2 MD5 collision attack by Wang et al.

Contents

2.1 Preliminaries

In the upcoming sections we describe the original attack on MD5 by Wang et al. [WY05] in detail. Their original paper is well complemented by [HPR04] which provides many insightful details. First we introduce the necessary preliminaries and the definition of MD5, followed by an overview of the attack and a detailed treatment of the various aspects.

2.1.1 32-bit words

MD5 is designed with a 32-bit computing architecture in mind and operates on *words* $(v_{31} \dots v_0)$ consisting of 32 bits $v_i \in \{0, 1\}$. These 32-bit words are identified with elements $v = \sum_{i=0}^{31} v_i 2^i$ of $\mathbb{Z}_{2^{32}}$ (a shorthand for $\mathbb{Z}/2^{32}\mathbb{Z}$). In this thesis we switch freely between the bitwise and $\mathbb{Z}_{2^{32}}$ representation of 32-bit words. We shall denote a 32-bit word in binary as $\texttt{00000000111111110000111100110101}_2$. For a more compact notation, we also use the well-known hexadecimal form $\texttt{00ff0f35}_{16}$.

For 32-bit words $X = (x_i)_{i=0}^{31}$ and $Y = (y_i)_{i=0}^{31}$ we use the following notation:

- $X \wedge Y = (x_i \wedge y_i)_{i=0}^{31}$ is the bitwise AND of X and Y;
- $X \vee Y = (x_i \vee y_i)_{i=0}^{31}$ is the bitwise OR of X and Y;
- $X \oplus Y = (x_i \oplus y_i)_{i=0}^{31}$ is the bitwise XOR of X and Y;
- $\overline{X} = (\overline{x_i})_{i=0}^{31}$ is the bitwise complement of X;

- $X[i]$ is the i-th bit x_i;
- $X + Y$ and $X - Y$ denote addition and subtraction, respectively, of X and Y in $\mathbb{Z}_{2^{32}}$;
- $RL(X, n)$ and $RR(X, n)$ are the cyclic left and right rotation, respectively, of X by n bit positions:

$$\begin{aligned} & RL(\mathtt{10100100111111111111111100000001}_2,\ 5) \\ = \quad & \mathtt{00111111111111111110000000110100}_2; \end{aligned}$$

- $w(X)$ denotes the Hamming weight $\sum_{i=0}^{31} x_i$ of $X = (x_i)_{i=0}^{31}$.

2.1.2 Endianness

When interpreting a 32-bit word from a string $b_{31} \dots b_0$ of 32 bits and vice versa, there are two main standards that can be used:

Big Endian: This is the most straightforward standard which interprets a bit string in order with the most significant bit first resulting in the 32-bit word $(b_{31} \dots b_0)$.

Little Endian: This standard partitions a bit string into strings of eight consecutive bits and reverses the order of these partitions while maintaining the bit order within each partition. The resulting bit string is then interpreted using Big Endian resulting in the 32-bit word:

$$(b_7 b_6 \dots b_1 b_0\ b_{15} b_{14} \dots b_9 b_8\ b_{23} b_{22} \dots b_{17} b_{16}\ b_{31} b_{30} \dots b_{25} b_{24}).$$

This standard is for example used on the x86 CPU architecture.

2.1.3 Binary signed digit representation

A *binary signed digit representation* (BSDR) for an $X \in \mathbb{Z}_{2^{32}}$ is a sequence $(k_i)_{i=0}^{31}$ such that

$$X = \sum_{i=0}^{31} k_i 2^i, \quad k_i \in \{-1, 0, 1\}.$$

For each non-zero X there exist many different BSDRs. The *weight* $w((k_i)_{i=0}^{31}) = \sum_{i=0}^{31} |k_i|$ of a BSDR $(k_i)_{i=0}^{31}$ is defined as the number of non-zero k_is.

A particularly useful type of BSDR is the Non-Adjacent Form (NAF), where no two non-zero k_i-values are adjacent. For any $X \in \mathbb{Z}_{2^{32}}$ there is no unique NAF, since we work modulo 2^{32} (making $k_{31} = +1$ equivalent to $k_{31} = -1$). However, uniqueness of the NAF can be enforced by the added restriction $k_{31} \in \{0, +1\}$. Among the BSDRs for a given $X \in \mathbb{Z}_{2^{32}}$, the NAF has minimal weight [MS06]. The NAF can be computed easily [Lin98] for a given $X \in \mathbb{Z}_{2^{32}}$ as $\mathrm{NAF}(X) = ((X + Y)[i] - Y[i])_{i=0}^{31}$ where Y is the 32-bit word $(0\ X[31]\ \dots\ X[1])$.

We use the following notation for a 32-digit BSDR Z:

- $Z[i]$ is the i-th signed bit of Z;
- $RL(Z,n)$ and $RR(Z,n)$ are the cyclic left and right rotation, respectively, of Z by n positions;
- $w(Z)$ is the weight of Z.
- $\sigma(Z) = \sum_{i=0}^{31} k_i 2^i \in \mathbb{Z}_{2^{32}}$ is the 32-bit word for which Z is a BSDR.

