7. We compute the chaining value b_2 by $b_2 = (b_1 + FF(c_2, d_2, a_2) + m_7 + t_8) \lll 19$ but $d_{2,17}$ changes from 0 to 1. However, the extra condition "$c_{2,17} = 0$" guarantees that $FF(c_{2,17}, d_{2,17}, a_{2,17}) = FF(0, 0 \to 1, a_{2,17}) = a_{2,17}$ unchanges. That implies that b_2 unchange.

8. We add the extra condition "$b_{2,17} = 0$".

9. We update the message word m_8 by $m_8 = (a_3 \ggg 3) - a_2 - FF(b_2, c_2, d_2) - t_9 = (a_3 \ggg 3) - a_2 - (FF(b_2, c_2, d_2^{\text{old}}) + 2^{16}) - t_9 = m_8^{\text{old}} - 2^{16}$ for the extra condition "$b_{2,17} = 0$".

10. $m_8 = m_8^{\text{old}} - 2^{16}$ will cause the complement of the bit $m_{8,17}$ to occur, then the complement of the bit $c_{5,26}$ ($c_5 = (c_4 + GG(d_5, a_5, b_4) + m_8 + t_{19}) \lll 9$) also occurs and the sufficient condition "$c_{5,26} = d_{5,26}$" will be fulfilled.

11. Finally, We update the message word m_9 by $m_9 = (d_3 \ggg 7) - d_2 - FF(c_2, b_2, a_2) - t_{10} = (d_3 \ggg 7) - (d_2^{\text{old}} + 2^{16}) - FF(c_2, b_2, a_2) - t_{10} = m_9^{\text{old}} - 2^{16}$.

We stress that the extra conditions must be in the first round of the compression function and must be fulfilled by the simple single-message modification. We will introduce single-message modification below.

Klima [Kli05b, Kli05a] first introduced an efficient and definite multi-message modification algorithms on MD5. His single-message modification methods is very useful to find the collision on both MD4 and MD5. By the introduction of the compression function of MD4 and MD5 in Section 2.2 and 2.3, respectively. We set $(a, b, c, d) = (a_0, b_0, c_0, d_0)$ first and do the single-message modification of MD4 as follows:

We randomly generate x but satisfy all the sufficient conditions of $a_{\lceil \frac{i}{4} \rceil}$,

$\Sigma_i = x \ggg s_i \quad m_{i-1} = \Sigma_i - a - FF(b, c, d) - t_i \quad a = x \qquad \text{if } i \mod 4 = 1$

We randomly generate x but satisfy all the sufficient conditions of $d_{\lceil \frac{i}{4} \rceil}$,

$\Sigma_i = x \ggg s_i \quad m_{i-1} = \Sigma_i - d - FF(a, b, c) - t_i \quad d = x \qquad \text{if } i \mod 4 = 2$

We randomly generate x but satisfy all the sufficient conditions of $c_{\lceil \frac{i}{4} \rceil}$,

$\Sigma_i = x \ggg s_i \quad m_{i-1} = \Sigma_i - c - FF(d,a,b) - t_i \quad c = x \qquad \text{if } i \mod 4 = 3$

We randomly generate x but satisfy all the sufficient conditions of $b_{\lceil \frac{i}{4} \rceil}$,

$\Sigma_i = x \ggg s_i \quad m_{i-1} = \Sigma_i - b - FF(c,d,a) - t_i \quad b = x \qquad \text{if } i \mod 4 = 0$

where x is a 32-bit word for $i = 1$ to 16.

The single-message modification of MD5 is as follows:

We randomly generate x but satisfy all the sufficient conditions of $a_{\lceil \frac{i}{4} \rceil}$,

$\Sigma_i = (x - b) \ggg s_i \quad m_{i-1} = \Sigma_i - a - FF(b,c,d) - t_i \quad a = x \qquad \text{if } i \mod 4 = 1$

We randomly generate x but satisfy all the sufficient conditions of $d_{\lceil \frac{i}{4} \rceil}$,

$\Sigma_i = (x - a) \ggg s_i \quad m_{i-1} = \Sigma_i - d - FF(a,b,c) - t_i \quad d = x \qquad \text{if } i \mod 4 = 2$

We randomly generate x but satisfy all the sufficient conditions of $c_{\lceil \frac{i}{4} \rceil}$,

$\Sigma_i = (x - d) \ggg s_i \quad m_{i-1} = \Sigma_i - c - FF(d,a,b) - t_i \quad c = x \qquad \text{if } i \mod 4 = 3$

We randomly generate x but satisfy all the sufficient conditions of $b_{\lceil \frac{i}{4} \rceil}$,

$\Sigma_i = (x - c) \ggg s_i \quad m_{i-1} = \Sigma_i - b - FF(c,d,a) - t_i \quad b = x \qquad \text{if } i \mod 4 = 0$

where x is a 32-bit word for $i = 1$ to 16.

The main philosophy of the single-message modification methods is that the message word m_i is not used to compute the chaining values but is able to be computed from the chaining values for $0 \leq i \leq 15$. After the computations of the single-message modification given by Klima, the 128-bit message block m is recovered wholly. Klima also mentioned that there is no sufficient conditions in the chaining values a_1 and d_1 in the first iteration of MD5. In the first round of the compression function of MD5, the chaining value a_1 and d_1 are computed using the message words m_0 and m_1, respectively. In the second round, the message words m_0 and m_1 are used to compute the chaining value b_5 and a_5, respectively. So in first 20 steps of the compression function of MD5, all message words of m are used exactly once except m_6 and m_{11}, that are used to compute the chaining values d_5 and c_5, respectively. Then in first 20 steps of the first iteration, all sufficient conditions of the chaining values except d_5 and c_5 can be fulfilled by the single-message modification introduced above easily. But the method given by Klima on how to modify m_0 and m_1 to satisfy the sufficient conditions on d_5 and c_5 is a brute-force method. He doesn't guarantee that the modification always succeeds, and it may need to restart the collision searching algorithm by re-generate some chaining values (that implies re-generate their corresponding message words as described in Section 3.4).

Sasaki et al. improve the message modification of the first iteration of MD5 [SNKO05]. In their paper, they claimed that they can satisfy all the sufficient conditions in first 23 steps of the first iteration with probability $\frac{1}{2}$. In other words, all sufficient conditions in a_i b_j c_i and d_i can be fulfilled by their message modification methods for $0 \leq i \leq 5 \quad \wedge \quad 0 \leq j \leq 4$. Similar to their results on MD4 [NSKO05], they also use the extra conditions to guarantee that later in the second round the message modification doesn't affect too many chaining values that are in the first round.

Later Liang and Lai improve the message modification techniques [LL05]. They introduced a new techniques, *small range searching techniques*, to do the multi-message modification. In [LL05, Chapter 5] they said that for the computation of the chaining value $N = ((L+\phi(X,Y,Z)+M+T) \lll s)+U$, they can modify the j^{th} bit of N by changing the j^{th} bit of U or the $(j-s)^{\mathrm{th}}$ bit of L, X, Y, Z, or M if these bits has no sufficient condition. But this was already used widely in type 1 or type 2 multi-message modification introduced before. They claimed that they can modify the j^{th} bit of N by changing the bits lower than the j^{th} bit of U and/or the bits lower than the $(j-s)^{\mathrm{th}}$ bit of L, X, Y, Z, and/or M by "carry" or "borrow" on these bits if these bits has no sufficient condition. But we think what they give to do the multi-message modification is less efficient than that given by Sasaki et al. Because the small range searching techniques need to search all the corresponding lower bits where there is no sufficient condition (i.e., the bit value is random) and modify them. Suppose there are total n the corresponding lower bits in U, L, X, Y, Z, and M. Then they may need total 2^n testing in searching these n bits and may fail in the worst case because no carry or borrow occurs. The small range searching technique doesn't guarantee that it must be able to modify the j^{th} bit of N successfully. But according to the introduction of the small range searching techniques by them, it is more flexible to do the multi-message modification than that Sasaki et al.'s. There is no extra condition to narrow the size of the message block M, and there are many possibilities how to do the multi-message modification by searching these n bits in U, L, X, Y, Z, and M. But the prerequisite is that the small range searching technique need always to work successfully. On the other hand, their small range searching techniques on Σ_{19} in both iterations will always work. Because we can select the corresponding bits in M randomly and modify it by the type 1 multi-message modification. If Σ_{20} in the first iteration is not what we want, we can regenerate b_5 randomly. In their papers of finding the MD5 collisions, they also give a type 2 multi-message modification on the sufficient conditions "$d_{5,18} = 1$", "$d_{5,30} = a_{5,30}$", and "$d_{5,32} = a_{5,32}$" in the first iteration and their corresponding extra conditions.

