Single-block collision attack on MD5

Marc Stevens

Cryptology Group, CWI
P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands
marc@marc-stevens.nl

January 19, 2012

Abstract

In 2010, Tao Xie and Dengguo Feng [XF10] constructed the first single-block collision for MD5 consisting of two 64-byte messages that have the same MD5 hash. Details of their attack, developed using what they call an evolutionary approach, has not been disclosed "for security reasons". Instead they have posted a challenge to the cryptology community to find a new different single-block collision attack for MD5. This paper answers that challenge by presenting a single-block collision attack based on other message differences together with an example colliding message pair. The attack is based on a new collision finding algorithm that exploits the low number of bitconditions in the first round. It uses a clever trick to choose message blocks that satisfy bitconditions up to step 22 and additionally uses three known tunnels to correct bitconditions up to step 25. The attack has an average runtime complexity equivalent to $2^{49.8}$ calls to MD5's compression function.

1 Introduction

The first MD5 collision by Wang et al. [WY05] in 2004 has set off an research impulse in the cryptanalysis of MD5. Their 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} calls to MD5's compression function. Since then many improvements have been made [YS05, SNKO05, LL05, Kli05, Ste06, Kli06]. Currently, collisions for MD5 based on Wang et al.'s 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. Xie and Feng [XF09] have described promising message differences that might lead to a very fast attack, with possible a runtime complexity of roughly 2^{10} compressions. However, they have not published an actual attack based on these message differences so far. Instead, they have constructed a MD5 collision attack with a complexity of about $2^{20.96}$ MD5 compressions using less promising message differences. Our fastest collision attack [SSA+09] is based on slightly different message block differences than those used by Wang et al. and has a theoretical computational cost of about 2^{16} compression function calls.

The above mentioned attacks have a limited potential for abuse due to the requirement that the intermediate hash values given as input to the collision attack have to be identical. This requirement is most easily fulfilled by having two identical prefixes. For that reason collision attacks of this form are called *identical-prefix* collision attacks. In 2007, a more powerful collision attack called a *chosen-prefix* collision attack was introduced [SLdW07] that removes this requirement. This additional freedom comes at a cost: chosen-prefix collision attacks are significantly slower and require more message blocks compared to identical-prefix collision attacks. Initially, a chosen-prefix collision could be found with an average computational cost of about 2^{49} compression function calls. Since then this attack has been improved to about 2^{39} compression function calls [SSA+09]. Also a very short chosen-prefix collision attack requiring only $512 + 84 = 596$ bits[1] and a computational cost of about $2^{53.2}$ compressions has been presented [SSA+09]. The most convincing abuse scenario of MD5 collision attacks was presented in 2009 when a rogue Certification Authority certificate signed by a trusted commercial Certification Authority was obtained using a chosen-prefix collision [SSA+09]. An overview of other abuse scenarios based on identical-prefix collisions and chosen-prefix collisions is given in [SLdW12].

Recently even single-block identical-prefix collisions have been found by Xie and Feng [XF10], although they do not present their new techniques or other details "for security reasons". Instead, they have made a challenge to the cryptographic community to find a different single-block identical-prefix collision attack. This paper answers this challenge by presenting a new single-block identical-prefix collision attack for MD5 and an example colliding message pair. Our new collision attack uses the three known best tunnels and a new algorithm that exploits the very low number of bitconditions in the first round to deal with a rather high number of bitconditions in the second round.

2 Preliminaries

2.1 Notation

2.1.1 32-bit words

MD5 is designed with a 32-bit computing architecture in mind and operates on words $(v_{31} \ldots v_0)$ 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 paper we switch freely between the bitwise and $\mathbb{Z}_{2^{32}}$ representation of 32-bit words. 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:

$$\begin{aligned} &RL(\mathtt{10100100111111111111111100000001}_2,\ 5) \\ = \;&\mathtt{10011111111111111111100000000110100}_2; \end{aligned}$$

1. When using chosen-prefixes having a bitlength of $512 \cdot N - 84$ with $N \in \mathbb{N}$, otherwise extra padding bits are required. Hence, the colliding messages are at least 1024 bits (2 message blocks) long.

2.1.2 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. 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.

