Could The 1-MSB Input Difference Be The Fastest Collision Attack For MD5 ?

Tao Xie+ FanBao Liu DengGuo Feng
The State Key Laboratory on Information Security, Chinese Academy of Science, Beijing
The Center for Soft-Computing and Cryptology, NUDT, Changsha, China
(hamishxie@vip.sina.com)

Abstract: So far, two different 2-block collision differentials, both with 3-bit input differences for MD5, have been found by Wang etc in 2005 and Xie etc in 2008 respectively, and those differentials have been improved later on to generate a collision respectively within around one minute and half an hour on a desktop PC. Are there more collision differentials for MD5? Can a more efficient algorithm be developed to find MD5 collisions? In this paper, we list the whole set of 1-bit to 3-bit input difference patterns that are possibly qualified to construct a feasible collision differential, and from which a new collision differential with only 1-MSB input difference is then analyzed in detail, finally the performances are compared with the prior two 3-bit collision attacks according to seven criteria proposed in this paper. In our approach, a two-block message is still needed to produce a collision, the first block being only one MSB different while the second block remains the same. Although the differential path appears to be computationally infeasible, most of the conditions that a collision differential path must satisfy can be fulfilled by multi-step modifications, and the collision searching efficiency can be much improved further by a specific divide-and-conquer technique, which transforms a multiplicative accumulation of the computational complexities into an addition by properly grouping of the conditional bits. In particular, a tunneling-like technique is applied to enhance the attack algorithm by introducing some additional conditions. As a result, the fastest attack algorithm is obtained with an averaged computational complexity of $2^{23.1}$ MD5 operations, which implies that it is able to search a collision on average in one second on a 2.66 Ghz Pentium4 PC for arbitrary random initial values. With a reasonable probability a collision can be found within milliseconds, allowing for instancing an attack during the execution of a practical protocol.
Key Words: MD5, Collision Attack, Collision Differentials, Differential Path.

1 Introduction

A hash function is a cryptographic primitive which outputs a fixed size message digest given a message of arbitrary size. The output value is used usually as the digital digest of the input message, so ideally a single bit flip in the input would cause averagely a half of the digest bits to change. Therefore, a cryptographic hash function is essentially a type of irreversible one-way function built with nonlinear operations. MD2, MD4 and MD5 are such hash functions that were developed in the early1990's by Ron Rivest at MIT for RSA Data Security. A description of these hash functions can be found in RSA Laboratories Technical Report TR-101 .

This paper mainly focuses on collision attacks on MD5. While it is postulated in RFC [1]that the difficulty of coming up with two messages having the same message digest is on the order of 2^{64} operations, research on collision attacks have never stopped since the publication of MD5. In 1992, Berson[2] showed that using differential cryptanalysis, it is possible in reasonable time to find two messages that produce the same digest for a single-round MD5. In 1993, Den Boer and Bosselaer[3] found pseudo-collisions for the compression function of MD5 with different initial values and common input. In 1996, Dobbertin [4] constructed collisions of the MD5 compression function, that is, MD5 collisions with a wrong initial value. In 2004, Wang et al. [5,6] succeeded in producing real collisions for the full MD5 hash function as well as collisions in a number of other hash functions including MD4, RIPEMD, and HAVAL-128.This new idea in their approach was to look for a collision after processing not one but two blocks of the message. Again at 2005 CRYPTO conference, Wang et al. [7] detailed the applications of their methods to the hash functions SHA0 and SHA1, with a generated collision for SHA0, and a description on how to obtain collisions in SHA1. Given the variety of hash functions efficiently attacked by Wang et al, it therefore seems worthwhile to seek a complete understanding of how this approach works, how much can it be improved, and whether it can be generalized.

Basically, Wang's differential collision attack is a hybrid differential cryptanalysis which takes advantages of both the modular difference and the XOR difference together. Wang et al have found a full two-block collision differential with its full differential path, which is computationally feasible, and for the first time constructed a real collision for MD5. Wang 's attack on MD5 has called its security especially in digital signature into question. Since the publication of [6], quite a number of researchers have worked on the optimization of the differential path and the set of sufficient conditions and hence the collision searching algorithms, resulted in a great improvement on the collision searching efficiency to $2^{24.8}$ MD5 operations as declared in [8], implying that a collision can be found in around one minute on a desktop PC. Practical attacks on real protocols and applications based on MD4 family functions have continuously been developed by different applications of Wang's collision. By using an *if-then-else* programming structure, two different Postscript files were created with the same MD5 digest to result in different texts when screening [9], and this attack was extended to other file formats in [10] . By Using Wang's approach to find a near-collision for different IVs and further using different differential paths to absorb the remaining difference, a pair of colliding X.509 certificates for two different distinguished name was found with the same MD5 digest [11]. Other applications of Wang's collision have been proposed to attack HMAC with several hash functions in [12,13].

To date, however, the method used by Wang et al. has been fairly difficult to grasp, and furthermore, the lack of some technical details and some small perhaps deliberately made errors (bugs) in the literature [6], might have constituted the appeal to have frustrated other cryptanalysts to grasp their technique. What is really inexplicable consists in that, no new two-block collision differentials have been found to be more efficient since Wang's paper [6], and it seems to remain a supernatural work to find a feasible collision differential and widely considered to be rely on one's experience and intuition. In 2007, a 1-bit input difference was used to construct a new collision attack [14], with the computational complexity of 2^{42} MD5 operations, but no details are known. In the same year, however, we have found another second 3-bit collision differential, which resulted in an initial collision searching algorithm with the computational complexity of 2^{36} MD5 operations [15] and an improvement has been made later on to reduce the computational complexity to 2^{30} MD5 operations [16]. The authors of this paper believe that there must exist other collision differentials that are more efficient than Wang's. In this paper, a whole set of 1-bit to 3-bit input differences is provided for the first time, one of the 1-MSB input differences is then analyzed in detail by presenting its full differential path and sufficient conditions. Based on these sufficient conditions, finally a collision searching algorithm is developed with an averaged computational complexity of $2^{23.1}$ MD5 operations, which is currently the fastest, implying that a collision can be found within around one second on a desktop PC.

The rest of this paper is organized as follows: In section 2, the definitions for the XOR difference, the modular difference as well as the signed difference are given, some properties especially with respect to the differential path design and extra condition derivation are presented. In section 3, the basic principle on how to find collision differentials is described, and a whole set of 1-bit to 3-bit input difference patterns is presented for the first time, in which the two published MD5 collision differentials are included. In section 4, some general and basic principles for differential path design are described, the basic condition derivation rules implicit in the auxiliary functions are presented, a new 2-block collision differential with only 1-MSB input difference is presented with the design of its full differential path, some extra conditions for preventing unexpected modular differences are also derived, and a specific divide-and-conquer technique is proposed to reduce the computational complexity. In section 5, a divide-and-conquer based collision searching algorithm is specialized for the 1-MSB collision differential, and a tunnel-like technique is applied to enhance the algorithm by introducing some additional conditions. Finally, in section 6, some evaluation criteria on collision differentials are given, and based on these criteria a comparison is made among the three collision differentials, and some suggestions for future research on hash collision attacks are given. In

Appendix A, a concise description of the MD5 algorithm is given to help understand this paper.

2 Some Properties of the Signed Difference

In this paper, let $<<<^s$ denote a left rotation of a word by s bits, '+' denote an addition modulo 2^{32}, $||$denote a concatenation operation, LSB and MSB denote the least and the most significant bit of a word respectively.

Let F_2 be the binary field, F_2^n be an *n*-dimensional vector space over F_2. Let $\mathbf{X}, \mathbf{X}^{\bullet} \in F_2^n$, $\Delta^{\oplus}\mathbf{X}$ denote the bitwise XOR difference between $\mathbf{X}$ and $\mathbf{X}^{\bullet}$, and is called the *XOR difference*, $\Delta^{+}\mathbf{X}$ denote the modular integer subtraction between $\mathbf{X}$ and $\mathbf{X}^{\bullet}$, and is called the *modular difference*, and $\Delta^{\pm}\mathbf{X}$ denote the bitwise difference between $\mathbf{X}$ and $\mathbf{X}^{\bullet}$, and is called the *signed difference*. For example, let $n = 10$, $\mathbf{X}$ =1001000101, $\mathbf{X}^{\bullet}$ =0000111010, then $\Delta^{\oplus}\mathbf{X}$, $\Delta^{+}\mathbf{X}$ and $\Delta^{\pm}\mathbf{X}$ are computed as follows:

$$\Delta^{\oplus}\mathbf{X} = \mathbf{X} \oplus \mathbf{X}^{\bullet} = \mathop{||}_{i=0}^{n-1} \mathbf{X}_i \oplus \mathbf{X}_i^{\bullet} = 1001111111;$$

$$\Delta^{+}\mathbf{X} = \left(\mathbf{X} - \mathbf{X}^{\bullet}\right)\mathrm{mod}\left(2^n\right) = \left(\sum_{i=0}^{n-1}\mathbf{X}_i 2^i - \sum_{i=0}^{n-1}\mathbf{X}_i^{\bullet} 2^i\right)\mathrm{mod}\left(2^n\right) = 1000001011;$$

$$\Delta^{\pm}\mathbf{X} = \mathop{||}_{i=0}^{n-1}\left(\mathbf{X}_i - \mathbf{X}_i^{\bullet}\right) = 1001\text{-}1\text{-}1\text{-}11\text{-}11.$$

For the sake of simplicity, we omit those 0's in the signed difference $\Delta^{\pm}\mathbf{X}$, and index the signed difference bits (+1 or -1) with their position identity instead, starting from 0 (the LSB) in $\mathbf{X}$. Using a 10-bit word as an example, the signed difference (1001-1-1-11-11) can be indexed as [9,6,-5,-4,-3,2,-1,0] (hope this notation is not confused with the reference citation).

