ABSTRACT

Title of Document:	A SURVEY OF THE ATTACK ON MD5.
	Prathap Sridharan, Master of Science, 2006
Directed By:	Professor Lawrence Washington, Mathematics

In Eurocrypt 2005, Wang et al. presented an exciting paper that showcased her method of breaking MD5 by attacking its collision resistance propery. However, Wang's paper does not give a thorough exposition of the attack and much of their techniques are shrouded in mystery. This paper attempts to explain Wang's attack on MD5 in greater detail by consolidating the various expository works on the subject.

A SURVEY OF THE ATTACK ON MD5.

By

Prathap Sridharan

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2006

Acknowledgements

I would like to thank my advisor, Professor Lawrence Washington, for his encouragement, constructive comments and helpful insights during the writing of this thesis. I would also like to thank Professor Jonathan Katz for giving me the idea to write this thesis, and providing me with the resources to conduct my research. Finally, I would like to thank Yiqun Lisa Yin and Phillip Hawkes for referring me to good sources of research on the topic.

Table of Contents

Acknowledgements ii
Table of Contents iii
Chapter 1: Introduction 1
1.1 Applications of Hash Functions 2
1.2 The Merkle-Damgård Construction for Hash Functions 4
1.3 Structure of the Thesis 6
Chapter 2: The MD5 Algorithm 7
Chapter 3: A Toolbox for Cryptanalysis of MD5 12
3.1 Modular Differences and XOR Differences 12
3.2 Modular Addition and Bit Rotation 17
3.3 Difference Propagation 20
Chapter 4: Wang's Attack on MD4 27
4.1 Computing Necessary Conditions for the Differential Characteristics 33
4.2 The Message Block Differentials and Differential Pattern 36
4.3 Message Modification 42
Bibliography 48

Chapter 1: Introduction

A cryptographic hash function is a hash function with certain additional security properties to make it suitable for use as a primitive in various information security applications, such as authentication and message integrity. A hash function takes a long string (or message) of any length as input and produces a fixed length string as output, sometimes termed a message digest or a digital fingerprint. In various standards and applications, the two most-commonly used hash functions are MD5 and SHA-1 ; however, as of 2005, security flaws have been identified in both algorithms.

Definition 1.1 (Cryptographic Hash Function) *A cryptographic hash function is a mapping*

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

where $\{0,1\}^*$ *denotes the set of bit strings of arbitrary length. The image* $h(X)$ *of some message* $X \in \{0,1\}^*$ *is called the hash value of X.*

Broadly speaking, a cryptographic hash function should behave as much as possible like a random function while still being deterministic and efficiently computable. There is no formal definition which captures all of the properties considered desirable for a cryptographic hash function. The properties below are generally considered prerequisites and a violation of any of these properties implies a weak hash function:

Preimage resistant: given h it should be computationally infeasible to find any m such that $h = \text{hash}(m)$.

Second preimage resistant: given an input m_1, it should be computationally infeasible to find another input, m_2 (not equal to m_1) such that hash(m_1) = hash(m_2).

Collision-resistant: it should be computationally infeasible to find two different messages m_1 and m_2 such that hash(m_1) = hash(m_2).

It should be noted that the meaning of "computationally infeasible" is very much a subjective phrase. One can define a problem to be "computationally infeasible" if solving it would require more than a pre-specified upper bound in space or computing speed. However, considering the current rate with which computing machines are being improved, what might be deemed "computationally infeasible" today might be perfectly feasible tomorrow.

1.1 Applications of Hash Functions

A particularly important application of hash functions occurs in the context of digital signature schemes. Digital signature is a type of method for authenticating digital information analogous to ordinary physical signatures on paper, but implemented using techniques from the field of public key cryptography.

Digital signature schemes rely on public-key cryptography. In public-key cryptography, each user has a pair of keys: one public and one private. The public key is distributed freely, but the private key is kept secret and confidential; another requirement is that it should be infeasible to derive the private key from the public key. A general digital signature scheme consists of three algorithms:

- A key generation algorithm
- A signing algorithm
- A verification algorithm

For example, consider the situation in which Bob sends a message to Alice and wants to be able to prove it came from him. Bob sends his message to Alice and attaches a digital signature. The digital signature is generated using Bob's private key, and takes the form of a simple numerical value (normally represented as a string of binary digits). On receipt, Alice can then check whether the message really came from Bob by running the verification algorithm on the message together with the signature and Bob's public key. If the verification algorithm accepts the message, then Alice can be confident that the message really was from Bob, because the signing algorithm is designed so that it is very difficult to forge a signature to match a given message (unless one has knowledge of the private key, which Bob has kept secret).

The problem with such a scheme is that the signature is usually about as big as the message itself. Thus, for efficiency reasons, Bob first applies a cryptographic hash function to the message before signing. This makes the signature much shorter and thus saves time since hashing is generally much faster than signing in implementations. However, if the message digest algorithm is insecure (for example, if it is possible to generate hash collisions), then it might be feasible to forge digital signatures.

To elaborate, suppose Alice and Eve agree to sign a document detailing some financial transactions between them. Also, suppose that Eve is dishonest and is able to produce two documents which are mapped to the same hash value and whose contents differ significantly. That is, suppose Eve was able to find another document with another set of conditions benefiting Eve that has the same hash value as the original financial document. Now, Alice mightbe in agreement with the conditions of the original document. So Eve asks Alice for her digital signature on the original document and what Eve receives is not only a valid signature for the original document but also for the forged document as well. The signature is valid for both messages because the verification process only refers to their common hash value. Eve can now replace one message by the other and claim that Alice signed the second message, the document that unfairly benefits her. Hence, it is important to require hash functions to be collision resistant.

1.2 The Merkle-Damgård Construction for Hash Functions

The Merkle-Damgård method or MD-design principle is a generic method of constructing a cryptographic hash function. A cryptographic hash function must be able to process an arbitrary-length message into a fixed-length output. This can be achieved by breaking the input up into a series of equal-sized blocks, and operating on them in sequence using a *compression function* that processes a fixed-length input into a shorter, fixed-length output, each time combining a block of the input with the output of the previous round.

Definition 1.2.1 (Compression Function) *A compression function is a mapping*

$$g : \{0,1\}^m \text{ x } \{0,1\}^l \rightarrow \{0,1\}^m$$

with $1 \leq m < l$ which can be evaluated efficiently. Here the $\{0,1\}^m$ part of the domain is some fixed parameter IV (initial value) and the compression function g is denoted by g_{IV} (and maps $\{0,1\}^l \rightarrow \{0,1\}^m$).

Typically an input message is padded such that the length of the padded input message is a multiple of l. An algorithm for the Merkle-Damgård construction proceeds as follows:

Given: Compression function $g : \{0,1\}^m \times \{0,1\}^l \rightarrow \{0,1\}^m$;

l-bit constant IV;

Input : Message M;

```
1. Break M into m-bit blocks $M_0, \ldots, M_{k-1}$, padding if necessary;
2. Let $h_0$ = IV;
3. For $i$ = 1 to $k$ let $h_i = g(h_{i-1}, M_{i-1})$;
4. Output $h_k$;
```

Since compression functions can be seen as small hash functions in themselves, it seems natural that the collision resistance of a compression function implies the collision resistance of the hash function. We make this more precise with the following definitions and theorem:

Definition 1.2.2 (Collision of the Compression Function) *A collision of the compression function g consists of an initial value IV and different inputs X and X' such that*

$$g_{IV}(X) = g_{IV}(X')$$

Theorem 1.2.4 ([5]) *Let g be a collision resistantcompression function and h be a hash function, constructed from g by using the MD-design principle. Then h is collision resistant.*

1.3 Structure of the Thesis

It seems intuitively clear that the property of collision resistance is the most important for hash functions and not surprisingly it is the target of most attacks on hash functions. The attack on the MD5 hash function by Xiaoyun Wang proved that MD5 is not collision resistant. In this thesis, we will attempt to explain Wang's attack on MD5. In order to understand the attack on MD5 we will need some background knowledge. In chapter 2 we explain the MD5 algorithm and some of its important characteristics. In chapter 3 we provide a toolbox for the cryptanalysis of MD5. Firstly, theorems pertaining to additions of integers modulo 2^n, bit rotations, and bitwise Boolean functions are presented. These theorems are taken from Magnus Daum's PhD thesis [2]. Most of the theorems will be stated without proof (for proofs see [2]). Secondly, we analyze difference propagation in hash functions. It seems intuitive that in order to understand the attack on the collision resistance of hash functions we would need to study the effect of input differences on the output differences. Particularly, we are interested in studying the conditions under which non-zero input differences produce a zero output difference. Finally, in chapter 4 we attempt to explain Wang's attack on MD5 using the background knowledge from the previous chapters.

Chapter 2: The MD5 Algorithm

The MD5 hash function belongs to a class of hash functions called the MD4-Family. The hash functions of this class use the iteration scheme as dictated by the Merkle-Damgård construction. In Crypto '89, Merkle and Damgård submitted a seminal article on the construction of hash functions using the iteration scheme. Inspired by this article, Rivest proposed the MD4 hash function, a predecessor of MD5, one year later. After cryptanalysis of MD4 revealed certain unexpected properties that raised concerns about its security, Rivest proposed the MD5 hash function in 1992. It incorporated many of the ideas used to design MD4 but with more emphasis on security rather than efficiency. Thus, Wang's attack on MD5 is applicable to MD4 as well. In fact, Wang's method produces collisions in MD4 much more quickly than in MD5.

MD5 processes a variable length message into a fixed-length output of 128 bits. The input message is broken up into chunks of 512-bit blocks. The message is padded so that its length in bits is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64-bit integer representing the length of the original message. Obviously, restricting the length of the message to 64 bits precludes the possibility of processing "arbitrarily long" messages but in practice a message length greater than $2^{64}-1$ is highly unlikely. Hence, for all practical purposes, we can consider the hash function as being able to process messages of arbitrary length. Details about the padding of an input message do not play any role in our explanation of Wang's attack. Thus we will assume that the input message length is a multiple of 512.

The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words (or registers), denoted *a*, *b*, *c* and *d*. As usual, 32-bit words are integers mod 2^{32}. These are initialized to certain fixed constants collectively called IV (or initial value). The main algorithm then operates on each 512-bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed *rounds*; each round is composed of 16 similar operations (or step operations) based on a non-linear function f_i, modular addition, and left rotation. Here $0 \leq i < 64$ denotes the i^{th} step operation.