As a more compact representation for a BSDR, we list all i-values with non-zero k_i and using an overline $\bar{i}$ to indicate a negative k_i. E.g., consider the following BSDR of $2^{23} - 2^0$:

$$\begin{aligned}
&\{0 \ldots 4, \bar{5}, \overline{23} \ldots \overline{27}, 28\} \\
&\quad = \quad \{0, 1, 2, 3, 4, \bar{5}, \overline{23}, \overline{24}, \overline{25}, \overline{26}, \overline{27}, 28\} \\
&\quad = \quad (k_i)_{i=0}^{31} \text{ with } \begin{cases} k_0 = k_1 = k_2 = k_3 = k_4 = k_{28} = +1; \\ k_5 = k_{23} = k_{24} = k_{25} = k_{26} = k_{27} = -1; \\ k_i = 0 \text{ for } 6 \leq i \leq 22 \text{ and } 29 \leq i \leq 31. \end{cases}
\end{aligned}$$

2.1.4 Related variables and differences

In collision attacks we consider two related messages M and M'. In this thesis any variable X related to the message M or its MD5 (or other hash function) calculation may have a corresponding variable X' related to the message M' or its hash calculation. Furthermore, for such a 'matched' variable $X \in \mathbb{Z}_{2^{32}}$ we define $\delta X = X' - X$ and $\Delta X = (X'[i] - X[i])_{i=0}^{31}$, which is a BSDR of δX. For a matched variable Z that is a tuple of 32-bit words, say $Z = (z_1, z_2, \ldots)$, we define δZ and ΔZ as $(\delta z_1, \delta z_2, \ldots)$ and $(\Delta z_1, \Delta z_2, \ldots)$, respectively.

2.2 Description of MD5

2.2.1 MD5 overview

MD5 works as follows on a given bit string M of arbitrary bit length, cf. [Riv92]:

1. *Padding.* Pad the message: first append a '1'-bit, next append the least number of '0'-bits to make the resulting bit length equivalent to 448 modulo 512, and finally append the bit length of the original unpadded message M as a 64-bit little-endian integer[7]. As a result the total bit length of the padded message $\widehat{M}$ is $512N$ for a positive integer N.

7. MD5 inherited the design choice for its predecessor MD4 of using little-endian so that for little-endian machines the conversion between bit strings and words is effortless. Since at the time little-endian machines were considered generally slower than big-endian machines, this conversion advantage was given to little-endian machines instead of big-endian machines[Riv90a], despite the mathematically more aesthetic definition of big-endian.

2. *Partitioning.* Partition the padded message $\widehat{M}$ into N consecutive 512-bit blocks $M_0, M_1, \ldots, M_{N-1}$.

3. *Processing.* To hash a message consisting of N blocks, MD5 goes through $N+1$ states IHV_i, for $0 \leq i \leq N$, called the *intermediate hash values*. Each intermediate hash value IHV_i is a tuple of four 32-bit words (a_i, b_i, c_i, d_i). For $i = 0$ it has a fixed public value called the *initial value* (IV):

$$(a_0, b_0, c_0, d_0) = (\texttt{67452301}_{16}, \texttt{efcdab89}_{16}, \texttt{98badcfe}_{16}, \texttt{10325476}_{16}).$$

   For $i = 1, 2, \ldots, N$ intermediate hash value IHV_i is computed using the MD5 compression function described in detail below:

$$IHV_i = \text{MD5Compress}(IHV_{i-1}, M_{i-1}).$$

4. *Output.* The resulting hash value is the last intermediate hash value IHV_N, expressed as the concatenation of the hexadecimal byte strings of the four words a_N, b_N, c_N, d_N, converted back from their little-endian representation. As an example the IV would be expressed as

$$\texttt{0123456789abcdeffedcba9876543210}_{16}.$$

2.2.2 MD5 compression function

The input for the compression function MD5Compress(IHV, B) consists of an intermediate hash value $IHV_{\text{in}} = (a, b, c, d)$ and a 512-bit message block B. The compression function consists of 64 *steps* (numbered 0 to 63), split into four consecutive *rounds* of 16 steps each. Each step t uses modular additions, a left rotation, and a non-linear function f_t, and involves an *Addition Constant* AC_t and a *Rotation Constant* RC_t. These are defined as follows (see also Table A-1):

$$AC_t = \left\lfloor 2^{32} \left|\sin(t+1)\right|\right\rfloor \ , \qquad 0 \leq t < 64,$$

$$(RC_t, RC_{t+1}, RC_{t+2}, RC_{t+3}) = \begin{cases} (7, 12, 17, 22) & \text{for } t = 0, 4, 8, 12, \\ (5, 9, 14, 20) & \text{for } t = 16, 20, 24, 28, \\ (4, 11, 16, 23) & \text{for } t = 32, 36, 40, 44, \\ (6, 10, 15, 21) & \text{for } t = 48, 52, 56, 60. \end{cases}$$

The non-linear function f_t depends on the round:

$$f_t(X, Y, Z) = \begin{cases} F(X, Y, Z) = (X \wedge Y) \oplus (\overline{X} \wedge Z) & \text{for } 0 \leq t < 16, \\ G(X, Y, Z) = (Z \wedge X) \oplus (\overline{Z} \wedge Y) & \text{for } 16 \leq t < 32, \\ H(X, Y, Z) = X \oplus Y \oplus Z & \text{for } 32 \leq t < 48, \\ I(X, Y, Z) = Y \oplus (X \vee \overline{Z}) & \text{for } 48 \leq t < 64. \end{cases} \tag{2.1}$$