Theorem 1 [17]. Let $\mathbf{X}, \mathbf{X}^{\bullet} \in F_2^n$ with some fixed signed difference $\Delta^{\pm}\mathbf{X}$, then their XOR difference $\Delta^{\oplus}\mathbf{X}$ and modular difference $\Delta^{+}\mathbf{X}$ are uniquely determined.

Proof: A more intuitive proof than that in [17] is given here. By the definitions of XOR difference $\Delta^{\oplus}\mathbf{X}$ and the signed difference $\Delta^{\pm}\mathbf{X}$ between $\mathbf{X}, \mathbf{X}^{\bullet} \in F_2^n$, we have

$\Delta^{\oplus}\mathbf{X} = \mathop{||}_{i=0}^{n-1}\left(\mathbf{X}_i \oplus \mathbf{X}_i^{\bullet}\right) = \mathop{||}_{i=0}^{n-1}\left|\mathbf{X}_i - \mathbf{X}_i^{\bullet}\right| = \mathop{||}_{i=0}^{n-1}\left|\Delta^{\pm}\mathbf{X}_i\right|$, i.e. the XOR difference $\Delta^{\oplus}\mathbf{X}$ is uniquely determined by $\Delta^{\pm}\mathbf{X}$. For each $\Delta^{\pm}\mathbf{X}_i = \mathbf{X}_i - \mathbf{X}_i^{\bullet}$, $\Delta^{\pm}\mathbf{X}_i$ has three possible values 0, 1 and -1.

When $\Delta^{\pm}\mathbf{X}_i$ =0, we have $\mathbf{X}_i = \mathbf{X}_i^{\bullet}$, which contributes nothing to the modular difference $\Delta^{+}\mathbf{X}$.

When $\Delta^{\pm}\mathbf{X}_i$ =1, we have $\mathbf{X}_i = 1$ and $\mathbf{X}_i^{\bullet} = 0$, which contributes 2^i to $\Delta^{+}\mathbf{X}$.

When $\Delta^{\pm}\mathbf{X}_i$ =-1, we have $\mathbf{X}_i = 0$ and $\mathbf{X}_i^{\bullet} = 1$, which contributes -2^i to $\Delta^{+}\mathbf{X}$.

Therefore, we have

$$\Delta^{\pm}\mathbf{X} = \mathop{||}_{i=0}^{n-1}\left(\mathbf{X}_i - \mathbf{X}_i^{\bullet}\right) = \mathop{||}_{i=0}^{n-1}\Delta^{\pm}\mathbf{X}_i \overset{\mathrm{mod}(2^n)}{\equiv} \sum_{i=0}^{n-1}\Delta^{\pm}\mathbf{X}_i 2^i \,\mathrm{mod}\left(2^n\right) = \Delta^{+}\mathbf{X}.$$

That means, $\Delta^{+}\mathbf{X}$ is uniquely determined by $\Delta^{\pm}\mathbf{X}$. □

Theorem 2. Let $\mathbf{X}, \mathbf{X}^{\bullet} \in F_2^n$, then the modular difference $\Delta^{+}\mathbf{X}$ is equivalent to the signed difference $\Delta^{\pm}\mathbf{X}$ in modulo 2^n, i.e. the $\Delta^{\pm}\mathbf{X}$ is the signed difference representation of the $\Delta^{+}\mathbf{X}$

with the corresponding XOR difference $\Delta^{\oplus}\mathbf{X}$. More precisely, we have

$$\Delta^{+}\mathbf{X}=\left(\mathbf{X}-\mathbf{X}^{\bullet}\right)\bmod\left(2^{n}\right)=\sum_{i=0}^{n-1}\left(\mathbf{X}_{i}-\mathbf{X}_{i}^{\bullet}\right)2^{i}\bmod\left(2^{n}\right)\overset{\bmod\left(2^{n}\right)}{\equiv}\Big\|_{i=0}^{n-1}\left(\mathbf{X}_{i}-\mathbf{X}_{i}^{\bullet}\right)=\Delta^{\pm}\mathbf{X}.$$

Proof: It is easy to verify the following deduction:

$$\Delta^{+}\mathbf{X}=\left(\mathbf{X}-\mathbf{X}^{\bullet}\right)\bmod\left(2^{n}\right)=\left(\sum_{i=0}^{n-1}\mathbf{X}_{i}2^{i}-\sum_{i=0}^{n-1}\mathbf{X}_{i}^{\bullet}2^{i}\right)\bmod\left(2^{n}\right)$$

$$=\sum_{i=0}^{n-1}\left(\mathbf{X}_{i}-\mathbf{X}_{i}^{\bullet}\right)2^{i}\bmod\left(2^{n}\right)=\sum_{i=0}^{n-1}\Delta^{\pm}\mathbf{X}_{i}2^{i}\bmod\left(2^{n}\right)\overset{\bmod\left(2^{n}\right)}{\equiv}\Delta^{\pm}\mathbf{X}$$

Thus, theorem 2 is proved! □

Theorem 1 and 2 reveal that a bijective mapping does exist between the signed difference $\Delta^{\pm}\mathbf{X}$ and the XOR difference $\Delta^{\oplus}\mathbf{X}$ plus the modular difference $\Delta^{+}\mathbf{X}$, for any $\mathbf{X},\mathbf{X}^{\bullet}\in F_{2}^{n}$.

Proposition 3. Given a modular difference 2^{k} between $\mathbf{X},\mathbf{X}^{\bullet}\in F_{2}^{n}$, there exist $n-k+1$ signed differences that match it; similarly, given the modular difference -2^{k} between $\mathbf{X},\mathbf{X}^{\bullet}\in F_{2}^{n}$, there also exist $n-k+1$ signed differences that match it.

Proof: Consecutively, we can simply have the equivalent transformations as follows:

$$2^{k}=2^{k+1}-2^{k}=2^{k+2}-2^{k+1}-2^{k}=\cdots=2^{n-1}-2^{n-2}-\cdots-2^{k}=-\left(2^{n-1}+2^{n-2}+\cdots+2^{k}\right)\bmod\left(2^{n}\right)$$

$$-2^{k}=-2^{k+1}+2^{k}=-2^{k+2}+2^{k+1}+2^{k}=\cdots=-2^{n-1}+2^{n-2}+\cdots+2^{k}=\left(2^{n-1}+2^{n-2}+\cdots+2^{k}\right)\bmod\left(2^{n}\right)$$

In particular, we always have $2^{n-1}=-2^{n-1}\bmod\left(2^{n}\right)$. Thus, proposition 3 is proved. □

Directly by proposition 3, the following scaling rules for the signed difference notation hold in terms of equivalent modular difference:

$$[k]=[k+1,-k]=[k+2,-(k+1),-k]=\cdots=[n-1,-(n-2),\cdots,-k]=[-(n-1),\cdots,-k],$$

$$[-k]=[-(k+1),k]=[-(k+2),(k+1),k]=\cdots=[-(n-1),n-2,\cdots,k]=[n-1,\cdots,k].$$

For example: [3]=[4,-3] = [5, -4,-3] =[9,-8,-7,-6,-5,-4,-3]= [-9,-8,-7,-6,-5,-4,-3],
[-7,6,-5,-3,-2,-1,0] = [9,8,7,6,-5,-3,-2,-1,0] =[-6,-5,-4,1,0] = [-7,4,1,0].

Theorem 4. Let $\Delta^{+}\mathbf{X}=2^{k}$, $\Delta^{\pm}\mathbf{X}=[k+l,-(k+l-1),\cdots,-k]$, $1\le l\le n-k-1$. Then, we have

$$\left(\Delta^{+}\mathbf{X}\right)^{<<<s}=\begin{cases}2^{k+s}\bmod\left(2^{n}\right) & \text{if } k+l+s\le n-1;\\ \left(2^{k+s}+1\right)\bmod\left(2^{n}\right) & \text{if } k+l+s>n-1, k+s\le n-1;\\ 2^{k+s-n}\bmod\left(2^{n}\right) & \text{if } k+s>n-1.\end{cases}$$

Similarly, let $\Delta^{+}\mathbf{X}=-2^{k}$, $\Delta^{\pm}\mathbf{X}=[-(k+l),k+l-1,\cdots,k]$, $1\le l\le n-k-1$. Then, we have

$$\left(\Delta^{+}\mathbf{X}\right)^{<<<s}=\begin{cases}-2^{k+s}\bmod\left(2^{n}\right) & \text{if } k+l+s\le n-1;\\ -\left(2^{k+s}+1\right)\bmod\left(2^{n}\right) & \text{if } k+l+s>n-1, k+s\le n-1;\\ -2^{k+s-n}\bmod\left(2^{n}\right) & \text{if } k+s>n-1.\end{cases}$$

Proof: For the first half of this proof, if $k+l+s\le n-1$, then we have

$$\begin{aligned}\left(\Delta^{+}\mathbf{X}\right)^{<<<s}&=[k+l,-(k+l-1),\cdots,-k]^{<<<s}\\&=[k+l+s,-(k+l+s-1),\cdots,-(k+s)]\\&=2^{k+s}\bmod\left(2^{n}\right).\end{aligned}$$

If $k+l+s>n-1$ and $k+s\le n-1$, then we have

$$
\begin{aligned}
\left(\Delta^{+}\mathbf{X}\right)^{<<<s} &= [k+l,-(k+l-1),\cdots,-k]^{<<<s} \\
&= [-(n-1),\cdots,-(k+s),k+l+s-n,-(k+l+s-n-1),\cdots,-0] \\
&= \left(2^{k+s}+1\right)\bmod\left(2^{n}\right).
\end{aligned}
$$

