MasterMath Cryptology 2015 - 2/2 - Cryptanalysis

Friday, 17 April, 2015 09:59

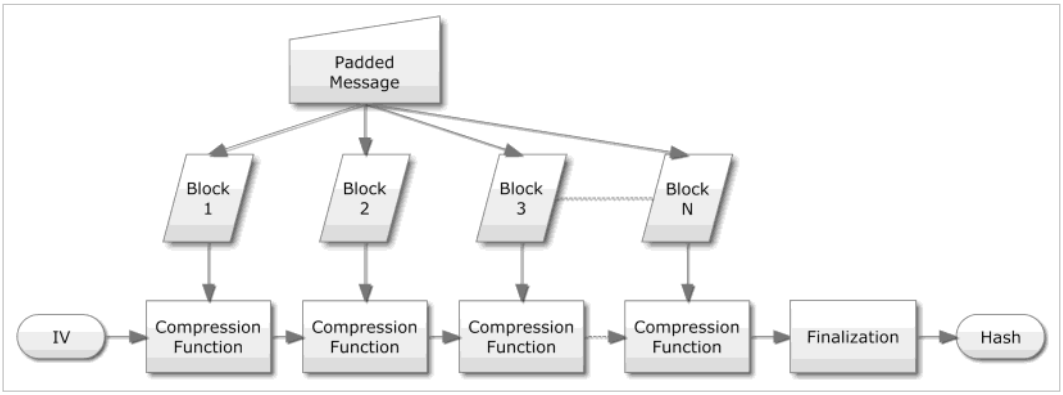
10. Hash Function Cryptanalysis (v3)

Cryptographic hash functions map messages of arbitrary size to a fixed size *hash*, e.g. a bitstring of length 256.
The most widely used hash functions are SHA-1, SHA-2-256 and SHA-2-512, whereas SHA-3 is the future standard.
SHA-1 is not collision resistant, its late predecessor and prior de facto standard MD5 is very weak.
All of these hash function standards are fixed hash functions and there is no family $\mathcal{F}$ to speak of, hence of the security properties discussed in Section 7 only aPre and aSec apply.
Remember that for fixed hash functions there is no formal definition of collision resistance due to the *Foundations-of-Hashing dilemma*.
Nevertheless many real world cryptographic systems depend on the *informal* collision resistance definition that for these fixed hash functions no one can find collisions faster than the birthday search.
Note that hash functions with N-bit output claim N-bit security against aPre, aSec(, Pre, ePre, Sec, eSec), but only $(N/2)$-bit security against collision resistance.

10.1. Hash Function Construction

Hash functions can process messages of arbitrary size, but do so by splitting the message into pieces and iteratively update its internal state using one piece.
The two most well-known methods are the Merkle-Damgard construction (independently proposed by Merkle and Damgard in 1989) used by MD5, SHA-1 and SHA-2-{224,256,384,512} and the Sponge construction (see http://www.infosec.sdu.edu.cn/cans2011/Joan%20SpongeCANS2011.pdf) known from SHA-3.

10.2. Merkle-Damgard construction



The Merkle-Damgard construction is based on a fixed-size *compression function* $f: \{0,1\}^N \times \{0,1\}^M \to \{0,1\}^N$ that takes as input the current N-bit state and a M-bit message piece and outputs the updated N-bit state.
The construction itself first unambigiously pads the message with 1s and 0s and the message bitlength, so that the total bitlength is a multiple of M, and splits the padded message $pad(M) = M_1||M_2||\cdots||M_l$ into l pieces M_i of bitlength M.
It then starts with an internal state CV_0 called the *chaining value* initialized to a fixed known value called the *initial value*:

$$CV_0 = IV.$$

Then for each message block it calls the compression function together with the last chaining value to produce the next chaining value:

$$CV_i = f(CV_{i-1}, M_i) \quad \text{for } i = 1, \dots, l$$

Finally it outputs the last chaining value CV_l as the hash (thus a trivial finalization).

10.2.1. Collision Reduction

For the Merkle-Damgard construction one can prove that to find a collision for the hash function one finds a collision for the compression function.
Thus one can argue that if one cannot find collisions for the compression function faster than the birthday search, then the same holds for the hash function.

Proof:

Let M and M' be two messages with the same hash $h(M) = h(M')$.
Then if M and M' are of different length then their last message pieces M_l and $M'_{l'}$ are different and together with the second last chaining values form a collision of f: $f(CV_{l-1}, M_l) = f(CV'_{l-1}, M'_{l'})$.
Otherwise M and M' have the same bitlength,
now consider the smallest $i \in \{1, \dots, l\}$ such that $(CV_{j-1}, M_j) = (CV'_{j-1}, M'_j)$ for all $j \geq$ i.
If $i = 1$ then M and M' are equal, which is a contradiction.
Finally we have $(CV_{i-2}, M_{i-1}) \neq (CV'_{i-2}, M'_{i-1})$ and $CV_{i-1} = CV'_{i-1}$, thus $(CV_{i-2}, M_{i-1}), (CV'_{i-2}, M'_{i-1})$ is a collision pair of f.

10.2.2. Weaknesses: length extension

A rather trivial weakness is that given any message M and hash $h(M)$, it is possible to compute the hash of the extended message $M||padding||S$ in time $O(|S|)$.
One can simply continue hashing given $h(M)$ using the pieces of S, naturally the final padding must use the total length including M, the padding of M and S.
This forms a problem for very simple *Message Authentication Codes* (MAC), where one computes a *tag* for a given message using a secret *key*.
Appending this *tag* to the message allows the receiver that knows the secret key to verify the message is authentic and has not been tampered with.
A very simple MAC would initialize CV_0 with the key, or equivalently output $h(key||message)$, and this would be secure if the hash function behaves as a random function.
However, due to the length extension attack, anyone can compute another valid $(message', tag')$-pair given $(message, tag)$-pair directly using the above length extension attack.