The 512-bit message block B is partitioned into sixteen consecutive 32-bit strings which are then interpreted as 32-bit words m_0, $m_1, \ldots, m_{15}$ (with little-endian byte ordering), and expanded to 64 words W_t, for $0 \leq t < 64$, (see also Table A-1):

$$W_t = \begin{cases} m_t & \text{for } 0 \leq t < 16, \\ m_{(1+5t) \bmod 16} & \text{for } 16 \leq t < 32, \\ m_{(5+3t) \bmod 16} & \text{for } 32 \leq t < 48, \\ m_{(7t) \bmod 16} & \text{for } 48 \leq t < 64. \end{cases}$$

We follow the description of the MD5 compression function from [HPR04] because its 'unrolling' of the cyclic state facilitates the analysis. For each step t the compression function algorithm uses a working state consisting of four 32-bit words Q_t, Q_{t-1}, Q_{t-2} and Q_{t-3} and calculates a new state word Q_{t+1}. The working state is initialized as $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$. For $t = 0, 1, \ldots, 63$ in succession, Q_{t+1} is calculated as follows:

$$\begin{aligned} F_t &= f_t(Q_t,\ Q_{t-1},\ Q_{t-2}); \\ T_t &= F_t + Q_{t-3} + AC_t + W_t; \\ R_t &= RL(T_t, RC_t); \\ Q_{t+1} &= Q_t + R_t. \end{aligned} \tag{2.2}$$

After all steps are computed, the resulting state words are added to the input intermediate hash value and returned as output:

$$\text{MD5Compress}(IHV_{\text{in}}, B) = (a + Q_{61},\ b + Q_{64},\ c + Q_{63},\ d + Q_{62}). \tag{2.3}$$

2.3 Collision attack overview

The collision attack against MD5 by X. Wang and H. Yu is based on differential cryptanalysis using a combination of the XOR difference and the modular difference, resulting in a differential cryptanalysis that uses the integer difference $(b' - b) \in \{-1, 0, +1\}$ for each bit b. Using this differential cryptanalysis they constructed *differential paths* for the compression function of MD5 which describe precisely how differences between the two input pairs (IHV_{in}, B) and (IHV'_{in}, B') propagate through the compression function's working states Q_t and Q'_t, resulting in a desired difference between the outputs. Based on the differential path they constructed a system of equations, called the *sufficient conditions*, over the bits of the working states Q_t and Q'_t that ensure that the desired differential path happens.[8] The purpose of this set of equations is to facilitate the search for blocks B and B' given IHV_{in} and IHV'_{in} such that the desired

8. As shown later in this thesis, the original analysis did not lead to 'sufficient conditions' that were truly sufficient. In particular, it was indirectly assumed that the desired propagation of a modular difference through the bitwise rotation in MD5 occurs with probability 1, even though in most cases this happens with probability slightly less than 1.

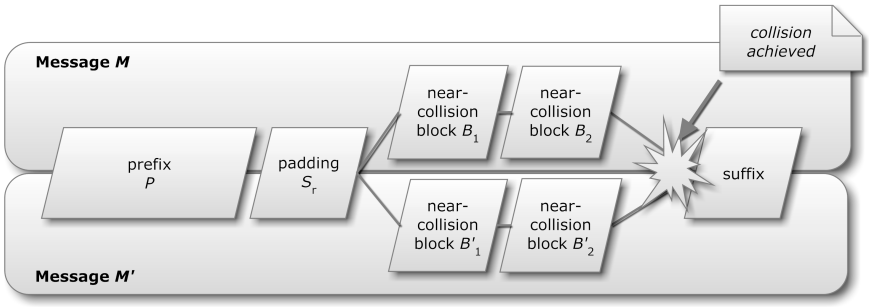

Figure 6: *Identical-prefix collision sketch*

differential path occurs, thus resulting in the desired output difference. Such an attack against the compression function, where for given IHV_{in} and IHV'_{in} a differential path is used to find message blocks B and B' such that a desired δIHV_{out} is obtained, is called a *near-collision attack*.

The attack takes as input two prefixes described as two equal-length sequences of message blocks $M_0, \ldots, M_{k-1}$ and $M'_0, \ldots, M'_{k-1}$ that resulting in identical IHV_k and IHV'_k. This condition is most easily fulfilled when both prefixes are identical, therefore collision attacks requiring $IHV_k = IHV'_k$ are called *identical-prefix collision attacks*.[9] For an arbitrary prefix P, we can obtain said message block sequences by appending a random bit string S_{r} of bit length less than 512 to P, such that the bit length of $P\|S_{\text{r}}$ is an integer multiple of 512. The padded prefix $P\|S_{\text{r}}$ is then split into message blocks $M'_0 = M_0, \ldots, M'_{k-1} = M_{k-1}$ of 512 bits.

A collision is generated using two consecutive pairs of message blocks (B_0, B'_0) and (B_1, B'_1), where $B'_0 \neq B_0$ and $B'_1 \neq B_1$, that are appended to the two prefixes. Together they result in a collision: $IHV'_{k+2} = IHV_{k+2}$. The collision attack outputs two messages $M = P||B_0||B_1$ and $M' = P'||B'_0||B'_1$ that result in the same MD5 hash. Due to the incremental nature of MD5, these messages can be further extended using an identical suffix S, resulting in the colliding messages $P||B_0||B_1||S$ and $P'||B'_0||B'_1||S$.