If $k+s>n-1$, then we have

$$
\begin{aligned}
\left(\Delta^{+}\mathbf{X}\right)^{<<<s} &= [k+l,-(k+l-1),\cdots,-k]^{<<<s} \\
&= [k+l+s-n,-(k+l+s-n-1),\cdots,-(k+s-n)] \\
&= 2^{k+s-n}\bmod\left(2^{n}\right).
\end{aligned}
$$

.Thus, the first half of theorem 4 is proved. The second half can be proved similarly. □

Theorem 5. Let $\Delta^{+}\mathbf{X}=2^{k}$, $\Delta^{\pm}\mathbf{X}=[-(n-1),\cdots,-k]$.Then, if $k+s\le n-1$, $\left(\Delta^{+}\mathbf{X}\right)^{<<<s}=\left(2^{k+s}-2^{s}+1\right)\bmod\left(2^{n}\right)$; otherwise, $\left(\Delta^{+}\mathbf{X}\right)^{<<<s}=\left(-2^{s}+2^{k+s-n}\right)\bmod\left(2^{n}\right)$. Similarly, let $\Delta^{+}\mathbf{X}=-2^{k}$, $\left(\Delta^{+}\mathbf{X}\right)^{<<<s}=\left(-2^{k+s}+2^{s}-1\right)\bmod\left(2^{n}\right)$; otherwise, $\left(\Delta^{+}\mathbf{X}\right)^{<<<s}=\left(2^{s}-2^{k+s-n}\right)\bmod\left(2^{n}\right)$.

Proof: Actually, theorem 5 complements one extreme case of theorem 4.

When $k+s\le n-1$, we have consecutive deduction as

$$\left(\Delta^{+}\mathbf{X}\right)^{<<<s}=[-(n-1),\cdots,-k]^{<<<s}=[-(n-1),\cdots,-(k+s),-(s-1),\cdots,-0]=\left(2^{k+s}-2^{s}+1\right)\bmod\left(2^{n}\right).$$

When $k+s>n-1$, we have consecutive deduction as

$$\left(\Delta^{+}\mathbf{X}\right)^{<<<s}=[-(n-1),\cdots,-k]^{<<<s}=[-(s-1),-(s-2),\cdots,-(k+s-n)]=\left(-2^{s}+2^{k+s-n}\right)\bmod\left(2^{n}\right).$$

The second half of theorem 5 can be proved similarly. □

Theorem 4 and 5 reveal how the carries due to modular addition or subtraction bring some unexpected modular differences when bit rotations are applied in the differential path, and from which some extra conditions can be derived to prevent the occurrence of unexpected modular differences.

3 Collision Differential Selection For MD5

3.1 General Principles

Single block or multi-block collision differentials always exist for any iterated hash function based on Merkle-Damgard theory, and the number of collision differentials may be numerous, but finite given the fact that MD5 puts a limit on the length of the message. To carry out a successful collision attack, the first and crucial step is to find an input difference pattern which can be controlled in the differential propagation process, so that the input differences can be eliminated by a single or multiple iterations in the final steps (four steps for MD5).

Given an input message difference, if a differential path exists that leads to a collision, then it is called a *feasible collision differential*, hence leading to a feasible differential path, otherwise an infeasible collision differential and differential path. If it is computationally feasible to fulfill the set of necessary conditions that maintain the differential path, then we call it a *computationally feasible collision differential*, hence a computationally feasible differential path, otherwise a computationally infeasible collision differential and differential path. In general, firstly a good collision differential should result in smaller and smaller differences beginning from round 2, so that an elimination of all differentials or most differentials can be achieved in the final round; secondly, the start differences in round 1 should be as far away as possible from the first step to ensure enough free message words in round 1, so that some states in round 2 can also be directly satisfied by these free message words through multi-step modifications. Wang et al have given the first collision differential [6] which properly meets the principles described above.

3.2 Input Difference Patterns

The first successful attack on the compression function of MD5 was proposed by Dobbertin [4]. The basic idea of Dobbertin's attack is to describe the whole compression function as a system of equations. These equations include the contents of the registers after various steps and the message words as variables, while the equations are mainly derived from the step operation and the message expansion. Using the concepts of inner collision and inner almost collision, the system of equations can be significantly simplified such that it becomes solvable with some special techniques including evolutionary approaches. Dobbertin's method can be used to produce real collisions for MD4 and collisions for the compression function of MD5, which is a pseudo collision attack for MD5. Inspired by Dobbertin's attack, Wang [6] developed her new technique to attack MD4-like hash functions, which can produce real collisions for MD5 and other MD4-like hash functions efficiently. The success of both Dobbertin's and Wang's attacks depends on selecting an appropriate input difference pattern so that the number of equations or the avalanche effects of the step operation can be minimized. In Dobbertin's attacks, only 1-bit input difference patterns are considered, while in Wang's attack, only a 3-bit input difference pattern is used.

By Dobbertin's attack, the system of equations consists of two inner collisions, one starting from step p_0 in the first round and ending at step p_1 in the second round, and the other starting from step p_2 in the third round and ending at step p_3 in the fourth round, both inner collisions are connected by a non-differential chain starting from step p_1+1 and ending at step p_2-1. The selection of 1-bit input difference patterns is to minimize the number of equations which depends significantly on the two inner collisions and thus on the choice of the initial bit difference step p_0. Considering the round-wise permutations $\sigma_k(i)$, a complete list of 1-bit difference patterns is given in [17], which shows that $p_0=15$ would be the best choice in terms of the number of equations to solve, but Dobbertin's choice was actually $m_{14,9}$ with $p_0=14$ instead, taking into account the inner almost collision.

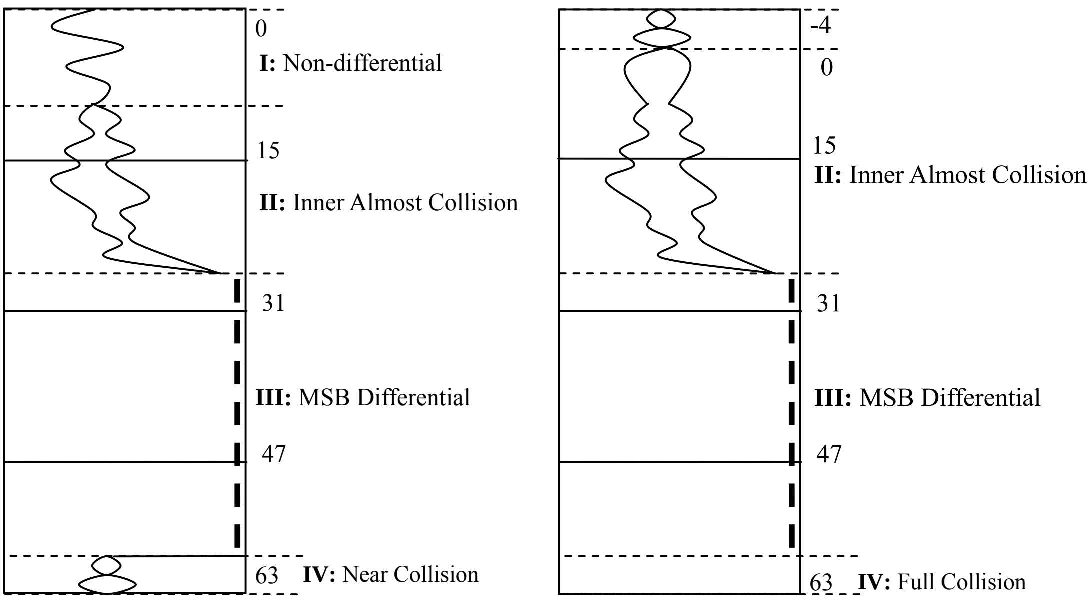

Figure1: Overview Of The Differential Collision Attack On MD5

Wang's attack on MD5 completely differs from Dobbertin's approach in that, it uses the first block to produce a near collision which can be further eliminated by the second block to generate a real collision. An input difference pattern should be selected such that in the first block there exist four differential sections, denoted as I,II,III and IV as demonstrated in Figure 1. Section I is a non-differential area without input differences, section II is a near (almost) collision area spanning across the first and second rounds, section III is a non-differential area plus MSB-only differential chain starting from the first input difference of the third round if section II is an inner collision, or a MSB-only differential chain directly derived from section II if section II is an inner almost collision. Section IV is a near collision area consisting of only a few of the last consecutive steps, better no more than 4 steps, which are then modulo 2^{32} added with the initial value to produce the chain input differences of the second block. Due to the chain input differences, section I does not exist in the second block, the chain input differences propagate all the way to the beginning of section II and constitute a lengthened section II starting from step 0, and the section III and IV in the second block correspond to the section III and IV in the first block, except that the differences in the last four consecutive steps are eliminated, i.e. turning a near collision into a full collision. This can be illustrated in Figure 1 by an overview of the differential scheme of the collision attack on MD5.

If only 1-bit to 3-bit input difference patterns are considered, we can thus make a list of all possible input difference patterns that are probably qualified to construct a feasible collision differential.

1-Bit Input Difference: If only 1-bit input difference is considered, the beginning MSB difference in section III must be derived from section II such that the differential can propagate along all the MSBs in section III to the beginning of section IV. By this specific requirement and some principles in section 3.1, a group of MSB-differences including $m_{5,31}$, $m_{8,31}$, $m_{11,31}$, $m_{14,31}$, and a group of non-MSB differences including $m_{2,7}$, $m_{4,20}$, $m_{5,10}$, $m_{6,8}$, $m_{8.25}$ $m_{9,27}$, $m_{11,15}$ and $m_{14,16}$ are probably qualified to construct a feasible collision differential.