4.2 Our Improvements

In this section, we will gave some improvements on multi-message modification techniques. We will introduce this in two parts, MD4 and MD5.

4.2.1 The Improvements on MD4

Although Naito et al.'s result [NSKO05] is efficient enough, but we find that three extra conditions given by them are not in the first round. So these three extra conditions can not be modified trivially by the simple single-message modification. We need to modify them by the multi-message modification techniques. One of the three extra conditions is the condition "$a_{5,20} = b_{4,20}$" for the sufficient condition "$c_{5,29} = d_{5,29}$" [NSKO05, Table 13]. Follow their type 2 multi-message-modification method for the sufficient condition "$c_{5,29} = d_{5,29}$", we can find that the update of the chaining value d_5 [NSKO05, Table 13, Step 18] is computed as $d_5 = (d_4 + GG(a_5, b_4, c_4) + m_4 + t_{18}) \lll 5$. The complement of the bit $c_{4,20}$ causes $GG(a_{5,20}, b_{4,20}, c_{4,20}) = GG(a_{5,20}, b_{4,20}, 0 \rightarrow 1) = a_{5,20} \wedge b_{4,20} \rightarrow (a_{5,20} \wedge b_{4,20}) \vee a_{5,20} \vee b_{4,20}$ and $a_{5,20} \wedge b_{4,20} = (a_{5,20} \wedge b_{4,20}) \vee a_{5,20} \vee b_{4,20}$ if and only if $a_{5,20} = c_{5,20}$. If the extra condition "$a_{5,20} = d_{4,20}$" is not fulfilled during the message modification for "$c_{5,29} = d_{5,29}$", d_5 changes. So when we do the multi-message modification for the chaining value a_5, besides their sufficient condition, we also need to consider the extra condition "$a_{5,20} = d_{4,20}$". If $a_{5,20} \neq b_{4,20}$, we need to use the following procedure to do the multi-message modification to cause a_5 satisfy the extra condition.

1. We complement the bit $a_{1,20}$.

2. We use the following equations to update the message words m_0, m_1, m_2, and m_3.

$$m_0 = (a_1 \ggg 3) - a_0 - FF(b_0, c_0, d_0) - t_1 \tag{4.1}$$

$$m_1 = (d_1 \ggg 7) - d_0 - FF(a_1, b_0, c_0) - t_2$$

$$m_2 = (c_1 \ggg 11) - c_0 - FF(d_1, a_1, b_0) - t_3$$

$$m_3 = (b_1 \ggg 19) - b_0 - FF(c_1, d_1, a_1) - t_4$$

$$m_4 = (a_2 \ggg 3) - a_1 - FF(b_1, c_1, d_1) - t_5$$

3. In equation 4.1, $m_0 = ((a_1 \pm 2^{19}) \ggg 3) - a_0 - FF(b_0, c_0, d_0) - t_1 = m_0^{\text{old}} \pm 2^{16}$. Then $a_5 = (a_4 + GG(b_4, c_4, d_4) + m_0 + t_{17}) \lll 3 = (a_4 +$

$GG(b_4, c_4, d_4) + (m_0^{\text{old}} \pm 2^{16}) + t_{17}) \lll 3 = a_5^{\text{old}} \pm 2^{19}$. This implies that the complement of the bit $a_{5,20}$ to occur.

The other two extra conditions are "$a_{5,22} = b_{4,22}$" and "$c_{5,31} = 1$" for the sufficient condition "$c_{5,32} = d_{5,32}$" [NSKO05, Table 15]. By their type 2 multi-message-modification method for the sufficient condition "$c_{5,32} = d_{5,32}$", we can find that the Boolean function ϕ_{18} of the chaining value d_5 [NSKO05, Table 15, Step 18] is computed as $GG(a_5, b_4, c_4)$. The complement of the bit $c_{4,22}$ causes $GG(a_{5,22}, b_{4,22}, 0 \to 1) = a_{5,22} \wedge b_{4,22} \to (a_{5,22} \wedge b_{4,22}) \vee a_{5,22} \vee b_{4,22}$ and $a_{5,22} \wedge b_{4,22} = (a_{5,22} \wedge b_{4,22}) \vee a_{5,22} \vee b_{4,22}$ if and only if $a_{5,22} = d_{5,22}$. If the extra condition "$a_{5,22} = d_{4,22}$" is not fulfilled during the message modification for "$c_{5,32} = d_{5,32}$", d_5 changes. So when we do the multi-message modification for the chaining value a_5, besides their sufficient condition, we need to consider the extra condition "$a_{5,22} = d_{4,22}$". If $a_{5,22} \neq b_{4,22}$, we need to use the following procedure to do the multi-message modification to cause a_5 to satisfy the extra condition.

1. We complement the bit $a_{1,22}$.

2. We use the following equations to update the message words m_0, m_1, m_2, m_3, and m_4.

$$m_0 = (a_1 \ggg 3) - a_0 - FF(b_0, c_0, d_0) - t_1 \tag{4.2}$$

$$m_1 = (d_1 \ggg 7) - d_0 - FF(a_1, b_0, c_0) - t_2$$

$$m_3 = (c_1 \ggg 11) - c_0 - FF(d_1, a_1, b_0) - t_3$$

$$m_4 = (b_1 \ggg 19) - b_0 - FF(c_1, d_1, a_1) - t_4$$

$$m_5 = (a_2 \ggg 3) - a_1 - FF(b_1, c_1, d_1) - t_5$$

3. In equation 4.2, $m_0 = ((a_1 \pm 2^{21}) \ggg 3) - a_0 - FF(b_0, c_0, d_0) - t_1 = m_0^{\text{old}} \pm 2^{18}$. Then $a_5 = (a_4 + GG(b_4, c_4, d_4) + m_0 + t_{17}) \lll 3 = (a_4 + GG(b_4, c_4, d_4) + (m_0^{\text{old}} \pm 2^{18}) + t_{17}) \lll 3 = a_5^{\text{old}} \pm 2^{21}$. This implies that the complement of the bit $a_{5,22}$ to occur.

Naito et al.'s type 2 multi-message modification for the sufficient condition "$c_{5,32} = d_{5,32}$" doesn't cause the complement of $c_{5,32}$ to occur directly. Instead, they add the extra condition "$c_{5,31} = 1$" and c_5 is updated by $c_5 = (c_4 + GG(d_5, a_5, b_4) + m_8 + t_{19}) \lll 9$. $c_{4,22} : 0 \rightarrow 1$ causes $c_5 = ((c_4^{\text{old}} + 2^{21}) + GG(d_5, a_5, b_4) + m_8 + t_{19}) \lll 9 = c_5^{\text{old}} + 2^{30}$. The extra condition "$c_{5,31} = 1$" causes the carry on $c_{5,31}$ to occur and implies that the complement of $c_{5,32}$ also occur. So if we need to do the multi-message modification for "$c_{5,32} = d_{5,32}$" and "$c_{5,31} = 0 \neq 1$", we need to use the following procedure to do the multi-message modification to cause c_5 to satisfy the extra condition.

1. We complement the bit $a_{3,25}$.

2. We use the following equations to update the message word m_8, m_9,

m_{10}, m_{11}, and m_{12}.

$$
\begin{aligned}
m_8 &= (a_3 \ggg 3) - a_2 - FF(b_2, c_2, d_2) - t_9 \qquad (4.3)\\
m_9 &= (d_3 \ggg 7) - d_2 - FF(a_3, b_2, c_2) - t_{10}\\
m_{10} &= (c_3 \ggg 11) - c_2 - FF(d_3, a_3, b_2) - t_{11}\\
m_{11} &= (b_3 \ggg 19) - b_2 - FF(c_3, d_3, a_3) - t_{12}\\
m_{12} &= (a_4 \ggg 3) - a_3 - FF(b_3, c_3, d_3) - t_{13}
\end{aligned}
$$

3. In equation 4.3, $m_8 = ((a_3^{\text{old}} \pm 2^{24}) \ggg 3) - a_2 - FF(b_2, c_2, d_2) - t_9 = m_8^{\text{old}} \pm 2^{21}$. Then $c_5 = (c_4 + GG(d_5, a_5, b_4) + m_8 + t_{19}) \lll 9 = (c_4 + GG(d_5, a_5, b_4) + (m_8 \pm 2^{21}) + t_{19}) \lll 9 = c_5^{\text{old}} \pm 2^{30}$. This implies that the complement of the bit $c_{5,31}$ to occur.