Recall that MD5 is only one hash function in the class of hash functions called MD4-Family. The non linear boolean functions used for the MD4-Family are:

$$\text{XOR}(X, Y, Z) = X \oplus Y \oplus Z$$

$$\text{MAJ}(X, Y, Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z)$$

$$\text{ITE}(X, Y, Z) = (X \wedge Y) \oplus \overline{(X} \wedge Z)$$

$$\text{ONX}(X, Y, Z) = (X \vee \overline{Y}) \oplus Z$$

Sometimes the functions ITE and ONX are applied with swapped parameters. Thus, for example, we will denote ITE(Z, X, Y) by ITE_{zxy}.

Since MD5 is a member of the MD4-Family, a subset of the functions from the above list is used. A different boolean function, f_i is used in each round of MD5. Note that MAJ is not used for MD5:

$$f_i(X, Y, Z) = \text{ITE} = (X \wedge Y) \vee (\neg X \wedge Z), \quad 0 \leq i \leq 15$$

$$f_i(X, Y, Z) = \text{ITE}_{zxy} = (X \wedge Z) \vee (Y \wedge \neg Z), \quad 16 \leq i \leq 31$$

$f_i(X, Y, Z) = \text{XOR} \quad = X \oplus Y \oplus Z, \qquad 32 \leq i \leq 47$

$f_i(X, Y, Z) = \text{ONX}_{xzy} \quad = Y \oplus (X \vee \neg Z), \qquad 48 \leq i \leq 63$

$\oplus, \wedge, \vee, \neg$ denote the XOR, AND, OR and NOT operations.

Let t_i and s_i denote step dependent constants, the + operator denote addition modulo 2^{32} and « denote the rotational left shift operator. If M is a 512 bit message block, then M = <m_0, m_1,…, m_{15}> where m_k is a 32 bit word. In each round of sixteen step operations, these sixteen 32 bit words are used exactly once and w_i denotes the round dependent permutations of these sixteen 32 bit words that make up the message block. The round dependent permutation is given as follows:

Let k Є {0,1,2,3} indicate the rounds. Then

$w_{16k+i} = m_{i \bmod 16}$, if $k = 0$

$w_{16k+i} = m_{5i+1 \bmod 16}$, if $k = 1$

$w_{16k+i} = m_{3i+5 \bmod 16}$, if $k = 2$

$w_{16k+i} = m_{7i \bmod 16}$, if $k = 3$

Let *[abcd i]* denote the following operation:

$$a = b + (a + f_i(b,c,d) + w_i + t_i)^{\ll s_i}$$

Then the algorithm for the MD5 hash function can be written as follows:

MD5(*x*)

```
external MD5-PAD
global $t_0,\ldots,t_{63}$
global $s_0,\ldots,s_{63}$
$y \leftarrow MD5-PAD(x)$
denote $y = M_1 \| M_2 \| \ldots \| M_n$, where each $M_k$ is a 512-bit block
$a = 0x67452301$
$b = 0xefcdab89$
```

```
c = 0x98badcfe
d = 0x10325476

for each $M_k \in \{M_1,...,M_n\}$
begin
        aa = a
        bb = b
        cc = c
        dd = d

        /*Do the following 16 operations for round 0*/
        [abcd 0]    [dabc 1]    [cdab 2]    [bcda 3]
        [abcd 4]    [dabc 5]    [cdab 6]    [bcda 7]
        [abcd 8]    [dabc 9]    [cdab 10]  [bcda 11]
        [abcd 12]  [dabc 13]  [cdab 14]  [bcda 15]

        /*Do the following 16 operations for round 1*/
        [abcd 16]  [dabc 17]  [cdab 18]  [bcda 19]
        [abcd 20]  [dabc 21]  [cdab 22]  [bcda 23]
        [abcd 24]  [dabc 25]  [cdab 26]  [bcda 27]
        [abcd 28]  [dabc 29]  [cdab 30]  [bcda 31]

        /*Do the following 16 operations for round 2*/
        [abcd 32]  [dabc 33]  [cdab 34]  [bcda 35]
        [abcd 36]  [dabc 37]  [cdab 38]  [bcda 39]
        [abcd 40]  [dabc 41]  [cdab 42]  [bcda 43]
        [abcd 44]  [dabc 45]  [cdab 46]  [bcda 47]

        /*Do the following 16 operations for round 3*/
        [abcd 48]  [dabc 49]  [cdab 50]  [bcda 51]
        [abcd 52]  [dabc 53]  [cdab 54]  [bcda 55]
        [abcd 56]  [dabc 57]  [cdab 58]  [bcda 59]
        [abcd 60]  [dabc 61]  [cdab 62]  [bcda 63]

        a = a + aa
        b = b + bb
        c = c + cc
        d = d + dd
end

return (a, b, c, d)
```

An important part of the step operations in MD5 are the non linear Boolean functions, which are applied bitwise to the registers. These functions have been chosen because:

- They support a strong avalanche effect, which means that small differences in the registers are mapped to large differences in only a few step operations.
- The functions are balanced, which means that $|f^{-1}(0)| = |f^{-1}(1)|$.
- The correlation between a boolean function and an arbitrary linear mapping $\{0,1\}^3 \to \{0,1\}$ is quite small so the boolean function is non linear.
- The Boolean functions produce their output from the bits of X, Y, and Z, in such a manner that if the input bits of X, Y, and Z are independent and unbiased, then the output bit of the corresponding function will be independent and unbiased.

Chapter 3: A Toolbox for Cryptanalysis of MD5

In this chapter we provide the background theory necessary for the cryptanalysis of MD5. In fact, much of the theory presented here is an indispensable tool for understanding other hash functions and block ciphers. After introducing some notation, we will present some theorems on the relationship between modular differences (mod 2^{32}) and xor differences in section 3.1. In 3.2 we will then focus our attention on the relationship between bit rotations and modular addition (mod 2^{32}) and explicitly analyze what happens when we interchange the application of the two operations. To conclude the background theory, we will investigate difference propagation in section 3.3.

3.1 Modular Differences and XOR Differences

Modular addition and xor addition are two of the most important operations used in the design of hash functions. To denote the difference between two 32 bit registers x and x' we have to consider:

XOR difference: $\Delta^{\oplus}x = x \oplus x'$ and Modular difference: $\Delta^{+}x = x - x' \bmod 2^{32}$

We state that all bit positions are indexed from 0. Since a 32 bit number will include many zeroes in its bit representation, and to avoid writing out such lengthy representations, we use the following notation:

$$[i_1,...,i_r] = (x_{n-1},...,x_0) \text{ where } x_{i_1} = ... = x_{i_r} = 1, \qquad \text{and}$$

$x_j = 0$ for all $j \notin \{i_1, ..., i_r\}$.

This means:

$$x = (x_{n-1}, ..., x_0) = 2^{i_1} + 2^{i_2} + ... + 2^{i_r}$$

On occasion, we will not only need to know the modular difference of two bits but also their exact values. To that end we introduce the concept of signed bitwise differences denoted by:

$$\Delta^{\pm} x = \ (x_{n-1} - x'_{n-1}, ..., x_0 - x'_0) \ \ where (x_i - x'_i) \in \{-1, 0, 1\}^n$$

To abbreviate values from $\{-1,0,1\}^{\mathrm{n}}$, we introduce the notation $\overline{i_k}$ to denote

$x_{i_k} = -1$. For example,

$[i_1, \overline{i_2}, \overline{i_3}, i_4] = \Delta^{\pm} x$ where $x_{i_1} = x_{i_4} = 1$, $x_{i_2} = x_{i_3} = 0$, $x'_{i_1} = x'_{i_4} = 0$,

$x'_{i_2} = x'_{i_3} = 1$ and

$x_j - x'_j = 0$ for all $j \notin \{i_1, ..., i_4\}$.

It should be noted that the signed bitwise difference $(\Delta^{\pm})$ has no direct relation to the modular difference (Δ^{+}).

Another piece of notation that will prove to be useful later pertains to how we can express step operations concisely and clearly. In most descriptions of MD5, the registers are labeled *a, b, c, d* and the step operations are defined as in Chapter 2. That is, the algorithm is usually defined in the form “first do F(*a,b,c,d*), then F(*d,a,b,c*), then F(*c,d,a,b*), then F(*b,c,d,a*)…" and so on. In the new notation we use the fact that in every step operation only one register is modified. We denote the content of the register changed in step *i* by R_i. If we initialize $R_{-1}=b$, $R_{-2}=c$, $R_{-3}=d$, $R_{-4}=a$, we denote the step operation in step *i* by

$$R_i = R_{i-1} + (R_{i-4} + f_i(R_{i-1}, R_{i-2}, R_{i-3}) + W_i + T_i)^{\ll s_i} \qquad (3.1.1)$$

Notice that with the above notation we can solve for R_{i-4} and W_i respectively:

$$R_{i-4} = (R_i - R_{i-1})^{\gg s_i} - f_i(R_{i-1},R_{i-2},R_{i-3}) - W_i - T_i \qquad (3.1.2)$$

$$W_i = (R_i - R_{i-1})^{\gg s_i} - f_i(R_{i-1},R_{i-2},R_{i-3}) - T_i - R_{i-4} \qquad (3.1.3)$$

We state the following very important theorem and its corollaries governing the relationship between modular differences, xor differences and signed bitwise differences mostly without proof.

Theorem 3.1.1 ([2]) *Let* $x, x' \in \{0,1\}^n$ *with some fixed signed bitwise difference* $\Delta^{\pm}x$. *Then the* $\oplus$*-difference* $\Delta^{\oplus}x$ *and the modular difference* $\Delta^{+}x$ *are uniquely determined.*

Proof.

This is obvious because knowing the signed bitwise difference directly gives you the values of x and x' in the bit positions that have non zero differences. So computing the xor difference and modular difference for these bit positions is trivial. For bit positions with a signed bitwise difference of zero, the xor difference and modular difference is zero because x and x' have the same values in these bit positions.