Although by some it may be considered a feature, not a bug, of the Merkle-Damgard based hash functions, it forces some (unexpected) care in constructing MACs from Merkle-Damgard based hash functions.

10.2.2. Weaknesses: Multi-collisions

Another weakness was exposed by Joux called the *multi-collision attack.*
The idea is that one can use the birthday search to find a one-block collision for the hash function:
$$f(IV, M_1) = f(IV, M'_1) \Rightarrow \; h(M_1) = h(M'_1)$$
And of course we can find another one-block collision using chaining value $f(IV, M_1)$:
$$f(f(IV, M_1), M_2) = f(f(IV, M_1), M'_2) \quad \Rightarrow \quad h(M_1||M_2) = h(M_1||M'_2)$$
But as $f(IV, M_1) = f(IV, M'_1)$, we actually have
$$h(M_1||M_2) = h(M_1||M'_2) = h(M'_1||M_2) = h(M'_1||M'_2)$$
Now iteratively construct t one-block collisions in this manner:
$$h(M_1||\cdots||M_{i-1}||M_i) = h(M_1||\cdots||M_{i-1}||M'_i) \quad \text{for } i = 1, \dots, t$$

For each of the t collisions one can choose either block M_i or M'_i, leading to a total of 2^t different messages that all hash to the same hash value.

10.2.3. Weaknesses: Faster second preimages against long messages

Against very long messages X of blocklength 2^l, a second preimage can be constructed in complexity $O\left(\max\left(2^{N/2+l}, 2^{N-l}\right)\right)$ using the following procedure by John Kelsey and Bruce Schneier (https://eprint.iacr.org/2004/304.pdf):

1. Construct an *expandable message*, i.e., a variant on the above multi-collision where the i-th collision consist of a single block part $|M_i| = M$ and a $(2^{i-1} + 1)$-block part $|M'_i| = (2^{i-1} + 1)M$.
Here we actually need *internal collisions*, i.e., a collision in the chaining value after processing only the message itself without any padding: all 2^t messages hash without padding to the same chaining value $\widehat{CV}$.
Also, all 2^t different messages that can be obtained have a different blocklength between t-blocks and $(t + 2^t - 1)$-blocks.
In particular, for any blocklength $n \in [t, t + 2^t - 1]$ we can construct a message m of blocklength n that hashes without padding to $\widehat{CV}$.
This step constructs $t = l$ collisions and requires finding l collisions of at most $2^{l-1} + 1$ blocks,
thus on average at most $O\left(l\left(2^{l-1} + 2\right)2^{N/2}\right)$ operations.

2. The original message M'' is of blocklength 2^l, thus there are 2^l chaining values $CV_1'', \dots, CV_{2^l}''$.
The idea is to find a block B such that $f\left(\widehat{CV}, B\right) \in \{CV''_{l+1}, \dots, CV''_{2^l}\}$, this requires on average about $O(2^{N-l})$ for large l (such that $(l + 1)/2^l$ is negligible).
Say $f\left(\widehat{CV}, B\right) = CV_r$, then we can construct another message Y that hashes to the same hash as M'':
   - For blocks 1 up to $(r - 1) \in [l, 2^l - 1]$, we choose the appropriate expandable message of length $r - 1$ that

hashes without padding to $\widehat{CV}$
- Block B will be the r-th block of Y, so chaining value $CV_r^Y = CV_r''$
- Blocks $(r+1)$ up to 2^l of Y are identical to those of M'', thus $CV_j^Y = CV_j''$ for $j = r+1, \dots, 2^l$. As M'' and Y have the same length $|Y| = |M''|$, it follows that Y has the same hash as M''.

10.2.4. Wide-pipe Merkle-Damgard

To counteract the above three weaknesses one can make a simple modification to the Merkle-Damgard construction.
As the above weaknesses depend either on the knowledge of the internal state or on collision attacks against the internal state, it suffices to make the internal state bitlength twice as big as the hash bitlength.
The so-called *wide-pipe* Merkle-Damgard construction uses an internal state twice as big as the output hash and uses a finalization function to produce the hash from the last internal state, which can be a simple truncation.
The length extension attacks against simple MACs don't work anymore as the attacker only knows the output hash and therefore has only partial knowledge about the last internal state that depends on the secret key.
Both the multi-collision attack and the second-preimage attack against long messages also don't work as they require more than $O(2^N)$ operations to find collisions in the internal state.

10.2.5. Instantiation using a block cipher

A block cipher $C = E_K(B)$ can be used to instantiate the compression function.
A trivial way would be to use a message block as the key input K, the current chaining value as the plaintext input B and use the output ciphertext C as the updated chaining value.
However, this is insecure and leads to a second pre-image attack of complexity $O(2^{N/2})$.

Meet-in-the-middle attack:

Given a 2-block message $M = M_1||M_2$ and chaining values $CV_0 = IV, CV_1 = f(CV_0, M_1), CV_2 = f(CV_1, M_2)$.
One can find a second preimage in $O(2^{N/2})$ operations if $f(CV, M) \coloneqq E_M(CV)$ using a *meet-in-the-middle attack* as follows:
Note that for fixed M, E is invertible, so given M, we can compute CV_1 from CV_2: $CV_1 = E_{M_2}^{-1}(CV_2)$.
Select $2^{N/2}$ random blocks M_1' and store the pairs $(f(IV, M_1') = E(M_1', IV), M_1')$.
Select $2^{N/2}$ random blocks M_2' and store the pairs $(E_{M_2'}^{-1}(CV_2), M_2')$.
There are 2^N pairs (M_1', M_2') and thus with high probability we can expect
to see one random collision pair (M_1', M_2') with

$$E_{M_1'}(IV) = E_{M_2'}^{-1}(CV_2).$$

It follows that $CV_0' = IV, CV_1' = E_{M_1'}(CV_0') = E_{M_1'}(IV) = E_{M_2'}^{-1}(CV_2)$,

$$CV_2' = E_{M_2'}(CV_1') = E_{M_2'}\left(E_{M_2'}^{-1}(CV_2)\right) = CV_2,$$

and that therefore $M' = M_1'||M_2'$ is a second preimage of $h(M)$ as it has the same hash as M.