Finally we compare our improvements with Naito et al.'s result. Here we assume that all bits of all chaining values are uniform under $\{0, 1\}$ and are totally independent. Let E_1 be the event that the sufficient condition "$c_{5,29} = d_{5,29}$" is not fulfilled and need to be modified by Naito et al.'s type 2 multi-message modification during the collision searching algorithm, E_2 be the event that the sufficient condition "$c_{5,32} = d_{5,32}$" is not fulfilled and need to be modified by their type 2 multi-message modification, and S be the successful event (i.e., the extra conditions described above are what we want). Then successful probabilities conditioned on E are listed in table 4.1. For the four possibilities of E, each is of probability $\frac{1}{4}$. So the successful probability of Naito et al.'s multi-message modification on MD4 is:

The Event E	The Probability $\Pr[S\|E]$
$\overline{E_1} \cap \overline{E_2}$	1
$\overline{E_1} \cap E_2$	$\frac{1}{4}$
$E_1 \cap \overline{E_2}$	$\frac{1}{2}$
$E_1 \cap E_2$	$\frac{1}{8}$

Table 4.1: The successful probability that some sufficient conditions are not fulfilled

$$
\begin{aligned}
&\Pr[S] \\
=&\Pr[\overline{E_1} \cap \overline{E_2}]\Pr[S|\overline{E_1} \cap \overline{E_2}] + \Pr[\overline{E_1} \cap E_2]\Pr[S|\overline{E_1} \cap E_2] \\
+&\Pr[E_1 \cap \overline{E_2}]\Pr[S|E_1 \cap \overline{E_2}] + \Pr[E_1 \cap E_2]\Pr[S|E_1 \cap E_2] \\
=&\frac{1}{4} \times 1 + \frac{1}{4} \times \frac{1}{4} + \frac{1}{4} \times \frac{1}{2} + \frac{1}{4} \times \frac{1}{8} \\
=&\frac{15}{32}
\end{aligned}
$$

Our modified version of their multi-message modification is of probability 1. Even if they restart the random generation of some message words for the failure as described in Section 3.4, our modified version is also about $\frac{32}{15}$ times faster than their multi-message modification. They also need 2 extra computations of the MD4 compression function to verify if the message pair (M, M') collide.

4.2.2 The Improvements on MD5

After surveying the previous papers about the multi-message modification techniques on MD5 [Kli05b, Kli05a, SNKO05, LL05], we find that something can be improved. As mentioned in Section 3.4, there are two iterations in the collision searching algorithm of MD5. We also introduce our improvements in two parts, the first iteration (or the first block) and the second iteration (or the second block).

- **The First Iteration:**
  - For the sufficient condition "$c_{5,32} = d_{5,32}$" in the first iteration, we do the following procedure.
    1. We complement the bit $c_{4,18}$.
    2. Because c_4 changes, we use the following equations to update the message words m_{14}, m_{15}, and m_0.

$$m_{14} = ((c_4 - d_4) \ggg 17) - c_3 - FF(d_4, a_4, b_3) - t_{15}$$

$$m_{15} = ((b_4 - c_4) \ggg 22) - b_3 - FF(c_4, d_4, a_4) - t_{16}$$

$$m_0 = ((a_5 - b_4) \ggg 5) - a_4 - GG(b_4, c_4, d_4) - t_{17}$$

    3. We add the extra condition "$b_{4,18} = a_{5,18} = 0$". Note that "$a_{5,18} = 0$" itself is a sufficient condition.
    4. We compute the chaining value d_5 by $d_5 = ((d_4 + GG(a_5, b_4, c_4) + m_6 + t_{18}) \lll 9) + a_5$ but $c_{4,18}$ changes. However, the extra condition "$b_{4,18} = a_{5,18}$" guarantees that $GG(a_{5,18}, b_{4,18}, c_{4,18}) = GG(0, 0, \overline{c^{\text{old}}_{4,18}}) = 0 \to 0$ unchanges. This also implies that d_5 unchanges.
    5. We update the chaining value c_5 by $c_5 = ((c_4 + GG(d_5, a_5, b_4) + m_{11} + t_{19}) \lll 14) + d_5 = (((c_4^{\text{old}} \pm 2^{17}) + GG(d_5, a_5, b_4) + m_{11} +$

$t_{19}) \lll 14) + d_5 = c_5^{\text{old}} \pm 2^{31}$. This implies that the complement of the bit $c_{5,32}$ occurs.

6. Because m_0 changes, we update the chaining value a_1 by $a_1 = ((a_0 + FF(b_0, c_0, d_0) + m_0 + t_1) \lll 7) + b_0$.

7. Because a_1 also changes, we use the following equations to update the message words m_1, m_2, m_3, and m_4.

$$m_1 = ((d_1 - a_1) \ggg 12) - d_0 - FF(a_1, b_0, c_0) - t_2$$

$$m_2 = ((c_1 - d_1) \ggg 17) - c_0 - FF(d_1, a_1, b_0) - t_3$$

$$m_3 = ((b_1 - c_1) \ggg 22) - b_0 - FF(c_1, d_1, a_1) - t_4$$

$$m_4 = ((a_2 - b_1) \ggg 7) - a_1 - FF(b_1, c_1, d_1) - t_5$$

Sasaki et al. also give a type 2 multi-message modification for the sufficient condition "$c_{5,32} = d_{5,32}$" in their paper [SNKO05, Table 6]. But there are six extra conditions in their multi-message modification method. As mentioned in Section 4.1, the extra condition is not necessary for the message modification techniques, they are used to avoid too many chaining values from changing instead. So if the message modification method need too many extra conditions, the size of the set of the collision message pair will become too small. Even if sometimes we need to add the extra condition to help us to do the multi-message modification, but we must decrease the total number of the extra conditions as possible as we can to enlarge the size of the set of the collision message pair. In our multi-message modification for the sufficient condition "$c_{5,32} = d_{5,32}$", we need only one extra condition "$c_{5,32} = d_{5,32}$".

- **The Second Iteration:**

  – For the sufficient condition "$d_{5,18} = 1$" in the second iteration, we do the following procedure.

    1. We complement the bit $d_{4,9}$.
    2. Because d_4 changes, we use the following equations to update the message words m_{13}, m_{14}, and m_{15}.

$$m_{13} = ((d_4 - a_4) \ggg 12) - d_3 - FF(a_4, b_3, c_3) - t_{14}$$

$$m_{14} = ((c_4 - d_4) \ggg 17) - c_3 - FF(d_4, a_4, b_3) - t_{15}$$

$$m_{15} = ((b_4 - c_4) \ggg 22) - b_3 - FF(c_4, d_4, a_4) - t_{16}$$

    3. We add the extra condition "$b_{4,9} = c_{4,9}$".
    4. Because the bit $d_{4,9}$ changes, we update $\Delta_{17,9}$ by $\Delta_{17,9} = GG(b_{4,9}, c_{4,9}, d_{4,9}) = GG(b_{4,9}, c_{4,9}, \overline{d^{\text{old}}_{4,9}}) = (b_{4,9} \to c_{4,9}) \vee (c_{4,9} \to b_{4,9})$. The extra condition "$b_{4,9} = c_{4,9}$" guarantees that $\Delta_{17,9}$ unchange. This also implies that $a_5 = ((a_4 + GG(b_4, c_4, d_4) + m_1 + t_{17}) \lll 5) + b_4$ unchanges.
    5. Finally, we update the chaining value d_5 by $d_5 = ((d_4 + GG(a_5, b_4, c_4) + m_6 + t_{18}) \lll 9) + a_5 = (((d_4^{\text{old}} \pm 2^8) + GG(a_5, b_4, c_4) + m_6 + t_{18}) \lll 9) + a_5 = d_5^{\text{old}} \pm 2^{17}$. This will cause the complement of the bit $d_{5,18}$ to occur.

– For the sufficient condition "$d_{5,32} = a_{5,32}$" in the second iteration, we do the following procedure.