Theorem 3.1.2 ([2]) *Let* $x, x' \in \{0,1\}^n$, $0 \le k \le n-1$ *and define*

$$l_0^{(k)} = \max\{0 \le j \le n-k \mid x_i = 0 \; for \; k \le i \le k+j-1\},$$
$$l_1^{(k)} = \max\{0 \le j \le n-k \mid x_i = 1 \; for \; k \le i \le k+j-1\},$$

Then it holds

$$\Delta^{+}x = 2^k \Leftrightarrow \begin{cases} \Delta^{\pm}x = [k+l_0^{(k)}, \overline{k+l_0^{(k)}-1}, \dots, \overline{k}], if\; k+l_0^{(k)} < n, \\ or\; \Delta^{\pm}x = [\overline{n-1}, \dots, \overline{k}], else \end{cases}$$
$$\Delta^{+}x = -2^k \Leftrightarrow \begin{cases} \Delta^{\pm}x = [\overline{k+l_1^{(k)}}, k+l_1^{(k)}-1, \dots, k], if\; k+l_1^{(k)} < n, \\ or \Delta^{\pm}x = [n-1, \dots, k], else. \end{cases}$$

The theorem shows that when transforming a modular difference into an $\oplus$ *-difference*, a modular difference of 2^k can affect more than just the kth bit of the $\oplus$ *-difference* or signed bitwise difference depending on the input bits. That is, for a given modular difference, there can be many XOR differences. For example, when the modular difference $x - x' = 2^6$ for some value *x*, then we have the following possibilities for the XOR difference:

- One bit difference in bit 6, i.e., $\Delta^{\oplus}x = 0x00000040$. This means that bit 6 in *x* is a 1 and bit 6 in x' is 0.
- Two bit difference where a carry is transferred from bit 6 to bit 7, i.e., $\Delta^{\oplus}x = 0x000000c0$. This means that $x_6 = 0, x_7 = 1$ and $x'_6 = 1, x'_7 = 0$.
- Three bit difference where a carry is transferred from bit 6 to bit 7 and then to bit 8, i.e., $\Delta^{\oplus}x = 0x000001c0$. This means that $x_6 = 0, x_7 = 0, x_8 = 1$ and $x'_6 = 1, x'_7 = 1, x'_8 = 0$.
- In general, there can be more carries propagating to further bits and the bit pattern is $x = 1000\ldots$ and $x' = 0111\ldots$

In case the modular difference is -2^6, the XOR difference remains unchanged but the values of x and $x^{'}$ are exchanged.

Corollary 3.1.3 ([2]) $\Delta^+ x = 2^k$ *or* $\Delta^+ x = -2^k \Rightarrow \exists l \geq 0 : \Delta^{\oplus} x = [k+l, ..., k]$.

The next corollary corrects the incorrect probabilities stated in [2].

Corollary 3.1.4 ([2]) *For fixed* $\Delta^+ x = 2^k, 0 \leq l \leq n-k-1$ *and* $x \in \{0,1\}^n$ *chosen uniformly at random*

$$\Pr(\Delta^{\pm} x = [k+l, \overline{k+l-1}, ..., \overline{k}]) = 2^{-(l+1)},$$
$$\Pr(\Delta^{\pm} x = [\overline{n-1}, ..., \overline{k}]) = 2^{-(n-k)}.$$

For fixed $\Delta^+ x = -2^k$ *the following probabilities hold:*

$$\Pr(\Delta^{\pm} x = [\overline{k+l}, k+l-1, ..., k]) = 2^{-(l+1)},$$
$$\Pr(\Delta^{\pm} x = [n-1, ..., k]) = 2^{-(n-k)}.$$

Thus in both cases we have

$$\Pr(\Delta^{\oplus} x = [k+l, ..., k]) = 2^{-(l+1)},$$
$$\Pr(\Delta^{\oplus} x = [n-1, ..., k]) = 2^{-(n-k)}$$

Proof.

For $\Delta^+ x = 2^k$,

$\Pr(\Delta^{\pm} x = [k+l, \overline{k+l-1}, ..., \overline{k}]) = \Pr(x_{k+l} = 1, x_{k+l-1} = ... = x_k = 0) = 2^{-(l+1)}$ and

$\Pr(\Delta^{\pm} x = [\overline{n-1}, ..., \overline{k}]) = \Pr(x_{n-1} = ... = x_k = 0) = 2^{-(n-k)}$

For $\Delta^+ x = -2^k$,

$\Pr(\Delta^{\pm} x = [\overline{k+l}, k+l-1, ..., k]) = \Pr(x_{k+l} = 0, x_{k+l-1} = ... = x_k = 1) = 2^{-(l+l)}$ and

$\Pr(\Delta^{\pm} x = [n-1, ..., k]) = \Pr(x_{n-1} = ... = x_k = 1) = 2^{-(n-k)}$.

We can also consider more complicated modular differences like $\Delta^{+}x = 2^{m_0} - 2^{m_1}, m_0 > m_1$. Here the signed bitwise difference is of the form

$$\Delta^{\pm}x = [m_0 + l_0, \overline{m_0 + l_0 - 1}, ..., \overline{m_0}, \overline{m_1 + l_1}, m_1 + l_1 - 1, ..., m_1]$$

as long as $m_0 > m_1 + l_1$, where $l_i = l_i^{(k_i)}, i = 0,1,$ are defined as in Theorem 3.1.2. As a concrete example, consider the modular difference $\Delta^{+}x = -1 - 2^{6} + 2^{23} - 2^{27}$ with a corresponding signed bitwise difference of the form

$$\Delta^{\pm}x = [0,1,2,3,4,\overline{5},6,7,8,9,10,\overline{11},\overline{23},\overline{24},\overline{25},26,27,28,29,30,\overline{31}].$$

3.2 Modular Addition and Bit Rotation

We start by defining the following notation

$$A = [A_l \,|_i\, A_r]$$

which means that for $A = (a_{n-1}, ..., a_0)$ we have $A_l = (a_{n-1}, ..., a_i)$ and $A_r = (a_{i-1}, ..., a_0)$.

We also define an indicator function by **1**, where

$$\mathbf{1}(x) = \begin{Bmatrix} 1, if\ x\ is\ true, \\ \\ 0, if\ x\ is\ false, \end{Bmatrix}$$

We assume that $0 < k < n$ and A and B are two integers such that $0 \le A, B < 2^n$ and

$$A = [A_l \,|_{n-k}\, A_r] \text{ and } B = [B_l \,|_{n-k}\, B_r].$$

Using this notation we can now state some important lemmas and theorems:

Lemma 3.2.1 ([2])

$$A^{«k} = [A_r \,|_k\, A_l],$$

$$A + B = [A_l + B_l + c_r^+ \,|_{n-k}\, A_r + B_r],$$

$$A - B = [A_l - B_l - c_r^- \,|_{n-k}\, A_r - B_r],$$

where

$$c_r^+ = \mathbf{1}(A_r +_{¢} B_r \geq 2^{n-k}),$$
$$c_r^- = \mathbf{1}(A_r < B_r)$$

are the carry bits coming from the right half of the computation.

With the knowledge of Lemma 3.2.1 we can deduce the following theorems which describe the error that occurs when we modify equations by reversing the order of addition and bit rotation:

Theorem 3.2.2 ([2])

$$(A+B)^{«k} - (A^{«k}+B^{«k}) = [-c^+ \,|_k\, c_r^+],$$

where

$$c^+ = \mathbf{1}(A +_{¢} B \geq 2^n),$$
$$c_r^+ = \mathbf{1}(A_r +_{¢} B_r \geq 2^{n-k})$$

are the carry bits from the full and right side additions respectively.

Theorem 3.2.3 ([2])

$$(A - B)^{«k} - (A^{«k} - B^{«k}) = c^- 2^k - c_r^-$$

where

$$c^- = \mathbf{1}(A < B),$$
$$c_r^- = \mathbf{1}(A_r < B_r)$$

are the carry bits coming from the full and right side subtractions respectively.

Theorem 3.2.4 ([2]) *Let $P_{\alpha,\beta}$ (with $\alpha, \beta \in \{0,1\}$) be the probability that*

$$(A+B)^{\lll k} - (A^{\lll k}+B^{\lll k}) = [-\alpha \,|_k\, \beta].$$

1. If we suppose A to be fixed and B to be chosen uniformly at random, then

$$P_{0,0} = 2^{-n}(2^{n-k} - A_r)(2^k - A_l)$$

2. *If we suppose A and B to be chosen independently and uniformly at random, then*

$$P_{0,0} = (1 + 2^{-(n-k)} + 2^{-k} + 2^{-n})/4$$

Theorem 3.2.5 ([2]) *Let $P_{\alpha,\beta}$ (with $\alpha, \beta \in \{0,1\}$) be the probability that*

$$(A-B)^{\lll k} - (A^{\lll k} - B^{\lll k}) = \alpha 2^k - \beta.$$

1. If we suppose A to be fixed and B to be chosen uniformly at random, then

$$P_{0,0} = 2^{-n}(A_r + 1)(A_l + 1)$$

2. If we suppose A to be chosen uniformly at random and B to be fixed, then

$$P_{0,0} = 2^{-n}(2^k - B_l)(2^{n-k} - B_r)$$

3. If we suppose A and B to be chosen independently and uniformly at random, then

$$P_{0,0} = (1 + 2^{-(n-k)} + 2^{-k} + 2^{-n})/4$$

The most important lesson to extract from the last two theorems is that if A and B are chosen uniformly at random then the most probable difference is zero. For example, we will often find it simpler to replace ($A^{\lll k}$-$B^{\lll k}$) with $(A\text{-}B)^{\lll k}$. Why? Because if A and B are chosen uniformly at random then the probability that the difference between the quantities ($A^{\lll k}$-$B^{\lll k}$) and $(A\text{-}B)^{\lll k}$ is zero is given by case 3 of Theorem

3.2.5. This also turns to be the most likely difference. This is important in our cryptanalysis because it enables us to simplify equations.

