A New Collision Differential For MD5 With Its Full Differential Path

Tao Xie[+] DengGuo Feng FangBao Liu
(hamishxie@vip.sina.com)

The State Key Laboratory on Information Security, Chinese Academy of Science
The Center for Soft-Computing and Cryptology, NUDT, China

【**Abstract**】Since the first collision differential with its full differential path was presented for MD5 function by Wang et al. in 2004, renewed interests on collision attacks for the MD family of hash functions have surged over the world of cryptology. To date, however, no cryptanalyst can give a second computationally feasible collision differential for MD5 with its full differential path, even no improved differential paths based on Wang's MD5 collision differential have appeared in literature. Firstly in this paper, a new differential cryptanalysis called signed difference is defined, and some principles or recipes on finding collision differentials and designing differential paths are proposed, the signed difference generation or elimination rules which are implicit in the auxiliary functions, are derived. Then, based on these newly found properties and rules, this paper comes up with a new computationally feasible collision differential for MD5 with its full differential path, which is simpler thus more understandable than Wang's, and a set of sufficient conditions considering carries that guarantees a full collision is derived from the full differential path. Finally, a multi-message modification-based fast collision attack algorithm for searching collision messages is specialized for the full differential path, resulting in a computational complexity of 2^{36} and 2^{32} MD5 operations, respectively for the first and second blocks. As for examples, two collision message pairs with different first blocks are obtained.

Key Words：MD5, differential cryptanalysis, collision attacks, collision differential, differential path design

1. Introduction

A hash function is a cryptographic primitive which computes a fixed size message digest from arbitrary size messages. The output value is used usually as the digital digest of the input message, so that a change of a single bit in the input would deterministically cause on average a half of the digest bits to change. Therefore, a cryptographic hash function is essentially a type of irreversible one-way functions built with nonlinear operations. MD2, MD4 and MD5 are 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 [1]. The widespread popularity of the MD family of hash functions is a testament to their innovative and successful design. Indeed, MD4 in particular has been used as the basis for the design of many other hash functions (including MD5, SHA-1, RIPEMD) and, MD5 is one of the most widely used hash functions in the world today, specially designed for 32-bit machine. MD5 is deployed in many applications, including SSL/TLS, IPSec, and many other cryptographic protocols. It is also commonly-used in implementations of time-stamping mechanisms, commitment schemes, and integrity-checking applications for online software, distributed file systems, and random-number generation.

There exists no sound mathematical security definition for cryptographic hash functions like MD5, but instead their security rely on the following intuitive notions: for a hash function $y = h(x)$ with domain $x \in D$ and range $y \in R$, we require the following three properties.

Pre-image Resistance: For a given $y \in R$, it should be hard to find a $x \in D$ such that $h(x) = y$.

Second Pre-image Resistance: For a given $x \in D$, it should be hard to find a distinct $x^{\bullet} \in D$ such that $h(x) = h(x^{\bullet})$.

Collision Resistance: It should be hard to find distinct $x, x^{\bullet} \in D$ such that $h(x) = h(x^{\bullet})$.

A hash function for which it is hard to find either a preimage or a second preimage is sometimes called a one-way hash function, whereas a hash function that possesses all the three properties described above is called a collision-resistant hash function. In particular, the term "hard" is used above as "computationally infeasible".

This paper mainly focused on collision attacks on MD5. While it is postulated in RFC [2]that the difficulty of coming up with two messages having the same message digest is on the order of 2^{64} operations, researches on collision attacks have never stopped since the publication of MD5. In 1992, Berson[3] 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[4] found pseudo-collisions for the compression function of MD5 with different initial values but common input. In 1996, Dobbertin[5,6] constructed collisions of the MD5 compression function, that is, MD5 collisions with a wrong initial value. In 2004, Wang et al.[7,8] succeeded in producing real collisions for the full MD5 hash function as well as collisions in a host 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[9]. 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 it can be improved, and how it can be generalized [10].

Fundamentally, Wang's differential collision attack is a hybrid differential cryptanalysis which takes advantages of both the modular difference and the XOR difference together. The modular differential cryptanalysis has been early used in the collision attacks of MD family of hash functions, with modular addition as the confusion method, including MD5, SHA0 and RIPEMD[3,11,12,13]. The XOR differential cryptanalysis is originally proposed for the cryptanalysis of the DES like block ciphers, with bitwise XOR operation as the confusion method [14]. Wang's main contribution to MD5 attack is that, they have found a full two-block collision differential with its full differential path, which is computationally feasible, and for the first time constructed a full message collision pair for MD5. Wang 's attack on MD5 has called its security especially in digital signature into question. To date, however, the method used by Wang et al has been fairly difficult to grasp, and furthermore, some small perhaps deliberately made errors (bugs) in the literature [8], might have constituted the appeal to have frustrated other cryptanalysts to grasp their technique. Since the publication of [8], quite a number of researchers have worked on the optimization of the set of sufficient conditions and hence the collision search algorithms, resulted in a great improvement on the collision search efficiency to 2^{30} MD5 operations as declared [10,15]. What is really inexplicable consists in that, no second differential path which is more computationally feasible than the original one has been published as yet, say nothing of the second collision differential with its full differential paths. The authors of this paper, however, believed that, there must exist other more efficient differential paths corresponding to Wang's collision differential and, even other collision differentials different from Wang's as well. In this paper, we present a new collision differential completely different from Wang's, offer a full differential path for this new collision differential, and the set of sufficient conditions to maintain the full differential path is also derived from the full differential path.

The rest of this paper is organized as follows: In section 2, we give a concise description of the MD5 algorithm and some notations used in this paper. Then in section 3, the definitions for XOR difference, modular difference as well as signed difference are given and a differentiation is provided, that the signed difference of a message word pair is a unique definition of both the corresponding XOR difference and modular difference. And in section 4, a new collision differential for MD5 with its full differential path is introduced, some principles or considerations on how to find collision differentials and design full differential path are made public, and the specific rules of how to generate or eliminate signed differences, which are implicit in the auxiliary functions, are derived. In section 5, a multi-message modification-based fast collision attack algorithm is specialized for the set of conditions which are derived directly from the full differential path.

Finally, in section 6, we summarize and conclude the paper, offer some suggests for verification and evaluation of collision differentials, and some remarks on differential collision attacks are also given.

2. The MD5 Message Digest Algorithm

There are several types of hash function construction methods, respectively based on block ciphers, modular arithmetic, knapsack problems, cellular automaton, algebraic matrices, or specially designed for message hashing which is called dedicated hash functions, like the MD family of hash functions. Dedicated hash functions are practically built on the Merkle-Damgard theory, which says that collision resistance of the compression function implies collision resistance of the hash function [17,18]. 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 a 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 variables (including four 32-bit words) which is updated during the processing of each block, H_0 is the initial value IVs specified in MD5 algorithm, and the final H_t is the hash value that we expect to obtain. The concrete padding rule is omitted here, since it has no influence on our attack.

The processing of the ith block $f(H_i, M_i)$ involves four round functions FF, GG, HH and II as follows: $H_{i+1} = f(H_i, M_i) = H_i + II(M_i, HH(M_i, GG(M_i, FF(M_i, H_i))))$.

The round functions are similar to one another in structure. The chaining variable H_i is treated as four-element shift register, with each element being one 32-bit word wide. The elements are referred to as a, b, c and d. 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 operation, in each step operation the register is used with one word from M_i. The step operation is formulated as a system of equations: $a = b + ((a + \Phi_j(b,c,d) + w_j + t_j) <<< s_j)$, $0 \le j < 64$. Where $\Phi_j(X,Y,Z)$ is an auxiliary function which varies from round to round; w_j is a word chosen from M_i, t_j and s_j are constant parameters associated with step j; and $<<< s_j$ signifies a s_j-bit left circular shift of a word. Note that each step involves four modular additions ($\bmod 2^{32}$), 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 (the values at the start of the compression function) to their final values (obtained after 64 steps), so that $f(H_i, M_i)$ cannot be inverted.

Here we define $\sum a_j = a + \Phi_j(b,c,d) + w_j + t_j$ for the extra conditions' derivation late in section 5.

The auxiliary functions $\Phi_j(X,Y,Z)$ each take three 32-bit words from the register of chaining variable and produce one 32-bit word as output. The auxiliary functions for each round are given as follows:

$\Phi_j(X,Y,Z) = F(X,Y,Z) = (X \wedge Y) \vee (\overline{X} \wedge Z), 0 \le j < 16$;

$\Phi_j(X,Y,Z) = G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \overline{Z}), 16 \le j < 32$;

$\Phi_j(X,Y,Z) = H(X,Y,Z) = X \oplus Y \oplus Z,\ 32 \le j < 48$;

$\Phi_j(X,Y,Z) = I(X,Y,Z) = Y \oplus (X \vee \overline{Z}), 48 \le j < 64$.