Thus in the most straightforward way, the attack can be used to create two messages M and M' with the same MD5 hash that only differ slightly in two subsequent blocks, as shown in Figure 6.

Note that all blocks $M_0, \ldots, M_{k-1}$ and $M_{k+2}, \ldots, M_{N-1}$ can be chosen arbitrarily and that only B_0, B'_0, B_1 and B'_1 are generated by the collision finding algorithm using the value of $IHV'_k = IHV_k$.

In the following few subsections we explain the various parts of the collision attack in more detail.

9. Since this collision attack requires $IHV_k = IHV'_k$ and results in $IHV_{k+2} = IHV'_{k+2}$, the attack can be chained endlessly.

2.4 Two message block collision

Wang et al.'s attack consists of two differential paths for two subsequent message blocks, which we refer to in order as the first and second differential path. We refer to B_0 and B_1 as the first and second near-collision block, respectively. The first differential path starts with any given $IHV_k = IHV'_k$ and introduces a difference between IHV_{k+1} and IHV'_{k+1}:

$$\delta IHV_{k+1} = (\delta a, \delta b, \delta c, \delta d) = (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25}).$$

These differences are canceled again by the second differential path. The first differential path is based on the following differences in the message block:

$$\delta m_4 = 2^{31}, \quad \delta m_{11} = 2^{15}, \quad \delta m_{14} = 2^{31}, \quad \delta m_i = 0 \text{ for } i \notin \{4, 11, 14\}.$$

The second differential path is based on the *negated* message block differences:

$$\delta m_4 = 2^{31}, \quad \delta m_{11} = -2^{15}, \quad \delta m_{14} = 2^{31}, \quad \delta m_i = 0 \text{ for } i \notin \{4, 11, 14\}.$$

Note that $-2^{31} = 2^{31}$ in $\mathbb{Z}_{2^{32}}$, so in fact δm_4 and δm_{14} are not changed by the negation.

These are very specific message block differences that were selected to ensure a low complexity for the collision finding algorithm as is shown later. In Section 6.4, we introduce even better message block differences.

2.5 Differential paths

As said before, this attack uses a combination of XOR differential cryptanalysis and modular differential cryptanalysis. The combination of both kinds of differences gives more information than each by themselves and both are aimed at a different representation of a 32-bit word: as a string of 32 bits or as an element of $\mathbb{Z}_{2^{32}}$. So instead of only the integer modular difference between two related 32-bit words X and X', this combination uses the integer differences ($-1, 0$ or $+1$) between each pair of bits $X[i]$ and $X'[i]$ for $0 \leq i \leq 31$. This difference is represented in a natural manner using BSDRs and thus can be denoted as ΔX, which is a BSDR of $\delta X = X' - X$:

$$\Delta X = (k_i), \quad k_i = X'[i] - X[i] \text{ for } 0 \leq i \leq 31.$$

Of all variables used in MD5's compression function, this combined differential cryptanalysis is only used on the Q_t variables since these are used in both the bitwise boolean function and modular addition. The R_t and T_t variables are used in bitwise rotation and modular addition. Nevertheless, only the modular differential is used for R_t and T_t as a simplification, since $\delta R_t = RL(T_t + \delta T_t, RC_t) - RL(T_t, RC_t)$ already holds with high probability for properly chosen δR_t and δT_t. The modular differential is used for the remaining variables as these are only used in modular addition.

The differential paths for both blocks (Table 2-3 and Table 2-5) were constructed specifically to create a collision in this manner. The differential paths describe precisely for each of the 64 steps of MD5 what the differences are in the working state and

how these differences pass through the boolean function and the rotation. More precisely, for MD5 a differential path can be described through the following sequences of differences: $(\delta m_t)_{t=0}^{15}$, $(\Delta Q_t)_{t=-3}^{64}$, $(\Delta F_t)_{t=0}^{63}$. Other differences can easily be derived: $\delta T_t = \delta F_t + \delta Q_{t-3} + \delta W_t$ and $\delta R_t = \delta Q_{t+1} - \delta Q_t$.

The first differential path starts without differences as $\delta IHV_k = 0$, but differences are introduced in step $t = 4$ by $\delta m_4 = 2^{31}$. The second differential path starts with the given δIHV_{k+1}. In both paths, all differences in the working state are canceled at step $t = 25$ by δm_{14}. Furthermore, from step $t = 34$, both paths use the same differential path, although with opposite signs. This structure can easily be seen in Table 2-3 and Table 2-5.