2.1.3 Related variables and differences

In collision attacks we consider two related messages M and M'. In this paper any variable X related to the message M or its MD5 calculation may have a corresponding variable X' related to the message M' or its MD5 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 Definition 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. As a result the total bit length of the padded message $\widehat{M}$ is $512N$ for a positive integer N.
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:

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

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 Definition of MD5Compress

MD5's compression function *MD5Compress* uses solely 32-bit words. The input for the compression function $\mathit{MD5Compress}(\mathit{IHV}_{\mathrm{in}}, B)$ consists of an intermediate hash value $\mathit{IHV}_{\mathrm{in}} = (a, b, c, d)$ consisting of four words 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:

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

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

The non-linear function f_t depends on the round:

$$f_t(X, Y, Z) = \begin{cases} F(X, Y, Z) = (X \wedge Y) \oplus (\overline{X} \wedge Z) & \text{for } 0 \leq t < 16, \\ G(X, Y, Z) = (Z \wedge X) \oplus (\overline{Z} \wedge Y) & \text{for } 16 \leq t < 32, \\ H(X, Y, Z) = X \oplus Y \oplus Z & \text{for } 32 \leq t < 48, \\ I(X, Y, Z) = Y \oplus (X \vee \overline{Z}) & \text{for } 48 \leq t < 64. \end{cases} \tag{1}$$

The message block B is partitioned into sixteen consecutive words $m_0, m_1, \ldots, m_{15}$ (with little-endian byte ordering), and expanded to 64 words W_t, for $0 \leq t < 64$, of 32 bits each:

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

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 maintains a working register with four state words Q_t, Q_{t-1}, Q_{t-2} and Q_{t-3} and calculates a new state word Q_{t+1}. With $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$, for $t = 0, 1, \ldots, 63$ in succession Q_{t+1} is calculated as follows:

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

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

$$\mathit{MD5Compress}(\mathit{IHV}_{\mathrm{in}}, B) = (a + Q_{61},\ b + Q_{64},\ c + Q_{63},\ d + Q_{62}). \tag{3}$$

2.3 Bitconditions

Our collision finding algorithm depends on a set of sufficient bitconditions on $(Q_t[b], Q'_t[b])$ that facilitates the search for a message block pair that satisfies a given differential path. Table 1 lists the possible bitconditions that may be used. We define $\mathfrak{q}_t[b]$ to denote the bitcondition on $(Q_t[b], Q'_t[b])$, and we define $\mathfrak{q}_t$ to denote all bitconditions $(\mathfrak{q}_t[i])_{i=0}^{31}$ on (Q_t, Q'_t).

Table 1: *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

3 Our single-block collision attack

3.1 Overview

For our single-block collision attack we have selected the following message differences:

$$\delta m_8 = 2^{25}, \quad \delta m_{13} = 2^{31}, \quad \delta m_i = 0 \text{ for } 0 \leq i < 16, i \neq 8, 13.$$

These message differences lead to the partial differential path presented in Table 2. We have chosen these message differences due to their properties similar to those of the message differences used by Xie and Fengg [XF10]: $\delta m_5 = 2^{10}$ and $\delta m_{10} = 2^{31}$.

In Section 3.2, we first construct a full differential path and a set of sufficient bitconditions, based on the partial differential path given in Table 2, using our techniques presented in [SLdW12]. Next, we present our new collision finding algorithm in Section 3.3 followed by the collision attack complexity analysis in Section 3.4. Finally, we present a found example colliding message pair each consisting of only a single block of 512 bits in Section 3.5.

3.2 Bitconditions

We used our differential path construction algorithm from [SLdW07][2] to construct a full differential path based on the partial differential path in Table 2.

In order to keep the number of necessary bitconditions in the first round to a minimum, we have chosen to perform the connection over the steps 18–21, instead of our usual choice for the steps 12–15. In particular, our new collision finding algorithm depends on a low number of bitconditions on the variables Q_2, Q_7, Q_8, Q_{12} and Q_{13}, whereas all bitconditions on the variables Q_{14} up to Q_{21} can be fulfilled freely. Differential paths of this form are usually only possible if they start with $\delta IHV_{\text{in}} = (0, 0, 0, 0)$, which is not the case for chosen-prefix collision attacks and the second near-collision attack in a two-block identical-prefix collision attack. However, a single-block identical-prefix collision has only one message block pair and its compression starts with $\delta IHV_{\text{in}} = (0, 0, 0, 0)$.