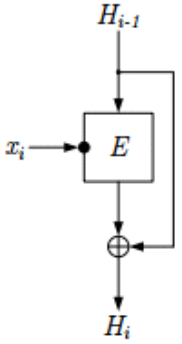
This attack can also be executed with very little memory using birthday search techniques like the distinguished point attack described in Section 7.3 (assuming block size $M \geq N$ is larger or equal to the state size):

Define $f_0: \{0,1\}^N \to \{0,1\}^N$ as $f_0(X) = E_{X||0^{M-N}}(IV)$
Define $f_1: \{0,1\}^N \to \{0,1\}^N$ as $f_1(X) = E_{X||0^{M-N}}^{-1}(CV_2)$
Select $\phi: \{0,1\}^N \to \{0,1\}$ such that $|\phi^{-1}(0)| = |\phi^{-1}(1)|$, e.g., $\phi(x_N \dots x_0) = x_0$.
Define $f_2: \{0,1\}^N \to \{0,1\}^N$ as $f_3(X) = f_{\phi(X)}(X)$

And use f_2 as iterator function in the birthday search described in Section 7.3.
Note that a collision $f_2(X) = f_2(Y)$ is only useful to construct a second preimage if $\phi(X) \neq \phi(Y)$.
This implies that a f_2-collision has probability 0.5 to be useful and that on average we expect to need to find 2 f_2-collisions.
See http://people.scs.carleton.ca/~paulv/papers/JoC97.pdf: as the expected number of collisions grows quadratically with the amount of work, we can expect to do $\sqrt{2}$ more work than normally.
Thus the expected complexity is $\sqrt{\pi 2^N/2} \cdot \sqrt{2} = \sqrt{\pi 2^N} = \sqrt{\pi} \cdot 2^{N/2}$.

Black, Rogaway and Shrimpton have proven that there are 12 basic ways of instantiating a compression function from a block cipher that leads to a secure hash function (see http://link.springer.com/content/pdf/10.1007%2F3-540-45708-9_21.pdf).
The most common secure way to do so is the Davies-Meyer construction:

$$CV_i = f(CV_{i-1}, M_i) \coloneqq E(M_i, CV_{i-1}) \oplus CV_{i-1}$$



Davies-Meyer mode:
$H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}$

10.3. A very broken hash function: MD5

MD5 is a hash function designed by Ron Rivest in 1991 and has been used a the de facto hash function for digital signatures up to 2004.
It has a hash size of 128 bits and thus was designed to offer 64-bit security against collisions and 128-bit security against (second) pre-image attacks.

> *Note that 64-bit and even 80-bit security is not sufficient in practice:*
> *The combined Bitcoin network currently performs $2^{58.3}$ SHA-2 hashes per second or $2^{83.2}$ SHA-2 hashes per year in Bitcoin mining (see https://blockchain.info/charts/hash-rate).*

As we will show in these lecture notes below, MD5 is weak and differential cryptanalysis can be used to construct collisions for MD5 within a second on modern PCs.

MD5 uses 32-bit words X which can be seen as an integer modulo 2^{32} and simultaneously as a 32-bit string:

$$X: \quad x_{31}x_{30}\cdots x_1x_0 \quad \leftrightarrow \quad \sum_{i=0}^{31} x_i 2^i \pmod{2^{32}}$$

We will write 32-bit words using hexadecimal notation, and use *word* when we mean a 32-bit word.
It uses the following operations on 32-bit words X:

- Addition and subtraction modulo 2^{32}
- Bitwise AND ($X \wedge Y$), XOR ($X \oplus Y$), OR ($X \vee Y$), NOT ($\bar{X}$) on 32-bit strings X and Y.
- Cyclic bitwise rotation of a 32-bit string by n bit positions:
  $RL(x_{31}x_{30}\cdots x_0, 1) = x_{30}\cdots x_0x_{31}$

It uses *Little Endian* ordering to parse a bit sequence or byte sequence into words, see http://en.wikipedia.org/wiki/Endianness .
In particular it means that a byte sequence $b_0b_1b_2b_3b_4b_5b_6b_7$ … (with integers $b_i \in [0,255]$) is parsed into a word sequence $w_0w_1w_2$ … using:

$$w_i = \sum_{j=0}^{3} b_{(4i+j)} 2^{8j}, \qquad \text{e.g.}, w_0 = b_0 2^0 + b_1 2^8 + b_2 2^{16} + b_3 2^{24}, w_1 = b_4 2^0 + b_5 2^8 + b_6 2^{16} + b_7 2^{24}.$$

The initial value is defined as the word tuple: $IV = (67452301, efcdab89, 98badcfe, 10325476)$.
The compression function takes as input a 4 word tuple $CV_{in} = (A, B, C, D)$ and a 16 word tuple $(m_0, \ldots, m_{15})$ and outputs a 4 word tuple CV_{out} that is computed as follows:

1. Let $W_0, \ldots, W_{63}$ and $F_i(b,c,d)$ defined as follows:

$W_i = m_i$ $BF_i(b,c,d) = (b \wedge c) \vee (\bar{b} \wedge d)$	for $i \in [0,15]$
$W_i = m_{1+5i \bmod 16}$ $BF_i(b,c,d) = (d \wedge b) \vee (\bar{d} \wedge c)$	for $i \in [16,31]$
$W_i = m_{5+3t \bmod 16}$ $BF_i(b,c,d) = b \oplus c \oplus d$	for $i \in [32,47]$
$W_i = m_{7t \bmod 16}$ $BF_i(b,c,d) = c \oplus (b \vee \bar{d})$	for $i \in [48,63]$