Where X, Y, Z are 32-bit words. The four words of the chaining variable register are initialized as $a = 0\text{x}67452301,\ b = 0\text{xefcdab89},\ c = 0\text{x}98\text{badcfe},\ d = 0\text{x}10325476.$

For a detailed description of MD5 algorithm, please refer to [2].

3. XOR Differential Plus Modular Differential = Signed Differential

The first published effort appears to have been the cryptanalysis of a block cipher called FEAL by Murphy [19]. This is followed by a number of papers by Biham and Shamir, who demonstrated this form of attack on a variety of encryption algorithms and hash functions; their results are summarized in [14]. In its basic version, differential cryptanalysis is a method that analyses the effect of particular differences in plaintext pairs on the difference of the resultant ciphertexts. These differences can be used to assign probabilities to the possible keys and to locate the most probable key. Although this scheme, as reported in[14], can successfully cryptanalyze DES with an effort on the order of 2^{47}, but the need to find 2^{47} chosen plaintexts make this attack of only theoretical interest. Usually, the difference between message word pair can be defined as XOR difference or modular difference. The XOR difference of message word pair is defined as a bitwise XOR operation of every two corresponding bits, while the modular difference is defined as a modular integer subtraction of the two binary message words. Berson might be the first to have made modular 2^{32} differential cryptanalysis on MD5 function [3,11~13]. Modular differential was employed late in analyzing the propagation properties of static difference of the quadratic function in RC6, which has lager propagation probability than the static XOR differential [20].

For any cryptographic system, Claude Shannon suggests two methods for frustrating statistical cryptanalysis: diffusion and confusion. In a binary block cipher, diffusion is achieved usually by repeatedly performing some permutations on the data followed by applying a function to that permutation, while confusion is achieved usually by the use of a complex substitution algorithm. In addition, both modular arithmetic operation and XOR operation are two types of confusion method that are widely employed in practical block ciphers. Whether XOR differential or modular differential or even both differentials should be chosen as the differential cryptanalytic tool, it depends on what confusion and diffusion methods are used in the cipher to be cryptanalyzed. The XOR difference is suitable for cryptanalysis of cipher in which the XOR operation is used as a confusion method, while the modular difference is adapted for cryptanalysis of cipher in which the modular addition is used as a confusion technique.

A slight mathematical analysis reveals that, a modular difference might map to many different XOR differences, and a XOR difference might also map to many different modular difference, too. This gives a proof that either modular difference or XOR difference can not separately be a proper measure of message word pair in differential cryptanalysis. Therefore, we define the third type of difference for message word pair, that is the bitwise difference representation between the binary message pair, called as signed bitwise difference or directly as signed difference. Both the modular difference and XOR difference of the message word pair are completely incorporated into the corresponding signed difference.

Given $w = 10$-bit messages as $\mathbf{X}_1$=1001000101 and $\mathbf{X}_2$=0000111010, the XOR difference denoted as $\bigtriangledown \mathbf{X}$, the modular difference denoted as $\bigtriangleup \mathbf{X}$ and the signed difference denoted as $\diamondsuit \mathbf{X}$ between $\mathbf{X}_1$ and $\mathbf{X}_2$ are formulated and computed, respectively as follows:

$\bigtriangledown \mathbf{X} = \mathbf{X}_1 \oplus \mathbf{X}_2 = \Big\|_{i=1}^{w} \mathbf{X}_{1,i} \oplus \mathbf{X}_{2,i}$ =1001111111;

$\bigtriangleup \mathbf{X} = (\mathbf{X}_1 - \mathbf{X}_2) \bmod (2^{10})$=1000001011;

$\diamond \mathbf{X} = \coprod_{i=1}^{w} \left(\mathbf{X}_{1,i} - \mathbf{X}_{2,i}\right)$=1001-1-1-11-11.

By mathematically analyzing the relation between the signed difference and the XOR difference plus modular difference, the following relations are obtained:

$$\triangle \mathbf{X} = \left(\mathbf{X}_1 - \mathbf{X}_2\right) \bmod \left(2^w\right) = \sum_{i=1}^{w} \left(\mathbf{X}_{1,i} - \mathbf{X}_{2,i}\right) \bullet 2^{i-1} \bmod \left(2^w\right) \overset{\bmod\left(2^w\right)}{\equiv} \coprod_{i=1}^{w} \left(\mathbf{X}_{1,i} - \mathbf{X}_{2,i}\right) = \diamond \mathbf{X}$$

That is, a bijective mapping does exist between the signed difference $\diamond \mathbf{X}$ and the XOR difference $\triangledown \mathbf{X}$ plus the modular difference $\triangle \mathbf{X}$, considering in a w-bit word, a MSB = +1 is equivalent to MSB = -1 within modulo 2^w. Namely, given a specific signed difference $\diamond \mathbf{X}$, the corresponding XOR difference and modular difference are both uniquely determined and vice versa. Many useful properties in differential path designing can be derived from the relation given above. Since the space is limited here, a detailed mathematic analysis on the full properties of the signed difference is going to be published as a separate paper.

For simplicity, we omit those "0"s in the signed difference $\diamond \mathbf{X}$, but index the signed difference bits (+1 or -1) with their bit position identity instead, starting from 1 (the LSB) in the word. Using 10-bit word as an example, the signed difference (1001-1-1-11-11) can be indexed as (10,7,-6,-5,-4,3,-2,1).

In differential cryptanalysis, a *differential* is a pair formed of an input difference and an output difference, whereas a differential *characteristic* is a sequence of differences where the difference after each round is given. A signed differential characteristic for a hash function is a sequence of signed differences where the signed differences after each step is given for an input difference. A signed differential characteristic is also called a signed differential path. For hash function collision attacks, it is to design a computationally feasible signed differential path which leads to a collision.

4. A New Collision Differential for MD5

4. 1 Some Principles for Collision Differential Finding

Single block or multi-block collision differentials always exist for any iterated hash function based on Merke-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 design or find an input message difference which can be controlled in the difference propagation during the computation of successive step operations, so that the difference can be eliminated by a single or multiple iterations in the final (four steps for MD5) steps.

Given an input message difference, if a full differential path can be computed to a collision, then it is called a feasible collision differential, hence the corresponding differential path a feasible differential path, otherwise we call it an infeasible collision differential. If the probability to satisfy the set of necessary conditions that maintain the differential path is computationally feasible, then we call it a computationally feasible collision differential, hence a corresponding computationally feasible differential path, otherwise a computationally infeasible collision differential and 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 intermediate states in round 2 can also be directly satisfied by these free message words. Wang has given the first collision differential [8] which properly meets the principles described above.

Wang's first two-block collision differential is listed below, and it is computationally feasible.

$\Delta M_0 = \left(M_0^{\bullet} - M_0\right) \bmod (2^{32})$ = (0,0,0,0, 2^{31},0,0,0,0,0,0, 2^{15},0,0, 2^{31},0);

$\Delta M_1 = \left(M_1^{\bullet} - M_1\right) \bmod (2^{32})$ = (0,0,0,0, 2^{31},0,0,0,0,0,0,- 2^{15},0,0, 2^{31},0);

$\Delta H_1 = \left(\Delta a, \Delta b, \Delta c, \Delta d\right) = \left(2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25}\right).$

In this section, we give a second collision differential, which is also computationally feasible, as follows:

$\Delta M_0 = \left(M_0^{\bullet} - M_0\right) \bmod (2^{32})$ = (0,0,0,0,0,0,-2^8,0,0, 2^{31},0,0,0,0,0, 2^{31});

$\Delta M_1 = \left(M_1^{\bullet} - M_1\right) \bmod (2^{32})$ = (0,0,0,0,0,0,2^8,0,0, 2^{31},0,0,0,0,0, 2^{31});

$\Delta H_1 = (\Delta a, \Delta b, \Delta c, \Delta d) = \left(2^{31} - 2^{23}, 2^{31} - 2^{23}, 2^{31} - 2^{23}, 2^{31} - 2^{23}\right)$.

Note that, the message block differential index starts at 0 and $0th$ word differential start at the left. For example, for ΔM_0 given by the author, the sixth word difference is -2^8 and the ninth word difference is 2^{31}.

4．2 Some Principles For Differential Path Designing

When a collision differential is found to be computationally feasible, the next work is to design a feasible differential path which leads to a collision. The basic design criterion is to decrease differentials in all rounds except round 1. In addition, less the differentials in round 1, better the differential path will be, provided it works. To design a good differential path with computational feasibility, the following principles could be benefited from if observed.