1. We complement the bit $d_{2,23}$.
2. We add the extra condition "$a_{2,23} = \overline{b_{1,23}}$".
3. Because d_2 changes, we use the following equations to update the message words m_5, m_6, m_7, m_8, and m_9.

$$\begin{aligned} m_5 &= ((d_2 - a_2) \ggg 12) - d_1 - FF(a_2, b_1, c_1) - t_6 \\ m_6 &= ((c_2 - d_2) \ggg 17) - c_1 - FF(d_2, a_2, b_1) - t_7 \\ m_7 &= ((b_2 - c_2) \ggg 22) - b_1 - FF(c_2, d_2, a_2) - t_8 \\ m_8 &= ((a_3 - b_2) \ggg 7) - a_2 - FF(b_2, c_2, d_2) - t_9 \\ m_9 &= ((d_3 - a_3) \ggg 12) - d_2 - FF(a_3, b_2, c_2) - t_{10} \end{aligned} \tag{4.4}$$

4. In equation 4.4, $FF(d_{2,23}, a_{2,23}, b_{1,23}) = FF(\overline{d^{\text{old}}_{2,23}}, a_{2,23}, b_{1,23}) = (b_{1,23} \rightarrow a_{2,23}) \vee (a_{2,23} \rightarrow b_{1,23})$. The extra condition "$a_{2,23} = \overline{b_{1,23}}$" cause the complement of $FF(d_{2,23}, a_{2,23}, b_{1,23})$ to occur. The word $m_6 = ((c_2 - (d_2^{\text{old}} \pm 2^{22})) \ggg 17) - c_1 - (FF^{\text{old}}(d_2, a_2, b_1) \pm 2^{22}) - t_7 = m_6^{\text{old}} \mp (2^5 + 2^{22})$. Then $d_5 = ((d_4 + GG(a_4, b_3, c_3) + m_6 + t_{18}) \lll 9) + a_5 = ((d_4 + GG(a_4, b_3, c_3) + (m_6^{\text{old}} \mp (2^5 + 2^{22})) + t_{18}) \lll 9) + a_5 = d_5^{\text{old}} \mp (2^{14} + 2^{31})$. This causes the complement of the bit $d_{5,32}$ to occur.

– For the sufficient condition "$c_{5,18} = 0$" in the second iteration, we do the following procedure.

1. We complement the bit $b_{2,4}$.
2. Because b_2 changes, we use the following equations to update the message words m_7, m_8, m_9, m_{10}, and m_{11}.

$$\begin{aligned}
m_7 &= ((b_2 - c_2) \ggg 22) - b_1 - FF(c_2, d_2, a_2) - t_8 \\
m_8 &= ((a_3 - b_2) \ggg 7) - a_2 - FF(b_2, c_2, d_2) - t_9 \\
m_9 &= ((d_3 - a_3) \ggg 12) - d_2 - FF(a_3, b_2, c_2) - t_{10} \\
m_{10} &= ((c_3 - d_3) \ggg 17) - c_2 - FF(d_3, a_3, b_2) - t_{11} \\
m_{11} &= ((b_3 - c_3) \ggg 22) - b_2 - FF(c_3, d_3, a_3) - t_{12} \qquad (4.5)
\end{aligned}$$

3. In equation 4.5, $m_{11} = ((b_3 - c_3) \ggg 22) - (b_2^{\text{old}} \pm 2^3) - FF(c_3, d_3, a_3) - t_{12} = m_{11}^{\text{old}} \mp 2^3$. Then $c_5 = ((c_4 + GG(d_5, a_5, b_4) + m_{11} + t_{19}) \lll 14) + d_5 = ((c_4 + GG(d_5, a_5, b_4) + (m_{11} \mp 2^3) + t_{19}) \lll 14) + d_5 = c_5^{\text{old}} \mp 2^{17}$. This implies that the complement of the bit $c_{5,18}$ to occur.

– For the sufficient condition "$c_{5,32} = d_{5,32}$" in the second iteration, we do the following procedure.

1. We complement the bit $c_{4,18}$.
2. Because c_4 changes, we use the following equations to update the message words m_{14} and m_{15}.

$$\begin{aligned}
m_{14} &= ((c_4 - d_4) \ggg 17) - c_3 - FF(d_4, a_4, b_3) - t_{15} \\
m_{15} &= ((b_4 - c_4) \ggg 22) - b_3 - FF(c_4, d_4, a_4) - t_{16}
\end{aligned}$$

3. We compute the chaining value a_5 by $a_5 = ((a_4 + GG(b_4, c_4, d_4) + m_0 + t_{17}) \lll 5) + b_4$ but $c_{4,18}$ changes. However, the sufficient condition "$d4, 18 = 1$" guarantees that $GG(b_{4,18}, c_{4,18}, d_{4,18}) = GG(b_{4,18}, \overline{c^{\text{old}}_{4,18}}, 0)$ is always equal to $b_{4,18}$. This also implies that a_5 unchanges.

4. We add the extra condition "$b_{4,18} = 0$".

5. We compute the chaining value d_5 by $d_5 = ((d_4 + GG(a_5, b_4, c_4) + m_6 + t_{18}) \lll 9) + a_5$ and $c_{4,18}$ changes. But the extra condition "$b_{4,18} = 0$" cause that $GG(a_{5,18}, b_{4,18}, c_{4,18}) = GG(a_{5,18}, b_{4,18}, \overline{c^{\text{old}}_{4,18}}) = (a_{5,18} \rightarrow b_{4,18}) \wedge (b_{4,18} \rightarrow a_{5,18})$. The bit "$a_{5,18} = 0$" itself is a sufficient condition, so $GG(a_{5,18}, b_{4,18}, c_{4,18}) = 0 \rightarrow 0$ unchanges. This also implies that d_5 unchanges.

6. Finally, we update the chaining value c_5 is by $c_5 = ((c_4 + GG(d_5, a_5, b_4) + m_{11} + t_{19}) \lll 14) + d_5 = (((c^{\text{old}}_4 \pm 2^{17}) + GG(d_5, a_5, b_4) + m_{11} + t_{19}) \lll 14) + d_5 = c^{\text{old}}_5 \pm 2^{31}$. This implies that the complement of the bit $c_{5,32}$ to occur.

– For the sufficient condition "$d_{6,32} = a_{6,32}$" in the second iteration, we do the following procedure.

1. We complement the bit $a_{3,23}$.

2. We add the extra conditions "$d_{3,23} = 1$" and "$c_{3,23} = 1$".

3. Because a_3 changes, so we use the following equations to up-

date the message words m_8, m_9, m_{10}, and m_{12}.

$$\begin{aligned}
m_8 &= ((a_3 - b_2) \ggg 7) - a_2 - FF(b_2, c_2, d_2) - t_9 \\
m_9 &= ((d_3 - a_3) \ggg 12) - d_2 - FF(a_3, b_2, c_2) - t_{10} \\
m_{10} &= ((c_3 - d_3) \ggg 17) - c_2 - FF(d_3, a_3, b_2) - t_{11} \qquad (4.6) \\
m_{12} &= ((a_4 - b_3) \ggg 7) - a_3 - FF(b_3, c_3, d_3) - t_{13}
\end{aligned}$$

4. Because a_3 also changes, we update the chaining value b_3 by $b_3 = ((b_2 + FF(c_3, d_3, a_3) + m_{11} + t_{12}) \lll 22) + c_3$. But the extra condition "$c_{3,23} = 1$" cause that $FF(c_{3,23}, d_{3,23}, a_{3,23}) = FF(1, d_{3,23}, \overline{a^{\text{old}}_{3,23}})$ is always equal to $d_{3,23}$ and unchanges. So b_3 unchanges.

5. In equation 4.6, the extra conditions "$d_{3,23} = 1$" cause that $FF(d_{3,23}, a_{3,23}, b_{2,23}) = FF(1, \overline{a^{\text{old}}_{3,23}}, b_{2,23}) = \overline{a^{\text{old}}_{3,23}}$. Then $m_{10} = ((c_3 - d_3) \ggg 17) - c_2 - (FF^{\text{old}}(d_3, a_3, b_2) \pm 2^{22}) - t_{11} = m^{\text{old}}_{10} \mp 2^{22}$. The word $d_6 = ((d_5 + GG(a_6, b_5, c_5) + m_{10} + t_{22}) \lll 9) + a_6 = ((d_5 + GG(a_6, b_5, c_5) + (m^{\text{old}}_{10} \mp 2^{22}) + t_{22}) \lll 9) + a_6 = d^{\text{old}}_6 \mp 2^{31}$. This implies that the complement of the bit $d_{6,32}$ occurs.

Finally we analyze the time complexity of our improvements on multi-message modification of MD5. In the first iteration of MD5, our improvements is as efficient as Sasaki et al.'s results [SNKO05]. But the size of the set of our collision message pair is larger than Sasaki et al.'s. We also find that some extra conditions in their multi-message modification can be erased. These extra conditions are as follows:

- The extra condition "$d_{3,4} = 0$" for the sufficient condition "$c_{5,18} = 0$" [SNKO05, Table 5]. No matter what $c_{5,18}$ is, we just need to complement it.

- The extra condition "$a_{4,1} = 1$" and "$d_{1,13} = 0$" for the sufficient condition "$a_{6,18} = b_{5,18}$" [SNKO05, Table 8]. No matter what $a_{4,1}$ and $d_{1,13}$ are, we just need to complement them.