2. Let the addition constants be $AC_i = \lfloor 2^{32} |\sin(i+1)| \rfloor$ for $i \in [0,63]$

3. Let the rotation constants RC_i be defined as follows:

$(RC_i, RC_{i+1}, RC_{i+2}, RC_{i+3}) = (7,12,17,22)$	for $i = 0,4,8,12$
$(RC_i, RC_{i+1}, RC_{i+2}, RC_{i+3}) = (5,9,14,20)$	for $i = 16,20,24,28$
$(RC_i, RC_{i+1}, RC_{i+2}, RC_{i+3}) = (4,11,16,23)$	for $i = 32,36,40,44$
$(RC_i, RC_{i+1}, RC_{i+2}, RC_{i+3}) = (6,10,15,21)$	for $i = 48,52,56,60$

4. Let $(a, b, c, d) = (A, B, C, D)$
5. For $i = 0, \dots, 63$ do
    5.1. $F_i = BF_i(b, c, d)$
    5.2. $T_i = a + F_i + AC_i + W_i$
    5.3. $R_i = RL(T_i, RC_i)$
    5.4. $(a, b, c, d) = (d, b + R, b, c)$
6. return $(A + a, B + b, C + c, D + d)$

Another way to define the compression function that unrolls the cyclic state (step 5.3) is by replacing the following steps:

4. Let $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$
5. For $i = 0, \dots, 63$ do
    5.1. $F_i = BF_i(Q_i, Q_{i-1}, Q_{i-2})$
    5.2. $T_i = Q_{i-3} + F_i + AC_i + W_i$
    5.3. $R_i = RL(T_i, RC_i)$
    5.4. $Q_{i+1} = Q_i + R_i$
6. return $CV_{out} = (A + Q_{61}, B + Q_{64}, C + Q_{63}, D + Q_{62})$

This description simplifies the cryptanalysis and is what we'll use.

We'll leave it to the reader to verify the equivalence of these two description.

t	AC_t	RC_t	W_t
0	$\texttt{d76aa478}_{16}$	7	m_0
1	$\texttt{e8c7b756}_{16}$	12	m_1
2	$\texttt{242070db}_{16}$	17	m_2
3	$\texttt{c1bdceee}_{16}$	22	m_3
4	$\texttt{f57c0faf}_{16}$	7	m_4
5	$\texttt{4787c62a}_{16}$	12	m_5
6	$\texttt{a8304613}_{16}$	17	m_6
7	$\texttt{fd469501}_{16}$	22	m_7
8	$\texttt{698098d8}_{16}$	7	m_8
9	$\texttt{8b44f7af}_{16}$	12	m_9
10	$\texttt{ffff5bb1}_{16}$	17	m_{10}
11	$\texttt{895cd7be}_{16}$	22	m_{11}
12	$\texttt{6b901122}_{16}$	7	m_{12}
13	$\texttt{fd987193}_{16}$	12	m_{13}
14	$\texttt{a679438e}_{16}$	17	m_{14}
15	$\texttt{49b40821}_{16}$	22	m_{15}

t	AC_t	RC_t	W_t
16	$\texttt{f61e2562}_{16}$	5	m_1
17	$\texttt{c040b340}_{16}$	9	m_6
18	$\texttt{265e5a51}_{16}$	14	m_{11}
19	$\texttt{e9b6c7aa}_{16}$	20	m_0
20	$\texttt{d62f105d}_{16}$	5	m_5
21	$\texttt{02441453}_{16}$	9	m_{10}
22	$\texttt{d8a1e681}_{16}$	14	m_{15}
23	$\texttt{e7d3fbc8}_{16}$	20	m_4
24	$\texttt{21e1cde6}_{16}$	5	m_9
25	$\texttt{c33707d6}_{16}$	9	m_{14}
26	$\texttt{f4d50d87}_{16}$	14	m_3
27	$\texttt{455a14ed}_{16}$	20	m_8
28	$\texttt{a9e3e905}_{16}$	5	m_{13}
29	$\texttt{fcefa3f8}_{16}$	9	m_2
30	$\texttt{676f02d9}_{16}$	14	m_7
31	$\texttt{8d2a4c8a}_{16}$	20	m_{12}

t	AC_t	RC_t	W_t
32	$\texttt{fffa3942}_{16}$	4	m_5
33	$\texttt{8771f681}_{16}$	11	m_8
34	$\texttt{6d9d6122}_{16}$	16	m_{11}
35	$\texttt{fde5380c}_{16}$	23	m_{14}
36	$\texttt{a4beea44}_{16}$	4	m_1
37	$\texttt{4bdecfa9}_{16}$	11	m_4
38	$\texttt{f6bb4b60}_{16}$	16	m_7
39	$\texttt{bebfbc70}_{16}$	23	m_{10}
40	$\texttt{289b7ec6}_{16}$	4	m_{13}
41	$\texttt{eaa127fa}_{16}$	11	m_0
42	$\texttt{d4ef3085}_{16}$	16	m_3
43	$\texttt{04881d05}_{16}$	23	m_6
44	$\texttt{d9d4d039}_{16}$	4	m_9
45	$\texttt{e6db99e5}_{16}$	11	m_{12}
46	$\texttt{1fa27cf8}_{16}$	16	m_{15}
47	$\texttt{c4ac5665}_{16}$	23	m_2