1) The presence of a differential path for round 1 is critical to the feasibility of the collision differential;
2) The differential path in round 2 must take only a small upper part of steps, so as to make it connect with the differential path in round 1;
3) More free message words before the start differential in round 1, better the differential path will be;
4) The path in round 2 must not make up a feedback with the path in round 1 through those free message words (directly related to the free state variables), so that multi-message modification works;
5) The differentials in the path section within the round 3 and 4 must be as small as possible so that only the path section around the final 4 steps have differentials (difference diffusion) if unavoidable;
6) Deduce a differential path bottom-up in a reverse computation way, starting from the inner collision step in round 2, and up to the step that is 4 or 5 steps away from the start differential in round 1, then compute the differentials up-down from the start differential in round 1 to meet with the bottom-up differential path;
7) Use the properties implicit in the signed difference in each backward or forward iteration, to generate new signed differences or eliminate unwanted signed differences, and this is the basic rule for any hash function;
8) Use the generation and elimination rules implicit in the auxiliary functions in each backward or forward iteration, and these are the special rules for a particular hash function.

4．3 MD5 Differential Iteration

A full MD5 differential path is composed of 64 consecutive differential steps of iteration.

Four consecutive state variable differentials ($\diamond a$, $\diamond d$, $\diamond c$ and $\diamond b$) are employed as input to the round function to generate the next output variable differential $\diamond a^{\bullet}$, we call this computation a MD5 differential iteration step.

In a MD5 iteration step, the signed differences in the next output variable can be :

1) Directly derived from the signed difference in the top state variable;
2) Directly derived from the signed difference in the last state variable;
3) Indirectly generated by the auxiliary functions used in the step, provided that at least one signed difference exists at the same bit position in the last three state variables:
   i) A signed difference can be generated in many ways;
   ii) Actually, any signed differences can be generated in the last three state variables in an almost magic way, by utilizing the properties implicit in the signed difference;
   iii) Almost all intelligence of differential path designing is focused here.
4) The signed difference generated by the auxiliary functions can be used to eliminate those derived directly from the top and last state variables.

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 signed differences, that are required by the next output variable but can not be directly derived from the top and last state variable; on the other hand, to employ the auxiliary functions to generate the complementary differentials for those directly derived from the top and last state variable, but not required by the next output variable, so that two complementary signed differences be eliminated.

4．4 The Signed Difference Rules Due to Auxiliary Functions

Part of the techniques that is shrouded in the differential path finally falls on the auxiliary functions. The auxiliary functions contribute to the generation and elimination of signed differences by bit-wise Boolean operation, and the properties implicit in the signed difference are employed to provide the context for the auxiliary functions as required. Table 1 only gives the rules of signed difference generation and elimination for the auxiliary function $F(X,Y,Z)=(X\wedge Y)\vee(\overline{X}\wedge Z)$ in round 1, the rules corresponding to other three auxiliary functions can be similarly analyzed.

Table 1：The signed difference rules derived from the function $F(X,Y,Z)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a																										
d	+1	+1	+1	+1	-1	-1	-1	-1	0 1	1 0	1 0	0 1	+1	+1	-1	-1	+1	+1	-1	-1	0 1	0 1			+1	-1
c	+1	+1	-1	-1	+1	+1	-1	-1	+1	+1	-1	-1	1 0	0 1	0 1	1 0	+1	-1	+1	-1	0 1	0 1	+1	-1		
b	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	0 1	0 1	0 1	0 1	+1	-1	1 0	1 0	0 1	0 1
$a^{\bullet}$	+1	+1	0	0	0	0	-1	-1	+1 1	+1 0	-1 0	-1 1	+1 0	+1 1	-1 1	-1 0	+1	+1 -1	-1 +1	-1	±1	±1	+1	-1	+1	-1

Some comments for table 1 are summarized below:

1) The three consecutive state variables each have signed difference at the same bit position: there are totally 8 situations, the first two situations produce a positive signed difference +1 and the final two situations a negative difference -1, whereas the other 4 situations will result in a bit value 0.
2) Two of the three consecutive state variables have signed difference at the same bit position: there are totally 12 situations, a signed difference +1 or -1 can be generated or eliminated in the first 8 situations, depending on the value of the bit that has no signed difference; whereas the final 4 situations will only result in a signed difference +1 or -1, and no difference elimination occurs.
3) Only one of the three consecutive state variables has signed difference at the same bit position: there are totally 6 situations, these situations can produce a signed difference +1 or -1 or directly result in a difference elimination, depending on the values of the other two bits that have no signed difference.

One particular situation which should be paid much attention to is that, when the top and last state variables have signed difference at the same bit position, there are totally 2 situations, i.e. the two signed differences being equal or complementary each other. If the objective in this step operation is to eliminate the signed difference due to the top and last state variable, and when two signed differences are equal, the two bits at the same position in the middle two state variables as it is concerned with, need to be specified so that a complementary difference will be generated to eliminate the top bit difference. This will, however, unfortunately result in new signed differences in the last two state variables. When the two bit differences are complementary each other, two bits at the same position in the middle state variables as it is concerned with, can be specified so that a complementary difference can be generated to eliminate the top bit difference but without introducing any new differences in the last two state variables.

In table 1, $a^{\bullet}$ is the next variable generated by the three consecutive state variables d , c and b according to the auxiliary function $F(b,c,d)=(b\wedge c)\vee(\overline{b}\wedge d)$, i.e. $a^{\bullet}=F(b,c,d)$. Where, +1 or -1 denote signed difference, unsigned numbers (1, 0) and the space represent no difference bit value or bit to be specified. The option for the next variable $a^{\bullet}$ if available, depends on the value of the corresponding bits without no

difference.

Following the differential path designing principles recommended for MD5, by utilizing the generation and elimination rules implicit in the signed differential and auxiliary functions, we have designed a full differential path for the collision differential given in section 4.1, they are listed in table 3 and table 5. Table 4 and table 6 are the set of sufficient conditions derived respectively from table 3 and table 5, message block pairs with the given collision differential can keep control of the differential paths till collision occurs, if the set of sufficient conditions are satisfied by the input messages.

In table 3, the first column denote the step operation number, the second column X_i denotes the 32-bit output variable of the step operation, the third column denotes the message word for M_0 in each step, the fourth column denotes the shift rotation number, the fifth column denotes the modular difference in the message word, the sixth column $\triangle X_i$ denotes the modular difference in the step operation output, the seventh column $\diamond X_i$ denotes the step operation output for M_0^* by signed differential. The empty items and unlisted steps have zero difference in the fifth and sixth columns. The same denotations are employed for table 5 which is specified for the iteration of the second message block M_1. By table 4 and 6, we count up the number of conditions for the first and second message blocks to be 222 (including 13 extra conditions) and 321 (including 15 extra conditions), respectively.

5. Multi-Message Modification Algorithm and Collision Pairs

The differential path shown in table 3 starts at the 7th step in round 1, hence 4 message words m_1, m_2, m_3 and m_4 remain free. This is the context that multi-message modification technique can take to improve the collision attack efficiency, by directly satisfying some necessary conditions corresponding to those free message words in round 2. The effect of new changes on the corresponding output variables in round 1 can be absorbed by computing new message words for successive steps, this method is due to [12], and further developed in [10,15,16]. We give the fast collision attack algorithm specialized for our collision differential path, based on the improved multi-message modification method.

5.1 Fast Attack Algorithm For The First Block

Step1: Select random 32-bit value for the output variables c_1 and b_1, and randomly select 32-bit value for the output variables from a_2 to b_4 but ensure that all the conditions in table 4 hold;

Step2: By the output values from c_1 to b_4 given in step1, message words from m_6 to m_{15} are computed by reformulating the step operation equation in the same way as follows:

$$m_6 = RR(c_2 - d_2, 17) - F(d_2, a_2, b_1) - c_1 - 0\text{xa8304613};$$

Step3: Randomly select 32-bit value for a_5, but make the conditions $a_{5,9} = 0$, $a_{5,23} = 1$ and $a_{5,27} = 1$ satisfied;

Step4: To compute d_5. If the condition $d_{5,23} = 0$ does not hold, then make $d_{5,23} = 0$ by modifying m_6, and this is followed by modifying c_1 to counteract the changes induced by the modified m_6;

$$m_6 = RR(d_5 - a_5, 9) - G(a_5, b_4, c_4) - d_4 - 0\text{xc040b340};$$
$$c_1 = RR(c_2 - d_2, 17) - F(d_2, a_2, b_1) - m_6 - 0\text{xa8304613};$$
$$d_5 = RL(G(a_5, b_4, c_4) + d_4 + m_6 + 0\text{xc040b340}, 9) + a_5;$$