Table 2: *Partial differential path*

t	δQ_t	δF_t	δW_t	$\delta T_t^{\dagger}$	$\delta R_t^{\dagger}$
26	$2^7 - 2^{22}$				
27	$-2^4 - 2^{25} + 2^{31}$				
28	$-2^4 - 2^{11} + 2^{25} + 2^{31}$				
29	$2^4 - 2^{16} - 2^{25}$	0	0	$2^7 - 2^{22}$	$2^{16} + 2^{31}$
30	$2^4 - 2^{25} + 2^{31}$	$2^4 + 2^{11} - 2^{25} + 2^{31}$	0	$2^{11} - 2^{26}$	$-2^8 + 2^{25}$
31	$2^4 - 2^8 + 2^{31}$	$2^4 - 2^{16} - 2^{25} + 2^{31}$	0	$-2^{11} - 2^{16}$	$-2^4 + 2^{31}$
32	-2^8	$+2^{25}$	0	$2^4 - 2^{16}$	$2^8 - 2^{20}$
33	-2^{20}	$-2^4 + 2^{20} + 2^{31}$	2^{25}	2^{20}	2^{31}
34	$-2^{20} + 2^{31}$	$2^8 + 2^{31}$	0	2^4	2^{20}
35	2^{31}	0	0	-2^8	2^{31}
36	0	$+2^{20}$	0	0	0
37	0	$+2^{31}$	0	-2^{20}	2^{31}
38	2^{31}	2^{31}	0	0	0
39	2^{31}	0	0	0	0
40	2^{31}	2^{31}	2^{31}	0	0
41 – 55	2^{31}	2^{31}	0	0	0
56	2^{31}	2^{31}	2^{25}	2^{25}	2^{31}
57	0	2^{31}	0	0	0
58	0	2^{31}	0	0	0
59	0	0	2^{31}	0	0
60	0	0	0	0	0
61	0	0	0	0	0
62	0	0	0	0	0
63	0	0	0	0	0
64	0				

† Note that $\delta T_t = \delta Q_{t-3} + \delta F_t + \delta W_t$ and $\delta R_t \in \{RL(X + \delta T_t, RC_t) - RL(X, RC_t) \mid X \in \mathbb{Z}_{2^{32}}\}$.

3.3 Algorithm

We present our new collision finding algorithm as Algorithm 1. It is designed to allow any number of bitconditions over Q_{14} up to Q_{21} by exploiting a large degree of freedom in the first round (very few bitconditions). Our algorithm is extended by the three known best tunnels, whose auxiliary bitconditions in the first round do not have an averse effect on the attack's runtime complexity.

Algorithm 1 can be roughly split into four parts:

1. *Instantiation (1.–3.)*: randomly choose values for Q_{14} up to Q_{21} satisfying the given bitconditions. These values directly imply values for m_6 (step 17), m_{11} (step 18), m_0 (step 19), m_5 (step 20) and Q_1 (step 0).

2. *Precomputations (4.–7.)*: first a lookup table is generated containing tuples of valid values for Q_2 up to Q_7 and Q_{13} that satisfy equations for steps 1 (uses m_1), 5 (uses m_5), 6 (uses m_6) and 16 (uses m_1) and the given bitconditions. The lookup table is indexed by the values of the bits of Q_7 and Q_{13} that are involved with Q_8 and Q_{12} due to indirect bitconditions. These bitpositions are marked with '1'-bits in the 32-bit word masks B_8 and B_{13} in Algorithm 1.

2. Implementation available at `http://code.google.com/p/hashclash`.

Table 3: *Bitconditions*

t	$\mathfrak{q}_t[31] \ldots \mathfrak{q}_t[0]$
-3 – 2	....
3	010.0.00 .0001... 01.1...0 00.000..
4	00000000 00000000 00000000 00000000
5	11101011 01111000 11010001 11011100
6	...1.1.. 1....111 ..1.111. ..1...11
7	0
8	....^0
9	00000000 00000000 000000.0 0!0000+-
10	00000000 00000000 000000.0 0.00000+
11	11111111 11111111 11111101 1.11111+
12	....1..00. ......+-
13	....^.^^ .^^^.... ..1.00+. ...^^^0+
14	010.0.00 .0001.^. 01+10--0 00.0001+
15	000.0.00 .00001+. 00+00--0 001000+-
16	0001010+ 100000+1 00+0-+-0 00100001
17	.0..10.+ 0+000++0 10+10--1 01+0.000
18	.01+1-+1 -+-+-01- --+0-+++ ---010+.
19	.-0++.+1 00+011-1 ++1----- +1--0+0.
20	..-10^0. 1+010.11 011.+100 0011+.+.
21	.^11+.1. .1+-..1+ 11.^0010 0-10.^..
22	..+.-... .-+...1+ ....-... .-.-^.^.
23	....1.0. .+-^..+0 ....-... .+.-...0
24	..^.1.0. .-0....0 ....0... 1+.+...1
25	0.....-. .0-...^. ....1... 1+.+...+
26	......0. .-...... ....1... +1.1....
27	-.....-. ..^....0 ....1... .1.-...^
28	-.....+. .^.....1 ....-... ^..-....
29	0.....-. ....- ....1..0 ...+....
30	-.....-. ....0 ....0... ...+....
31	-.....1. ....1 ....- ...+....
32	......0. ...!.... ....- ....
33	!....... ...-.... ....!....
34	+....... ...-.... ....! ....
35	+....... ....
36	....!.... ....
37	!....... ....
38	+....... ....
39 – 56	+....... ....
57	1....... ....
58	0....... ....
59 – 64	....