2-Bit Input Difference: Since 2-bit input differences cannot themselves produce a section III (MSB differential chain) in the third round, at least one MSB difference at the beginning steps in section III must be derived from section II so that 3-MSB differences (1-bit plus 2-bit) can be combined in the third round to form an area of section III. By this specific requirement and some principles in section 3.1, an appropriate composition of two 1-bit input differences can be a 2-bit input difference that is probably qualified to construct a feasible collision differential. Basically, there are three ways to select the two 1-bit input differences, i.e., to select both from the MSB-difference group, or from the non-MSB difference group, or one from each group. Therefore, many 2-bit input differences can be selected by the above three ways. For example, a couple of two MSB-differences $m_{5,31}$ and $m_{11,31}$ or $m_{8,31}$ and $m_{14,31}$, a couple of two non-MSB differences $m_{5,10}$ and $m_{9,27}$, and a couple of MSB-difference $m_{8,31}$ and non-MSB difference $m_{5,10}$ can all be qualified to construct a feasible collision differential. For the sake of limited space, we only give three different 2-bit input differences in Table 1.

3-Bit Input Difference: Since 3-bit input differences can themselves produce a section III (a MSB differential chain) in the third round, no derivation of differences from section II is needed. The 3-bit input differences should be arranged in such a way that, after left rotations, the first bit difference in the third round must be a MSB, and this MSB difference is combined with the second and third MSB input differences within four steps to build a consecutive MSB differential chain, which will propagate along the way to the beginning of section IV. By this specific requirement and some principles in section 3.1, only the corresponding bit differences of m_2, m_4, m_6, m_9, m_{11}, m_{13} and m_{15} in the end of the fourth round can be used to produce the beginning MSB difference in the third round. Due to the round-wise message permutation in the third round, m_2 and m_{15} cannot be the beginning MSB difference, while the corresponding bit differences in m_4, m_6, m_9, m_{11} and m_{13} are qualified as the first input difference, namely, $m_{4,20}$, $m_{6,8}$, $m_{9,27}$, $m_{11,15}$ and $m_{13,27}$. Therefore, we have five groups of 3-bit input difference patterns, each consisting of 3 words, each word having 1-bit difference. More specifically, $m_{4,20}$, $m_{7,31}$ and $m_{13,31}$ constitute the first group, $m_{6,8}$, $m_{9,31}$ and $m_{15,31}$ the second group, $m_{9,27}$, $m_{12,31}$ and $m_{2,31}$ the third group, $m_{11,15}$, $m_{14,31}$ and $m_{4,31}$ the fourth group, and $m_{13,27}$, $m_{0,31}$ and $m_{6,31}$ the fifth group. Being required of enough free message words in the first round, the fifth group cannot be a good collision differential.

Table 1. The Most Probable Input Difference Patterns For Collision Attacks On MD5

Bit Differences	Number	Section I	Section II	Section III	Section IV
$m_{4,20}$, $m_{7,31}$, $m_{13,31}$	3-bit	1-4	5-31	32-60	61-64
$m_{6,8}$, $m_{9,31}$, $m_{15,31}$	3-bit	1-6	7-25	26-58	59-64
$m_{2,31}$, $m_{9,27}$, $m_{12,31}$	3-bit	1-2	3-32	33-63	64-64
$m_{4,31}$, $m_{11,15}$, $m_{14,31}$	3-bit	1-4	5-26	27-61	62-64
$m_{2,7}$	1-bit	1-2	3-30	31-62	63-64
$m_{4,20}$	1-bit	1-4	5-24	25-60	61-64
$m_{6,8}$	1-bit	1-6	7-18	19-58	59-64
$m_{9,27}$	1-bit	1-9	10-25	26-63	64-64
$m_{11,15}$	1-bit	1-11	12-19	20-61	62-64
$m_{5,10}$, $m_{9,27}$	2-bit	1-5	6-25	26-63	64-64
$m_{5,31}$, $m_{11,31}$	2-bit	1-5	6-21	22-64	
$m_{8,31}$, $m_{11,21}$	2-bit	1-8	9-31	32-64	
				III + IV = MSB Diff. Chain	
$m_{5,31}$	1-bit	1-5	6-21	22-64	
$m_{8,31}$	1-bit	1-8	9-28	29-64	
$m_{11,31}$	1-bit	1-11	12-19	20-64	
$m_{14,31}$	1-bit	1-14	15-26	27-64	
$m_{5,10}$	1-bit	1-5	6-29	30-64	
$m_{8.25}$	1-bit	1-8	9-30	31-64	
$m_{11,21}$	1-bit	1-11	12-31	32-64	
$m_{14,16}$	1-bit	1-14	15-32	33-64	

For the sake of clarity, we make a list of the whole set of input difference patterns in Table 1, that are most probably qualified to construct a feasible collision attack on MD5. It is interesting to note that, all the collision differentials that have already been published are included in Table 1. From Table 1, we can see that Wang's choice is perhaps the best one in the four groups of 3-bit input differences, the 2-bit input differences are derived from the 1-bit input differences, and can be regarded as compositions of them. So at this stage, we only need to pay attention to those 1-bit input differences. However, it is not a trivial task to tell which 1-bit input difference is more appropriate for collision differential, unless a comprehensive step-by-step analysis is performed.

3.3 Three Feasible Collision Differentials

For a comparison, in Table 2 we make a list of all the collision differentials that have been published, with their chain output differentials $\Delta^{+}H_1$ and $\Delta^{+}H_2$ together.

Table 2. The Three 2-Block Collision Differentials Published For MD5

Δ^+	No.1 Collision Differential[6]	No.2 Collision Differential[15]	No.3 Collision Differential
$\Delta^+ M_0$	0,0,0,0,2^{31},0,0,0,0,0,0,2^{15},0,0,2^{31},0	0,0,0,0,0,0,-2^{8},0,0, 2^{31},0,0,0,0,0, 2^{31}	0,0,0,0,0,0,0,0,2^{31},0,0,0,0,0,0,0
$\Delta^+ M_1$	0,0,0,0,2^{31},0,0,0,0,0,0,-2^{15},0,0,2^{31},0	0,0,0,0,0,0,2^{8},0,0, 2^{31},0,0,0,0,0, 2^{31}	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
$\Delta^+ H_1$	2^{31}, $2^{31}+2^{25}$, $2^{31}+2^{25}$, $2^{31}+2^{25}$	$2^{31}-2^{23}$, $2^{31}-2^{23}$, $2^{31}-2^{23}$, $2^{31}-2^{23}$	2^{31}, 2^{31}, 2^{31}, 2^{31}
$\Delta^+ H_2$	0, 0, 0, 0	0, 0, 0, 0	0, 0, 0, 0

4 Design of Differential Paths

4.1 General Principles

When an input difference pattern is found to be probably qualified to construct a feasible collision differential, the next task is to design a feasible differential path which leads to a collision. The basic design criterion is to minimize the Hamming weight of the differential path, which counts the number of bit differences in the differential path, especially the differential section III and IV. In addition, section II is the most critical part of the differential path, a successful design of the differential path usually depends on it. When designing a differential path, a smart trial-and-error method is necessary in the backward-and-forward construction process. To design a good differential path with as small Hamming weight as possible, the following principles would be benefited from if observed.

1) Deduce a differential path bottom-up in a backward way, starting from the first inner collision in section II (in the second round), up to four or five steps away from the first input difference step in the first round;
2) Deduce the differential up-down from the first input difference step in the first round so that it can link-up with the bottom-up differential;
3) In general, the start input difference in section II is applied in the step operation in such a way that, all differences that are needed by the bottom-up differential can be generated within five steps;
4) Employ the properties implicit in the signed difference to extend the signed differences as required in each backward or forward step, and this is the basic rule suitable for any hash functions;
5) Use the generation and elimination rules implicit in the auxiliary functions in each backward or forward step, and these are the special rules derived from the particular hash function.

4.2 MD5 Differential Propagation

An MD5 differential path is composed of 64 consecutive steps of state differences.

Four consecutive signed differences (in order of $\Delta^{\pm}a$, $\Delta^{\pm}d$, $\Delta^{\pm}c$ and $\Delta^{\pm}b$) are employed as input to the step operation function to generate the next signed difference $\Delta^{\pm}a^{\bullet}$, we call this computation a step of MD5 differential iteration.

In an MD5 step of differential iteration, the modular differences in the next step can be :

1) Directly derived from the modular difference of $\Delta^{\pm}a$ in state variable a ;
2) Directly derived from the modular difference of $\Delta^{\pm}b$ in state variable b ;
3) Indirectly generated by the auxiliary function, provided that at least one signed difference exists at the same bit position in the last three state variables b , c and d :
   i) A modular difference can be generated in quite a few ways;
   ii) Actually, modular differences can be generated from the last three state variables b , c and d in a variety of ways, by utilizing both the properties implicit in the signed difference and the rules implicit in the auxiliary functions;
   iii) Almost all intelligence of differential path designing is focused here, and for the basic differential propagation rules with respect to the four auxiliary functions, please refer to [15].

4) The modular differences generated by the auxiliary function can be used to cancel out those modular differences derived directly from the top or last state variables a and b .