3.3 Difference Propagation

Before we directly dive into the subject of difference propagation in hash functions, it is worthwhile to mention a few sentences on the measurement of avalanche effect in hash functions. Because the ideal of a cryptographic hash function is to behave like a random function, a hash function is designed so that it has a strong avalanche effect. This means that an average of one half of the output bits should change whenever a single input bit is complemented. The avalanche factor tries to mathematically abstract the desirable property of high nonlinearity between input and output bits, and specifies that the hashes of messages from a close neighborhood in the domain are dispersed over the whole range. Another property of random functions and thus desired of good hash functions is *completeness*. Completeness is defined as the fact that every output bit depends on all the input bits, and not a proper subset of them. The concept of completeness and the avalanche effect can be combined to define what is called the *strict avalanche criterion*. A cryptographic hash function satisfies the strict avalanche criterion when each output bit changes with a probability of $\frac{1}{2}$ whenever a single input bit is complemented. There are ways to measure the strength of the avalanche factor of hash functions. Though MD5 demonstrates good avalanche effect, it can be empirically shown that it behaves far from a random function ([2]).

Recall that in section 3.1 we introduced a notation to express step operations. That is we denoted the content of the register changed after step operation i by R_i and the formula for a step operation was given by equation 3.1.1. We also showed that the step operations can be reversed (cf. equation 3.1.2, 3.1.3), i.e. we can go backwards through all the steps by computing R_{i-4} from $R_{i-3}, R_{i-2}, R_{i-1}$, and R_i. Typically, the focus of the attacks on the collision resistance property of hash functions is on the difference between register values generated by the different messages rather than on the actual register values themselves. In other words, consider two different messages resulting in input words W_i and $W_i^{'}$ in the i^{th} step operation. Denote the computed register values by R_i and $R_i^{'}$ respectively. Then we are generally interested in $\Delta^{+} R_i$ and how this difference propagates in the computation of successive step operations. It turns out that in Wang's collision finding scheme this difference has a structural pattern which can be exploited to break MD5.

It can be shown that MD5 has a much stronger avalanche effect in the forward direction than in the reverse direction ([2]). Although we are merely speculating, it seems intuitive that this effect can be exploited, when looking for a differential pattern, because it is much easier to control small differences when computing backwards than forwards. The notion of a differential pattern will be properly explained in the next chapter. However, for now it will suffice to know that a differential pattern provides some sort of structure to the output differential of each step operation by explicitly stating what that modular and xor differential must be after each step operation. The attack will then succeed with a high probability if we can find two different messages whose output differential (modular and xor) after each step operation matches the differential pattern. No one really knows how Wang computed the differential pattern but we can make some intelligent guesses as to how she arrived at the pattern. Thus what follows is purely speculation. In chapter 4 we will give another speculative method by which Wang might have computed the differential pattern.

Note that in order for a collision to occur we must have $\Delta^+ R_{60} = \Delta^+ R_{61} = \Delta^+ R_{62} = \Delta^+ R_{63} = 0$. Then using equation 3.1.2, we can approximately compute:

$$\Delta^+ R_{59} = (\Delta^+ R_{63} - \Delta^+ R_{62})^{\gg s_{63}} - f_{63}(\Delta^+ R_{62}, \Delta^+ R_{61}, \Delta^+ R_{61}) - \Delta^+ W_{63}.$$

Roughly speaking, we can continue in this fashion and build a differential pattern bottom up that needs to be satisfied in order for a collision to occur. Because of the weak avalanche effect of MD5 in the reverse direction, we may approximately compute the differential pattern in the reverse direction as well.

To be more concrete, we start by fixing some modular differences $\Delta^+ R_{i-3}, ..., \Delta^+ R_i$ for register values and $\Delta^+ W_i$ for input word value after step *i*. Strictly speaking, using equation 3.1.2 to compute $\Delta^+ R_i$ as above isn't mathematically correct. However, Corollary 3.1.4 and Theorem 3.2.5 will tell us with what probability using such a reformulation is correct. It turns out that this probability is sufficiently high. For example, if we have a fixed difference $\Delta^+ R = 2^t$ for registers R and R', and let s denote the number of bits to rotate by, then by the notation used in section 3.2 we have,

$$2^t = \left\{ \begin{array}{l} [0\,|_{n-s}\, 2^t], if\ t < n-s, \\ \\ [2^{t+s-n}\,|_{n-s}\, 0], if\ t \ge n-s \end{array} \right\}$$

and using Theorem 3.2.5, we obtain the following corollary.

Corollary 3.3.1 ([2]) *Let* $0 < s < n$ *and* $\Delta^+ R = 2^t$ *with* $0 \le t < n$, *and denote* $\Delta^+(R^{\lll s}) = R^{\lll s} - R'^{\lll s}$. *Then for R chosen uniformly at random*

$$\Pr(\Delta^+(R^{\lll s}) = 2^{t+s}) = 1 - 2^{t+s-n}, if\ t < n-s,$$
$$\Pr(\Delta^+(R^{\lll s}) = 2^{t+s-n}) = 1 - 2^{t-n}, if\ t \ge n-s.$$

Thus as long as $(n-t)$ or $(n-s)-t$ is not too small, rotating the difference is a good approximation to the difference of the rotated values. This is the reason that we can replace $\Delta^+((R_i - R_{i-1})^{\ggg s_i})$ with $(\Delta^+ R_i - \Delta^+ R_{i-1})^{\ggg s_i}$ in the reformulation to compute $\Delta^+ R_i$.

Analyzing the differential resulting from the boolean functions is a little more complicated. When a modular difference $\Delta^+ R_i \ne 0$ is used in the boolean function, we need to make assumptions about the signed bitwise difference $\Delta^{\pm} R_i$. By Corollary 3.1.4, for a modular difference of 2^k or -2^k, the most likely (with probability $\frac{1}{2}$) signed bitwise difference is $[k]$ or $[\overline{k}]$ respectively. We will use this fact and information from Table 3.1 in the following example to illustrate how the differential pattern is computed.

Example 3.3.1

Suppose we want to have a collision appearing in step 25 of the compression function of MD5. This means that we must have

$$\Delta^+R_{25} = \ldots = \Delta^+R_{22} = 0.$$

We proceed by computing the differential pattern backwards as follows:

Step 25:

$$\Delta^+R_{21} = (\Delta^+R_{25} - \Delta^+R_{24})^{\gg 9} - f_{25}(\Delta^+R_{24}, \Delta^+R_{23}, \Delta^+R_{22}) - \Delta^+W_{25}.$$

Given that $\Delta^+R_{25} = \ldots = \Delta^+R_{22} = 0$, we have $\Delta^+R_{21} = -\Delta^+W_{25}$.

Here we can introduce an input difference, say, $\Delta^+W_{25} = 2^9$ and thus we have established $\Delta^+R_{21} = -2^9$ as part of the differential pattern.

Step 24:

$$\Delta^+R_{20} = (\Delta^+R_{24} - \Delta^+R_{23})^{\gg 5} - f_{24}(\Delta^+R_{23}, \Delta^+R_{22}, \Delta^+R_{21}) - \Delta^+W_{24}.$$

Because $\Delta^+R_{24} = \Delta^+R_{23} = 0$, we only need to concern ourselves with $\Delta^+R_{21} = -2^9$ in the boolean function. First of all, we need to make an assumption on the signed bitwise difference $\Delta^{\pm}R_{21}$. As stated earlier, by Corollary 3.1.5, $\Delta^{\pm}R_{21} = [\overline{9}] = -2^9$ with probability $\frac{1}{2}$. Since $f_{24} = ITE_{zxy}$, from Table 3.1, we see that the 9th bit of f_{24}:

$$f_{24}(\Delta^+R_{23}, \Delta^+R_{22}, \Delta^+R_{21}) = ITE_{zxy}(\Delta^+R_{23}, \Delta^+R_{22}, \Delta^+R_{21}) = ITE_{zxy}(0, 0, -1) = x - 1.$$

Because this depends on the actual value of x, the 9th bit of f_{24} is zero with probability $\frac{1}{2}$. Now, choosing not to introduce an input difference in Δ^+W_{24}, we see that with probability $\frac{1}{2}$ we have $\Delta^+R_{20} = 0$.

Why do we choose not to introduce an input difference in Δ^+W_{24}? Because minimal input differences causes minimal output differences. Since our goal is maximize our chance of finding different inputs that map to the same output, we only introduce an input difference when necessary. The inspiration for this idea comes from Hans Dobbertin's attack on the compression function of MD5 [8].

Step 23:

$\Delta^+R_{19} = (\Delta^+R_{23} - \Delta^+R_{22})^{\gg 20} - f_{23}(\Delta^+R_{22}, \Delta^+R_{21}, \Delta^+R_{20}) - \Delta^+W_{23}$.

Again, $\Delta^+R_{21} = -2^9$ appears in the boolean function. Since $f_{23} = ITE_{zxy}$, from Table 3.1, we see that the 9th bit of f_{23}:

$f_{23}(\Delta^+R_{22}, \Delta^+R_{21}, \Delta^+R_{20}) = ITE_{zxy}(\Delta^+R_{22}, \Delta^+R_{21}, \Delta^+R_{20}) = ITE_{zxy}(0, -1, 0) = -x.$

Because this depends on the actual value of x, the 9th bit of f_{23} is zero with probability $\frac{1}{2}$. Now, choosing not to introduce an input difference in Δ^+W_{23}, we see that with probability $\frac{1}{2}$ we have $\Delta^+R_{19} = 0$.

Step 22:

$\Delta^+R_{18} = (\Delta^+R_{22} - \Delta^+R_{21})^{\gg 14} - f_{22}(\Delta^+R_{21}, \Delta^+R_{20}, \Delta^+R_{19}) - \Delta^+W_{22}$.

Since $\Delta^+R_{21} = -2^9$ appears in the rotation part of the equation, we use Corollary 3.3.1 with $t = 9$, $s = 14$, $n = 32$, to deduce that

$(\Delta^+R_{22} - \Delta^+R_{21})^{?\,14} = (0-(-2^9))^{?\,14} = (2^9)^{?\,14} = 2^{27}$ with probability $1-2^{-9}$.

As in step 23, we see that the 9th bit of f_{22} is zero with probability $\frac{1}{2}$. We again do not introduce an input difference for Δ^+W_{22} and we have $\Delta^+R_{18} = 2^{27}$ as part of the differential pattern.

We can continue in this fashion and determine differential characteristics for all 64 steps. This concludes our treatment of difference propagation and consequently our presentation of the background knowledge necessary to understand Wang's attack.