Includes bitconditions for tunnels (see Table 4). Bitconditions (and rotations) are only verified up to Q_{29}. Variations of the above differential path over steps $t = 29, \ldots, 63$ are allowed. E.g., differential paths with different signs of $\Delta Q_t[31]$.

3. *Main loop (8.–11.)*: iterate over valid values for Q_8 up to Q_{12} satisfying bitconditions and the step equation for step 11 using m_{11}: Find all values in the lookup table that satisfy all indirect bitconditions between Q_7 and Q_8, and between Q_{12} and Q_{13} using the index. For each of these values, the variables Q_{-3} up to Q_{16} are known and thus the entire message

Algorithm 1 Single-block collision finding algorithm

Input: $IHV_{\text{in}} \in (\mathbb{Z}_{2^{32}})^4$;
Output: $M \neq M' \in \{0,1\}^{512}$ such that $\mathit{MD5Compress}(IHV_{\text{in}}, M) = \mathit{MD5Compress}(IHV_{\text{in}}, M')$;
Uses the bitconditions given in Table 3 and the tunnels shown in Table 4.

Algorithm:

1. Initialize Q_{-3}, Q_{-2}, Q_{-1} and Q_0 with IHV_{in}
2. Randomly choose $Q_{14}, \ldots, Q_{21}$ satisfying bitconditions†
3. Compute m_6, m_{11}, m_0 and m_5 at steps $t = 17, 18, 19, 20$, and Q_1 (at step 0)
4. Generate lookup table:
5. Loop over all values for Q_3, Q_4, Q_5 and Q_6 that satisfy bitconditions†:
6. Compute Q_7 (at step 6), Q_2 (at step 5), m_1 (at step 1) and Q_{13} (at step 16)
7. If Q_7, Q_2 and Q_{13} satisfy bitconditions‡ then
append tuple $(Q_2, Q_3, Q_6, Q_7, Q_{13})$ to lookup table at index $(Q_7 \wedge B_8, Q_{13} \wedge B_{13})$
8. Loop over all values for Q_9, Q_{10}, Q_{11} and Q_{12} that satisfy bitconditions†:
9. Compute Q_8 at step 11
10. If Q_8 satisfies bitconditions (including indirect ones involving Q_9) then
loop over all $(Q_2, Q_3, Q_6, Q_7, Q_{13})$ in lookup table at index $(Q_8 \wedge B_8, Q_{12} \wedge B_{13})$:
11. Compute $m_0, \ldots, m_{15}$ (at steps $0, \ldots, 15$) and Q_{22} and Q_{23}
12. If Q_{22} and Q_{23} satisfy bitconditions then
Loop over tunnel $\mathcal{T}_4$ to alter m_3, m_4 and m_7:
13. Compute m_4 (step 4) and Q_{24} (step 23)
14. If Q_{24} satisfies bitconditions then
loop over tunnel $\mathcal{T}_9$ to alter m_8, m_9 and m_{12}:
15. Compute m_9 (step 9) and Q_{25} (step 24)
16. If Q_{25} satisfies bitconditions then
loop over tunnel $\mathcal{T}_{14}$ to alter m_2, m_3, m_6 (at steps 6 and 17), m_{13} and m_{14}:
17. Compute m_{14}, m_3, m_8, m_{13} (steps 14, 3, 8, 13) and Q_{26} – Q_{29} (steps 25–28)
18. If Q_{26}, Q_{27}, Q_{28} and Q_{29} satisfy bitconditions then:
19. Let $M = (m_i)_{i=0}^{15}$ and $M' = M + \delta M$
20. If $\mathit{MD5Compress}(IHV_{\text{in}}, M) = \mathit{MD5Compress}(IHV_{\text{in}}, M')$ then **return** (M, M').
21. Repeat steps *1.* through *20.* (until a collision is found)