Step5: To compute c_5. If the conditions $c_{5,27} = d_{5,27}$ and $c_{5,32} = 0$ do not hold, then modify $b_{2,18}$ or both

$b_{2,13}$ and $a_{3,13}$ together. By the modified b_2 and a_3, the message word m_{11} is recomputed, and followed by a update of c_5;

$$m_{11} = RR(b_3 - c_3, 22) - F(c_3, d_3, a_3) - b_2 - 0\text{x}895\text{cd}7\text{be};$$
$$c_5 = RL(G(d_5, a_5, b_4) + c_4 + m_{11} + 0\text{x}265\text{e}5\text{a}51, 14) + d_5;$$

Step6: Randomly select 32-bit value for b_5, but make the conditions $b_{5,23} = c_{5,23}$ and $b_{5,32} = 0$ satisfied, then make a update of $m_0, m_1, a_1, d_1, m_2, m_3, m_4$ and m_5 in sequence;

$$m_0 = RR(b_5 - c_5, 20) - G(c_5, d_5, a_5) - b_4 - 0\text{xe}9\text{b}6\text{c}7\text{aa};$$
$$m_1 = RR(a_5 - b_4, 5) - G(b_4, c_4, d_4) - a_4 - 0\text{xf}61\text{e}2562;$$
$$a_1 = RL(F(b_0, c_0, d_0) + a_0 + m_0 + 0\text{xd}76\text{aa}478, 7) + b_0;$$
$$d_1 = RL(F(a_1, b_0, c_0) + d_0 + m_1 + 0\text{xe}8\text{c}7\text{b}756, 12) + a_1;$$
$$m_2 = RR(c_1 - d_1, 17) - F(d_1, a_1, b_0) - c_0 - 0\text{x}242070\text{db};$$
$$m_3 = RR(b_1 - c_1, 22) - F(c_1, d_1, a_1) - b_0 - 0\text{xc}1\text{bdceee};$$
$$m_4 = RR(a_2 - b_1, 7) - F(b_1, c_1, d_1) - a_1 - 0\text{xf}57\text{c}0\text{faf};$$
$$m_5 = RR(d_2 - a_2, 12) - F(a_2, b_1, c_1) - d_1 - 0\text{x}4787\text{c}62\text{a};$$

Step7: To compute a_6. If the condition $\sum a_{6,27} = 1$ does not hold, then modify both $d_{2,7}$ and $c_{2,7}$ together. By the modified d_2 and c_2, m_5 is recomputed under the pre-specified extra condition $a_{2,7} = b_{1,7}$, while m_6 is not changed. Recompute a_6 and make a update of m_7, m_8, m_9 and m_{10} in sequence, so that b_2, a_3, d_3 and c_3 keep unchanged;

$$m_5 = RR(d_2 - a_2, 12) - F(a_2, b_1, c_1) - d_1 - 0\text{x}4787\text{c}62\text{a};$$
$$a_6 = RL(G(b_5, c_5, d_5) + a_5 + m_5 + 0\text{xd}62\text{f}105\text{d}, 5) + a_5;$$

Step8: Go on with next step operations that have not been processed above, go to step6 provided that at least one condition in table 4 does not hold; if the number of iterations exceeds a pre-specified bound, then go to step 1; if all conditions in table 4 hold, store the first message block and output aa_0, bb_0, cc_0 and dd_0 to the second message block attack algorithm as chaining value, then stop.

5.2 Fast Attack algorithm for the second block

Step1: Randomly select 32-bit value for the output variables from a_1 to c_4 but ensure that all the conditions in table 6 hold;

Step2: By the output values from a_1 to c_4 given in step1 and the initial value aa_0, bb_0, cc_0 and dd_0 obtained in the fast attack algorithm for the first block, message words from m_0 to m_{14} are computed by reformulating the step operation equations in sequence, in the same way as follows:

$$m_0 = RR(a_1 - bb_0, 7) - F(bb_0, cc_0, dd_0) - aa_0 - 0\text{xd}76\text{aa}478;$$

Step3: Randomly select 32-bit value for the output variable b_4, but ensure that all the conditions for b_4 in table 6 hold, then compute m_{15} by reformulating the step operation equation as follows:

$$m_{15} = RR(b_4 - c_4, 22) - F(c_4, d_4, a_4) - b_3 - 0\text{x}49\text{b}40821;$$

Step4: To compute a_5. If conditions $a_{5,9} = 1, a_{5,23} = 1$ and $a_{5,27} = 1$ do not hold, modify $d_{1,2}$ and $d_{1,31}$ or both

$d_{1,16}$ and $c_{1,16}$ together. By the modified d_1 or c_1, a update of m_1, m_2, m_3, m_4, m_5 or m_6 is made in sequence, and this is followed by a recomputation of a_5 according to the updated m_1;

$$m_1 = RR(d_1 - a_1, 12) - F(a_1, bb_0, cc_0) - dd_0 - 0\text{xe8c7b756};$$

$$a_5 = RL(G(b_4, c_4, d_4) + a_4 + m_1 + 0\text{xf61e2562}, 5) + b_4;$$

Step5: To compute d_5. If condition $d_{5,23} = 0$ does not hold, then modify $c_{2,31}$. By the modified c_2, m_6 is updated, hence d_5 is recomputed, and this is followed by a sequential update of m_7, m_8, m_9 and m_{10};

$$m_6 = RR(c_2 - d_2, 17) - F(d_2, a_2, b_1) - c_1 - 0\text{xa8304613};$$

$$d_5 = RL(G(a_5, b_4, c_4) + d_4 + m_6 + 0\text{xc040b340}, 9) + a_5;$$

Step6: To compute c_5. If conditions $c_{5,27} = d_{5,27}$ and $c_{5,32} = 0$ do not hold, then modify $b_{3,3}$ or both $b_{2,18}$ and $a_{3,18}$ together. By the modified b_2 or b_3, m_{11} is updated, hence c_5 is recomputed, and this is followed by a sequential update of m_{12}, m_{13}, m_{14} and m_{15}, or $m_7, m_8, m_9, m_{10}, m_{11}$ and m_{12} as well;

$$m_{11} = RR(b_3 - c_3, 22) - F(c_3, d_3, a_3) - b_2 - 0\text{x895cd7be};$$

$$c_5 = RL(G(d_5, a_5, b_4) + c_4 + m_{11} + 0\text{x265e5a51}, 14) + c_4;$$

Step7: To compute b_5 .If conditions $b_{5,23} = c_{5,23}$ and $b_{5,32} = 0$ do not hold, then go to step3;

Step8: To compute a_6. If condition $\sum a_{6,27} = 1$ does not hold, then go to step3;

Step9: To compute d_6. If condition $\sum d_{6,23} = 0$ does not hold, then modify both $c_{3,8}$ and $b_{3,8}$ together. By the modified c_3 and b_3, m_{10} is updated under the pre-specified extra condition $d_{3,8} = a_{3,8}$, while m_{11} is not changed. And then, d_6 is recomputed, this is followed by a sequential update of m_{12}, m_{13}, m_{14} and m_{15};

$$m_{10} = RR(c_3 - d_3, 17) - F(d_3, a_3, b_2) - c_2 - 0\text{xffff5bb1};$$

$$d_6 = RL(G(a_6, b_5, c_5) + d_5 + m_{10} + 0\text{x02441453}, 9) + a_6;$$

Step10: Go on with next step operations that have not been processed above, and go to step3 provided that at least one condition in table 6 does not hold; if the number of iterations exceeds a pre-specified bound, then go to step 1; if all conditions in table 6 hold, store the second message block and come to a full stop.

5.3 Computational Complexity Analysis

According to the fast attack algorithms for two message blocks and table 4 and table 6, there are totally 36 conditions that need to be probabilistically satisfied in round 2 to round 4 for the first block, whereas the probability of satisfying the condition ($d_{16,25}$ or $\sim b_{15,25}$)=1 exceed $1 - 0.5^2 = 0.75$, consequently the computational complexity of the first block does not exceed 2^{36} MD5 operations. There are totally 32 conditions that need to be probabilistically satisfied in round 2 to round 4 for the second block, whereas the probability of satisfying the condition ($d_{16,25}$ or $\sim b_{15,25}$)=1 exceeds $1 - 0.5^2 = 0.75$, hence the computational complexity of finding the second block does not exceed 2^{32} MD5 operations. Two collision pairs are presented in table 2, which are obtained by the fast attack algorithms described above.