With the properties implicit in the signed difference, in each forward or backward differential iteration step, the critical technique will most probably be, on the one hand, to employ the auxiliary functions to generate those modular differences, required by the next output signed differences $\Delta^{\pm}a^{\bullet}$ but not directly derived from the top or last signed differences $\Delta^{\pm}a$ and $\Delta^{\pm}b$; on the other hand, to employ the auxiliary functions to generate the complementary modular differences for those directly derived from the top and last signed differences $\Delta^{\pm}a$ and $\Delta^{\pm}b$, but not required by the next output signed differences $\Delta^{\pm}a^{\bullet}$, so that two complementary signed differences be eliminated together.

4.3 Basic Conditions Due To Signed Differences

A bit that a value must be specified to keep control of the differential path, is called a conditional bit, a set of bit specifications on all the conditional bits is called sufficient if it will definitely lead to a collision when all are imposed on. In particular, two bits may be relatively specified to include two situations, for example, $a_{i,j} = d_{i,j}$ or $a_{i,j} \neq d_{i,j}$.

All the bit specifications due to the signed difference bits in the state variables are called basic conditions. Every basic condition is incidental to a signed difference bit in a state variable within two steps, in other words, a bit cannot become a basic condition if there exists no signed difference bit on the same position in a state variable within two steps. Each state variable works as different component in three consecutive step operations, consequently a bit difference in a state variable will produce at most five basic conditions, which are uniquely determined by the auxiliary function applied. As for the ITE function $F(X,Y,Z)$ used in the first round, one condition is the difference bit itself, one or two conditions depend on if there are modular differences derived from when it works as the selection component, two conditions are defined by the bit of being selected or not in the ITE function. For the three different auxiliary functions of MD5, we give the basic condition derivation rules in Table 3.

Table 3. Basic Condition Derivation Rules For Auxiliary Functions In MD5

$F(b_i,c_i,d_i)=(b_i \wedge c_i)\vee(\overline{b_i}\wedge d_i),\ 0\le i<32$																
$\Delta^{\pm}F_i$	0		-1		+1		$\Delta^{\pm}F_i$	0		±1		$\Delta^{\pm}F_i$	0		±1	
d_i	0	1	0	1	1	0	d_i	*		*		$\Delta^{\pm}d_i$	±1		±1	
c_i	0	1	1	0	0	1	$\Delta^{\pm}c_i$	±1		±1		c_i	*		*	
$\Delta^{\pm}b_i$	±1		-1	+1	-1	+1	b_i	0		1		b_i	1		0	
$H(b_i,c_i,d_i)=b_i\oplus c_i\oplus d_i,\ 0\le i<32$																
$\Delta^{\pm}H_i$	±1		∓1				$\Delta^{\pm}H_i$	±1		∓1		$\Delta^{\pm}H_i$	±1		∓1	
d_i	0	1	0	1			d_i	0	1	0	1	$\Delta^{\pm}d_i$	±1			
c_i	0	1	1	0			$\Delta^{\pm}c_i$	±1				c_i	0	1	0	1
$\Delta^{\pm}b_i$	±1						b_i	0	1	1	0	b_i	0	1	1	0
$I(b_i,c_i,d_i)=c_i\oplus(b_i\vee\overline{d_i}),\ 0\le i<32$																
$\Delta^{\pm}I_i$	0		-1		+1		$\Delta^{\pm}I_i$	∓1			±1	$\Delta^{\pm}I_i$	0		±1	∓1
d_i	0		1		1		d_i	0	0	1	1	$\Delta^{\pm}d_i$	±1			
c_i	*		0	1	1	0	$\Delta^{\pm}c_i$	±1				c_i	*		0	1
$\Delta^{\pm}b_i$	±1		-1	+1	-1	+1	b_i	0	1	1	0	b_i	1		0	

In Table 3, each auxiliary function has one bit signed difference at one of the three components, denoted in order as b_i, c_i and d_i . When the component b has signed difference $\Delta^{\pm}b_i$, the other two component bits c_i and d_i must be (or relatively) specified according to the output bit signed difference $\Delta^{\pm}F_i$, $\Delta^{\pm}H_i$ or $\Delta^{\pm}I_i$ as required by the differential path. The condition derivation rules are listed in the columns of every auxiliary function, each having three situations. The '0's in the $\Delta^{\pm}F_i$, $\Delta^{\pm}H_i$ and $\Delta^{\pm}I_i$ rows represent non-difference output, while the '0's in other rows represent the conditional bit value. The asterisk " * "denotes an arbitrarily specified bit. In a similar way, a condition derivation table can be easily obtained (but omitted here for the limited space) for the situation, where there are signed differences at the same bit position of two components in the auxiliary functions.

By the principles and rules in section 4.1 to 4.3, we give the basic differential paths with respect to the described 1-MSB input difference $\Delta^{+}m_8 = 2^{31}$ for two blocks in Table 6 and Table 7.

4.4 Extra Conditions Due to Carries and Rotations

Besides the basic conditions that must be fulfilled, some extra conditions must be satisfied to prevent the occurrence of some possible unexpected modular differences due to the carries or overflow. By Theorems 4 and 5 in section 2, unexpected carries and even overflows are always possible when the part $\sum a_i$ of step operation is implemented, since $\sum a_i$ will probably have a much lengthened signed difference representation of a equal modular difference, and probably again the rotation operation will just break it off. Therefore, the set of sufficient conditions must include both the basic conditions and the extra conditions, and fortunately, most of the extra conditions are fulfilled with much high probabilities.

By Theorems 4 and 5, no extra conditions are needed for the differential path of the second block, and the extra conditions for the first block are included with the following groups of equations: $\sum d_{3,6\sim19} = 0$, $\sum a_{4,21\sim24} = 0$, $\sum a_{5,22\sim24} = 0$, $\sum d_{5,29\sim31} = 0$, $\sum c_{5,7\sim17} = 0$, $\sum a_{6,24\sim26} = 1$, $\sum d_{6,11\sim22} = 0$, $\sum c_{6,21\sim31} = 0$, $\sum b_{6,9\sim11} \neq 101$, $\sum b_{6,15\sim31} = 1$, $\sum a_{7,26} = 1$ or $\sum a_{7,24\sim25} = 11$, $\sum c_{7,7\sim17} = 0$, $\sum b_{7,29\sim31} = 1$.

As an example, $\sum d_{3,6\sim19} = 0$ means that at least one equation of $\sum d_{3,6} = 0$, $\sum d_{3,7} = 0$,…, $\sum d_{3,19} = 0$ must hold.

4.5 Condition Fulfillment: Divide-and-Conquer

There always exist conditions that can hardly be satisfied by direct modifications, these conditions have to be probabilistically fulfilled through random or brute force search, which compose the computational complexity of collision attack algorithm. For example, if there exist k conditions that can only be probabilistically fulfilled in the round 2, 3 and 4, then the computational complexity will be around 2^k hash operations, which is a multiplicative accumulation on the conditions. One idea is to change the multiplicative accumulation of computational complexities into an additional accumulation by properly grouping the conditional bits that cannot be directly modified, so that the previously fulfilled groups of conditions will not be violated by later searches. This will result in a specific divide-and-conquer technique for hash collision attack, which will greatly reduce the computational complexity to be determined actually by the maximal group of conditions.

To be more precise, if the k conditions can be divided into p groups, namely G_1, G_2 … and G_p, and we have $\sum_{i=1}^{p} |G_i| = k$. Let $G_{\max} = \max\{G_1, G_2, \cdots, G_p\}$ be the largest group with the most conditions. If groups G_1 to G_i are not violated by the search of group G_{i+1}'s satisfaction and so on, then the computational complexity for the k conditions will be reduced to an additive accumulation of the complexities for groups G_1, G_2 … and G_p instead, and the group $G_{\max}$ will be representative of the whole computational complexity, provided that there exist enough free message bits to be searched for each group.

According to the principle that the previously used message words or bits must not be further modified later, these probabilistically satisfied conditions can be grouped mainly by the step orders. In this way, the conditions in round 2,3 and 4 can be divided into three groups, more specifically, a_5, d_5 and c_5 constitute the first group, b_5, a_6, d_6 and c_6 the second group, b_6 the third group, finally the state variables from a_7 to the end b_{16} constitute the fourth group (or both the third and the fourth groups constitute the final group). The first group relies directly on the brute force search on the free bits of a_5, but indirectly on m_1 to fulfill the conditions that cannot be satisfied by direct modification in d_5 and c_5. The second group relies directly on the brute force search on the free bits of b_5, but indirectly on m_0 to fulfill the conditions in a_6, d_6 and c_6, which are all probabilistically satisfied. The conditions in a_7 are satisfied directly through a brute force search on the four selected bits in a_3, but indirectly on m_9, while the final group relies mainly on the brute force search directly on the free bits of b_1, but indirectly on m_3, m_4 and m_7 (and m_8, m_9 and m_{12} due to the selected bits search on a_3) to fulfill the conditions in the state variables from b_6 to the end b_{16}.

4.6 Additional Conditions: Change Absorption

Take the first block as an example. In the third group, the brute-force search on the free bits of b_1 will certainly make changes on d_2 and c_2, and indirectly on m_5 and m_6 if the d_2 and c_2 remain the same, this will result in a conflict with the previously used message words m_5 and m_6, since m_5 and m_6 are used respectively in state variables d_5 (in the first group) and a_6 (in the second group). To avoid this type of conflicts, as components of the choose function $F(X,Y,Z)$ the state variables a_2 and d_2 need to be specified in the format of 0x00000000 and 0xffffffff respectively, so that the brute force search on the state variable b_1 will be absorbed in the re-computation of d_2 and c_2. For the same reason, the four couples of bits (namely, $d_{3,1}$, $d_{3,2}$, $d_{3,17}$, $d_{3,31}$, and $c_{3,1}$, $c_{3,2}$, $c_{3,17}$, $c_{3,31}$ respectively) in the state variables d_3 and c_3 need to be 0 and 1, respectively, so that the brute force search on the four free bits of a_3 can be absorbed in the re-computation of d_3 and c_3, resulting in no changes on the previously used m_{10} and m_{11}. The second block is treated in a similar way.