† Ignoring any indirect bitconditions involving variables Q_i whose values are not yet known.
‡ Including all indirect bitconditions involving these variables and variables Q_i whose values are known.
Note: The words $B_8, B_{13} \in \mathbb{Z}_{2^{32}}$ are constants such that $B_t[b] = 1 \Leftrightarrow \mathfrak{q}_t[b] =$ '^' for $t \in \{8, 13\}, b \in \{0, \ldots, 31\}$.

block is determined. Compute Q_{22} and Q_{23} and if they satisfy the given bitconditions then do the last part.

4. *Tunnels* (*12.–20.*): use the three known best tunnels (see Table 4) to make very precise corrections to the message block pair such that all bitcondition up to Q_{23} remain fulfilled. For all message block pairs that satisfy bitconditions $\mathfrak{q}_{-3}$ up to $\mathfrak{q}_{29}$, check whether the message block pair forms a collision.

Our algorithm uses the three known best tunnels that are described in Table 4 and allow to efficiently satisfy bitconditions $\mathfrak{q}_{24}$, $\mathfrak{q}_{25}$ and $\mathfrak{q}_{26}$.

Table 4: *Tunnels for MD5.*

Tunnel	Change	Affected	Extra bitconditions*	Bitmask$^{\diamond}$
$\mathcal{T}_4$	$Q_4[b]$	$m_3, m_4, m_7, Q_{24}..Q_{64}$	$Q_5[b] = 0, Q_6[b] = 1$	$\texttt{14872e23}_{16}$
$\mathcal{T}_9$	$Q_9[b]$	$m_8, m_9, m_{12}, Q_{25}..Q_{64}$	$Q_{10}[b] = 0, Q_{11}[b] = 1$	$\texttt{fffffdbc}_{16}$
$\mathcal{T}_{14}$	$Q_{14}[b]$ $Q_3[b]$	$m_{13}, m_{14}, m_6, Q_{26}..Q_{64}$ m_2, m_3	$Q_{15}[b] = Q_{16}[b] = 0, Q_3[b] = Q_{14}[b]^{\dagger}$ $Q_4[b] = 0, Q_5[b] = 1^{\ddagger}$	$\texttt{eb78d1dc}_{16}$

⋆ Extra bitconditions refer only to $Q_t[b]$ and not to $Q'_t[b]$. E.g., $Q_5[b] = 0$ is met by both $\mathfrak{q}_5[b] =$ '0' and $\mathfrak{q}_5[b] =$ '+'.
⋄ A 32-bit word whose '1'-bits describe the bits b that are used for tunnels in combination with Table 3.
† Bitcondition $\mathfrak{q}_3[b] =$ '.' and no other indirect bitconditions may involve $Q_3[b]$.
‡ An extra bitcondition $Q_{12}[b] = Q_{13}[b]$ may be used such that the change in m_{14} can be accurately predicted.

3.4 Complexity analysis

We have implemented our single-block collision attack in C++. The sources and a Windows executable can be found at `http://marc-stevens.nl/research`. Our algorithm can be freely parallelized by using different instantiations for each thread.

Our implementation frequently shows the number of message block pairs it has found that satisfies bitconditions $\mathfrak{q}_{-3}, \ldots, \mathfrak{q}_{29}$, together with the wall time that has passed. Based on these numbers, we have experimentally determined the average runtime complexity of finding one message block pair that satisfies bitconditions $\mathfrak{q}_{-3}, \ldots, \mathfrak{q}_{29}$ being equivalent to about $2^{15.96}$ MD5 compressions[3]. Furthermore, we have experimentally determined that the probability that a message block pair satisfying $\mathfrak{q}_{26}$, $\mathfrak{q}_{27}$, $\mathfrak{q}_{28}$ and $\mathfrak{q}_{29}$ results in a collision is about $2^{-33.85}$. Hence, our single-block collision attack has a runtime cost equivalent to about $2^{15.96} \cdot 2^{33.85} = 2^{49.81}$ MD5 compressions.

3.5 Results

Based on our complexity analysis and a number of computers available for our collision search, we estimated that it would take approximately five weeks. As this was feasible enough, we started the actual search.