Below we show a fraction of the first differential path:

Table 2-1: *Fraction from Table 2-3*

$\mathbf{t}$	$\Delta\mathbf{Q_t}$ **(BSDR of** $\delta\mathbf{Q_t}$**)**	$\delta\mathbf{F_t}$	$\delta\mathbf{w_t}$	$\delta\mathbf{T_t}$	$\mathbf{RC_t}$
13	$\{\overline{24}, 25, 31\}$	$-2^{13}+2^{31}$	0	-2^{12}	12
14	$\{31\}$	$2^{18}+2^{31}$	2^{31}	$2^{18}-2^{30}$	17
15	$\{3, \overline{13}, 31\}$	$2^{25}+2^{31}$	0	$-2^{7}-2^{13}+2^{25}$	22
16	$\{\overline{29}, 31\}$	2^{31}	0	2^{24}	5
17	$\{31\}$	2^{31}	0	0	9
18	$\{31\}$	2^{31}	2^{15}	2^{3}	14
19	$\{17, 31\}$	2^{31}	0	-2^{29}	20

These two differential paths were made by hand with great skill and intuition by Wang et al. In order to construct faster collisions attacks or for collision attacks that do not require $IHV'_k = IHV_k$ we need to be able to construct differential paths preferably algorithmically instead of by hand.

2.6 Sufficient conditions

Wang et al. use *sufficient conditions* to efficiently search for message blocks for which these differential paths hold. These sufficient conditions guarantee that the necessary carries and correct boolean function differences happen. Each condition gives the value of a bit $Q_t[i]$ of the working state either directly or indirectly as shown in Table 2-2. Later on, we generalize and extend these conditions to also include the value of the related bit $Q'_t[i]$.

These conditions are only used to find a block B on which the message differences can be applied to find B' and should guarantee that the differential path happens. They can be derived for any differential path and there can be many different possible sets of sufficient conditions.

However, it should be noted that the sufficient conditions given by Wang et al. are not sufficient at all, as those conditions do not guarantee that in each step the differences are rotated correctly. In fact, as we show later on, one does not want

Table 2-2: *Sufficient bitconditions.*

Symbol	condition on $\mathbf{Q_t[i]}$	direct/indirect
.	no condition	direct
0	$Q_t[i] = 0$	direct
1	$Q_t[i] = 1$	direct
^	$Q_t[i] = Q_{t-1}[i]$	indirect

See Table B-1 for a complete listing of all possible bitconditions used in this thesis.

sufficient conditions for the full differential path as this increases the collision finding complexity significantly.

2.7 Collision finding

Using these sufficient conditions one can efficiently search for a block B. The most basic *collision finding algorithm* chooses random working state words $Q_1, \ldots, Q_{16}$ that fulfill their sufficient conditions for the first round. The corresponding message words can directly be computed:

$$m_t = RR(Q_{t+1} - Q_t, RC_t) - f_t(Q_t, Q_{t-1}, Q_{t-2}) - Q_{t-3} - AC_t \text{ for } 0 \leq t \leq 15.$$

As all the bits in the message block are now determined, the other working state words $Q_{17}, \ldots, Q_{64}$ are now also completely determined. Thus the remaining sufficient conditions have to be fulfilled probabilistically and directly result in the complexity of this basic collision finding algorithm. Wang et al. used the following improvements over this basic algorithm:

Message modification: When a certain condition in the second round fails, one can use message modification. This is a substitution formula specially made for this condition on the message block B. In the case that this condition does not hold applying this substitution has the effect that this condition now does hold without interfering with other previous conditions. To use message modification, auxiliary conditions have to be imposed on the first round. Specifically, at least one auxiliary condition should be imposed to avoid testing the same message block B twice (once directly, once through another message block $\widehat{B}$ to which this substitution is applied).

Early stop: Stop at the step where the first sufficient condition fails and there is no message modification. This improvement reduces the average number of MD5 compression function steps that have to be evaluated.

An example of message modification is the following. When testing a block for the sufficient conditions in Table 2-4, suppose $Q_{17}[31] = 1$ instead of 0. This can be corrected by replacing m_1, m_2, m_3, m_4, m_5 with:

1. $\widehat{m}_1 \leftarrow (m_1 + 2^{26})$ which results in a different $\widehat{Q}_2$;

2. $\widehat{m}_2 \leftarrow RR(Q_3 - \widehat{Q}_2, 17) - Q_{-1} - F(\widehat{Q}_2, Q_1, Q_0) - AC_2$;
3. $\widehat{m}_3 \leftarrow RR(Q_4 - Q_3, 22) - Q_0 - F(Q_3, \widehat{Q}_2, Q_1) - AC_3$;
4. $\widehat{m}_4 \leftarrow RR(Q_5 - Q_4, 7) - Q_1 - F(Q_4, Q_3, \widehat{Q}_2) - AC_4$;
5. $\widehat{m}_5 \leftarrow RR(Q_6 - Q_5, 12) - \widehat{Q}_2 - F(Q_5, Q_4, Q_3) - AC_5$.

The first line is the most important, here m_1 is changed such that $\widehat{Q}_{17}[31] = 0$, assuming Q_{13} up to Q_{16} remain unaltered. The added $+2^{26}$ in m_1 results in an added $+2^{31}$ in $Q_{17}[31]$, hence $\widehat{Q}_{17}[31] = 0$. The four other lines simply change m_2, m_3, m_4 and m_5 such that Q_3 up to Q_{16} remain unaltered by the change from Q_2 to $\widehat{Q}_2$. Since there are no conditions in the first block that involve Q_2, all sufficient conditions up to Q_{17} are left unaltered. However, there is a side-effect: with probability 1/2 the rotation in step 16 actually results in an additional added $+2^0$ in $Q_{17}[31]$, which then with probability 1/8 interferes with the condition imposed on $Q_{17}[3]$. Fortunately, this side-effect can be avoided entirely by negating the substitution whenever $T_{16}[26] = 1$.