t	AC_t	RC_t	W_t
48	$\texttt{f4292244}_{16}$	6	m_0
49	$\texttt{432aff97}_{16}$	10	m_7
50	$\texttt{ab9423a7}_{16}$	15	m_{14}
51	$\texttt{fc93a039}_{16}$	21	m_5
52	$\texttt{655b59c3}_{16}$	6	m_{12}
53	$\texttt{8f0ccc92}_{16}$	10	m_3
54	$\texttt{ffeff47d}_{16}$	15	m_{10}
55	$\texttt{85845dd1}_{16}$	21	m_1
56	$\texttt{6fa87e4f}_{16}$	6	m_8
57	$\texttt{fe2ce6e0}_{16}$	10	m_{15}
58	$\texttt{a3014314}_{16}$	15	m_6
59	$\texttt{4e0811a1}_{16}$	21	m_{13}
60	$\texttt{f7537e82}_{16}$	6	m_4
61	$\texttt{bd3af235}_{16}$	10	m_{11}
62	$\texttt{2ad7d2bb}_{16}$	15	m_2
63	$\texttt{eb86d391}_{16}$	21	m_9

MD5's Addition Constants AC_t (in hexadecimal notation), Rotation Constants RC_t and message block permutations W_t for each step $t = 0, \ldots, 63$.

10.3.1. Properties of the compression function

1. Each message word is used four times, which makes it very hard given arbitrary values for CV_{in} and CV_{out} to find a message block B that hashes CV_{in} to $CV_{out} = f(CV_{in}, B)$.

2. Each step is a permutation on the state:
for fixed $Q_{i-2}, Q_{i-1}, Q_i, W_i$ there is a bijective mapping between Q_{i-3} and Q_{i+1}.

3. A consequence of 2. is that the 64 steps are a permutation on the state:
for fixed message block, there is a bijective mapping between $(Q_0, Q_{-1}, Q_{-2}, Q_{-3})$ and $(Q_{61}, Q_{62}, Q_{63}, Q_{64})$.
This means you can indeed view MD5's compression function as a block cipher in Davies-Meyer mode.
Note that decryption (the inverse mapping) is as fast as encryption.

4. In each step the message word fully contributes to the output, i.e.:
For fixed $Q_{i-3}, Q_{i-2}, Q_{i-1}, Q_i$, there is a bijective mapping between W_i and Q_{i+1}

5. For the 'decryption' the same holds, i.e.:
For fixed $Q_{i-2}, Q_{i-1}, Q_i, Q_{i+1}$, there is a bijective mapping between W_i and Q_{i-3}

10.4. Modular differential cryptanalysis

One of the first attempts to break MD5 was using modular differential cryptanalysis,
i.e., differential cryptanalysis using difference $\delta X := X' - X \ mod \ 2^{32}$
(e.g., see http://link.springer.com/content/pdf/10.1007%2F3-540-47555-9_6.pdf).
Den Boer and Bosselaers (see http://link.springer.com/content/pdf/10.1007%2F3-540-48285-7_26.pdf) showed in 1993 that one can find collisions for the compression function of MD5 of the form:

$f(CV, M) = f(CV', M')$ with $\delta CV = (\delta a, \delta b, \delta c, \delta d) = (2^{31}, 2^{31}, 2^{31}, 2^{31})$ and $\delta M = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$.

The idea is that for all steps $i = 0, \dots, 63$, given $\delta Q_i = \delta Q_{i-1} = \delta Q_{i-2} = \delta Q_{i-3} = 2^{31}$,
it holds that $\delta F_i = 2^{31}$ with probability 1 (for $i = 32, \dots, 47$) or probability 0.5 (otherwise).
Note that as there is no difference in the boolean function inputs in bit positions 0,...,30, there can also be no output difference in bit positions 0,...,30, i.e., $\delta F_i \in \{2^{31}, 0\}$.
For each bit position we can view the boolean function as a 3-to-1 substitution box and thus determine from DDTs that $\delta F_i = 2^{31}$ with probability 1 for $i = 32, \dots, 47$ and probability 0.5 otherwise.

In which case it follows that:

$\delta T_i = \delta Q_{i-3} + \delta F_i + \delta AC_i + \delta W_i = 2^{31} + 2^{31} + 0 + 0 \ mod \ 2^{32} = 0$;
$\delta R_i = 0$;
$\delta Q_{i+1} = \delta Q_i + \delta R_i = 2^{31}$.

Then after 64 steps we find a collision:

$\delta A + \delta Q_{61} = 2^{31} + 2^{31} \ mod \ 2^{32} = 0$
$\delta B + \delta Q_{64} = 2^{31} + 2^{31} \ mod \ 2^{32} = 0$
$\delta C + \delta Q_{63} = 2^{31} + 2^{31} \ mod \ 2^{32} = 0$
$\delta D + \delta Q_{62} = 2^{31} + 2^{31} \ mod \ 2^{32} = 0$

For random CV and M, the success probability is 2^{-48}, requiring 2^{48} attempts and 2^{49} compression function calls.
However, an important difference between block cipher cryptanalysis and hash function cryptanalysis, is that in hash function cryptanalysis the attacker knows all the intermediate computations as there is no secret key.
This can provide an immediate speed-up:
Assume we have found a (CV, M)-pair such that $\delta T_0 = \cdots = \delta T_{15} = 0$ as desired, requiring 2^{16} attempts.
Then note that by changing $m_{15} = m_{15}'$ to another value, we do not change δF_{15}, so in fact we now know 2^{32} different (CV, M)-pairs with $\delta T_0 = \cdots = \delta T_{15} = 0$.
For these pairs the probability that also $\delta T_{16} = \cdots = \delta T_{63} = 0$ is 2^{-32}, thus among those 2^{32} pairs we can expect to find one that leads to a collision.
The expected cost is now $O(2^{17} + 2^{33})$ compression function calls.
By applying this idea also to the first 15 steps one can further reduce the expected cost to $O(32 + 2^{33})$.