It was our fortune that a collision was found a bit earlier, namely after only three weeks. We present our found example colliding message pair in Table 5. The two colliding messages can also be downloaded at `http://marc-stevens.nl/research`.

3. Measured on an Intel Core2 Q9550 cpu.

Table 5: *Example single-block collision – in hexadecimal notation*

Message 1

```
4d c9 68 ff 0e e3 5c 20 95 72 d4 77 7b 72 15 87
d3 6f a7 b2 1b dc 56 b7 4a 3d c0 78 3e 7b 95 18
af bf a2 00 a8 28 4b f3 6e 8e 4b 55 b3 5f 42 75
93 d8 49 67 6d a0 d1 55 5d 83 60 fb 5f 07 fe a2
```

Message 2

```
4d c9 68 ff 0e e3 5c 20 95 72 d4 77 7b 72 15 87
d3 6f a7 b2 1b dc 56 b7 4a 3d c0 78 3e 7b 95 18
af bf a2 02 a8 28 4b f3 6e 8e 4b 55 b3 5f 42 75
93 d8 49 67 6d a0 d1 d5 5d 83 60 fb 5f 07 fe a2
```

Common MD5 hash

```
008ee33a9d58b51cfeb425b0959121c9
```

4 Concluding remarks

Tao Xie and Dengguo Feng [XF10] have posted a challenge to the cryptology community to find a new different single-block collision attack for MD5. In this paper we have met their challenge by presenting a new single-block collision attack for MD5 that is based on other message differences than those used by Xie and Feng. Our single-block collision attack has a runtime complexity equivalent to about $2^{49.8}$ calls to MD5's compression function. Since Xie and Feng disclosed very little details about their attack, we are unable to compare the complexities of these two single-block collision attacks for MD5.

Furthermore, we have successfully implemented our collision attack which resulted in an example colliding message block pair. Our implementation sources, a Windows executable and the two colliding 64-byte messages can be found at `http://marc-stevens.nl/research`.

References

[HPR04] Philip Hawkes, Michael Paddon, and Gregory G. Rose, *Musings on the Wang et al. MD5 Collision*, Cryptology ePrint Archive, Report 2004/264, 2004.

[Kli05] Vlastimil Klima, *Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications*, Cryptology ePrint Archive, Report 2005/102, 2005.

[Kli06] Vlastimil Klima, *Tunnels in Hash Functions: MD5 Collisions Within a Minute*, Cryptology ePrint Archive, Report 2006/105, 2006.

[LL05] Jie Liang and Xuejia Lai, *Improved Collision Attack on Hash Function MD5*, Cryptology ePrint Archive, Report 2005/425, 2005.

[Riv92] Ronald L. Rivest, *The MD5 Message-Digest Algorithm*, Internet Request for Comments, April 1992, RFC 1321.

[SLdW07] Marc Stevens, Arjen K. Lenstra, and Benne de Weger, *Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities*, EUROCRYPT (Moni Naor, ed.), Lecture Notes in Computer Science, vol. 4515, Springer, 2007, pp. 1–22.

[SLdW12] Marc Stevens, Arjen K. Lenstra, and Benne de Weger, *Chosen-prefix collisions for MD5 and applications*, 2012, to appear in: Internation Journal of Applied Cryptography.

[SNKO05] Yu Sasaki, Yusuke Naito, Noboru Kunihiro, and Kazuo Ohta, *Improved Collision Attack on MD5*, Cryptology ePrint Archive, Report 2005/400, 2005.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger, *Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate*, CRYPTO (Shai Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, 2009, pp. 55–69.

[Ste06] Marc Stevens, *Fast Collision Attack on MD5*, Cryptology ePrint Archive, Report 2006/104, 2006.

[WY05] Xiaoyun Wang and Hongbo Yu, *How to Break MD5 and Other Hash Functions*, EUROCRYPT (Ronald Cramer, ed.), Lecture Notes in Computer Science, vol. 3494, Springer, 2005, pp. 19–35.

[XF09] Tao Xie and Dengguo Feng, *How To Find Weak Input Differences For MD5 Collision Attacks*, Cryptology ePrint Archive, Report 2009/223, 2009.

[XF10] Tao Xie and Dengguo Feng, *Construct MD5 Collisions Using Just A Single Block Of Message*, Cryptology ePrint Archive, Report 2010/643, 2010.

[YS05] Jun Yajima and Takeshi Shimoyama, *Wang's sufficient conditions of MD5 are not sufficient*, Cryptology ePrint Archive, Report 2005/263, 2005.