- The extra condition "$a_{3,23} = 0$" for the sufficient condition "$d_{6,32} = a_{6,32}$" [SNKO05, Table 10]. No matter what $a_{3,23}$ is, we just need to complement it.

So the size of the set of the collision message pair can be enlarged by erasing these four extra conditions. Because the collision searching algorithm so far doesn't satisfy all the sufficient conditions, they need to restart the whole collision searching algorithm many times. If we add these extra conditions, we must reset these extra conditions when restarting the whole collision searching algorithm. This is because that these extra conditions are broken when doing the message modification in the second round. So we need to recover them by resetting them. But other conditions, including sufficient conditions and other extra conditions, are not broken when doing the message modification in the second round. Our improvements is more efficient than Liang and Lai's results [LL05]. Note that the collision searching algorithm will restart the whole algorithm many times. As mentioned before, the small range searching algorithm given by Liang and Lai [LL05] is less efficient than Sasaki et al.'s multi-message modification methods [SNKO05]. So in each time, a small time increase will cause that the time of finding the collision increase longer because the algorithm will execute inside the loop many times.

For the second iteration of MD5, Sasaki et al. don't give any multi-message modification method. We combine Liang and Lai's results [LL05] and our improvements introduced before. We can modify for the sufficient conditions "$a_{5,4} = b_{4,4}$", "$a_{5,16} = b_{4,16}$", "$a_{5,18} = 0$", and "$d_{5,30} = a_{5,30}$" by using Liang and Lai's type 1 multi-message modification methods, and for the sufficient conditions "$d_{5,18} = 1$", "$d_{5,32} = a_{5,32}$", "$c_{5,18} = 0$", "$c_{5,32} = d_{5,32}$", and "$d_{6,32} = a_{6,32}$" by using our own type 2 multi-message modification methods. Because Sasaki et al. don't give any multi-message modification method for the second iteration, so our improvements are more efficient than theirs. We give 5 more type 2 multi-message modification on 5 sufficient conditions than Liang and Lai's results for the second round. In these 5 sufficient conditions, Liang and Lai either use the small range searching techniques or restart the whole collision searching algorithm. So our improvements are more efficient than Liang and Lai's results.

Chapter 5

Our Implementation on the Collision Searching Algorithm

5.1 A brief description of the Implementation

In this section, we describe our implementation of the collision searching algorithm briefly. The code is available on `http://www.cs.nctu.edu.tw/~gtchen/Codes/md4coll.c` and `http://www.cs.nctu.edu.tw/~gtchen/Codes/md5coll.c` for MD4 and MD5, respectively. The MD4 code was developed independently, and Stach has his own implementation [Sta05a]. The MD5 code was derived from Stach's implementation[Sta05b], although most of them was already modified by us. Stach is the first man who implemented the collision searching algorithm of MD4 and MD5 and published the codes. We can find that there are 7 kinds of all the bits of all the chaining values. Stach also found that and implemented it in his code, although we have already found that before the release of his implementation. Now we consider the j^{th} bit of the chaining value x in the i^{th} step (we denote it as x_j), and the 7 kinds of the bit x_j:

1. There is no sufficient condition here, i.e., this bit of the chaining value x_j can be set to 0 or 1.

2. This bit of the chaining value x_j here must set to be 0.

3. This bit of the chaining value x_j here must set to be 1.

4. This bit of the chaining value x_j here must set to be the value of the same bit of the last updated chaining value (the j^{th} bit of the chaining value in the $(i-1)^{\text{th}}$ step).

5. This bit of the chaining value x_j here must set to be the complement of the same bit of the last updated chaining value(the j^{th} bit of the chaining value in the $(i-1)^{\text{th}}$ step).

6. This bit of the chaining value x_j here must set to be the value of the same bit of the last two updated chaining value (the j^{th} bit of the chaining value in the $(i-2)^{\text{th}}$ step).

7. This bit of the chaining value x_j here must set to be the complement of the same bit of the last two updated chaining value (the j^{th} bit of the chaining value in the $(i-2)^{\text{th}}$ step).

In the above statements, if we consider the sufficient condition x_j in the chaining value x, y_j is the same bit of the chaining value y with respective to x_j. So we need to construct the bit masks for representing the 7 types of the bits of the chaining values. But for the bit of the chaining values that has no sufficient condition, we can ignore it. So we construct the 6 bit-mask for every chaining values during the computation of the compression function. We define the six 32-bit bit mask $S_{i,k}$, where $0 \leq k \leq 5$, for the i^{th} step chaining value x as follows:

1. The bit $S_{i,0,j}$ is 1 if x_j must set to be 0, otherwise $S_{i,0,j}$ is 0.

2. The bit $S_{i,1,j}$ is 1 if x_j must set to be 1, otherwise $S_{i,1,j}$ is 0.

3. The bit $S_{i,2,j}$ is 1 if x_j must set to be the value of the same bit of the last updated chaining value, otherwise $S_{i,2,j}$ is 0.

4. The bit $S_{i,3,j}$ is 1 if x_j must set to be the complement of the same bit of the last updated chaining value, otherwise $S_{i,3,j}$ is 0.

5. The bit $S_{i,4,j}$ is 1 if x_j must set to be the value of the same bit of the last two updated chaining value, otherwise $S_{i,4,j}$ is 0.

6. The bit $S_{i,5,j}$ is 1 if x_j here must set to be the complement of the same bit of the last two updated chaining value, otherwise $S_{i,5,j}$ is 0.

We don't need to modify Wang et al.'s MD4 collision searching algorithm as described in Section 3.4. But we need to modify Wang et al.'s MD5 collision searching algorithm as described in Section 3.4 to optimize the collision searching algorithm. Our MD5 collision searching algorithm now will run as follows:

1. We use the following procedure to generate the first iteration message block M_1.

   (a) We set $\mathrm{IV}_0 = \mathrm{IV} = a_0||b_0||c_0||d_0$.

   (b) For the sufficient conditions in the first round, we do the single-message modification by Klima's idea [Kli05a] as follows:

$$x \overset{\mathrm{u}}{\leftarrow} Z_{2^{32}},$$

$$c_1 = [x \wedge \overline{S_{3,0}}] \vee [x \vee S_{3,1}].$$

$$x \overset{\mathrm{u}}{\leftarrow} Z_{2^{32}},$$

$$b_1 = [x \wedge \overline{S_{4,0}}] \vee [x \vee S_{4,1}]$$

$$\vee [(x \wedge \overline{S_{4,2}}) \vee (c_1 \wedge S_{4,2})] \vee [(x \wedge \overline{S_{4,3}}) \vee (\overline{c_1} \wedge S_{4,3})].$$

$$x \overset{\mathrm{u}}{\leftarrow} Z_{2^{32}},$$

$$a_2 = [x \wedge \overline{S_{5,0}}] \vee [x \vee S_{5,1}]$$

$$\vee [(x \wedge \overline{S_{5,2}}) \vee (b_1 \wedge S_{5,2})] \vee [(x \wedge \overline{S_{5,3}}) \vee (\overline{b_1} \wedge S_{5,3})]$$

$$\vee [(x \wedge \overline{S_{5,4}}) \vee (c_1 \wedge S_{5,4})] \vee [(x \wedge \overline{S_{5,5}}) \vee (\overline{c_1} \wedge S_{5,5})].$$

   Then for $i = 6$ to 16,

$$\text{If } i \mod 4 = 1, \quad x \overset{\mathrm{u}}{\leftarrow} Z_{2^{32}},$$

$$a_{\lceil \frac{i}{4} \rceil} = [x \wedge \overline{S_{i,0}}] \vee [x \vee S_{i,1}]$$

$$\vee [(x \wedge \overline{S_{i,2}}) \vee (b_{\lceil \frac{i-1}{4} \rceil} \wedge S_{i,2})] \vee [(x \wedge \overline{S_{i,3}}) \vee (\overline{b_{\lceil \frac{i-1}{4} \rceil}} \wedge S_{i,3})]$$

$$\vee [(x \wedge \overline{S_{i,4}}) \vee (c_{\lceil \frac{i-2}{4} \rceil} \wedge S_{i,4})] \vee [(x \wedge \overline{S_{i,5}}) \vee (\overline{c_{\lceil \frac{i-2}{4} \rceil}} \wedge S_{i,5})]$$

$$\Sigma_i = (a_{\lceil \frac{i}{4} \rceil} - b_{\lceil \frac{i-1}{4} \rceil}) \ggg s_i$$

$$m_{i-1} = \Sigma_i - a_{\lceil \frac{i-4}{4} \rceil} - FF(b_{\lceil \frac{i-1}{4} \rceil}, c_{\lceil \frac{i-2}{4} \rceil}, d_{\lceil \frac{i-3}{4} \rceil}) - t_i$$