Although den Boer and Bosselaers were able to find collisions for the compression function, the attack did not produce collisions for MD5, as finding message blocks (M_1, M_1') leading to the required chaining value difference $\delta CV_1 = (2^{31}, 2^{31}, 2^{31}, 2^{31})$ remained an open problem.

Let's call this type of collision *dBB-collisions*.

10.5. Combined modular and bitwise differential cryptanalysis

The first successful collision attack on MD5 was due to Xiaoyun Wang and her team
(see http://link.springer.com/content/pdf/10.1007%2F11426639_2.pdf for the original paper,
however this paper provides exposes how the attack works: https://eprint.iacr.org/2004/264.pdf).

They used a number of key ingredients to make their attack successful:

1. They used modular and bitwise differentials together to tackle both modular additions and the bitwise operations with high probabilities, i.e., they used the difference

$\Delta X = (X'[i] - X[i])_{i=0}^{31}$ (bitwise signed difference between the bits of X' and X)

for variables that enter the boolean function: Q_i
and the modular difference for other variables: W_i, F_i, T_i, R_i.

2. They chose very specific δCV and δM to make their attack work, assume these fixed for now. They were chosen to create a high probability differential path over steps 16,...,63. Over steps 0,...,15 they created a very complex differential path that starts with δCV and ends with the difference at step 16 required by the high probability differential.

3. Given an exact differential path, i.e., given all bitwise differences $\delta M, \Delta Q_i, \delta F_i, \delta T_i, \delta R_i$,

one can determine *sufficient conditions* only on variables Q_i and T_i related to M (and thus not on variables Q'_i and T'_i related to M') such that:
if the sufficient conditions are satisfied and $\delta CV_{in}, \delta M$ hold then the different path is followed exactly obtaining $\delta CV_{out} = \delta CV_{in} + (\delta Q_{61}, \delta Q_{64}, \delta Q_{63}, \delta Q_{62})$.

4. Given a partial solution (CV, M) (and thus $(CV', M') = (CV, M) + (\delta CV, \delta M)$) that satisfies the sufficient conditions up to Q_t and T_t for some t one may make small changes to M into $\widehat{M}$ such that $(CV, \widehat{M})$ is also a partial solution that satisfies the sufficient conditions up to Q_t and T_t. This is called a message modification technique and can considerably speed up the attack. E.g., the changes we made to m_{15} to generate 2^{32} solutions in the den Boer and Bosselaers attack is a simple example.

10.5.1. Bitwise signed difference

The bitwise signed difference can be described by a *binary signed digit representation.*
A binary digit representation is a sequence $b_{31} \dots b_0 = (b_i)_{i=0}^{31}$ with $b_i \in \{0,1\}$,
in contrast a binary signed digit representation is a sequence $b_{31} \dots, b_0 = (b_i)_{i=0}^{31}$ with $b_i \in \{-1,0,1\}$.
We will call a 32-bit binary signed digit representation a BSDR
and we can define the difference ΔX as the BSDR $(X'[i] - X[i])_{i=0}^{31}$, where $X[i]$ denotes the i-th bit x_i of $X = x_{31} \dots x_0$.
Let the map $\sigma: \{-1,0,1\}^{32} \to \{0,1\}^{32}$ from a BSDR Y to a word be defined as:

$$\sigma(Y) = \sigma\big((y_i)_{i=0}^{31}\big) = \sum_{i=0}^{31} y_i 2^i \mod 2^{32}$$

Also let $hw(Y)$ denote the hamming weight (number of non-zero digits) for words Y as well as for BSDRs Y.

Given BSDR ΔX we can uniquely determine the word δX: $\delta X = \sigma(\Delta X)$.
However, for any given non-zero word Z there are many possible BSDRs Y such that $Z = \sigma(Y)$.
Let's take as an example $Z = 1$, then the following BSDRs Y have $\sigma(Y) = Z$:

- $y_0 = 1, \quad y_1 = \dots = y_{31} = 0$
- $y_0 = -1, \quad y_1 = 1, \quad y_2 = \dots = y_{31} = 0$
- $y_0 = y_1 = -1, \quad y_2 = 1, \quad y_3 = \dots = y_{31} = 0$
- $\dots$
- $y_0 = \dots = y_{30} = -1, \quad y_{31} = 1$
- $y_0 = \dots = y_{31} = -1$

We call this effect carry propagation.
More generally for any BSDR Y and for any two consecutive signed bits (y_i, y_{i+1}), we can make the following exchanges without changing $\sigma(Y)$:

- $(1,0) \leftrightarrow (-1,1)$ as $2^i = -2^i + 2^{i+1}$
- $(-1,0) \leftrightarrow (1,-1)$ as $-2^i = 2^i - 2^{i+1}$

Also, we can exchange between $y_{31} = -1$ and $y_{31} = 1$ without changing $\sigma(Y)$ as $+2^{31} \equiv -2^{31} \mod 2^{32}$.

A BSDR $Y = (y_i)_{i=0}^{31}$ has *Non-Adjacent-Form (NAF)* if no two adjacent bits $y_i\ y_{i+1}$ are both non-zero.
One can prove that NAF BSDRs have minimal hamming weight,
i.e., given NAF BSDR Y there exists no BSDR Y' such that $\sigma(Y) = \sigma(Y')$ and $hw(Y') < hw(Y)$.
If we further demand that a NAF BSDR has $y_{31} \in \{0,1\}$, then for each word Z there is a unique NAF BSDR.

10.5.2. Sufficient conditions

Let's consider the exact differential path for the den Boer and Bosselaers attack:
We have

- $\Delta Q_j = (\pm 1, 0, \dots, 0) \Rightarrow \delta Q_j = \sigma(\Delta Q_j) = 2^{31}$
- $\delta W_i = 0$
- $\delta F_i = 2^{31}$
- $\delta T_i = 2^{31} + 2^{31} = 0$
- $\delta R_i = 0$

For $j = -3, \dots, 64$, and $i = 0, \dots, 63$.