$\Delta^{\pm}x$	$\Delta^{\pm}y$	$\Delta^{\pm}z$	$\Delta^{\pm}$XOR	$\Delta^{\pm}$ITE	$\Delta^{\pm}$MAJ	$\Delta^{\pm}$ONX
0	0	0	0	0	0	0
0	0	1	$1-2(x\oplus y)$	$1-x$	$x\oplus y$	$-1-2(x-1)y$
0	0	-1	$2(x\oplus y)-1$	$x-1$	$-(x\oplus y)$	$1+2(x-1)y$
0	1	0	$1-2(x\oplus z)$	x	$x\oplus z$	$(1-x)(2z-1)$
0	1	1	0	1	1	$-x$
0	1	-1	0	$2x-1$	0	x
0	-1	0	$2(x\oplus z)-1$	$-x$	$-(x\oplus z)$	$(1-x)(1-2z)$
0	-1	1	0	$1-2x$	0	$-x$
0	-1	-1	0	-1	-1	x
1	0	0	$1-2(y\oplus z)$	$y-z$	$y\oplus z$	$y(1-2z)$
1	0	1	0	y	1	$y-1$
1	0	-1	0	$y-1$	0	$1-y$
1	1	0	0	$1-z$	1	0
1	1	1	1	1	1	-1
1	1	-1	-1	0	1	1
1	-1	0	0	$-z$	0	$1-2z$
1	-1	1	-1	0	1	0
1	-1	-1	1	-1	-1	0
-1	0	0	$2(y\oplus z)-1$	$z-y$	$-(y\oplus z)$	$y(2z-1)$
-1	0	1	0	$1-y$	0	$y-1$
-1	0	-1	0	$-y$	-1	$1-y$
-1	1	0	0	z	0	$2z-1$
-1	1	1	-1	1	1	0
-1	1	-1	1	0	-1	0
-1	-1	0	0	$z-1$	-1	0
-1	-1	1	1	0	-1	-1
-1	-1	-1	-1	-1	-1	1

Table 3.1: Propagation of *signed* bitwise differences.

Chapter 4: Wang's Attack on MD4

In this chapter we will discuss Wang's attack on MD5 in detail. First we present a high level view of the attack by giving the general algorithm and further dissect the algorithm in the following sections.

The objective of the attack is to find two colliding 1024-bit messages. Let $M = (M_0, M_1)$ be 1024-bit message such that $|M_0| = |M_1| = 512$. Also define

$$\delta_0 = (0,0,0,0,2^{31},0,0,0,0,0,0,2^{15},0,0,2^{31},0)$$

$$\delta_1 = (0,0,0,0,2^{31},0,0,0,0,0,0,-2^{15},0,0,2^{31},0),$$

and let $M^{'} = (M_0^{'}, M_1^{'})$ be another 1024-bit message such that $M_0^{'} = M_0 + \delta_0$ and $M_1^{'} = M_1 + \delta_1$. If M and $M^{'}$ are colliding messages then MD5 (M) = MD5 ($M^{'}$). Note that the message block differential index starts at 0 and 0th word differential starts at the left. For example, for δ_0 the fourth word differential is 2^{31} and the eleventh word differential is 2^{15}.

Let $\Delta^{+} R_i = R_i - R_i^{'}$. From the previous chapter, we know that $\Delta^{+} R_i$ and $\Delta^{\oplus} R_i$ represent the modular difference and xor difference respectively of the output after the i^{th} step operation. In Wang's attack these differences must attain certain pre-specified values after each step operation called a *differential characteristic* (cf. Table 4.1 below). For example, in the case of modular differences, $\Delta^{+} R_3 = 0$ whereas $\Delta^{+} R_4 = 2^6$. Recall from section 3.1 that as a consequence of Theorem 3.1.3, a particular modular differential does not determine a unique xor differential. In addition, given a nonzero xor differential alone for the i^{th} step operation, we cannot determine the exact bit values for $R_i^{'}$ and R_i. In Wang's method, xor differences after each step operation are implicitly given by specifying values of certain bits of R_i and $R_i^{'}$. To show this, we need another notation. All bit positions are indexed from 0:

$X^{'} = X[i_1, -i_2, -i_3, i_4, i_5]$ denotes that

$X_{i_1} = X_{i_4} = X_{i_5} = 0$ and $X_{i_2} = X_{i_3} = 1$ whereas,

$X_{i_1}^{'} = X_{i_4}^{'} = X_{i_5}^{'} = 1$ and $X_{i_2}^{'} = X_{i_3}^{'} = 0$ and,

$X_i^{'} = X_i$ for all $i \notin (i_1, i_2, i_3, i_4, i_5)$.

For example, in the 4th step operation, $R_4^{'} = R_4[6, ..., 21, -22]$, which means that for R_4 bit 6 ($R_{i,6}$) to bit 21 ($R_{i,21}$) are set to 0, and bit 22 ($R_{i,22}$) is set to 1, whereas for $R_4^{'}$ bit 6 to bit 21 are set to 1, and bit 22 is set to 0. All other bits are the same for R_4 and $R_4^{'}$. Thus we know both the xor difference and modular difference of R_4 and $R_4^{'}$. The collection of these differential characteristics is collectively called the *differential pattern*. Since we are processing 1024-bit messages each consisting of two 512-bit blocks, the differential pattern is composed of 128 differential characteristics; 64 for the first block and 64 for the second block. Table 4.1 gives the differential characteristics for the first block. These tables were copied from [1] with modifications to reflect our notation. The first column denotes the step operation number, the second column denotes the output value of the step operation, the third column denotes the message word for M_0 in each step, the fourth column denotes the shift rotation, the fifth column denotes the modular difference in the message word, the sixth column denotes the modular difference in the step operation output, and the seventh column denotes the step operation output for $M_0^{'}$. The empty items and the unlisted steps have zero differences in the fifth and sixth columns.

The idea of the attack is to find two messages such that their step operation differentials match the differential pattern laid out by Wang, which consequently results in a collision. Furthermore, Wang computes what are called necessary conditions for the differential characteristics to hold with improved probability. These necessary conditions dictate the actual values in certain bit positions of the output of a step operation. For example, to ensure that the differential characteristics in step *i* are satisfied with high probability, the necessary conditions will require that certain bit values of $R_{i-4},...,R_i$ are set to 1 and certain other bits are set to 0. For example, the necessary conditions for the differential characteristics to hold in step 16 are $R_{16,3} = R_{15,3}, R_{16,17} = R_{15,15}, R_{16,17} = 0, R_{16,31} = 0$. As a matter of fact, the seventh column of Table 4.1 also gives some of the necessary conditions because they explicitly provide the bit values of the step operation output. Obviously given a randomly chosen 1024-bit message M and a second message $M^{'}$ computed as above, it is more likely that the all the differential characteristics are not satisfied and thus yields no collision. However, Wang uses message modification techniques to ensure that the necessary conditions are met, which in turn ensures that the differential characteristics are satisfied with high probability. Given this the algorithm for finding a collision in MD5 proceeds as follows:

1. Repeat the following steps until all first block differential characteristics are satisfied:

(i) Select a random 512-bit block M_0.

(ii) Use message modification techniques on M_0 to ensure that most of the necessary conditions for the first block differential pattern are met.

(iii) Let $M_0^{'} = M_0 + \delta_0$ and apply the compression function to check that the step operation differentials satisfy the differential characteristics for the first block as laid out in Table 4.1.

2. Repeat the following steps until a collision is found:

(i) Select a random 512-bit block M_1.

(ii) Use message modification techniques on M_1 to ensure that most of the necessary conditions for the second block differential pattern are met.

(iii) Let $M_1^{'} = M_1 + \delta_1$ and apply the compression function (with the state variables determined from the output of the first block) to check if there is a collision.

Notice that we insist that the message modification techniques result in satisfying only *most* of the necessary conditions. It is natural to ask: why not require that the message modification techniques result in satisfying *all* the necessary conditions? The answer is that it is computationally inefficient to satisfy all conditions because there aren't fast message modification algorithms to do so. Instead, it is computationally more efficient to run the algorithm probabilistically by using clever message modification techniques to satisfy as many conditions as possible. There are also many other questions to be answered here. Given the differential pattern, how are the necessary conditions determined? How are the initial message block differentials δ_0 and δ_1 chosen? How is the differential pattern chosen? What are message modification techniques? All these questions will be answered in the following sections with concrete examples.

Step	The output in the i^{th} step of M_0	W_i	S_i	$\Delta^+ W_i$	$\Delta^+ R_i$	The output in the i^{th} Step for $M_0^{'}$
3	R_3	m_3	22			
4	R_4	m_4	7	-2^{31}	2^6	$R_4[6,...,21,-22]$
5	R_5	m_5	12		$2^6-2^{23}-2^{31}$	$R_5[-6,23,31]$
6	R_6	m_6	17		$1+2^6-2^{23}+2^{27}$	$R_6[6,7,8,9,10,-11,-23,-24,-25,26,27,28,29,30,31,0,1,2,3,4,-5]$
7	R_7	m_7	22		$-1+2^{15}+2^{17}+2^{23}$	$R_7[0,15,-16,17,18,19,-20,-23]$
8	R_8	m_8	7		$-1+2^6-2^{31}$	$R_8[-0,1,6,7,-8,-31]$
9	R_9	m_9	12		$-2^{12}-2^{31}$	$R_9[-12,13,31]$
10	R_{10}	m_{10}	17		$-2^{30}-2^{31}$	$R_{10}[30,31]$
11	R_{11}	m_{11}	22	-2^{15}	$2^7+2^{13}-2^{31}$	$R_{11}[7,-8,13,...,18,-19,31]$
12	R_{12}	m_{12}	7		$-2^{24}-2^{31}$	$R_{12}[-24,25,31]$
13	R_{13}	m_{13}	12		-2^{31}	$R_{13}[31]$
14	R_{14}	m_{14}	17	-2^{31}	$-2^3+2^{15}-2^{31}$	$R_{14}[3,-15,31]$
15	R_{15}	m_{15}	22		$2^{29}-2^{31}$	$R_{15}[-29,31]$
16	R_{16}	m_1	5		-2^{31}	$R_{16}[31]$
17	R_{17}	m_6	9		-2^{31}	$R_{17}[31]$
18	R_{18}	m_{11}	14	-2^{15}	$-2^{17}-2^{31}$	$R_{18}[17,31]$
19	R_{19}	m_0	20		-2^{31}	$R_{19}[31]$
20	R_{20}	m_5	5		-2^{31}	$R_{20}[31]$
21	R_{21}	m_{10}	9		-2^{31}	$R_{21}[31]$