If $i \mod 4 = 2, \quad x \overset{\mathrm{u}}{\leftarrow} Z_{2^{32}},$

$$d_{\lceil \frac{i}{4} \rceil} = [x \wedge \overline{S_{i,0}}] \vee [x \vee S_{i,1}]$$

$$\vee [(x \wedge \overline{S_{i,2}}) \vee (a_{\lceil \frac{i-1}{4} \rceil} \wedge S_{i,2})] \vee [(x \wedge \overline{S_{i,3}}) \vee (\overline{a_{\lceil \frac{i-1}{4} \rceil}} \wedge S_{i,3})]$$

$$\vee [(x \wedge \overline{S_{i,4}}) \vee (b_{\lceil \frac{i-2}{4} \rceil} \wedge S_{i,4})] \vee [(x \wedge \overline{S_{i,5}}) \vee (\overline{b_{\lceil \frac{i-2}{4} \rceil}} \wedge S_{i,5})]$$

$$\Sigma_i = (d_{\lceil \frac{i}{4} \rceil} - a_{\lceil \frac{i-1}{4} \rceil}) \ggg s_i$$

$$m_{i-1} = \Sigma_i - d_{\lceil \frac{i-4}{4} \rceil} - FF(a_{\lceil \frac{i-1}{4} \rceil}, b_{\lceil \frac{i-2}{4} \rceil}, c_{\lceil \frac{i-3}{4} \rceil}) - t_i$$

If $i \mod 4 = 3, \quad x \overset{\mathrm{u}}{\leftarrow} Z_{2^{32}},$

$$c_{\lceil \frac{i}{4} \rceil} = [x \wedge \overline{S_{i,0}}] \vee [x \vee S_{i,1}]$$

$$\vee [(x \wedge \overline{S_{i,2}}) \vee (d_{\lceil \frac{i-1}{4} \rceil} \wedge S_{i,2})] \vee [(x \wedge \overline{S_{i,3}}) \vee (\overline{d_{\lceil \frac{i-1}{4} \rceil}} \wedge S_{i,3})]$$

$$\vee [(x \wedge \overline{S_{i,4}}) \vee (a_{\lceil \frac{i-2}{4} \rceil} \wedge S_{i,4})] \vee [(x \wedge \overline{S_{i,5}}) \vee (\overline{a_{\lceil \frac{i-2}{4} \rceil}} \wedge S_{i,5})]$$

$$\Sigma_i = (c_{\lceil \frac{i}{4} \rceil} - d_{\lceil \frac{i-1}{4} \rceil}) \ggg s_i$$

$$m_{i-1} = \Sigma_i - c_{\lceil \frac{i-4}{4} \rceil} - FF(d_{\lceil \frac{i-1}{4} \rceil}, a_{\lceil \frac{i-2}{4} \rceil}, b_{\lceil \frac{i-3}{4} \rceil}) - t_i$$

If $i \mod 4 = 0, \quad x \overset{\mathrm{u}}{\leftarrow} Z_{2^{32}},$

$$b_{\lceil \frac{i}{4} \rceil} = [x \wedge \overline{S_{i,0}}] \vee [x \vee S_{i,1}]$$

$$\vee [(x \wedge \overline{S_{i,2}}) \vee (c_{\lceil \frac{i-1}{4} \rceil} \wedge S_{i,2})] \vee [(x \wedge \overline{S_{i,3}}) \vee (\overline{c_{\lceil \frac{i-1}{4} \rceil}} \wedge S_{i,3})]$$

$$\vee [(x \wedge \overline{S_{i,4}}) \vee (d_{\lceil \frac{i-2}{4} \rceil} \wedge S_{i,4})] \vee [(x \wedge \overline{S_{i,5}}) \vee (\overline{d_{\lceil \frac{i-2}{4} \rceil}} \wedge S_{i,5})]$$

$$\Sigma_i = (b_{\lceil \frac{i}{4} \rceil} - c_{\lceil \frac{i-1}{4} \rceil}) \ggg s_i$$
$$m_{i-1} = \Sigma_i - b_{\lceil \frac{i-4}{4} \rceil} - FF(c_{\lceil \frac{i-1}{4} \rceil}, d_{\lceil \frac{i-2}{4} \rceil}, a_{\lceil \frac{i-3}{4} \rceil}) - t_i$$

where x is a 32-bit word.

(c) For the sufficient conditions in the chaining value a_5, we do the single-message modification as follows:

$$x \overset{u}{\leftarrow} Z_{2^{32}},$$
$$a_5 = [x \wedge \overline{S_{17,0}}] \vee [x \vee S_{17,1}]$$
$$\vee [(x \wedge \overline{S_{17,2}}) \vee (b_4 \wedge S_{17,2})] \vee [(x \wedge \overline{S_{17,3}}) \vee (\overline{b_4} \wedge S_{17,3})]$$
$$\vee [(x \wedge \overline{S_{17,4}}) \vee (c_4 \wedge S_{17,4})] \vee [(x \wedge \overline{S_{17,5}}) \vee (\overline{c_4} \wedge S_{17,5})]$$
$$\Sigma_{17} = (a_5 - b_4) \ggg 5$$
$$m_0 = \Sigma_{17} - a_4 - FF(b_4, c_4, d_4) - t_{17}$$

We recover the chaining value a_1 as follows:

$$a_1 = ((a_0 + FF(b_0, c_0, d_0) + m_0 + t_1) \lll 7) + b_0$$

(d) We use the multi-message modification introduced in Chapter 4 to let the chaining values d_5 and c_5 to satisfy all their corresponding sufficient conditions.

(e) For the sufficient conditions in the chaining value b_5, we do the single-message modification as follows:

$$x \xleftarrow{\text{u}} Z_{2^{32}},$$

$$b_5 = [x \wedge \overline{S_{20,0}}] \vee [x \vee S_{20,1}]$$

$$\vee [(x \wedge \overline{S_{20,2}}) \vee (c_5 \wedge S_{20,2})] \vee [(x \wedge \overline{S_{20,3}}) \vee (\overline{c_5} \wedge S_{20,3})]$$

$$\vee [(x \wedge \overline{S_{20,4}}) \vee (d_5 \wedge S_{17,4})] \vee [(x \wedge \overline{S_{20,5}}) \vee (\overline{d_5} \wedge S_{20,5})]$$

$$\Sigma_{20} = (b_5 - c_5) \ggg 20$$

$$m_1 = \Sigma_{20} - b_4 - FF(c_5, d_5, a_5) - t_{20}$$

We recover the chaining value d_1 as follows:

$$d_1 = ((d_0 + FF(a_1, b_0, c_0) + m_1 + t_2) \lll 12) + a_1$$

Then we recover the message words m_2, m_3, and m_4 as follows:

$$m_2 = ((c_1 - d_1) \ggg 17) - c_0 - FF(d_1, a_1, b_0) - t_3$$

$$m_3 = ((b_1 - c_1) \ggg 22) - b_0 - FF(c_1, d_1, a_1) - t_4$$

$$m_4 = ((a_2 - b_1) \ggg 7) - a_1 - FF(b_1, c_1, d_1) - t_5$$

(f) For the remaining sufficient conditions that can be fulfilled by the multi-message modification as described in Chapter 4, if the corresponding bits of the chaining values are not the same as them, we correct them by using the multi-message modification introduced in Chapter 4. For all the sufficient conditions that can't be fulfilled by the message modification techniques, we check if the chaining value x in the i^{th} step during the computation of the compression function and its corresponding sufficient conditions are equivalent using the following equation:

$$\begin{aligned}&(x \wedge (S_{i,0} \vee S_{i,1} \vee S_{i,2} \vee S_{i,3} \vee S_{i,4} \vee S_{i,5}))\\=&(x \wedge S_{i,0}) \vee (y \wedge S_{i,2}) \vee (\overline{y} \wedge S_{i,3}) \vee (z \wedge S_{i,4}) \vee (\overline{z} \wedge S_{i,5}) \vee S_{i,1}\end{aligned} \tag{5.1}$$

where y and z are the chaining values in the $(i-1)^{\text{th}}$ and $(i-2)^{\text{th}}$ step, respectively.

If it is not the case, we go back to step 1c.

(g) Then the first message block $M_1 \triangleq m_1||m_2||\ldots||m_{15}$ and $\text{IV}_1 \triangleq (a_0 + a_{16})||(b_0 + b_{16})||(c_0 + c_{16})||(d_0 + d_{16})$.