Comparing Table 2-4 and Table 2-6, there are fewer sufficient conditions for the first block. In general, this means that more message modification techniques can be applied on the first block, resulting in a larger speedup compared to the second block. However, there are eight sufficient bitconditions for the second block on the input IHV_{k+1}. Since the value of IHV_{k+1} depends only on the values of IHV_k and the first block B_0, these eight sufficient bitconditions have to be fulfilled probabilistically by the first block rather than the second block.

The original attack can find collisions for MD5 in about 15 minutes up to an hour on an IBM P690 with a computational cost equivalent to about 2^{39} compression function calls. Since then many improvements have been made [YS05, SNKO05, LL05, Kli05, Ste06, Kli06]. Currently, collisions for MD5 based on these differential paths can be found in several seconds on a single powerful PC with a computational cost equivalent to about $2^{24.1}$ compression function calls. These faster attacks use techniques based on *Tunnels* [Kli06], controlling rotations in the first round [Ste06] and additional differential paths. Our fastest collision attack [SSA$^+$09b] is based on slightly different message block differences and has a theoretical computational cost of about 2^{16} compression function calls (see Section 6.4). Xie and Feng [XF09] claim to have found message differences that may lead to an even faster attack. Based on a rather optimistic cost function they claim an estimated complexity of 2^{10} compressions. Besides this claim, they have constructed a MD5 collision attack with a complexity of about $2^{20.96}$ MD5 compressions using less promising message differences. Recently even single-block identical-prefix collisions have been found by Xie and Feng [XF10], although they do not present their new techniques 'for security reasons'.

2.8 Tables: differential paths and sufficient conditions

Table 2-3: *Wang et al.'s first differential path*

$\mathbf{t}$	$\Delta\mathbf{Q_t}$ (BSDR of $\delta\mathbf{Q_t}$)	$\delta\mathbf{F_t}$	$\delta\mathbf{w_t}$	$\delta\mathbf{T_t}$	$\mathbf{RC_t}$
$0-3$	0	0	0	0	RC_t
4	0	0	2^{31}	2^{31}	7
5	$\{6\dots21,\overline{22}\}$	$2^{11}+2^{19}$	0	$2^{11}+2^{19}$	12
6	$\{\overline{6},23,31\}$	$-2^{10}-2^{14}$	0	$-2^{10}-2^{14}$	17
7	$\{0\dots4,\overline{5},6\dots10,\overline{11},$ $\overline{23}\dots\overline{25},26\dots31\}$	$-2^{2}+2^{5}+2^{10}$ $+2^{16}-2^{25}-2^{27}$	0	$-2^{2}+2^{5}+2^{10}$ $+2^{16}-2^{25}-2^{27}$	22
8	$\{0,15,\overline{16},17,18,$ $19,\overline{20},\overline{23}\}$	$2^{6}+2^{8}+2^{10}$ $+2^{16}-2^{24}+2^{31}$	0	$2^{8}+2^{10}+2^{16}$ $-2^{24}+2^{31}$	7
9	$\{\overline{0},1,6,7,\overline{8},\overline{31}\}$	$2^{0}+2^{6}-2^{20}$ $-2^{23}+2^{26}+2^{31}$	0	$2^{0}-2^{20}+2^{26}$	12
10	$\{\overline{12},13,31\}$	$2^{0}+2^{6}+2^{13}-2^{23}$	0	$2^{13}-2^{27}$	17
11	$\{30,31\}$	$-2^{0}-2^{8}$	2^{15}	$-2^{8}-2^{17}-2^{23}$	22
12	$\{7,\overline{8},13\dots18,\overline{19},31\}$	$2^{7}+2^{17}+2^{31}$	0	$2^{0}+2^{6}+2^{17}$	7
13	$\{\overline{24},25,31\}$	$-2^{13}+2^{31}$	0	-2^{12}	12
14	$\{31\}$	$2^{18}+2^{31}$	2^{31}	$2^{18}-2^{30}$	17
15	$\{3,\overline{15},31\}$	$2^{25}+2^{31}$	0	$-2^{7}-2^{13}+2^{25}$	22
16	$\{\overline{29},31\}$	2^{31}	0	2^{24}	5
17	$\{31\}$	2^{31}	0	0	9
18	$\{31\}$	2^{31}	2^{15}	2^{3}	14
19	$\{17,31\}$	2^{31}	0	-2^{29}	20
20	$\{31\}$	2^{31}	0	0	5
21	$\{31\}$	2^{31}	0	0	9
22	$\{31\}$	2^{31}	0	2^{17}	14
23	0	0	2^{31}	0	20
24	0	2^{31}	0	0	5
25	0	0	2^{31}	0	9
$26-33$	0	0	0	0	RC_t
34	0	0	2^{15}	2^{15}	16
35	$\delta Q_{35}=2^{31}$	2^{31}	2^{31}	0	23
36	$\delta Q_{36}=2^{31}$	0	0	0	4
37	$\delta Q_{37}=2^{31}$	2^{31}	2^{31}	0	11
$38-49$	$\delta Q_t=2^{31}$	2^{31}	0	0	RC_t
50	$\delta Q_{50}=2^{31}$	0	2^{31}	0	15
$51-59$	$\delta Q_t=2^{31}$	2^{31}	0	0	RC_t
60	$\delta Q_{60}=2^{31}$	0	2^{31}	0	6
61	$\delta Q_{61}=2^{31}$	2^{31}	2^{15}	2^{15}	10
62	$\delta Q_{62}=2^{31}+2^{25}$	2^{31}	0	0	15
63	$\delta Q_{63}=2^{31}+2^{25}$	2^{31}	0	0	21
64	$\delta Q_{64}=2^{31}+2^{25}$	$\times$	$\times$	$\times$	$\times$