22	R_{22}	m_{15}	14			R_{22}
23	R_{23}	m_4	20	-2^{31}		R_{23}
24	R_{24}	m_9	5			R_{24}
25	R_{25}	m_{14}	9	-2^{31}		R_{25}
26	R_{26}	m_3	14			R_{26}
…	…	…	…	…	…	…
33	R_{33}	m_8	11			R_{33}
34	R_{34}	m_{11}	16	-2^{15}	-2^{31}	$R_{34}[31]$
35	R_{35}	m_{14}	23	-2^{31}	-2^{31}	$R_{35}[31]$
36	R_{36}	m_1	4		-2^{31}	$R_{36}[31]$
37	R_{37}	m_4	11	-2^{31}	-2^{31}	$R_{37}[31]$
38	R_{38}	m_7	16		-2^{31}	$R_{38}[31]$
…	…	…	..	…	…	…
44	R_{44}	m_9	4		-2^{31}	$R_{44}[31]$
45	R_{45}	m_{12}	11		-2^{31}	$R_{45}[31]$
46	R_{46}	m_{15}	16		-2^{31}	$R_{46}[31]$
47	R_{47}	m_2	23		-2^{31}	$R_{47}[31]$
48	R_{48}	m_0	6		-2^{31}	$R_{48}[31]$
49	R_{49}	m_7	10		-2^{31}	$R_{49}[-31]$
50	R_{50}	m_{14}	15	-2^{31}	-2^{31}	$R_{50}[31]$
51	R_{51}	m_5	21		-2^{31}	$R_{51}[-31]$
…	….	…	…	…	…	…
57	R_{57}	m_{15}	10		-2^{31}	$R_{57}[-31]$
58	R_{58}	m_6	15		-2^{31}	$R_{58}[31]$
59	R_{59}	m_{13}	21		-2^{31}	$R_{59}[31]$
60	$R_{60}=R_{60}+R_{-4}$	m_4	6	-2^{31}	-2^{31}	$R_{60}[31]$
61	$R_{61}=R_{61}+R_{-3}$	m_{11}	10	-2^{15}	-2^{31}	$R_{61}[25,31]$
62	$R_{62}=R_{62}+R_{-2}$	m_2	15		-2^{31}	$R_{62}[-25,26,31]$
63	$R_{63}=R_{63}+R_{-1}$	m_9	21		-2^{31}	$R_{63}[25,-31]$

Table 4.1. The differential characteristics for the first block

4.1 Computing Necessary Conditions for the Differential Characteristics

The best way to explain this is to do an example. The following example is from [3]. Let us determine the necessary conditions for the differential characteristic in step 4 for the first message block (cf. Table 4.1). Typically the step operation of the *i^th^* step is given by equation 3.1.1. We can rewrite the step operation as follows:

$$Q_i = f_i(R_{i-1}, R_{i-2}, R_{i-3}) + R_{i-4} + W_i + T_i$$

$$R_i = R_{i-1} + (Q_i^{= s_i})$$

We can reformulate the modular differential: $\Delta^+ R_i = \Delta^+ R_{i-1} + ((Q_i)^{= s_i} - (Q_i^{'})^{= s_i})$ by using Corollary 3.2.5 to produce:

$$\Delta^+ R_i = \Delta^+ R_{i-1} + ((\Delta^+ Q_i)^{= s_i}), \text{ where}$$

$$\Delta^+ Q_i = \Delta^+ f_i(R_{i-1}, R_{i-2}, R_{i-3}) + \Delta^+ R_{i-4} + \Delta^+ W_i \qquad \textit{(4.1.1)}$$

Though it is not indicated in Tables 4.1, $\Delta^+ f_i$ is an important intermediate step in computing $\Delta^+ R_i$. However Hawkes et al. [3], provide this information and we will use it to compute the necessary conditions. It isn't hard to arrive at the values for $\Delta^+ f_i$, given that we have equation 4.1.1. In our example, in order to have $\Delta^+ R_5 = 2^6 - 2^{23} - 2^{31}$ given that we also have $\Delta^+ R_4 = 2^6$ and $s_5 = 12$, it is easy to see that $\Delta^+ Q_5 = -2^{19} - 2^{11}$, which implies that $\Delta^+ f_5 = -2^{19} - 2^{11}$ because $\Delta^+ R_1 = \Delta^+ W_5 = 0$. We are given $\Delta^+ R_2 = 0, \Delta^+ R_3 = 0, \Delta^+ R_4 = 2^6$, and we want $\Delta^+ f_5 = -2^{19} - 2^{11}$.

- *Obtaining the correct $\Delta^{\oplus} R_4$:*

Because $\Delta^+ R_2 = \Delta^+ R_3 = 0$, the only way to obtain $\Delta^+ f_5 = -2^{19} - 2^{11}$ is to have $\Delta^{\oplus} R_4$ propagate into higher order bits via carries. This should be clear because

$\Delta^+ R_4 = 2^6$ is the only non zero differential we have to work with in f_5. This means that we have $R_{4,k} = 0$ for $6 \le k \le 19$ and $R_{4,20} = 1$ (cf. Theorem 3.1.2). Wang's attack dictates that $R_{4,k} = 0$ for $6 \le k \le 21$ and $R_{4,22} = 1$, although $R_{4,k} = 0$ for $6 \le k \le 19$ and $R_{4,20} = 1$ is more probable. However, we shall be satisfied with Wang's conditions. Thus so far we have the conditions:

$$R_{4,6} = R_{4,7} = \ldots = R_{4,21} = 0 \text{ and } R_{4,22} = 1.$$

- *Obtaining the correct $\Delta^+ f_5$:*

We want to figure out the fewest conditions necessary to obtain $\Delta^+ f_5 = -2^{19} - 2^{11}$. To do this we make use of what we obtained previously. Specifically, we have $\Delta^{\pm} R_{4,11} = \Delta^{\pm} R_{4,19} = -1$ (Note the use of signed bitwise difference). Thus, somehow we need to arrive at $\Delta^{\pm} f_{5,11} = \Delta^{\pm} f_{5,19} = -1$.

We have two cases to consider:

(i) *Zero value bits of $\Delta^{\oplus} R_4$:*

For this case we consider $\Delta^{\oplus} R_{4,k} = 0$ for $0 \le k \le 5$ and $23 \le k \le 31$. The Boolean function is $f_5(R_4, R_3, R_2) = (R_4 \wedge R_3) \vee (\neg R_4 \wedge R_2)$. Observe that this function evaluates according to the following rule:

If ($R_{4,k} = 1$) then "output $R_{3,k}$" else "output $R_{2,k}$".

That is why this function is called the "If then Else" function and consequently abbreviated as ITE. Thus we have,

$-$Select $f_{5,k} = R_{3,k}$ and $f'_{5,k} = R'_{3,k}$ when $R_{4,k} = R'_{4,k} = 1$, or

$-$Select $f_{5,k} = R_{2,k}$ and $f'_{5,k} = R'_{2,k}$ when $R_{4,k} = R'_{4,k} = 0$.

Since $\Delta^{+}R_2 = \Delta^{+}R_3 = 0$ and consequently $\Delta^{\oplus}R_2 = \Delta^{\oplus}R_3 = 0$, we see that $\Delta^{+}f_5 = \Delta^{\oplus}f_5 = 0$ for these bits and no conditions are required.

(ii) *Nonzero value bits of* $\Delta^{\oplus}R_4$ *:*

For $6 \le k \le 21$,

We have $R_{4,k} = 0$ and $R^{'}_{4,k} = 1$, which implies that

$f_{5,k} = R_{2,k}$ and $f^{'}_{5,k} = R^{'}_{3,k}$.

For $k = 21, k = 20, 12 \le k \le 18, 6 \le k \le 10,$ we require that $f_{5,k} = f^{'}_{5,k}$,

which implies that $R_{2,k} = R^{'}_{3,k} = R_{3,k}$.

For $k = 19$ and $k = 11$, we want $\Delta^{\pm}f_{5,k} = -1$ which implies

that $R^{'}_{3,k} - R_{2,k} = R_{3,k} - R_{2,k} = 1$. Here the minus sign represents signed bitwise subtraction. This results in $R_{3,k} = 1$ and $R_{2,k} = 0$.

For $k = 22$,

We have $R_{4,k} = 1$ and $R^{'}_{4,k} = 0$, which implies that

$f_{5,k} = R_{3,k}$ and $f^{'}_{5,k} = R^{'}_{2,k}$.

Thus the conditions obtained for the 4th step operation are:

$R_{4,k} = 0$ for $6 \le k \le 21$,

$R_{4,k} = 1$ for $k = 22$,

$R_{3,k} = 1$ for $k = 11$ and $k = 19$,

$R_{2,k} = 0$ for $k = 11$ and $k = 19$,

$R_{3,k} = R_{2,k}$ for $6 \le k \le 10, 12 \le k \le 18, k = 20, k = 21$.

4.2 *The Message Block Differentials and Differential Pattern*

The calculation of the differential pattern and message block differentials are the heart of Wang's attack on MD5, and not coincidentally the differential pattern and message block differentials are shrouded in mystery. However, we can make intelligent conjectures as to how the differential pattern was computed, and how the message block differentials were chosen. Recall that in section 3.3, we speculated on how to compute the differential pattern by taking advantage of the weak avalanche factor of MD5 in the reverse direction. In this section we present yet another idea on how to compute the differential pattern. We also present an idea that could potentially explain how the message block differentials are chosen. Both the ideas and the illustrated example are due to [4]. Before we delve into the illustration, a remark is in order. Wang states that the message block differentials are picked so that the differential pattern in round 3 and round 4 of MD5 for each message block are satisfied with high probability. Given what we know now about difference propagation, this claim should not seem unreasonable. The differential pattern in Wang's attack is there to ensure that a collision will take place. So by the time the step output differentials have arrived at round 3, further propagation of these step output and message block differentials had better satisfy the differential characteristics in round 3 and round 4 if we are to have a good chance at arriving at a collision. Roughly speaking then we would expect to work backwards to arrive at a differential pattern for round 1 and round 2 similar to example 3.3.1.