Consider step 0 then $\Delta Q_0 = \Delta Q_{-1} = \Delta Q_{-2} = \Delta Q_{-3} = (\pm 1, 0, \dots, 0)$.
As the boolean function operates bitwise and there are no differences in the bit positions 0 up to 30,
we see that $\Delta F_0[b] = 0$ for $b = 0, \dots, 30$.
Now consider bit position 31, we have three input bits, thus eight possible values:

$Q_0[31]Q_{-1}[31]Q_{-2}[31]$	$Q'_0[31]Q'_{-1}[31]Q'_{-2}[31]$	$F_0[31]$	$F'_0[31]$	$\Delta F_0[31]$
000	111	0	1	1

001	110	1	1	0
010	101	0	0	0
011	100	1	0	-1
100	011	0	1	1
101	010	0	0	0
110	001	1	1	0
111	000	1	0	-1

To achieve $\delta F_0 = 2^{31}$ there are only 4 allowed values for these three input bits: 000, 011, 100, 111.
We see that $\delta F_0 = 2^{31}$ if and only if $Q_{-1}[31] = Q_{-2}[31]$.
The condition $Q_{-1}[31] = Q_{-2}[31]$ is necessary and sufficient to obtain $\delta Q_1 = 2^{31}$, and only involves variables related to M and not variables related to M'.
The same argument is made for steps 1, … , 15 that use the same boolean function and we find 16 sufficient conditions:

$$Q_{i-1}[31] = Q_{i-2}[31] \text{ for } i = 0, \dots, 15.$$

For steps $i = 16, \dots, 31$, we can make a similar argument using boolean function $BF_{16} = \cdots = BF_{31}$:

$Q_i[31]Q_{i-1}[31]Q_{i-2}[31]$	$Q'_i[31]Q'_{i-1}[31]Q'_{i-2}[31]$	$F_i[31]$	$F'_i[31]$	$\Delta F_i[31]$
000	111	0	1	1
001	110	0	1	1
010	101	1	1	0
011	100	0	0	0
100	011	0	0	0
101	010	1	1	0
110	001	1	0	-1
111	000	1	0	-1

To achieve $\delta F_i = 2^{31}$ there are only 4 allowed values for these three input bits: 000, 001, 110, 111.
We see that $\delta F_i = 2^{31}$ if and only if $Q_i[31] = Q_{i-1}[31]$.
The condition $Q_i[31] = Q_{i-1}[31]$ is necessary and sufficient to obtain $\delta Q_{i+1} = 2^{31}$, and only involves variables related to M and not variables related to M'.
So we find another 16 sufficient conditions:

$$Q_i[31] = Q_{i-1}[31] \text{ for } i = 16, \dots, 31.$$

For steps $i = 32, \dots, 47$ we see that $\delta F_i = 2^{31}$ regardless of the values of $Q_i[31]Q_{i-1}[31]Q_{i-2}[31]$:

$Q_i[31]Q_{i-1}[31]Q_{i-2}[31]$	$Q'_i[31]Q'_{i-1}[31]Q'_{i-2}[31]$	$F_i[31]$	$F'_i[31]$	$\Delta F_i[31]$
000	111	0	1	1
001	110	1	0	-1
010	101	1	0	-1
011	100	0	1	1
100	011	1	0	-1
101	010	0	1	1
110	001	0	1	1
111	000	1	0	-1

However, for steps $i = 48, \dots, 63$, we can determine that: $c \oplus (b \vee \bar{d})$

$Q_i[31]Q_{i-1}[31]Q_{i-2}[31]$	$Q'_i[31]Q'_{i-1}[31]Q'_{i-2}[31]$	$F_i[31]$	$F'_i[31]$	$\Delta F_i[31]$
000	111	1	0	-1
001	110	0	0	0
010	101	0	1	1
011	100	1	1	0
100	011	1	1	0
101	010	1	0	-1
110	001	0	0	0
111	000	0	1	1

To achieve $\delta F_i = 2^{31}$ there are again only 4 allowed values for the three input bits: 000, 010, 101, 111.
We see that $\delta F_i = 2^{31}$ if and only if $Q_i[31] = Q_{i-2}[31]$.
The condition $Q_i[31] = Q_{i-2}[31]$ is necessary and sufficient to obtain $\delta Q_{i+1} = 2^{31}$, and only involves variables related to M and not variables related to M'.
So we find yet another 16 sufficient conditions:

$Q_i[31] = Q_{i-2}[31]$ for $i = 48, \dots, 63$.

Combining these we find the following sufficient conditions for den Boer and Bosselaers attack:

- $Q_{14}[31] = Q_{13}[31] = \cdots = Q_{-2}[31]$
- $Q_{31}[31] = Q_{30}[31] = \cdots = Q_{15}[31]$
- $Q_{63}[31] = Q_{61}[31] = Q_{59}[31] = \cdots = Q_{47}[31]$
- $Q_{62}[31] = Q_{60}[31] = Q_{58}[31] = \cdots = Q_{46}[31]$

Having found the set of sufficient conditions we have now reduced the problem of finding a pair $(CV, M), (CV', M')$ that follows the differential path to the problem of finding a (CV, M) that satisfies the sufficient conditions.
This provides significant advantages:

- we don't have to keep track of the (CV', M') computation anymore
- Also they allow to work one step ahead:
  By ensuring Q_{i+1} satisfies the conditions in step i, in fact we ensure that the correct differences occur in step $i + 1$.