Table 8 and Table 9 are obtained by respectively modifying Table 6 and Table 7 as described above, additional conditions are appended for absorbing the changes due to random or brute-force searches.

5 Collision Searching Algorithms

In general, a collision searching algorithm is fundamentally determined by the corresponding differential path, a good differential path corresponds to an intrinsically efficient algorithm. The objective of designing an algorithm for a collision differential path is to reduce the number of probabilistically fulfilled conditions as much as possible, this can be achieved by some methods such as single-step modification, multi-step modification and the tunneling-like techniques. In this paper, by properly grouping of conditional bits, we particularly transform the multiplicative computational complexities into additional accumulation, which is the divide-and-conquer technique introduced in Sections 4.5 and 4.6. As a result, the actual computational complexity is dramatically reduced. The collision searching algorithm has been implemented which is available from the website http://www.is.iscas.ac.cn/gnomon. As a result of our computation, a collision pair is given in Table 4 with its MD5 digest.

5.1 An Algorithm For The First Block

Step 1: Randomly initialize the state variables c_1 and b_1, set a_2 and d_2 to be 0x00000000 and 0xffffffff. Randomly initialize the state variables from c_2 to a_5 so that all the relevant conditions in Sections 4.3 and 4.4 satisfied. This can be done first to satisfy conditions in Sections 4.3, then check by the extra conditions in Section 4.4 if there exist invalid carries in d_3 and a_4, if so, repeat the random initialization again; otherwise, compute the message words from m_6 to m_{15} according to their corresponding step equations and based on the state variables from c_1 to b_4;

Step 2: Do the brute force search on the free bits of a_5, and for each of the brute force search, execute the following steps. When the brute force search is over, then go to step 1;

Step 3: Randomly initialize d_5 but with all the relevant conditions in Sections 4.3 and 4.4 satisfied, compute m_6 according to the d_5 step equation, then make an update of c_2. Check if there exist conditions unsatisfied in c_2, then go to step 2;

Step 4: Compute c_5, check if there exist conditions unsatisfied for c_5, then modify b_2 and a_3 or directly c_4 to compute m_{11}, so that the conditions for c_5 can be satisfied from the less significant bit to more significant bits. If $c_{5,20} = 0$ or $c_{5,21} = 1$, go to step 2 since no modifications can be applied; otherwise, initialize b_5 so that all the relevant conditions in Sections 4.3 and 4.4 are satisfied, and compute m_1 according to the a_5 step equation;

Step 5: Do the brute force search on the free bits of b_5 one binary combination each time. Compute m_0 according to the b_5 step equation, then make an update of a_1 and d_1, and compute m_5 according to the d_2 step equation. If the brute force search on the free bits in b_5 is over, go to step 2;

Step 6: Compute a_6, check if there exist any conditions unsatisfied for a_6, then go to step 5;

Step 7: Compute d_6, check if there exist any conditions unsatisfied for d_6, then go to step 5;

Step 8: Compute c_6, check if there exist any conditions unsatisfied for c_6, then go to step 5; otherwise, compute m_2, m_3 and m_4 according to the c_1, b_1 and a_2 step equations;

Step 9: Do the brute force search on the free bits of b_1, compute m_4 according to the a_2 step equation to make an update of b_6. Check if all the relevant conditions in Sections 4.3 and 4.4 for b_6 are satisfied, if so, then compute m_3 and m_7; otherwise do step9 again. If the brute force search on the free bits in b_1 is over, go to step 5;

Step 10: Do the brute force search on the four free bits ($a_{3,1}$, $a_{3,2}$, $a_{3,17}$ and $a_{3,31}$) of a_3 one binary combination each time, compute m_9 according to the d_3 step equation to make an update of a_7. Check if all the relevant conditions in Sections 4.3 and 4.4 for a_7 are satisfied, if so, then compute m_8 and m_{12}; otherwise do step 10 again. If the search on the free bits in a_3 is over, go to step 9;

Step 11: Compute the next step operation till the last one, check if there exists a relevant unsatisfied condition in Sections 4.3 and 4.4, then go to step 10 (called early stop); otherwise, output the chain variables $a_{16} + a_0$, $b_{16} + b_0$, $c_{16} + c_0$ and $d_{16} + d_0$ to the algorithm for the second block.

5.2 An Algorithm For The Second Block

Step 1: Randomly initialize a_1 and b_1 so that they satisfy all the relevant conditions in Sections 4.3

and 4.4. Set a_2 and d_2 to be 0x00000000 and 0x7fffffff so that all the relevant conditions in Sections 4.3 and 4.4 satisfied; randomly initialize the state variables from c_2 to a_3 so that all the relevant conditions in Sections 4.3 and 4.4 satisfied; set d_3 and c_3 to be 0x00000000 and 0x7fffffff so that all the relevant conditions in Sections 4.3 and 4.4 satisfied; randomly initialize the state variables from b_3 to c_4 so that all the relevant conditions in Sections 4.3 and 4.4 satisfied. Compute the message words from m_0 to m_{14} according to their corresponding step equations and based on the state variables from a_1 to c_4 ;

Step 2: Randomly initialize b_4 so that all the relevant conditions in Sections 4.3 and 4.4 satisfied, search all the free bits of b_4 for the following computations. If the prescribed limit on the number of random search tries is over, then go to step 1;

Step 3: Compute the state variables from a_5 to c_6 , check if there exist any conditions unsatisfied for the state variables from a_5 to c_6 , then go to step 2;

Step 4: Randomly initialize b_1 but with all its conditions satisfied. Do the brute force search on the free bits of b_1 one binary combination each time, compute m_4 according to the a_2 step equation to make an update of b_6 . Check if there exist any conditions unsatisfied for b_6 , if so, repeat the random initialization again; otherwise, compute m_3 and m_7 . If the prescribed limit on the number of brute force search tries is over, go to step2;

Step 5: Randomly initialize a_3 so that all the relevant conditions in Sections 4.3 and 4.4 satisfied. Do the brute force search on the free bits of a_3 one binary combination each time, compute m_9 according to the d_3 step equation to make an update of a_7 . Check if there exist any conditions unsatisfied for a_7 , if so, repeat the random initialization again; otherwise, compute m_8 and m_{12} . If the prescribed limit on the number of brute force search tries is over, go to step4;

Step 6: Compute the next step operation till the last one, check if there exists a relevant unsatisfied condition in Sections 4.3 and 4.4, then go to step 5; otherwise output the collision blocks.

Table 4. A Collision Example With The MD5 Digest (Underlined Bits With Difference）

M_0	0x6f5405b5, 0xb891efe, 0xae153522, 0x3dd541ab, 0x77cfac08, 0xb4ae7077, 0xb14ec779, 0xa7ccf30, 0xf1c56954, 0x70dc3345, 0x5eda46a1, 0xc9fc1730, 0x948b9be, 0x2ef76cad, 0x86149360, 0x3bcecd25
M_1	0x1dea12a, 0x50179204, 0x6a2ab7f9, 0x80e06efa, 0x1da137c9, 0x22032f7e, 0x3af27c94, 0xbfd0dda2, 0x54dd5054, 0xde27de3, 0x328eb6dc, 0x1da31980, 0xf0a9c456, 0x720e6177, 0xe5ac6c8f, 0x15ab7afc
M_0^*	0x6f5405b5, 0xb891efe, 0xae153522, 0x3dd541ab, 0x77cfac08, 0xb4ae7077, 0xb14ec779, 0xa7ccf30, 0x71c56954, 0x70dc3345, 0x5eda46a1, 0xc9fc1730,0x948b9be, 0x2ef76cad, 0x86149360, 0x3bcecd25
M_1^*	0x1dea12a, 0x50179204, 0x6a2ab7f9, 0x80e06efa, 0x1da137c9, 0x22032f7e, 0x3af27c94, 0xbfd0dda2, 0x54dd5054, 0xde27de3, 0x328eb6dc, 0x1da31980, 0xf0a9c456, 0x720e6177, 0xe5ac6c8f, 0x15ab7afc
	MD5 value: 0x281e1404 0x596131cd 0x9cd2262c 0xa5aa822f

5.3 Computational Complexity Analysis

There totally exist 47 and 31 conditions respectively in the first block and the second block starting from the second round, which must be probabilistically fulfilled, the computational complexity would be around 2^{47} and 2^{31} MD5 operations if only multi-step modifications are applied. When the divide-and-conquer technique is applied, these conditions are divided into four groups, each group of conditions are independently and probabilistically fulfilled without violating each other, resulting in a great decrease in the computational complexity. In details, the condition fulfillment in the first block can be divided into five phases as follows:

Phase 1: Phase 1 includes step 1 and step 2. Since in this phase only direct modifications are needed, the computational complexity is a constant denoted as C ;
Phase 2: Phase 3 includes step 3 and step 4. Direct modifications coexist with probabilistic condition fulfillment in this phase, and only three conditions (one for step d_5 and two for step c_5) are probabilistically fulfilled without violating the previously satisfied conditions in phase 1, resulting in a computational complexity of less than 2^3 MD5 operations;
Phase 3: Phase 3 includes step 5 to step 8. In this phase, totally 15 conditions are probabilistically fulfilled without violating the previously satisfied conditions in phase 1 and phase 2, resulting in a computational complexity of less than 2^{15} MD5 operations;
Phase 4: Phase 4 includes only step 9, only two conditions are probabilistically fulfilled, resulting in a computational complexity of less than 2^2 MD5 operations;
Phase 5: Phase 5 includes step 10 to step 11. There totally exist 27 conditions that can only be probabilistically fulfilled without violating the previously satisfied conditions in phase 1, 2 and 3. Since a single try involves at least (3+1) steps of operation, in average it results in a computational complexity of $2^{23.1}$ MD5 operations (In Appendix B, a detailed calculation of the computational complexity is described).