2. We use the following procedure to generate the second iteration message block M_2.

   (a) We set $\text{IV}_1 = a_0||b_0||c_0||d_0$

   (b) We set $i_{\text{low}} = 1$ and $i_{\text{high}} = 16$.

   (c) For the sufficient conditions in the first round, we do the single-message modification as follows:

For $i = i_{\text{low}}$ to i_{high}.

If $i \mod 4 = 1, \quad x \overset{\text{u}}{\leftarrow} Z_{2^{32}}$,

$$a_{\lceil \frac{i}{4} \rceil} = [x \wedge \overline{S_{i,0}}] \vee [x \vee S_{i,1}]$$

$$\vee [(x \wedge \overline{S_{i,2}}) \vee (b_{\lceil \frac{i-1}{4} \rceil} \wedge S_{i,2})] \vee [(x \wedge \overline{S_{i,3}}) \vee (\overline{b_{\lceil \frac{i-1}{4} \rceil}} \wedge S_{i,3})]$$

$$\vee [(x \wedge \overline{S_{i,4}}) \vee (c_{\lceil \frac{i-2}{4} \rceil} \wedge S_{i,4})] \vee [(x \wedge \overline{S_{i,5}}) \vee (\overline{c_{\lceil \frac{i-2}{4} \rceil}} \wedge S_{i,5})]$$

$$\Sigma_i = (a_{\lceil \frac{i}{4} \rceil} - b_{\lceil \frac{i-1}{4} \rceil}) \ggg s_i$$

$$m_{i-1} = \Sigma_i - a_{\lceil \frac{i-4}{4} \rceil} - FF(b_{\lceil \frac{i-1}{4} \rceil}, c_{\lceil \frac{i-2}{4} \rceil}, d_{\lceil \frac{i-3}{4} \rceil}) - t_i$$

If $i \mod 4 = 2, \quad x \overset{\text{u}}{\leftarrow} Z_{2^{32}}$,

$$d_{\lceil \frac{i}{4} \rceil} = [x \wedge \overline{S_{i,0}}] \vee [x \vee S_{i,1}]$$

$$\vee [(x \wedge \overline{S_{i,2}}) \vee (a_{\lceil \frac{i-1}{4} \rceil} \wedge S_{i,2})] \vee [(x \wedge \overline{S_{i,3}}) \vee (\overline{a_{\lceil \frac{i-1}{4} \rceil}} \wedge S_{i,3})]$$

$$\vee [(x \wedge \overline{S_{i,4}}) \vee (b_{\lceil \frac{i-2}{4} \rceil} \wedge S_{i,4})] \vee [(x \wedge \overline{S_{i,5}}) \vee (\overline{b_{\lceil \frac{i-2}{4} \rceil}} \wedge S_{i,5})]$$

$$\Sigma_i = (d_{\lceil \frac{i}{4} \rceil} - a_{\lceil \frac{i-1}{4} \rceil}) \ggg s_i$$

$$m_{i-1} = \Sigma_i - d_{\lceil \frac{i-4}{4} \rceil} - FF(a_{\lceil \frac{i-1}{4} \rceil}, b_{\lceil \frac{i-2}{4} \rceil}, c_{\lceil \frac{i-3}{4} \rceil}) - t_i$$

If $i \mod 4 = 3, \quad x \overset{\text{u}}{\leftarrow} Z_{2^{32}}$,

$$c_{\lceil \frac{i}{4} \rceil} = [x \wedge \overline{S_{i,0}}] \vee [x \vee S_{i,1}]$$

$$\vee [(x \wedge \overline{S_{i,2}}) \vee (d_{\lceil \frac{i-1}{4} \rceil} \wedge S_{i,2})] \vee [(x \wedge \overline{S_{i,3}}) \vee (\overline{d_{\lceil \frac{i-1}{4} \rceil}} \wedge S_{i,3})]$$

$$\vee [(x \wedge \overline{S_{i,4}}) \vee (a_{\lceil \frac{i-2}{4} \rceil} \wedge S_{i,4})] \vee [(x \wedge \overline{S_{i,5}}) \vee (\overline{a_{\lceil \frac{i-2}{4} \rceil}} \wedge S_{i,5})]$$

$$\Sigma_i = (c_{\lceil \frac{i}{4} \rceil} - d_{\lceil \frac{i-1}{4} \rceil}) \ggg s_i$$

$$m_{i-1} = \Sigma_i - c_{\lceil \frac{i-4}{4} \rceil} - FF(d_{\lceil \frac{i-1}{4} \rceil}, a_{\lceil \frac{i-2}{4} \rceil}, b_{\lceil \frac{i-3}{4} \rceil}) - t_i$$

If $i \mod 4 = 0, \quad x \overset{\text{u}}{\leftarrow} Z_{2^{32}}$,

$$b_{\lceil \frac{i}{4} \rceil} = [x \wedge \overline{S_{i,0}}] \vee [x \vee S_{i,1}]$$

$$\vee [(x \wedge \overline{S_{i,2}}) \vee (c_{\lceil \frac{i-1}{4} \rceil} \wedge S_{i,2})] \vee [(x \wedge \overline{S_{i,3}}) \vee (\overline{c_{\lceil \frac{i-1}{4} \rceil}} \wedge S_{i,3})]$$

$$\vee [(x \wedge \overline{S_{i,4}}) \vee (d_{\lceil \frac{i-2}{4} \rceil} \wedge S_{i,4})] \vee [(x \wedge \overline{S_{i,5}}) \vee (\overline{d_{\lceil \frac{i-2}{4} \rceil}} \wedge S_{i,5})]$$

$$\Sigma_i = (b_{\lceil \frac{i}{4} \rceil} - c_{\lceil \frac{i-1}{4} \rceil}) \ggg s_i$$

$$m_{i-1} = \Sigma_i - b_{\lceil \frac{i-4}{4} \rceil} - FF(c_{\lceil \frac{i-1}{4} \rceil}, d_{\lceil \frac{i-2}{4} \rceil}, a_{\lceil \frac{i-3}{4} \rceil}) - t_i$$

where x is a 32-bit word.

(d) Then for the remaining sufficient conditions that can be fulfilled by the multi-message modification as described in Chapter 4, if the corresponding bits of the chaining values are not the same as them, we correct them by using the multi-message modification introduced in Chapter 4. For all the sufficient conditions that can't be fulfilled by the message modification techniques, we check if the chaining value x in the i^{th} step during the computation of the compression function and its corresponding sufficient conditions are equivalent using the equation 5.1. If it is not the case, we set $i_{\text{low}} = 15$, $i_{\text{high}} = 16$ and then go back to step 2c.

(e) Then the second message block is set as $M_2 \triangleq m_1 || m_2 || \ldots || m_{15}$.

3. Finally, if the message $M = M_1||M_2$ satisfies all the correct sufficient conditions, $M' = M_1'||M_2' = (M_1 + \Delta M_1)||(M_2 + \Delta M_2)$ and M will collide.

In the above MD5 collision finding procedure, if $S_{i,j} = 0$ for $1 \leq i \leq 64 \wedge 0 \leq j \leq 5$, we can ignore it for efficiency.

5.2 Analysis of Our Implementation

Because the MD4 collision searching algorithm is very efficient, so we don't compare its performance in this section. But according to our execution experiments, our implementation is also more efficient than Stach's. You can download our implementation code and compare it with Stach's by compiling and executing both ones. Our implementation of the MD4 collision searching algorithm gives a collision pair in a very short time even in the worst case. But Stach's takes more than 1 minute in the worst case. In this section, we compare our implementation of MD5 collision searching algorithm with Stach's. We put our experiment data on `http://www.cs.nctu.edu.tw/~gtchen/Data/md5.tbz` and the modified version of Stach's code on `http://www.cs.nctu.edu.tw/~gtchen/Codes/md5coll-orig_modified.c` for comparison.

5.2.1 Correctness Analysis

In Stach's implementation of the collision searching algorithms of MD5, the second block does not always exist for any the first block whose message differential is ΔM_1 and hash value differential is Δh_1 as described in section 3.1. In `http://www.stachliu.com/collisions.html`, he claimed that "some block #1's don't have block #2 solutions." But we think what he claimed is not correct. As long as the differential value of the internal chaining values are the same as Wang et al.'s, the probability that the message pair M and $M' = M + \Delta M$ collide is always 1. So in our opinion, there may be some sufficient conditions lost in his implementation. We do the experiments by running our implementation and Stach's implementation 50 times respectively to measure the correctness and the performance of the implementations. We assume that the experiment whose execution time exceeds 12 hours would fail to find the collision pair. According to the experiments on Stach's implementation, we find that 50% of the executions exceed 12 hours. But all the execution experiments of our implementation don't exceed 9 hours. Although some experiments of Stach's implementation don't give the collision pair, but it always find the first iteration message block.