Table2: Two pairs of collision blocks with the corresponding MD5 values (underlined bits with difference）

M_0	0x9d133b36, 0xec2a44c9, 0x7e32bcdd, 0x5498e911, 0x78b8c3b6, 0x1a629661, 0xb456c47f, 0xccec27cb, 0x7c1eede2, 0xd3934205, 0xfa24921c, 0x3bb373cc, 0x0aabfa31, 0xa7bc4d44, 0xba91559, 0x567d9653
M_1	0x1522683e, 0x3e598084, 0xbc74fad, 0x854881fa, 0xcec6c0fb, 0x9ee808b5, 0x1acbe9f8, 0xe77779b2, 0xc81afd90, 0xa85ec52a, 0x6d16ba45, 0x629b30e8, 0x6e00673, 0xa232e472, 0xedcaac9, 0xb7d754f6
M_0^*	0x9d133b36, 0xec2a44c9, 0x7e32bcdd, 0x5498e911, 0x78b8c3b6, 0x1a629661, 0xb456c37f, 0xccec27cb, 0x7c1eede2, 0x53934205, 0xfa24921c, 0x3bb373cc, 0xaabfa31, 0xa7bc4d44, 0xba91559, 0xd67d9653
M_1^*	0x1522683e, 0x3e598084, 0xbc74fad, 0x854881fa, 0xcec6c0fb, 0x9ee808b5, 0x1acbeaf8, 0xe77779b2, 0xc81afd90, 0x285ec52a, 0x6d16ba45, 0x629b30e8, 0x6e00673, 0xa232e472, 0xedcaac9, 0x37d754f6
	MD5 value: 0x5812e272 0x23d0af39 0xaa5d4744 0xe6eba22d
M_0	0x57d07f13, 0x52afeaaa, 0xaeb4dd08, 0x547968e8, 0x483c5bc6, 0x184e7b7a, 0x3c90d783, 0x06a13ef3, 0x58c55475, 0xc43133c9, 0xf1f92143, 0x740d40ae, 0x17a9f89d, 0x6062bd01, 0xa2adc70a, 0x9d38855d
M_1	0x3381ac8c, 0x3f126c23, 0x698959b9, 0xe8f77bb3, 0x7ca95c2b, 0x164f075a, 0x04be0ef7, 0xa585f0a8, 0x4c12fd8b, 0x7bdec156, 0x705cda20, 0x669f2d24, 0xe60c23b0, 0xf35c6097, 0xfa38665c, 0xbf1d3b8b
M_0^*	0x57d07f13, 0x52afeaaa, 0xaeb4dd08, 0x547968e8, 0x483c5bc6, 0x184e7b7a, 0x3c90d683, 0x06a13ef3, 0x58c55475, 0x443133c9, 0xf1f92143, 0x740d40ae, 0x17a9f89d, 0x6062bd01, 0xa2adc70a, 0x1d38855d
M_1^*	0x3381ac8c, 0x3f126c23, 0x698959b9, 0xe8f77bb3, 0x7ca95c2b, 0x164f075a, 0x04be0ff7, 0xa585f0a8, 0x4c12fd8b, 0xfbdec156, 0x705cda20, 0x669f2d24, 0xe60c23b0, 0xf35c6097, 0xfa38665c, 0x3f1d3b8b
	MD5 value: 0x2f9f3cf8 0xd16d7fb4 0xf7e726ec 0xbcb232a7

6 Summary and Conclusions

MD5 function is a widely deployed cryptographic algorithm for digital digest and message authentication, along with the evolution of techniques for collision differential path designing, more new feasible collision differentials will be found. A new collision differential with its full differential path might offer more new design criteria for the next generation of hash function, and raise a shocking alarm for those who are still working with MD5. In this paper, a new differential cryptanalysis called signed differential is proposed, some principles or recipes for MD5 collision differential finding and differential path designing are offered, the signed difference generation or elimination rules implicit in auxiliary functions are derived. Based on this background of knowledge, this paper presents a second two-block collision differential with its computationally feasible full differential path, the set of conditions sufficient to maintain the differential path is derived. By the fast collision attack algorithms, at most 2^{36} and 2^{32} MD5 operations are needed to obtain the first and second collision blocks, respectively.

It will continue to be an uneasy work and perhaps partly depends on your lucks to find a feasible collision differential, and furthermore, it is probably going to be a challenging work to design a full differential path with respect to a known collision differential. Whether a collision differential is feasible or not, it depends on if you can design a full differential path leading to a collision. The computational feasibility of a full differential path is determined by the number of necessary conditions that can only be probabilistically satisfied, or directly measured by the computational complexity. Given a full differential path, it will not be a difficult work to design a specialized (fast) collision attack algorithm, while the debugging process for collision attack takes pains.

A collision differential can be evaluated according to the following five factors:

1) Whether the differential path depends on the fixed IVs of hash function or not?
2) The number of bits with difference in collision blocks;
3) The number of blocks comprising of the collision differential;

4) The number of necessary conditions which must be satisfied to make collision;
5) The number of necessary conditions in all rounds except the first round, or the number of necessary conditions that can not be satisfied directly by message modification;

Considering practically cryptanalytic attacks, a differential path which does not rely on the fixed initial value IVs will obviously be better than that must rely on it; a collision differential which has less bits with difference will more easily be used to construct meaningful attacks; a collision differential with less blocks will be more efficient for practical attacks; less the conditions necessary to maintain the full differential path, higher the density of collision message will be; less the computational complexity of finding collision blocks, more probably an instant collision attack will succeed.

Reference

1. M.J.B. Robshaw. MD2, MD4, MD5, SHA and other Hash Functions. RSA Laboratories Technical Report TR-101,version 4.0, July 24, 1995.
2. Ron Rivest. The MD5 message-digest algorithm. Internet Request for Comment RFC 1321, Internet Engineering Task Force, April 1992.
3. T. A. Berson. Differential Cryptanalysis Mod 2^{32} with application to MD5. In Advances in Cryptology, Proceedings of EUROCRYPT'92, pages71~80, 1992.
4. B. den. Boer, A. Bosselaers. Collisions for the compression function of MD5, Advances in Cryptology, Eurocrypt'93 Proceedings, Springer-Verlag, 1994.
5. H. Dobbertin. Cryptanalysis of MD5 compress, presented at the rump session of Eurocrypt'96.
6. H. Dobbertin. The status of MD5 after a recent attack, CryptoBytes 2 (2), 1996, ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf.
7. 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.
8. X.Y. Wang, Hongbo Yu, How to Break MD5 and Other Hash Functions, EUROCRYPT 2005, LNCS 3494, pp.19-35, Springer-Verlag, 2005.
9. X.Y Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1, Crypt'2005, LNCS 3621, pp17~36.
10. J. Black, M.Cochran , T Highland. A study of MD5 Attacks: insights and improvements, FSE2006
11. E. Biham. On the Applicability of Differential Cryptanalysis to Hash Functions. In E.I.S.S Workshop on Cryptographic Hash Functions, pages25-27, 1992.
12. F. Chabaud, A. Faux. Differential Collision on SHA-0, CRYPTO1998, LNCS1462, pp.56~71,1998
13. E. Biham, R. Chen, *Near collision for SHA-0,* Advances in Cryptology, Crypto'04, 2004, LNCS 3152, pp. 290-305.
14. E. Biham, A. Shamir. Differential Cryptanalysis of the Data Encryption Standard, Springer-Verlag, 1993.
15. V. Klima. Finding MD5 collisions on a notebook PC using multi-message modifications. In International Scientific Conference Security and Protection of Information, May , 2005
16. Liang J. and Lai X.: Improved Collision Attack on Hash Function MD5, Cryptology ePrint Archive: Report 425/2005, 23 Nov 2005, http://eprint.iacr.org/2005/425.pdf.
17. I. B. Damgard. A design principle for hash functions, Advances in Cryptology,Crypto'89 Proceedings, Springer-Verlag, 1990.
18. Praveen Gauravaram,William Millan and Juanma Gonzalez Neito. Some thoughts on Collision Attacks in the Hash Functions MD5, SHA-0 and SHA-1. Cryptology ePrint Archive, Report 2005/391, 2005. Available at: http://eprint.iacr.org/.
19. S. Murphy. "The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts." Journal of Cryptology, No.3, 1990.
20. S. Cotini, R.L. Rivest, M.J.B. Robshaw, Y. Lisa Yin. Security of the RC6*TM* Block Cipher, http://www.rsasecurity.com/rsalabs/rc6/.