Due to the separation of the four phases above, the total computational complexity for the first block is an additive accumulation of that in all the four phases, which means that the computational complexity is $C + 2^3 + 2^{15} + 2^2 + 2^{23.1}$, instead of a multiplicative accumulation, which would be $C + 2^3 \times 2^{15} \times 2^2 \times 2^{27} \approx 2^{47}$. A similar analysis on the second block shows that the averaged computational complexity for the second block is $2^{19.1}$ MD5 operations.

6 Summary, Comparison and Suggestions

In this paper, firstly, a whole list of 1-bit to 3-bit eligible input differences is presented, and the supernatural appearance of collision differential selection is thus revealed. Secondly, a new 1-MSB input difference pattern is developed to be the currently fastest collision attack algorithm for MD5, with an averaged computational complexity of $2^{23.1}$ MD5 operations, implying that a common desktop PC can easily produce MD5 collisions. Thirdly, a divide-and-conquer technique specific for hash collision attacks is proposed with a concrete application of it. Finally, some technical details related to the derivation of the basic conditions and the extra conditions are presented. This paper will help the cryptology community to further grasp the recent techniques on hash cryptanalysis.

A collision differential can be evaluated according to the following seven criteria:

1) Whether or not the differential path depends on the fixed IVs of hash function;
2) The number of message blocks constituting a collision differential;
3) The number of free words in the message;
4) The number of bit differences in the messages;
5) The number of the conditional bits which must be satisfied to yield a collision;
6) The number of the conditional bits excluding the first round;
7) The averaged computational complexity of finding a collision.

Considering the real-world cryptanalytic attacks, a differential path which does not rely on the fixed initial IVs will obviously be better than those must rely on it, a collision differential which has more free words, less input differences and sufficient conditions will be more easily used to construct meaningful attacks, a collision differential with less message blocks and probabilistically fulfilled conditions will be more efficient for practical attacks. The less the conditions necessary to maintain a full differential path, the higher the density of collision message will be; the less the average computational complexity of finding a collision, the more feasible an attack on practical protocols based on hash function will be. For the three collision differentials on MD5 that have been published, we make a comparison in Table 5 based on the above criteria. From Table 5, the 1-MSB input difference exceeds the other two 3-bit input differences, in terms of free message words, bit differences, sufficient conditions and especially the computational complexity.

Table 5. Performance Comparison For The Three Collision Differentials

	No.1 [6]	No.2 [16]	No.3	Notes
depend on fixed IVs	no	no	no	IVs are free
number of blocks	2	2	2	2-block collision
number of free words	1~2	1~4	1~6	Steps indexed by 1~64
number of diff. bits	3-bit/3-bit	3-bit/3-bit	1-bit/0-bit	No.3 is specific
number of all conditions	290 / 309	205 / 306	264 / 47	exclude extra conditions
number of prob. conditions	43 / 36	38 / 35	84 / 31	exclude extra conditions
computational complexity	$2^{24.8}$	2^{30}	$2^{23.1}$	averaged
time / a collision (averaged.)	1 min.	30 min.	1 sec.	2.66 GHZ PC

By the seven criteria above, in this paper the 1-MSB input difference $m_{8,32}$ may not be the best choice for the MD5 collision differential, probably there exists better choices from the other input differences in Table 1, as the $m_{8,32}$-based collision differential was developed before the whole set of input differences being found. Hence, an open problem is whether other collision differentials from Table 1 have better performance than the 1-MSB collision differential.

Another problem that remains open is that, given a collision differential, how to efficiently design a differential path so that the computation of real collision is mostly efficient. Despite of some initial work in this direction [14, 18], it is worth to perform a further study on how to intelligently and automatically design a good differential path.

References

1. Ron Rivest. The MD5 message-digest algorithm. Internet Request for Comment RFC 1321, Internet Engineering Task Force, April 1992.
2. T. A. Berson. Differential Cryptanalysis Mod 2^{32} with application to MD5. In Advances in Cryptology, Proceedings of EUROCRYPT'92, pages71~80, 1992.
3. B. den. Boer, A. Bosselaers. Collisions for the compression function of MD5, Advances in Cryptology, Eurocrypt'93 Proceedings, Springer-Verlag, 1994.
4. H. Dobbertin. Cryptanalysis of MD5 compress, presented at the rump session of Eurocrypt'96.
5. X.Y. Wang, F.D. Guo, X.J. Lai, H.B. Yu, Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD, rump session of Crypto'04, E-print, 2004.
6. X.Y. Wang, Hongbo Yu, How to Break MD5 and Other Hash Functions, EUROCRYPT 2005, LNCS 3494, pp.19-35, Springer-Verlag, 2005.
7. X.Y Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1, Crypt'2005, LNCS 3621, pp17~36.
8. Vlastimil Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105, 2006. http://eprint.iacr.org/.
9. Magnus Daum and Stefan Lucks. Hash Collisions (The Poisoned Message Attack) "The Story of Alice and her Boss". Presented at the rump session of Eurocrypt '05.
10. Max Gebhardt, Georg Illies, and Werner Schindler. A Note on the Practical Value of Single Hash Collisions for Special File Formats. In Jana Dittmann, editor, Sicherheit, volume 77 of LNI, pages 333–344. GI, 2006.
11. Marc Stevens, Arjen Lenstra, and Benne de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities, EUROCRYPT 2007 (Moni Naor, ed.), LNCS, vol. 4515, Springer, 2007, pp. 1–22.
12. Scott Contini and Yiqun Lisa Yin. Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions. In Xuejia Lai and Kefei Chen, editors, ASIACYPT, volume 4284 of Lecture Notes in Computer Science. Springer, 2006.
13. Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. On the Security of HMAC and NMAC Based on

HAVAL, MD4, MD5, SHA-0 and SHA-1. In Roberto De Prisco and Moti Yung, editors, SCN, volume 4116 of Lecture Notes in Computer Science, pages 242–256. Springer, 2006.
14. Y. Sasaki, L. Wang, N. Kunihiro, and K. Ohta. New Message Differences for Collision Attacks on MD4 and MD5. *IEICE Transactions*, 91-A(1):55-63, 2008.
15. Tao Xie, Dengguo Feng, Fanbao Liu. A New Collision Differential For MD5 With Its Full Differential Path, Cryptology ePrint Archive (2008/230), http://eprint.iacr.org/.
16. Tao Xie, Dengguo Feng, Fanbao Liu. An Improved Path for Xie's first Collision Differential of MD5, technical paper, 2008.6.
17. M. Daum. Cryptanalysis of Hash Functions of the MD4-Family. PhD thesis, Ruhr-University of Bochum, 2005.
18. C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, Advances in Cryptology - ASIACRYPT 2006, Proceedings, volume 4284 of Lecture Notes in Computer Science, pages 1-20. Springer, 2006.

Appendix A MD5 Function

Practically, a Merkle-Damgard structure-based hash function is iterated by a compression function $Y = f(X)$, which compresses l-bit message block X to an s-bit hash value Y, where $l > s$. For MD5, $l = 512$, $s = 128$. For a padded message M with multiple (t) of l-bit blocks, the iteration process can be described as: $H_{i+1} = f(H_i, M_i)$, $0 \le i \le t-1$, where $M = (M_0, M_1, ..., M_{t-1})$, H_i is the 128-bit chaining variable (including four 32-bit words) which is updated during the processing of each block, H_0 is the prescribed initial value IVs in MD5 algorithm, and the final H_t is the digest that we expect to obtain. The concrete padding rule is omitted here, since it has no influence on our attack.

The whole process of the *i*-th block $f(H_i, M_i)$ can be defined as follows: $H_{i+1} = f(H_i, M_i) = H_i + II(M_i, HH(M_i, GG(M_i, FF(M_i, H_i))))$, where four round functions FF, GG, HH and II are involved. All round functions are similar to one another in structure. The chaining variable H_i is treated as a four-element shift register, with each element being one 32-bit word, referred to as a_0, b_0, c_0 and d_0, respectively. Each 512-bit block M_i is divided into 16 32-bit words, denoted as $M_i = (m_0, m_1, ..., m_{15})$, each round consists of 16 steps of operations, in each step of operation, the register is updated with one word from M_i. The 64 step operations form a system of equations: $a_{i+1} = b_i + \left((a_i + \Phi_j(b_i, c_i, d_i) + w_j + t_j)^{<<<s_j}\right)$, $0 \le i \le 16$, $1 \le j \le 64$, where a_i, b_i, c_i and b_i ($1 \le i \le 16$) are the internal state variables, $\Phi_j(X, Y, Z)$ is an auxiliary function which varies from round to round, w_j is a word chosen from $(m_0, m_1, ..., m_{15})$ by a round-wise message permutation $\sigma_k(i), k = 0,1,2,3$, $i = 0,1,...,15$, t_j and s_j are constant parameters associated with step j. Note that each step operation involves four modular additions ($\mathrm{mod}\, 2^{32}$), one auxiliary function and one $<<<$ operation. As the step operation of MD5 is reversible, the compression function $f(H_i, M_i)$ uses a feed-forward operation which adds the initial value H_i of the register to their final values, so that $f(H_i, M_i)$ cannot be inverted.