Because Wang states that the message block differentials are picked so that the third and fourth round differential characteristics are satisfied with high probability, we will start by analyzing the differential pattern in these rounds. We begin by making the following important observations which apply to both the first and second blocks:

- For the step operations in round 3 and round 4 that have nonzero modular difference, these modular differences are exactly $-2^{31} = 2^{31}$ (cf. Table 4.1).
- The modular difference in the last few steps of round 2 and the first few steps of round 3 is zero (cf. Table 4.1).
- The Boolean function used in round 3 is $f_i = XOR$ for $32 \le i \le 47$. This is important because f_i is a linear function, i.e. $f_i(x \oplus u, y \oplus v, z \oplus w) = f_i(x, y, z) \oplus f_i(u, v, w)$. Observe that any change in a particular bit position of any *one* of the three input words necessarily results in a change of the output in that same bit position.

Let us just consider the differential pattern for the first block. So how is the modular difference of 2^{31} propagated through round 3 and round 4? To answer this we make use of the facts that the differential is zero in the last few steps of round 2 and the first few steps of round 3, and the Boolean function is linear. The first bit difference in round 3 is introduced in step 34 due to the message block difference.

<u>Step 34:</u>

We have $R'_{34} = R'_{33} + (R'_{30} + f_{34}(R'_{33}, R'_{32}, R'_{31}) + W'_{34} + T_{34})^{\lll 16}$.

Because the differential in the last few steps of round 2 and round 3 is zero, and $W'_{34} = m'_{11} = m_{11} + 2^{15}$, we can write:

$R'_{34} = R_{33} + (R_{30} + f_i(R_{33}, R_{32}, R_{31}) + m_{11} + 2^{15} + T_{34})^{\lll 16}$.

From Theorem 3.2.4, we can infer that $R_{34}^{'} = R_{34} + (2^{15})^{\lll 16} = R_{34} + 2^{31}$ as stated in Table 4.1.

<u>*Step 35:*</u>

Here $W_{35}^{'} = m_{14}^{'} = m_{14} + 2^{31}$.

So we have $R_{35}^{'} = R_{34}^{'} + (R_{31} + f_{35}(R_{34}^{'}, R_{33}, R_{32}) + m_{14} + 2^{31} + T_{35})^{\lll 23}$.

Substituting $R_{34} + 2^{31}$ for $R_{34}^{'}$ we obtain,

$$R_{35}^{'} = R_{34} + 2^{31} + (R_{31} + f_{35}(R_{34} + 2^{31}, R_{33}, R_{32}) + m_{14} + 2^{31} + T_{35})^{\lll 23}.$$

Using the linearity property of f_{35} we get,

$$R_{35}^{'} = R_{34} + 2^{31} + (R_{31} + f_{35}(R_{34}, R_{33}, R_{32}) + 2^{31} + m_{14} + 2^{31} + T_{35})^{\lll 23}$$

$$= 2^{31} + R_{34} + (R_{31} + f_{35}(R_{34}, R_{33}, R_{32}) + m_{14} + T_{35})^{\lll 23} = R_{35} + 2^{31}.$$

<u>*Step 36:*</u>

Here no difference is introduced by the message word. Thus making substitutions and using the linearity property of the Boolean function, we get:

$$R_{36}^{'} = R_{35}^{'} + (R_{32} + f_{36}(R_{35}^{'}, R_{34}^{'}, R_{33}) + W_{36} + T_{36})^{\lll 4}$$

$$= R_{35} + 2^{31} + (R_{32} + f_{36}(R_{35} + 2^{31}, R_{34} + 2^{31}, R_{33}) + W_{36} + T_{36})^{\lll 4}$$

$$= R_{35} + 2^{31} + (R_{32} + f_{36}(R_{35}, R_{34}, R_{33}) + (2^{31} + 2^{31}) + W_{36} + T_{36})^{\lll 4}$$

$$= R_{35} + 2^{31} + (R_{32} + f_{36}(R_{35}, R_{34}, R_{33}) + W_{36} + T_{36})^{\lll 4} = R_{36} + 2^{31}.$$

<u>*Step 37:*</u>

Here a message word difference is introduced. That is, $W_{37}^{'} = m_{4}^{'} = m_{4} + 2^{31}$.

$$R'_{37} = R'_{36} + (R_{33} + f_{37}(R'_{36}, R'_{35}, R'_{34}) + m_4 + 2^{31} + T_{37})^{\lll 11}$$

$$= R_{36} + 2^{31} + (R_{33} + f_{37}(R_{36} + 2^{31}, R_{35} + 2^{31}, R_{34} + 2^{31}) + m_4 + 2^{31} + T_{37})^{\lll 11}$$

$$= R_{36} + 2^{31} + (R_{33} + f_{37}(R_{36}, R_{35}, R_{34}) + (2^{31} + 2^{31} + 2^{31}) + m_4 + 2^{31} + T_{37})^{\lll 11}$$

$$= R_{36} + 2^{31} + (R_{33} + f_{37}(R_{36}, R_{35}, R_{34}) + m_4 + T_{37})^{\lll 11} = R_{37} + 2^{31}.$$

Since no message block differentials are used for the rest of the steps in round 3, it is easy to see how this differential of 2^{31} propagates down to these steps. How this differential propagates through round 4 is not well known. However, we can make an educated guess about how this phenomenon might occur. Let us look at Table 4.2 which is due to [4]. This table tells how changes in the input of the Boolean function used in the fourth round affect the output of the function. Here we can see cases where changes in the input do not change the output value and this phenomenon is not uncommon. For example, for $ONX(x, y, z)$ where $x = 0, y = 1, z = 0$, we can see that flipping x does not change the output. Thus it seems possible that this absorbing quality of ONX contributes to the propagation of the differential.

x	y	z	$\Delta x \Rightarrow \Delta F$	$\Delta y \Rightarrow \Delta F$	$\Delta z \Rightarrow \Delta F$
0	0	0		X	X
0	0	1	X	X	X
0	1	0		X	X
0	1	1	X	X	X
1	0	0		X	
1	0	1	X	X	
1	1	0		X	
1	1	1	X	X	

Table 4.2. Output differences for F $= f_i$ $48 \le i < 64$

From our analysis above, it seems that the following algorithm might be used to determine the message block differential and the step operation output differential pattern:

1. Assume that message block differentials can be introduced such that the step operation output differential for the first few steps of round 3 is zero.
2. Let *i* denote the first step operation in round 3 for which it is decided that the step operation output differential should be nonzero. In Wang's method, this was step 34. Since this differential must be 2^{31}, and must also propagate down to the successive steps as illustrated in our example above, set $W_i' = W_i + 2^{31-s_i}$, $W_{i+1}' = W_{i+1} + 2^{31}$, $W_{i+3}' = W_{i+3} + 2^{31}$. Then, without introducing any further message block differences, the step operation output differential of 2^{31} should propagate to the rest of the step operations in round 3. Similar to Example 3.3.1, we introduce as few input differences as possible to minimize complications.
3. Using the message differential chosen in the previous step, find a differential pattern in the first and second rounds such that the step operation output differential for the last four steps of the second round is zero. To understand how this might be possible, let us look at Table 4.3 and Table 4.4 which are due to [4]. These tables tell us about the correlation between changes in the input and output of the Boolean functions used in the first and second rounds respectively. Again for these functions, it is common that changes in the input do not translate into changes in the output. Thus it seems plausible that the

absorbing quality of these Boolean functions could aid in searching for such a differential pattern in the first and second round.

x	y	z	$\Delta x \Rightarrow \Delta F$	$\Delta y \Rightarrow \Delta F$	$\Delta z \Rightarrow \Delta F$
0	0	0			X
0	0	1	X		X
0	1	0	X		X
0	1	1			X
1	0	0		X	
1	0	1	X	X	
1	1	0	X	X	
1	1	1		X	

Table 4.3. Output differences for F = f_i $0 \leq i < 16$

x	y	z	$\Delta x \Rightarrow \Delta F$	$\Delta y \Rightarrow \Delta F$	$\Delta z \Rightarrow \Delta F$
0	0	0		X	
0	0	1	X		
0	1	0		X	X
0	1	1	X		X
1	0	0		X	X
1	0	1	X		X
1	1	0		X	
1	1	1	X		

Table 4.4. Output differences for F = f_i $16 \leq i < 32$

In conclusion we stress, yet again, that the analysis given in this section is pure speculation and Wang has yet to publish a thorough exposition of her method of computing the message block differentials and the differential characteristics of the step operation outputs.

4.3 Message Modification

In section 4.1 we illustrated how to compute the necessary conditions for the differential pattern to hold. It turns out that most of the conditions are for the step operations in the first round and this proves to be a very important characteristic of the structure of the differential pattern. For example, when processing the first block, there are a total of 290 conditions and most of them pertain to the first round of the compression function. When we pick a random message for the first block, as dictated by the algorithm in the beginning of this chapter, it is more likely that we will have to modify it to satisfy the conditions for the first block. By having most of the conditions in the first round, modifying the message block does not affect previous computations because there are no previous computations. For instance, modifying the 32-bit word m_1 of the message block to meet the conditions in step 1(round 1) is easier than having to modify m_1 to meet the conditions in step 16 (round 2). This is because a modification in round 2 affects the step operation differentials computed in round 1. Thus we will have to go back to round 1 and make adjustments.

There are two types of message modification techniques. The message modification technique used to modify messages to satisfy the necessary conditions of the first round is called *single message modification*. In order to improve the probability of satisfying the differential pattern in the second round, messages will need to be modified to satisfy the conditions in this round. The message modification technique used for the second round is called *multi message modification*. We will now explain both message modification techniques with illustrations that are due to [4].

The idea behind single message modification is quite simple. We simply execute the following steps until the differential characteristics of the first round have been satisfied:

a. Pick random values for $R_0,...,R_{15}$, and flip their bits until all the conditions of round 1 are met.
b. Compute the message words using equation 3.1.3. That is, evaluate $m_i = W_i = (R_i - R_{i-1})^{? s_i} - f_i(R_{i-1}, R_{i-2}, R_{i-3}) - T_i - R_{i-4}$.