Table 3: The Differential Path For The First Block

Step	Chaining variables X_i	w_i	s_i	$\triangle w_i$	$\triangle X_i$	$\diamond X_i$ (signed difference)
6	d_2	m_5	12			
7	c_2	m_6	17	-2^8	-2^{25}	c_2[26,27,28,29,30,31, -32]
8	b_2	m_7	22		$-2^{18}-2^{25}$	b_2[19,20,21, 22,-23,-26]
9	a_3	m_8	7		-2^{18}	a_3[-19]
10	d_3	m_9	12	2^{31}	$2^1+2^5+2^{10}-2^{18}+2^{31}$	d_3[2,-6,-7,8,-11,…,-16,17,19,-20,32]
11	c_3	m_{10}	17		$-2^0-2^{23}-2^{28}-2^{31}$	c_3[-1,24,25,26,-27,-29, -32]
12	b_3	m_{11}	22		$2^{18}-2^{31}$	b_3[19,-32]
13	a_4	m_{12}	7		-2^{22}	a_4[-23]
14	d_4	m_{13}	12		$2^{13}+2^{17}$	d_4[14, -18, -19, -20, -21,22]
15	c_4	m_{14}	17		-2^8	c_4[-9]
16	b_4	m_{15}	22	2^{31}		b_4
17	a_5	m_1	5		-2^{26}	a_5[-27]
18	d_5	m_6	9	-2^8	2^{22}	d_5[23]
19	c_5	m_{11}	14			c_5
20	b_5	m_0	20			b_5
21	a_6	m_5	5		2^{31}	a_6[*32]
22	d_6	m_{10}	9			d_6
23	c_6	m_{15}	14	2^{31}		c_6
24	b_6	m_4	20			b_6
25	a_7	m_9	5	2^{31}		a_7
26	d_7	m_{14}	9			d_7
...	...	...	...	...	...	...
44	b_{11}	m_6	23	-2^8	2^{31}	b_{11}[*32]
45	a_{12}	m_9	4	2^{31}	2^{31}	a_{12}[*32]
46	d_{12}	m_{12}	11		2^{31}	d_{12}[*32]
47	c_{12}	m_{15}	16	2^{31}	2^{31}	c_{12}[*32]
48	b_{12}	m_2	23		2^{31}	b_{12}[32]
...	...	...	...	...	...	...
57	a_{15}	m_8	6		2^{31}	a_{15}[-32]
58	d_{15}	m_{15}	10	2^{31}	2^{31}	d_{15}[32]
59	c_{15}	m_6	15	-2^8	$-2^{23}+2^{31}$	c_{15}[-24,-32]
60	b_{15}	m_{13}	21		$-2^{23}+2^{31}$	b_{15}[-24, 32]
61	$aa_0=a_{16}+a_0$	m_4	6		$-2^{23}+2^{31}$	**a_{16}[24, -25, -32], $aa_0'=aa_0$[-24, *32]**
62	$dd_0=d_{16}+d_0$	m_{11}	10		$-2^{23}+2^{31}$	**d_{16}[-24, 32], $dd_0'=dd_0$[24, -25, 32]**
63	$cc_0=c_{16}+c_0$	m_2	15		$-2^{23}+2^{31}$	**c_{16}[24, -25, 32], $cc_0'=cc_0$[24, -25, 32]**
64	$bb_0=b_{16}+b_0$	m_9	21	2^{31}	$-2^{23}+2^{31}$	**b_{16}[-24, *32], $bb_0'=bb_0$[-24, *32]**

Table 4: The Sufficient Conditions For The First Block

Conditions derived from table 3		Extra conditions derived from $\sum_i$
a_2	$a_{2,26}=0$, $a_{2,29}=1$	
d_2	$d_{2,19}=1$, $d_{2,20}=0$, $d_{2,26}=0$, $d_{2,27}=a_{2,27}$, $d_{2,28}=a_{2,28}$, $d_{2,29}=0$, $d_{2,30}=a_{2,30}$, $d_{2,31}=a_{2,31}$, $d_{2,32}=a_{2,32}$	
c_2	$c_{2,19}=0$, $c_{2,20}=1$, $c_{2,21}=d_{2,21}$, $c_{2,22}=d_{2,22}$, $c_{2,23}=d_{2,23}$, $c_{2,26}=0$, $c_{2,27}=0$, $c_{2,28}=0$, $c_{2,29}=0$, $c_{2,30}=0$, $c_{2,31}=0$, $c_{2,32}=1$	
b_2	$b_{2,2}=0$, $b_{2,7}=0$, $b_{2,12}=0$, $b_{2,16}=0$, $b_{2,17}=1$, $b_{2,19}=0$, $b_{2,20}=0$, $b_{2,21}=0$, $b_{2,22}=0$, $b_{2,23}=1$, $b_{2,26}=1$, $b_{2,27}=0$, $b_{2,28}=0$, $b_{2,29}=0$, $b_{2,30}=0$, $b_{2,31}=0$, $b_{2,32}=0$	
a_3	$a_{3,2}=1$, $a_{3,6}=b_{2,6}$, $a_{3,7}=1$, $a_{3,8}=b_{2,8}$, $a_{3,12}=1$, $a_{3,13}=b_{2,13}$, $a_{3,14}=b_{2,14}$, $a_{3,15}=b_{2,15}$, $a_{3,16}=1$, $a_{3,17}=0$, $a_{3,19}=1$, $a_{3,20}=1$, $a_{3,21}=0$, $a_{3,22}=1$, $a_{3,23}=0$, $a_{3,26}=0$, $a_{3,27}=1$, $a_{3,28}=1$, $a_{3,29}=1$, $a_{3,30}=1$, $a_{3,31}=0$, $a_{3,32}=0$	
d_3	$d_{3,1}=a_{3,1}$, $d_{3,2}=0$, $d_{3,6}=1$, $d_{3,7}=1$, $d_{3,8}=0$, $d_{3,11}=1$, $d_{3,12}=1$, $d_{3,13}=1$, $d_{3,14}=1$, $d_{3,15}=1$, $d_{3,16}=1$, $d_{3,17}=0$, $d_{3,19}=0$, $d_{3,20}=1$, $d_{3,21}=0$, $d_{3,22}=0$, $d_{3,23}=0$, $d_{3,24}=a_{3,24}$, $d_{3,25}=a_{3,25}$, $d_{3,26}=1$, $d_{3,27}=a_{3,27}$, $d_{3,29}=0$, $d_{3,32}=0$	$\sum d_{3,20}=0 \Leftarrow d_{3,31}=1$ $\sum d_{3,31}=0 \Leftarrow d_{3,10}=1, a_{3,10}=0$ $a_{3,11}=b_{2,11}$
c_3	$c_{3,1}=1$, $c_{3,2}=1$, $c_{3,6}=0$, $c_{3,7}=1$, $c_{3,8}=1$, $c_{3,11}=1$, $c_{3,12}=1$, $c_{3,13}=1$, $c_{3,14}=1$, $c_{3,15}=1$, $c_{3,16}=1$, $c_{3,17}=1$, $c_{3,19}=1$, $c_{3,20}=0$, $c_{3,24}=0$, $c_{3,25}=0$, $c_{3,26}=0$, $c_{3,27}=1$, $c_{3,29}=1$, $c_{3,32}=1$	
b_3	$b_{3,1}=0$, $b_{3,2}=1$, $b_{3,6}=1$, $b_{3,7}=1$, $b_{3,8}=1$, $b_{3,11}=1$, $b_{3,12}=0$, $b_{3,13}=1$, $b_{3,14}=1$, $b_{3,15}=1$, $b_{3,16}=0$, $b_{3,17}=1$, $b_{3,19}=0$, $b_{3,20}=1$, $b_{3,23}=c_{3,23}$, $b_{3,24}=0$, $b_{3,25}=1$, $b_{3,26}=0$, $b_{3,27}=0$, $b_{3,29}=0$, $b_{3,32}=1$	$\sum b_{3,29} \sim \sum b_{3,31}$ not all 1's
a_4	$a_{4,1}=1$, $a_{4,9}=1$, $a_{4,14}=1$, $a_{4,18}=b_{3,18}$, $a_{4,19}=1$, $a_{4,20}=1$, $a_{4,21}=b_{3,21}$, $a_{4,22}=b_{3,22}$, $a_{4,23}=1$, $a_{4,24}=1$, $a_{4,25}=1$, $a_{4,26}=1$, $a_{4,27}=1$, $a_{4,29}=1$	$\sum a_{4,25}=0 \Leftarrow b_{3,31}=0$, $a_{4,31}=1, a_{4,32}=1$
d_4	$d_{4,9}=1$, $d_{4,14}=0$, $d_{4,18}=1$, $d_{4,19}=1$, $d_{4,20}=1$, $d_{4,21}=1$, $d_{4,22}=0$, $d_{4,23}=0$, $d_{4,32}=0$	
c_4	$c_{4,9}=1$, $c_{4,14}=0$, $c_{4,18}=0$, $c_{4,19}=0$, $c_{4,20}=0$, $c_{4,21}=0$, $c_{4,22}=0$, $c_{4,23}=1$, $c_{4,27}=0$	$\sum c_{4,29} \sim \sum c_{4,31}$ not all 0's
b_4	$b_{4,9}=1$, $b_{4,14}=0$, $b_{4,18}=0$, $b_{4,19}=0$, $b_{4,20}=0$, $b_{4,21}=0$, $b_{4,22}=1$, $b_{4,23}=0$, $b_{4,27}=1$	
a_5	$a_{5,9}=0$, $a_{5,23}=1$, $a_{5,27}=1$	
d_5	$d_{5,23}=0$	
c_5	$c_{5,27}=d_{5,27}$, $c_{5,32}=0$	
b_5	$b_{5,23}=c_{5,23}$, $b_{5,32}=0$	$\sum a_{6,27}=1 \Leftarrow a_{2,7}=b_{1,7}$, $c_{2,7}=d_{2,7}$
c_6	$c_{6,32}=d_{6,32}$	$\sum d_{6,23}=0$， $\sum b_{11,9}=1$
c_{12}-b_{14}	$b_{12,32}=d_{12,32}$, $a_{13,32}=c_{12,32}$, $d_{13,32}=b_{12,32}$, $c_{13,32}=a_{13,32}$, $b_{13,32}=d_{13,32}$, $a_{14,32}=c_{13,32}$, $d_{14,32}=b_{13,32}$, $c_{14,32}=a_{14,32}$, $b_{14,32}=d_{14,32}$	
a_{15}	$a_{15,24}=0$, $a_{15,32}=c_{14,32}+1$	
d_{15}	$d_{15,24}=1$, $d_{15,32}=b_{14,32}$	
c_{15}	$c_{15,24}=1$, $c_{15,25}=0$, $c_{15,32}=a_{15,32}$	
b_{15}	$b_{15,24}=1$, $b_{15,32}=d_{15,32}$	
a_{16}	$a_{16,24}=0$, $a_{16,25}=1$, $a_{16,32}=c_{15,32}$	
d_{16}	$d_{16,24}=1$, ($d_{16,25}$ or $\sim b_{15,25}$)=1, $d_{16,32}=b_{15,32}$, $dd_{0,24}=0$, $dd_{0,25}=1$	
c_{16}	$c_{16,24}=0$, $c_{16,25}=1$, $c_{16,32}=a_{16,32}+1$, $cc_{0,24}=0$, $cc_{0,25}=1$, $cc_{0,32}=dd_{0,32}$	
b_{16}	$bb_{0,24}=1$	