Table 2-4: *Wang et al.'s first block sufficient bitconditions*

t	Conditions on Q_t: $b_{31} \dots b_0$	#
3	....0... ....0... .0......	3
4	1....... 0^^^1^^^ ^^^^1^^^ ^0......	19
5	1...1.0. 01..0000 00000000 001..1.1	22
6	0000001^ 01111111 10111100 0100^0^1	32
7	00000011 11111110 11111000 00100000	32
8	00000001 1..10001 0.0.0101 01000000	28
9	11111011 ...10000 0.1^1111 00111101	28
10	01...... 0..11111 1101...0 01....00	17
11	00...... ....0001 1100...0 11....10	15
12	00....^^ ....1000 0001...1 0.......	14
13	01....01 ....1111 111....0 0...1...	14
14	0.0...00 ....1011 111....1 1...1...	14
15	0.1...01 .... 1....... ....0...	6
16	0.1.....	2
17	0....... ......0. ^....... ....^...	4
18	0.^..... ......1.	3
19	0....... ......0.	2
20	0.......	1
21	0....... ......^.	2
22	0.......	1
23	0.......	1
24	1.......	1
25 − 45	....	0
46	....	0
47	....	0
48	m.......	1
49	m.......	1
50	#.......	1
51	m.......	1
52	m.......	1
53	m.......	1
54	m.......	1
55	m.......	1
56	m.......	1
57	m.......	1
58	m.......	1
59	m.......	1
60	#.....0.	2
61	m.....1.	2
62	m.....0.	2
63	m.....0.	2
64	....	0
	Sub-total # IV conditions in Table 2-6	**8**
	Total # conditions	**289**

Uses bitconditions as defined in Table B-1 limited to only the variables $Q_i[j]$, not $Q'_i[j]$.

Table 2-5: *Wang et al.'s second differential path*

$\mathbf{t}$	$\Delta\mathbf{Q_t}$ (BSDR of $\delta\mathbf{Q_t}$)	$\delta\mathbf{F_t}$	$\delta\mathbf{w_t}$	$\delta\mathbf{T_t}$	$\mathbf{RC_t}$
-3	$\{31\}$	$\times$	$\times$	$\times$	$\times$
-2	$\{25, 31\}$	$\times$	$\times$	$\times$	$\times$
-1	$\{\overline{25}, 26, 31\}$	$\times$	$\times$	$\times$	$\times$
0	$\{25, 31\}$	2^{31}	0	0	7
1	$\{25, 31\}$	2^{31}	0	2^{25}	12
2	$\{\overline{5}, 25, 31\}$	2^{25}	0	$2^{31}+2^{26}$	17
3	$\{\overline{5}, \overline{6}, 7, \overline{11}, 12, \overline{16} \ldots \overline{20}, 21, \overline{25} \ldots \overline{29}, 30, 31\}$	$-2^{11}-2^{21}+2^{25}$ $-2^{27}+2^{31}$	0	$-2^{11}-2^{21}-2^{26}$	22
4	$\{1, 2, 3, \overline{4}, 5, \overline{25}, 26, 31\}$	$2^{1}-2^{3}-2^{18}$ $+2^{26}+2^{30}$	2^{31}	$2^{1}+2^{2}-2^{18}$ $+2^{25}+2^{26}+2^{30}$	7
5	$\{0, \overline{6}, 7, 8, \overline{9}, \overline{10}, \overline{11}, 12, 31\}$	$-2^{4}-2^{5}-2^{8}-2^{20}$ $-2^{25}-2^{26}+2^{28}+2^{30}$	0	$-2^{4}-2^{8}-2^{20}$ $-2^{26}+2^{28}-2^{30}$	12
6	$\{16, \overline{17}, 20, \overline{21}, 31\}$	$2^{3}-2^{5}-2^{10}-2^{11}$ $-2^{16}-2^{21}-2^{25}$	0	$2^{3}-2^{10}-2^{21}-2^{31}$	17
7	$\{6, 7, 8, \overline{9}, 27, \overline{28}, 31\}$	$2^{16}-2^{27}+2^{31}$	0	$-2^{1}+2^{5}+2^{16}$ $+2^{25}-2^{27}$	22
8	$\{\overline{15}, 16, \overline{17}, 23, 24, 25, \overline{26}, 31\}$	$-2^{6}+2^{16}+2^{25}$	0	$2^{0}+2^{8}+2^{9}$ $+2^{16}+2^{25}-2^{31}$	7
9	$\{\overline{0}, 1, \overline{6}, \overline{7}, \overline{8}, 9, 31\}$	$2^{0}+2^{16}-2^{26}+2^{31}$	0	$2^{0}-2^{20}-2^{26}$	12
10	$\{12, 31\}$	$2^{6}+2^{31}$	0	-2^{27}	17
11	$\{31\}$	2^{31}	-2^{15}	$-2^{17}-2^{23}$	22
12	$\{\overline{7}, 13 \ldots 18, \overline{19}, 31\}$	$2^{17}+2^{31}$	0	$2^{0}+2^{6}+2^{17}$	7
13	$\{\overline{24} \ldots \overline{29}, 30, 31\}$	$-2^{13}+2^{31}$	0	-2^{12}	12
14	$\{31\}$	$2^{18}+2^{30}$	2^{31}	$2^{18}+2^{30}$	17
15	$\{3, 15, 31\}$	$-2^{25}+2^{31}$	0	$-2^{7}-2^{13}-2^{25}$	22
16	$\{\overline{29}, 31\}$	2^{31}	0	2^{24}	5
17	$\{31\}$	2^{31}	0	0	9
18	$\{31\}$	2^{31}	-2^{15}	2^{3}	14
19	$\{17, 31\}$	2^{31}	0	-2^{29}	20
20	$\{31\}$	2^{31}	0	0	5
21	$\{31\}$	2^{31}	0	0	9
22	$\{31\}$	2^{31}	0	2^{17}	14
23	0	0	2^{31}	0	20
24	0	2^{31}	0	0	5
25	0	0	2^{31}	0	9
$26-33$	0	0	0	0	RC_t
34	0	0	-2^{15}	-2^{15}	16
35	$\delta Q_{35} = 2^{31}$	2^{31}	2^{31}	0	23
36	$\delta Q_{36} = 2^{31}$	0	0	0	4
37	$\delta Q_{37} = 2^{31}$	2^{31}	2^{31}	0	11
$38-49$	$\delta Q_t = 2^{31}$	2^{31}	0	0	RC_t
50	$\delta Q_{50} = 2^{31}$	0	2^{31}	0	15
$51-59$	$\delta Q_t = 2^{31}$	2^{31}	0	0	RC_t
60	$\delta Q_{60} = 2^{31}$	0	2^{31}	0	6
61	$\delta Q_{61} = 2^{31}$	2^{31}	-2^{15}	-2^{15}	10
62	$\delta Q_{62} = 2^{31} - 2^{25}$	2^{31}	0	0	15
63	$\delta Q_{63} = 2^{31} - 2^{25}$	2^{31}	0	0	21
64	$\delta Q_{64} = 2^{31} - 2^{25}$	$\times$	$\times$	$\times$	$\times$