We iterate the fact that these conditions are "necessary" and not "sufficient" as incorrectly stated by Wang. Thus satisfying the conditions doesn't guarantee that the differential pattern in round 1 will automatically hold. We might need to run the single message modification technique until the first round differential pattern holds. The time complexity required to satisfy the differential pattern in the first round is much less than the overall collision finding algorithm. Thus we are guaranteed to satisfy the differential characteristics of the first round.

The best way to explain the multi message modification technique is through examples. We will illustrate a simple example first and then illustrate a slightly more complex example to communicate the gist of the technique.

Example 4.3.1

Suppose that in the 16th step operation (first step of round 2), we have $R_{16,31} = 1$. Then as dictated by the necessary conditions, we will need to correct this bit to $R_{16,31} = 0$ by modifying the message word, W_{16}, used in the step operation. Since $W_{16} = m_1$, and m_1 is used in the 1st step operation (second step operation in round

1), modifying m_1 could change R_1, which consequently could propagate through out the steps in round 1. Then the differential characteristics of round 1 would no longer be satisfied.

We first begin by modifying m_1 by adding 2^{26} to it. That is, $m_1^{new} = m_1 + 2^{26}$. This corrects $R_{16,31}$ by flipping the 1 to 0. How? Observe that the shift amount in the 16th step is 5. Thus adding 2^{26} to the message word is tantamount to adding $(2^{26})^{= 5} = 2^{31}$ to the step operation output R_{16}, which results in flipping its most significant bit. In order to correct the change to R_1, we recomputed R_1 with the new m_1 as follows:

$$R_1^{new} = R_0 + (R_{-3} + f_1(R_0, R_{-1}, R_{-2}) + m_1^{new} + T_1)^{= 12}.$$

To ensure none of the other successive step operations in round 1 are modified, we recomputed $m_2, m_3, m_4,$ and m_5 to absorb the change to made to R_1. After the 5th step operation R_1^{new} has been completely absorbed. Thus we have,

$$m_2^{new} = (R_2 - R_1^{new})^{?\ 17} - R_{-2} - f_2(R_1^{new}, R_0, R_{-1}) - T_2$$

$$m_3^{new} = (R_3 - R_2)^{?\ 22} - R_{-1} - f_3(R_2, R_1^{new}, R_0) - T_3$$

$$m_4^{new} = (R_4 - R_3)^{?\ 7} - R_0 - f_4(R_3, R_2, R_1^{new}) - T_4$$

$$m_5^{new} = (R_5 - R_4)^{?\ 12} - R_1^{new} - f_5(R_4, R_3, R_2) - T_5$$

We have now corrected $R_{16,31}$ and ensured that the first round differential characteristics are satisfied as well. This method of computing new message words for successive steps to absorb the effect of R_1^{new} is due to [7].

Example 4.3.2

In this example we show how to satisfy the conditions in steps 16, 17, 18 and 19. This technique is originally due to [9]. There is a total of 10 conditions:

<u>*Step 16:*</u>

$R_{16,3} = R_{15,3}, R_{16,16} = R_{15,15}, R_{16,17} = 0, R_{16,31} = 0.$

<u>*Step 17:*</u>

$R_{17,17} = 1, R_{17,29} = R_{16,29}, R_{17,31} = 0.$

<u>*Step 18:*</u>

$R_{18,17} = 0, R_{18,31} = 0.$

<u>*Step 19:*</u>

$R_{20,31} = 0.$

The algorithm to satisfy these 10 conditions as written in [4] is:

1. Choose $R_2, ..., R_{15}$ such that they satisfy the conditions in the first round.
2. For $6 \le i \le 15$, compute m_i:

   $$m_i = (R_i - R_{i-1})^{?\, s_i} - f_i(R_{i-1}, R_{i-2}, R_{i-3}) - R_{i-4} - T_i.$$

3. Pick a random value for R_{16} with its conditions satisfied, and compute

   $$R_{17} = R_{16} + (R_{13} + f_{17}(R_{16}, R_{15}, R_{14}) + W_{17} + T_{17})^{?\, 9}$$

   $$R_{18} = R_{17} + (R_{14} + f_{18}(R_{17}, R_{16}, R_{15}) + W_{18} + T_{18})^{?\, 14}$$

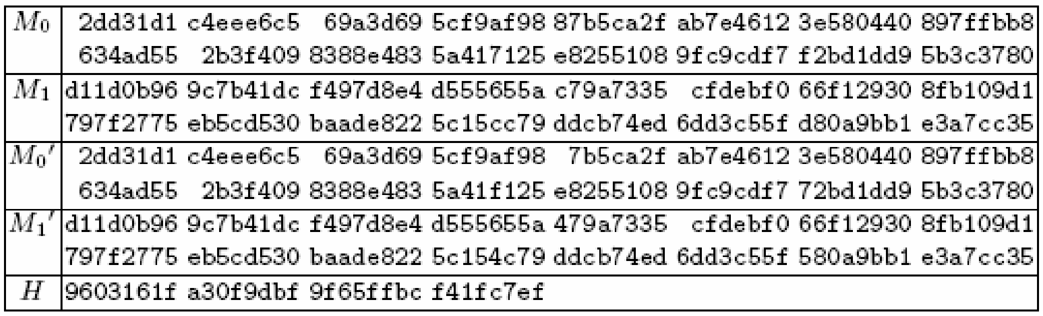
   until conditions for R_{17} and R_{18} are satisfied. Since there is only a total of 9 conditions for R_{16}, R_{17}, and R_{18}, it can be done quickly. However it may be the case that no value of R_{16} satisfies the conditions for R_{17} and R_{18} because

R_{17} and R_{18} are functions of R_{13}, R_{14}, and R_{15} as well, which might prove to be too inflexible. In this case, we will have to pick new values for $R_2,...,R_{15}$ and start the algorithm all over again. If the conditions for $R_{16},...,R_{18}$ are satisfied then proceed to step 4 otherwise go to step 1.

4. Pick a random value for R_{19} with its only condition fulfilled, and compute $W_{19} = m_0 = (R_{19} - R_{18})^{?\ 20} - R_{15} - f_{19}(R_{18}, R_{17}, R_{16}) - T_{19}$.
5. Compute R_0 from the new value of m_0 computed in the previous step.
6. Compute $W_{16} = m_1$ by rearranging the formula for the 16th step operation.
7. Compute R_1 from the new value of m_1 computed in the previous step.
8. To absorb the new value of R_1, compute $m_2,...,m_5$:

$$\begin{aligned}&\mathit{for}\ 2 \le i \le 5,\\ &m_i = (R_i - R_{i-1})^{?\ s_i} - R_{i-4} - f_i(R_{i-1}, R_{i-2}, R_{i-3}) - T_i\end{aligned}$$

After computing step 8, 33 conditions in the remaining step operation remain unsatisfied. Thus we should expect to pick 2^{33} messages for the first block before all the differential characteristics for the first block are met. This means that the time complexity of the overall collision finding algorithm can be improved by clever algorithms to perform multi message modification techniques and to satisfy more second round conditions. Since there isn't a general method to satisfy second round differentials, the algorithms are quite specialized and invented from having clever insights into how the differentials propagate. Actually Wang does not perform the complex multi message modification as detailed in example 4.3.2 which improves the complexity of the collision finding algorithm. Instead she opts for the multi message modification similar to the one in example 4.3.1. Thus she is left with 37 unsatisfied conditions for the first block and consequently her time complexity to find the first block is 2^{39} MD5 operations. Similarly, 30 conditions are left unsatisfied for the second block and her time complexity to find the second block is 2^{32} MD5 operations. In conclusion, the goal of the message modification techniques is to satisfy as many of the conditions as possible in the first and second round. This will then drastically increase the probability of satisfying the differential characteristics which consequently increases the collision probability. We give a collision found by Wang, where H is the hash value:



M_0	2dd31d1	c4eee6c5	69a3d69	5cf9af98	87b5ca2f	ab7e4612	3e580440	897ffbb8
	634ad55	2b3f409	8388e483	5a417125	e8255108	9fc9cdf7	f2bd1dd9	5b3c3780
M_1	d11d0b96	9c7b41dc	f497d8e4	d555655a	c79a7335	cfdebf0	66f12930	8fb109d1
	797f2775	eb5cd530	baade822	5c15cc79	ddcb74ed	6dd3c55f	d80a9bb1	e3a7cc35
M_0'	2dd31d1	c4eee6c5	69a3d69	5cf9af98	7b5ca2f	ab7e4612	3e580440	897ffbb8
	634ad55	2b3f409	8388e483	5a41f125	e8255108	9fc9cdf7	72bd1dd9	5b3c3780
M_1'	d11d0b96	9c7b41dc	f497d8e4	d555655a	479a7335	cfdebf0	66f12930	8fb109d1
	797f2775	eb5cd530	baade822	5c154c79	ddcb74ed	6dd3c55f	580a9bb1	e3a7cc35
H	9603161f	a30f9dbf	9f65ffbc	f41fc7ef				

Bibliography

[1] WANG, X., AND YU, H. How to break MD5 and other hash functions, Eurocrypt 2005, LNCS 3494, pp. 19-35, 2005.

[2] DAUM, M., Cryptanalysis of hash function of the MD4-family, PhD thesis, Ruhr-University of Bochum, May 2005. http://www.cits.ruhr-uni-bochum.de/imperia/md/content/magnus/dissmd4.pdf.

[3] HAWKES, P., PADDON, M., AND ROSE, G.G. Musings on the Wang et al. MD5 collisions, October 2004. http://iacr.eprint.org/2004/264.

[4] BLACK, J., COCHRAN, M., HIGHLAND, T., A study of the MD5 attacks: insights and improvements, FSE 2006. http://www.cs.colorado.edu/~jrblack/papers/md5e-full.pdf.

[5] STINSON, D., Cryptography: Theory and Practice, Second Edition, pp. 129-130.

[6] RIVEST, R., The MD5 Message Digest Algorithm, Request for Comments (RFC 1320), Internet Activities Board, Internet Privacy Task Force, 1992.

[7] CHAUBAUD, F., JOUX, A., Differential Collisions in SHA-0, Crypto 1998, LNCS 1462, pp. 56-71, 1998.

[8] DOBBERTIN, H., Cryptanalysis of MD5 Compress, Presented at rump session of Eurocrypt 1996.

[9] KLIMA, V., Finding MD5 collisions on a notebook using multi-message modification technique. In International Scientific Conference Security and Protection of Information (May 2005).