Table 5: The Differential Path For The Second Block

Step	Chaining variables X_i	w_i	s_i	$\triangle w_i$	$\triangle X_i$	$\diamond X_i$ (signed difference)
IV	aa_0,dd_0,cc_0,bb_0				aa_0[-24, 32], dd_0[24,-25, 32], cc_0[24, -25, 32], bb_0[-24, 32]	
1	a_1	m_0	7		-2^{23}	a_1[-24]
2	d_1	m_1	12		-2^3-2^{23}	d_1[-4,-24]
3	c_1	m_2	17		$-2^3-2^9-2^{23}$	c_1[-4,10,11,12,13,-14,-24]
4	b_1	m_3	22		$2^0+2^2-2^9-2^{14}+2^{21}-2^{23}$	b_1[1,-3…-8,9,-10,15,16,-17,22,24…27,-28]
5	a_2	m_4	7		-2^{18}	a_2[-19]
6	d_2	m_5	12		$2^{12}+2^{21}$	d_2[-13,-14,15,22]
7	c_2	m_6	17	2^8	-2^{22}	c_2[-23]
8	b_2	m_7	22		$2^{24}-2^{31}$	b_2[25,-32],
9	a_3	m_8	7		$-2^{24}+2^{31}$	a_3[25…29,-30,32]
10	d_3	m_9	12	2^{31}	$2^1+2^5+2^9+2^{31}$	d_3[2,-6,7,-10…-22,23,32]
11	c_3	m_{10}	17		$-2^0-2^{28}+2^{31}$	c_3[-1,-29, 32]
12	b_3	m_{11}	22		$2^{18}+2^{31}$	b_3[19,-32]
13	a_4	m_{12}	7		-2^{21}	a_4[22,23,-24]
14	d_4	m_{13}	12		$2^{13}+2^{17}$	d_4[14, 18]
15	c_4	m_{14}	17		-2^8	c_4[-9]
16	b_4	m_{15}	22	2^{31}		b_4
17	a_5	m_1	5		-2^{26}	a_5[-27]
18	d_5	m_6	9	2^8	2^{22}	d_5[23]
19	c_5	m_{11}	14			c_5
20	b_5	m_0	20			b_5
21	a_6	m_5	5		2^{31}	a_6[*32]
22	d_6	m_{10}	9			d_6
23	c_6	m_{15}	14	2^{31}		c_6
24	b_6	m_4	20			b_6
25	a_7	m_9	5	2^{31}		a_7
26	d_7	m_{14}	9			d_7
...	...	...	...	...	...	...
44	b_{11}	m_6	23	2^8	2^{31}	b_{11}[*32]
45	a_{12}	m_9	4	2^{31}	2^{31}	a_{12}[*32]
46	d_{12}	m_{12}	11		2^{31}	d_{12}[*32]
47	c_{12}	m_{15}	16	2^{31}	2^{31}	c_{12}[*32]
48	b_{12}	m_2	23		2^{31}	b_{12}[32]
...	...	...	...	...	...	...
57	a_{15}	m_8	6		2^{31}	a_{15}[-32]
58	d_{15}	m_{15}	10	2^{31}	2^{31}	d_{15}[32]
59	c_{15}	m_6	15	2^8	$2^{23}+2^{31}$	c_{15}[24,-32]
60	b_{15}	m_{13}	21		$2^{23}+2^{31}$	b_{15}[24, 32]
61	$a_{16}+aa_0$	m_4	6		$2^{23}+2^{31}$	$\mathbf{a_{16}+aa_0=a_{16}'+aa_0'}$
62	$d_{16}+dd_0$	m_{11}	10		$2^{23}+2^{31}$	$\mathbf{d_{16}+dd_0=d_{16}'+dd_0'}$
63	$c_{16}+cc_0$	m_2	15		$2^{23}+2^{31}$	$\mathbf{c_{16}+cc_0=c_{16}'+cc_0'}$
64	$b_{16}+bb_0$	m_9	21	2^{31}	$2^{23}+2^{31}$	$\mathbf{b_{16}+bb_0=b_{16}'+bb_0'}$