5.2.2 Performance Analysis

In this section, we compare the execution time of our implementation of MD5 collision finding algorithm with Stach's. We summary the result of the total fifty times experiments per implementation in table 5.1. In `http://www.stachliu.com/collisions.html`, he said "average run time on P4 1.6ghz PC — 45 minutes." But according to our experiments on P4 2.8GHz PC, the result is quite different from what he said. We should mention that

	Average Case	Best Case	Worst Case	Sucessful Probability
Our Implementation	1h48m10s	4m20s	8h50m01s	1
Stach's Implementation	2h06m13s	8m29s	5h58m57s	0.5

Table 5.1: The Execution Time of the Experiments

the execution time of his implementation in the worst case is much better than ours.

Chapter 6

Conclusions

After Wang et al. publish their MD4 and MD5 collision searching algorithms, many researchers publish their improvements. After their improvements were given, the collision searching algorithms become more and more definitive and efficient. In this thesis, we also give our own improvements on the message modification techniques. Then we implement the MD4 and MD5 collision searching algorithms to show that our improvements are efficient enohgh to run on modern PCs. Some cryptographic scholars think that the collision resistance requirement is not necessary for a cryptographic hash function in all cryptographic applications. Because it is infeasible for any probabilistic polynomial time adversary to break all cryptographic applications even their internal cryptographic hash function is not collision resistant. For example, if the adversary Eve want to fake Alice's certificate, it is infeasible for him to do this even the certificate use a non-collision resistant cryptographic hash function. So we can relax the collision resistance requirements in some cryptographic applications, but not in all of them. As mentioned in Section

2.1, some cryptographic applications need stronger security requirements of the hash functions, not just the three basic ones. If a hash function is not second preimage resistant, the hash function is broken wholly. So how to find the second preimage of the existing hash functions is an interesting problem. Yu et al. [YWZW05] gave their discovery on finding the second preimage of the hash function MD4. But they didn't really find the real second preimage of the hash function MD4. Instead, they gave a collision path that is of higher probability to find the second preimage of MD4 than Wang et al.'s [WLF$^+$05, Chapter 6]. For a randomly chosen message block M, they gave a sufficiently efficient algorithm to modify M to $\widehat{M}$, where the hamming distance of M and $\widehat{M}$ is very small, and the second preimage of $\widehat{M}$ can be computed efficiently. But their results are not practical enough to damage the use of MD4 in the real world for digital signatures, certificates, MACs, and so on. Instead, we want an algorithm to find the second preimage for any message, even though the time complexity of the algorithm is a little bigger. As long as the time complexity of the second preimage finding algorithm is not too large to run on modern computers, we can accept it. So how to find the second preimage for arbitrary message of the dedicated hash functions, such as MD4, MD5, SHA-1, is an interesting problem for solving.

Bibliography

[BR96] R. Baldwin and R. Rivest. The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms. Request for Comments (RFC 2040), October 1996.

[Bra90] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *STOC*, pages 209–218, 1998.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[Cra05] Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark,* *May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[Dam89] Ivan Damgård. A design principle for hash functions. In Brassard [Bra90], pages 416–427.

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. Ripemd-160: A strengthened version of ripemd. In Dieter Gollmann, editor, *Fast Software Encryption*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1996.

[FIP95] Secure hash standard (shs). Federal Information Processing Standard (FIPS) Publication 180-1, National Institute of Standards and Technology (NIST), April 1995.

[FIP02] Secure hash standard (shs). Federal Information Processing Standard (FIPS) Publication 180-2, National Institute of Standards and Technology (NIST), August 2002.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–. IEEE Computer Society, 2003.

[HPR04] Philip Hawkes, Michael Paddon, and Gregory G. Rose. Musings on the wang et al. md5 collision. Cryptology ePrint Archive, Report 2004/264, 2004. `http://eprint.iacr.org/2004/264.pdf`.

[Kal92] B. Kaliski. The MD2 Message-Digest Algorithm. Request for Comments (RFC 1319), April 1992.

[Ken05] S. Kent. IP Encapsulating Security Payload (ESP). Request for Comments (RFC 4303), December 2005.

[Kli05a] Vlastimil Klima. Finding md5 collisions on a notebook pc using multi-message modifications. Cryptology ePrint Archive, Report 2005/102, 2005. `http://eprint.iacr.org/2005/102.pdf`.

[Kli05b] Vlastimil Klima. Finding md5 collisions v a toy for a notebook. Cryptology ePrint Archive, Report 2005/075, 2005. `http://eprint.iacr.org/2005/075.pdf`.

[KM05] Lars R. Knudsen and John Erik Mathiassen. Preimage and collision attacks on md2. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 255–267. Springer, 2005.

[LdW05] Arjen K. Lenstra and Benne de Weger. On the possibility of constructing meaningful hash collisions for public keys. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, volume 3574 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 2005.

[LL05] Jie Liang and Xuejia Lai. Improved collision attack on hash function md5. Cryptology ePrint Archive, Report 2005/425, 2005. `http://eprint.iacr.org/2005/425.pdf`.

[LWdW05] Arjen Lenstra, Xiaoyun Wang, and Benne de Weger. Colliding x.509 certificates. Cryptology ePrint Archive, Report 2005/067, 2005. `http://eprint.iacr.org/2005/067.pdf`.

[Mer89] Ralph C. Merkle. One way hash functions and des. In Brassard [Bra90], pages 428–446.

[Mik04] Ondrej Mikle. Practical attacks on digital signatures using md5 message digest. Cryptology ePrint Archive, Report 2004/356, 2004. `http://eprint.iacr.org/2004/356.pdf`.

[Mul04] Frédéric Muller. The md2 hash function is not one-way. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 2004.

[NSKO05] Yusuke Naito, Yu Sasaki, Noboru Kunihiro, and Kazuo Ohta. Improved collision attack on md4 with probability almost 1. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2005.

[RB00] Vincent Rijmen and Paulo S. L. M. Barreto. The WHIRLPOOL hash function. First open NESSIE Workshop record, November

2000. The document is available at http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html.

[RC97] N. Rogier and Pascal Chauvaud. Md2 is not secure without the checksum byte. *Des. Codes Cryptography*, 12(3):245–251, 1997.

[Riv90] Ronald L. Rivest. The md4 message digest algorithm. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer, 1990.

[Riv92a] R. Rivest. The MD4 Message-Digest Algorithm. Request for Comments (RFC 1320), April 1992.

[Riv92b] R. Rivest. The MD5 Message-Digest Algorithm . Request for Comments (RFC 1321), April 1992.

[SNKO05] Yu Sasaki, Yusuke Naito, Noboru Kunihiro, and Kazuo Ohta. Improved collision attack on md5. Cryptology ePrint Archive, Report 2005/400, 2005. http://eprint.iacr.org/2005/400.pdf.

[SNY$^+$06] Yu Sasaki, Yusuke Naito, Jun Yajima, Takeshi Shimoyama, Noboru Kunihiro, and Kazuo Ohta. How to construct sufficient condition in searching collisions of md5. Cryptology ePrint Archive, Report 2006/074, 2006. http://eprint.iacr.org/.

[Sta05a] Patrick Stach. MD4 Collision Generation— Faster implementation of techniques in "Cryptanalysis for Hash Functions MD4 and RIPEMD". `http://www.stachliu.com/md4coll.c`, 2005.

[Sta05b] Patrick Stach. MD5 Collision Generation— Faster implementation of techniques in "How to Break MD5 and Other Hash Functions". `http://www.stachliu.com/md5coll.c`, 2005.

[WLF$^+$05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions md4 and ripemd. In Cramer [Cra05], pages 1–18.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In Cramer [Cra05], pages 19–35.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[YS05] Jun Yajima and Takeshi Shimoyama. Wang's sufficient conditions of md5 are not sufficient. Cryptology ePrint Archive, Report 2005/263, 2005. `http://eprint.iacr.org/2005/263.pdf`.

[YWZW05] Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The second-preimage attack on md4. In Yvo Desmedt, Huaxiong

Wang, Yi Mu, and Yongqing Li, editors, *CANS*, volume 3810 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[ZPS92] Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry. Haval - a one-way hashing algorithm with variable length of output. In Jennifer Seberry and Yuliang Zheng, editors, *ASIACRYPT*, volume 718 of *Lecture Notes in Computer Science*, pages 83–104. Springer, 1992.