For the sake of understanding how and where some extra conditions are derived from late in section 5, which are used to prevent the possible unexpected modular differences due to the joint effect of both modular addition and left rotation, we define part of the step operation as $\sum a_{i+1} = a_i + \Phi_j(b_i, c_i, d_i) + w_j + t_j$.

For a detailed description of MD5 algorithm, please refer to [1].

The auxiliary function and the round-wise permutation $\sigma_k(i)$ for each round are given as follows:

$$\Phi_j(X,Y,Z)=F(X,Y,Z)=(X\wedge Y)\vee(\overline{X}\wedge Z), 1\le j\le 16;$$
$$\Phi_j(X,Y,Z)=G(X,Y,Z)=(X\wedge Z)\vee(Y\wedge \overline{Z}), 17\le j\le 32;$$
$$\Phi_j(X,Y,Z)=H(X,Y,Z)=X\oplus Y\oplus Z,\ 33\le j\le 48;$$
$$\Phi_j(X,Y,Z)=I(X,Y,Z)=Y\oplus(X\vee \overline{Z}), 49\le j\le 64.$$

$$w_{j+1}=\begin{cases} m_j, & 0\le j<16;\\ m_{1+5j \bmod 16}, & 16\le j<32;\\ m_{5+3j \bmod 16}, & 32\le j<48;\\ m_{7j \bmod 16}, & 48\le j<64.\end{cases}$$

Where X,Y,Z are 32-bit words. The auxiliary functions $\Phi_j(X,Y,Z)$ each takes three consecutive 32-bit words from the register of chaining variables as input and produces one 32-bit word as output. The four words in the chaining variable register are initialized as

$a=$ 0x67452301, $b=$ 0xefcdab89, $c=$ 0x98badcfe, $d=$ 0x10325476.

Appendix B Estimation of The Averaged Computational Cost

No concrete method of estimating the averaged computational cost has been described in literature related to MD5 collision attacks. Assume a bit condition being fulfilled with probability 1/2, the computational complexity of obtaining a MD5 collision is usually presumed to be 2^k MD5 operations if there exist k conditional bits that can only be probabilistically imposed on. Actually, the practically averaged computational cost should be calculated step by step by the distribution of conditional bits in MD5 steps. To be more precisely, assume there are k_i conditional bits for MD5 step i, $\sum_{i=u}^{v} k_i = k$ and $u\le i\le v$, and r_i steps of operation are needed for the probabilistically fulfillment of conditions in step i, then a precise estimation of the averaged computational complexity or cost in MD5 operations can be given for the collision searching algorithm as $N_C=\frac{1}{64}\sum_{i=u}^{v} r_i 2^{k-\sum_{j=u}^{i-1}k_j}$, where $\sum_{j=u}^{u-1}k_j=0$, provided that there are enough free message bits (no less than the number of conditional bits) to be searched.

Generally in MD5, the computational complexity is determined by the maximal group $G_{\max}$ of conditional bits which is usually the final group being composed of the MD5 steps from b_6 or a_7 to the final b_{16}, i.e. $u=b_6$ or a_7 (MD5 step 24 or 25), $v=b_{16}$ (MD5 step 64). By this deduction and further assume there are r_0 steps of operation for the search of free message bits before the computation of the step b_6 or a_7, the steps of operation for step $i(\ge u)$ will be $r_i=r_0+i-u+1$ and so on. Then, we have $N_C=\frac{1}{64}\sum_{i=u}^{64}(r_0+i-u+1)2^{k-\sum_{j=u}^{i-1}k_j}$, where u =24 or 25 (depending on the grouping scheme). For the MD5 collision searching algorithm in this paper, $r_0=3$. Hence for the first block, $N_{C1}=2^{27}\times\frac{4}{64}+2^{25}\times\frac{5}{64}+\cdots+2^{1}\times\frac{42}{64}\approx 2^{23.1}$, for the second block $N_{C2}=2^{23}\times\frac{4}{64}+2^{22}\times\frac{5}{64}+\cdots+2^{1}\times\frac{42}{64}\approx 2^{19.1}$.

Appendix C The Differential Paths

Table 6: The Basic Differential Path Using $\Delta^{+}m_8 = 2^{31}$ (Block1).

t	Bits Qt: $a_0 \ldots a_{31}$	#
1~6	******** ******** ******** ********	0
7	******1* ******** ******** ********	1
8	****0*0* ******** **0***** ****1***	4
9	****1^+^ ^^^^**0*1 0*0^^^^^ ^^^^0^^*	23
10	****01-- --+**1^0 1*+++++++ +++++++-*	24
11	^**^11+0 001^^+++ -1-00111 0000000*	29
12	+**-++-0 110++000 00++++-1 1000101*	29
13	0**0011* 1**00111 0-10000* ****+01^	22
14	11*1110* +**10*** *011111* ***00--+	20
15	*1*-0*00 +******* *1****1* ***01000	13
16	*-1*1*10 0******* *0000*+* ***-*001	15
17	**0^+*+- 0******* *11111** ^*******	12
18	*^****** *01****0 *---+1^* ***^****	11
19	****^*^^ *11****0 ***01+** 00******	11
20	******** *++****- *^^10*** 11******	9
21	***0**** ******** *****^** --******	4
22	***1**** *^^****^ ******** *****00*	6
23	***+**** ******** ******** ^^****11*	5
24	******** ******** ******** *****++*	2
25	***^**** ******** *0****** ********	2
26	******** ******** *1****** *****^^*	3
27	******** ******** *+****** ********	1
28	******** ******** ******** ********	0
29	******** ******** *^****** *******0	2
30	******** ******** ******** ********	0
31	******** ******** ******** *******-	1
32~47	******** ******** ******** *******-	0
48~55	******** ******** ******** *******-	8
56	******** ******** ******** *******+	1
57	******** ******** ******** *******-	1
58	******** ******** ******** *******+	1
59	******** ******** ******** *******-	1
60	******** ******** ******** *******+	1
61	******** ******** ******** *******-	1
62	******** ******** ******** *******+	1
63	******** ******** ******** *******-	1
64	******** ******** ******** *******+	0

Table 7: The Basic Differential Path Using $\Delta^{+}m_i = 0, 0 \le i < 16$ (Block2).

t	Bits Qt: $a_0 \ldots a_{31}$	#
-3	******** ******** ******** *******+	0
-2	******** ******** ******** *******+	0
-1	******** ******** ******** *******+	1
0	******** ******** ******** *******+	1
1~31	******** ******** ******** *******+	31
32~47	******** ******** ******** *******+	0
48~63	******** ******** ******** *******+	16

Table 8: The Modified Differential Path With Additional Absorbing Bits (Block1).

t	Bits Qt: $a_0 \ldots a_{31}$	#
1~4	******** ******** ******** ********	0
5_t	00000000 00000000 00000000 00000000	32
6_t	11111111 11111111 11111111 11111111	32
7	******1* ******** ******** ********	1
8	****0*0* ******** **0***** ****1***	4
9	****1^+^ ^^^^**0*1 0*0^^^^^ ^^^^0^^*	23
10_t	*00*01-- --+**1^0 10+++++++ +++++++-0	28
11_t	^11^11+0 001^^+++ -1-00111 00000001	32
12	+**-++-0 110++000 00++++-1 1000101*	29
13	0**0011* 1**00111 0-10000* ****+01^	22
14	11*1110* +**10*** *011111* ***00--+	20
15	*1*-0*00 +******* *1****1* ***01000	13
16	*-1*1*10 0******* *0000*+* ***-*001	15
17	**0^+*+- 0******* *11111** ^*******	12
18	*^****** *01****0 *---+1^* ***^****	11
19	****^*^^ *11****0 ***01+** 00******	11
20	******** *++****- *^^10*** 11******	9
21	***0**** ******** *****^** --******	4
22	***1**** *^^****^ ******** *****00*	6
23	***+**** ******** ******** ^^****11*	5
24	******** ******** ******** *****++*	2
25	***^**** ******** *0****** ********	2
26	******** ******** *1****** *****^^*	3
27	******** ******** *+****** ********	1
28	******** ******** ******** ********	0
29	******** ******** *^****** *******0	2
30	******** ******** ******** ********	0
31	******** ******** ******** *******-	1
32~47	******** ******** ******** *******-	0
48~55	******** ******** ******** *******-	8
56	******** ******** ******** *******+	1
57	******** ******** ******** *******-	1
58	******** ******** ******** *******+	1
59	******** ******** ******** *******-	1
60	******** ******** ******** *******+	1
61	******** ******** ******** *******-	1
62	******** ******** ******** *******+	1
63	******** ******** ******** *******-	1
64	******** ******** ******** *******+	0

Table 9: The Modified Differential Path With Additional Absorbing Bits (Block2).

t	Bits Qt: $a_0 \ldots a_{31}$	#
-3	******** ******** ******** *******+	0
-2	******** ******** ******** *******+	0
-1	******** ******** ******** *******+	1
0	******** ******** ******** *******+	1
1~4	******** ******** ******** *******+	4
5_t	00000000 00000000 00000000 0000000+	32
6_t	11111111 11111111 11111111 1111111+	32
7~9	******** ******** ******** *******+	3
10_t	00000000 00000000 00000000 0000000+	32
11_t	11111111 11111111 11111111 1111111+	32
12~31	******** ******** ******** *******+	20
32~47	******** ******** ******** *******+	0
48~63	******** ******** ******** *******+	16

Notes: In Tables 6,7,8 and 9, '+' denote a positive flip(0→1), '-' a negative flip(1→0), 0(1) the conditional bit value, '^' the bit equal to the up bit, '! ' the bit not equal to the up bit, '*' the free bit, 't' the MD5 step, '#' the number of conditions for each step.