Table 2-6: *Wang et al.'s second block sufficient bitconditions*

t	Conditions on Q_t: $b_{31} \dots b_0$	#
-2	`......0. ........ ........ ........`	(1)
-1	`^....01. ........ ........ ........`	(3)
0	`^....00. ........ ........ ..0.....`	(4)
	Total # IV conditions for first block	(8)
1	`!...010. ..1..... ....0... .10.....`	8
2	`^^^^110. ..0^^^^0 ...^1... ^10..00.`	20
3	`^011111. ..011111 ...01..1 011^^11.`	23
4	`^011101. ..000100 ...00^^0 0001000^`	26
5	`!10010.. ..101111 ...01110 01010000`	25
6	`^..0010. 1.10..10 1..01100 01010110`	24
7	`!..1011^ 1.00..01 1..11110 00.....1`	20
8	`^..00100 0.11..10 1.....11 11....^0`	18
9	`^..11100 0.....01 0..^..01 11....01`	17
10	`^....111 1....011 1100..11 11....00`	18
11	`^....... ....^101 1100..11 11....11`	15
12	`^^^^^^^^ ....1000 0001.... 1.......`	17
13	`!0111111 ....1111 111..... 0...1...`	17
14	`^1000000 ....1011 111..... 1...1...`	17
15	`01111101 ........ 0....... ....0...`	10
16	`0.1..... ........ ........ ........`	2
17	`0....... ......0. ^....... ....^...`	4
18	`0.^..... ......1. ........ ........`	3
19	`0....... ......0. ........ ........`	2
20	`0....... ........ ........ ........`	1
21	`0....... ......^. ........ ........`	2
22	`0....... ........ ........ ........`	1
23	`0....... ........ ........ ........`	1
24	`1....... ........ ........ ........`	1
$25-45$	`........ ........ ........ ........`	0
46	`........ ........ ........ ........`	0
47	`........ ........ ........ ........`	0
48	`m....... ........ ........ ........`	1
49	`m....... ........ ........ ........`	1
50	`#....... ........ ........ ........`	1
51	`m....... ........ ........ ........`	1
52	`m....... ........ ........ ........`	1
53	`m....... ........ ........ ........`	1
54	`m....... ........ ........ ........`	1
55	`m....... ........ ........ ........`	1
56	`m....... ........ ........ ........`	1
57	`m....... ........ ........ ........`	1
58	`m....... ........ ........ ........`	1
59	`m....... ........ ........ ........`	1
60	`#.....0. ........ ........ ........`	2
61	`m.....1. ........ ........ ........`	2
62	`m.....1. ........ ........ ........`	2
63	`m.....1. ........ ........ ........`	2
64	`........ ........ ........ ........`	0
	Total # conditions	**312**

Uses bitconditions as defined in Table B-1 limited to only the variables $Q_i[j]$, not $Q'_i[j]$.