Table 6: The Sufficient Conditions For The Second Block

Conditions derived from table 5		Extra conditions derived from $\sum_i$
a_1	$a_{1,4}=bb_{0,4}$, $a_{1,10}=0$, $a_{1,11}=0$, $a_{1,13}=0$, $a_{1,14}=1$, $a_{1,24}=1$, $a_{1,25}=1$	$\sum a_{1,25}=1$
d_1	$d_{1,3}=0$, $d_{1,4}=1$, $d_{1,5}=1$, $d_{1,8}=1$, $d_{1,10}=0$, $d_{1,11}=1$, $d_{1,12}=a_{1,12}$, $d_{1,13}=1$, $d_{1,14}=0$, $d_{1,15}=1$, $d_{1,17}=1$, $d_{1,22}=0$, $d_{1,24}=1$, $d_{1,25}=0$, $d_{1,26}=1$, $d_{1,28}=0$, $d_{1,32}=0$	
c_1	$c_{1,1}=d_{1,1}$, $c_{1,3}=1$, $c_{1,4}=1$, $c_{1,5}=0$, $c_{1,6}=d_{1,6}$, $c_{1,7}=d_{1,7}$, $c_{1,8}=0$, $c_{1,9}=d_{1,9}$, $c_{1,10}=0$, $c_{1,11}=0$, $c_{1,12}=0$, $c_{1,13}=0$, $c_{1,14}=1$, $c_{1,15}=0$, $c_{1,16}=d_{1,16}$, $c_{1,17}=0$, $c_{1,22}=d_{1,22}$, $c_{1,24}=1$, $c_{1,25}=1$, $c_{1,26}=0$, $c_{1,27}=d_{1,27}$, $c_{1,28}=1$	
b_1	$b_{1,1}=0$, $b_{1,3}=1$, $b_{1,4}=1$, $b_{1,5}=1$, $b_{1,6}=1$, $b_{1,7}=1$, $b_{1,8}=1$, $b_{1,9}=0$, $b_{1,10}=1$, $b_{1,11}=0$, $b_{1,12}=1$, $b_{1,13}=1$, $b_{1,14}=1$, $b_{1,15}=0$, $b_{1,16}=0$, $b_{1,17}=1$, $b_{1,19}=c_{1,19}$, $b_{1,22}=0$, $b_{1,24}=0$, $b_{1,25}=0$, $b_{1,26}=0$, $b_{1,27}=0$, $b_{1,28}=1$	$\sum b_{1,32}=0$ <= $d_{1,22}=0$ $b_{1,21}=1$,$c_{1,21}=0$
a_2	$a_{2,1}=1$, $a_{2,3}=0$, $a_{2,5}=1$, $a_{2,6}=1$, $a_{2,7}=0$, $a_{2,8}=1$, $a_{2,9}=1$, $a_{2,10}=0$, $a_{2,11}=1$, $a_{2,12}=1$, $a_{2,13}=1$, $a_{2,14}=1$, $a_{2,15}=0$, $a_{2,16}=0$, $a_{2,17}=0$, $a_{2,19}=1$, $a_{2,22}=0$, $a_{2,23}=1$, $a_{1,24}=1$, $a_{2,25}=0$, $a_{2,26}=0$, $a_{2,27}=0$, $a_{2,28}=0$	
d_2	$d_{2,1}=1$, $d_{2,3}=1$, $d_{2,4}=0$, $d_{2,5}=1$, $d_{2,6}=0$, $d_{2,7}=1$, $d_{2,8}=1$, $d_{2,9}=0$, $d_{2,10}=1$, $d_{2,13}=1$, $d_{2,14}=1$, $d_{2,15}=0$, $d_{2,16}=1$, $d_{2,17}=1$, $d_{2,19}=0$, $d_{2,22}=0$, $d_{2,23}=0$, $d_{2,24}=0$, $d_{2,25}=1$, $d_{2,26}=1$, $d_{2,27}=1$, $d_{2,28}=0$, $d_{2,32}=0$	
c_2	$c_{2,13}=0$, $c_{2,14}=0$, $c_{2,15}=1$, $c_{2,19}=1$, $c_{2,22}=1$, $c_{2,23}=1$, $c_{2,24}=0$, $c_{2,25}=1$, $c_{2,26}=0$, $c_{2,30}=1$, $c_{2,32}=1$	
b_2	$b_{2,12}=0$, $b_{2,13}=1$, $b_{2,14}=1$, $b_{2,15}=1$, $b_{2,16}=0$, $b_{2,17}=0$, $b_{2,21}=0$, $b_{2,23}=0$, $b_{2,25}=0$, $b_{2,26}=1$, $b_{2,27}=c_{2,27}$, $b_{2,28}=c_{2,28}$, $b_{2,29}=c_{2,29}$, $b_{2,30}=0$, $b_{2,31}=0$, $b_{2,32}=1$	$\sum b_{2,10}=1$ <= $b_{2,31}=0$
a_3	$a_{3,2}=b_{2,2}$, $a_{3,6}=b_{2,6}$, $a_{3,7}=b_{2,7}$, $a_{3,10}=b_{2,10}$, $a_{3,11}=b_{2,11}$, $a_{3,12}=1$, $a_{3,13}=1$, $a_{3,14}=1$, $a_{3,15}=1$, $a_{3,16}=1$, $a_{3,17=}1$, $a_{3,18}=b_{2,18}$, $a_{3,19}=b_{2,19}$, $a_{3,20}=b_{2,20}$, $a_{3,21}=1$, $a_{3,22}=b_{2,22}$, $a_{3,23}=1$, $a_{3,25}=0$, $a_{3,26}=0$, $a_{3,27}=0$, $a_{3,28}=0$, $a_{3,29}=0$, $a_{3,30}=1$, $a_{3,32}=0$	
d_3	$d_{3,1}=a_{3,1}$, $d_{3,2}=0$, $d_{3,6}=1$, $d_{3,7}=0$, $d_{3,10}=1$, $d_{3,11}=1$, $d_{3,12}=1$, $d_{3,13}=1$, $d_{3,14}=1$, $d_{3,15}=1$, $d_{3,16}=1$, $d_{3,17}=1$, $d_{3,18}=1$, $d_{3,19}=1$, $d_{3,20}=1$, $d_{3,21}=1$, $d_{3,22}=1$, $d_{3,23}=0$, $d_{3,25}=1$, $d_{3,26}=1$, $d_{3,27}=1$, $d_{3,28}=1$, $d_{3,29}=1$, $d_{3,30}=1$, $d_{3,32}=0$	$\sum d_{3,30} \sim \sum c_{3,31}$ not all 1's
c_3	$c_{3,1}=1$, $c_{3,2}=0$, $c_{3,6}=0$, $c_{3,7}=1$, $c_{3,10}=0$, $c_{3,11}=1$, $c_{3,12}=1$, $c_{3,13}=1$, $c_{3,14}=1$, $c_{3,15}=1$, $c_{3,16}=1$, $c_{3,17}=1$, $c_{3,18}=1$, $c_{3,19}=1$, $c_{3,20}=1$, $c_{3,21}=1$, $c_{3,22}=1$, $c_{3,23}=1$, $c_{3,25}=0$, $c_{3,26}=0$, $c_{3,27}=0$, $c_{3,28}=0$, $c_{3,29}=1$, $c_{3,30}=1$, $c_{3,32}=0$	
b_3	$b_{3,1}=0$, $b_{3,2}=1$, $b_{3,6}=1$, $b_{3,7}=1$, $b_{3,10}=1$, $b_{3,11}=1$, $b_{3,12}=0$, $b_{3,13}=1$, $b_{3,14}=1$, $b_{3,15}=0$, $b_{3,16}=1$, $b_{3,17}=1$, $b_{3,18}=1$, $b_{3,19}=0$, $b_{3,20}=1$, $b_{3,21}=1$, $b_{3,22}=1$, $b_{3,23}=c_{3,23}$, $b_{3,24}=c_{3,24}$, $b_{3,29}=0$, $b_{3,32}=1$	$\sum b_{3,29} \sim \sum b_{3,31}$ not all 1's
a_4	$a_{4,1}=1$, $a_{4,9}=1$, $a_{4,14}=1$, $a_{4,18}=1$, $a_{4,19}=0$, $a_{4,22}=0$, $a_{4,23}=0$, $a_{4,24}=1$, $a_{4,29}=1$	$\sum a_{4,25}=1$ <= $a_{4,32}=1$, $a_{4,31}=0$,$b_{3,31}=1$
d_4	$d_{4,9}=1$, $d_{4,14}=0$, $d_{4,18}=0$, $d_{4,19}=1$, $d_{4,22}=0$, $d_{4,23}=0$, $d_{4,24}=1$, $d_{4,32}=0$	
c_4	$c_{4,9}=1$, $c_{4,14}=0$, $c_{4,18}=0$, $c_{4,22}=1$, $c_{4,23}=1$, $c_{4,24}=1$, $c_{4,27}=0$	$\sum c_{4,29} \sim \sum c_{4,31}$ not all 0's,
b_4	$b_{4,9}=0$, $b_{4,14}=0$, $b_{4,18}=0$, $b_{4,23}=0$, $b_{4,27}=1$	
a_5	$a_{5,9}=1$, $a_{5,23}=1$, $a_{5,27}=1$	
d_5	$d_{5,23}=0$	
c_5	$c_{5,27}=d_{5,27}$, $c_{5,32}=0$	
b_5	$b_{5,23}=c_{5,23}$, $b_{5,32}=0$	$\sum d_{6,23}=0$<= $b_{3,8}=c_{3,8}$,$d_{3,8}=a_{3,8}$
c_6	$c_{6,32}=d_{6,32}$	$\sum a_{6,27}=1$, $\sum b_{11,9}=0$
c_{12}-b_{14}	$b_{12,32}=d_{12,32}$, $a_{13,32}=c_{12,32}$, $d_{13,32}=b_{12,32}$, $c_{13,32}=a_{13,32}$, $b_{13,32}=d_{13,32}$, $a_{14,32}=c_{13,32}$, $d_{14,32}=b_{13,32}$, $c_{14,32}=a_{14,32}$, $b_{14,32}=d_{14,32}$	
a_{15}	$a_{15,24}=0$, $a_{15,32}=c_{14,32}+1$, $d_{15,24}=1$, $d_{15,32}=b_{14,32}$	
c_{15}	$c_{15,24}=0$, $c_{15,25}=0$, $c_{15,32}=a_{15,32}$	
b_{15}	$b_{15,24}=0$, $b_{15,32}=d_{15,32}$	
a_{16}	$a_{16,24}=1$, $a_{16,25}=0$, $a_{16,32}=c_{15,32}$	
d_{16}	$d_{16,24}=0$, ($d_{16,25}$ or $\sim b_{15,25}$)=1, $d_{16,32}=b_{15,32}$	
c_{16}	$c_{16,24}=1$, $c_{16,25}=0$, $c_{16,32}=a_{16,32}+1$	