10.5.3. Computing with sufficient conditions

To determine the sufficient conditions for differential paths and even to determine differential paths we can construct Sufficient Condition Tables for the boolean functions.
In these tables we can lookup given current sufficient conditions for its inputs to determine the possible output differences and the respective new sufficient conditions for each output difference.
Let us first define sufficient conditions represented by a symbol:

Symbol $q_t[b]$	$Q_t[b]$	$Q'_t[b]$
.	0/1	$Q_t[b]$
X	0/1	$\overline{Q_t[b]}$
0	0	0
1	1	1
+	0	1
-	1	0
^	$Q_{t-1}[b]$	$Q'_{t-1}[b]$
!	$\overline{Q_{t-1}[b]}$	$\overline{Q'_{t-1}[b]}$
M	$Q_{t-2}[b]$	$Q'_{t-2}[b]$
#	$\overline{Q_{t-2}[b]}$	$\overline{Q'_{t-2}[b]}$
?	$Q_t[b] = 1 \vee Q_{t-2}[b] = 0$	$Q_t[b]$

These conditions define values for both $Q_t[b]$ and $Q'_t[b]$ to fully describe the differential path, for the attack however we can limit ourselves to only to the conditions over $Q_t[b]$.
Using this representation den Boer and Bosselaers differential path can be described by 68x32 table over bits $0, \dots, 31$ of $Q_{-4}, \dots, Q_{64}$:

```
    bit 31                                  0
Q-3:    |X....... ........ ........ ........|
Q-2:    |X....... ........ ........ ........|
Q-1:    |^....... ........ ........ ........|
Q0:     |^....... ........ ........ ........|
Q1:     |^....... ........ ........ ........|
Q2:     |^....... ........ ........ ........|
Q3:     |^....... ........ ........ ........|
Q4:     |^....... ........ ........ ........|
Q5:     |^....... ........ ........ ........|
Q6:     |^....... ........ ........ ........|
Q7:     |^....... ........ ........ ........|
Q8:     |^....... ........ ........ ........|
Q9:     |^....... ........ ........ ........|
Q10:    |^....... ........ ........ ........|
```

```
Q11:    |^....... ........ ........ ........|
Q12:    |^....... ........ ........ ........|
Q13:    |^....... ........ ........ ........|
Q14:    |^....... ........ ........ ........|
Q15:    |X....... ........ ........ ........|
Q16:    |^....... ........ ........ ........|
Q17:    |^....... ........ ........ ........|
Q18:    |^....... ........ ........ ........|
Q19:    |^....... ........ ........ ........|
Q20:    |^....... ........ ........ ........|
Q21:    |^....... ........ ........ ........|
Q22:    |^....... ........ ........ ........|
Q23:    |^....... ........ ........ ........|
Q24:    |^....... ........ ........ ........|
Q25:    |^....... ........ ........ ........|
Q26:    |^....... ........ ........ ........|
Q27:    |^....... ........ ........ ........|
Q28:    |^....... ........ ........ ........|
Q29:    |^....... ........ ........ ........|
Q30:    |^....... ........ ........ ........|
Q31:    |^....... ........ ........ ........|
Q32:    |X....... ........ ........ ........|
Q33:    |X....... ........ ........ ........|
Q34:    |X....... ........ ........ ........|
Q35:    |X....... ........ ........ ........|
Q36:    |X....... ........ ........ ........|
Q37:    |X....... ........ ........ ........|
Q38:    |X....... ........ ........ ........|
Q39:    |X....... ........ ........ ........|
Q40:    |X....... ........ ........ ........|
Q41:    |X....... ........ ........ ........|
Q42:    |X....... ........ ........ ........|
Q43:    |X....... ........ ........ ........|
Q44:    |X....... ........ ........ ........|
Q45:    |X....... ........ ........ ........|
Q46:    |X....... ........ ........ ........|
Q47:    |X....... ........ ........ ........|
Q48:    |M....... ........ ........ ........|
Q49:    |M....... ........ ........ ........|
Q50:    |M....... ........ ........ ........|
Q51:    |M....... ........ ........ ........|
Q52:    |M....... ........ ........ ........|
Q53:    |M....... ........ ........ ........|
Q54:    |M....... ........ ........ ........|
Q55:    |M....... ........ ........ ........|
Q56:    |M....... ........ ........ ........|
Q57:    |M....... ........ ........ ........|
Q58:    |M....... ........ ........ ........|
Q59:    |M....... ........ ........ ........|
Q60:    |M....... ........ ........ ........|
Q61:    |M....... ........ ........ ........|
Q62:    |M....... ........ ........ ........|
Q63:    |M....... ........ ........ ........|
Q64:    |X....... ........ ........ ........|
```

The boolean function has only three inputs and thus we will consider triples ABC of conditions on bits $(Q_t[b], Q_{t-1}[b], Q_{t-2}[b])$ (e.g., (...), (+01), (^^X), ...).

We define $\mathcal{L}$ as the set of triples ABC of conditions that are *local*, i.e., no condition refers to bits $Q_j[c]$ outside the triple:

$$\mathcal{L} = \{ABC \in \{\,.\ \mathrm{X}\ +\ -\ 0\ 1\ \hat{}\ !\ \mathrm{M}\ \#\ ?\,\}^3 \ \mid\ B \notin \{\,\mathrm{M}\ \#\ ?\,\} \wedge\ C \notin \{\,\hat{}\ !\ \mathrm{M}\ \#\ ?\,\}\}$$

For triples $ABC \in \mathcal{L}$ we define the set I_{ABC} as the set of all tuples

$$(x, y, z, x', y', z') = (Q_t[b], Q_{t-1}[b], Q_{t-2}[b], Q'_t[b], Q'_{t-1}[b], Q'_{t-2}[b])$$

that satisfy the conditions ABC.

E.g. $I_{+\hat{}.} = \{(0,0,0,1,0,0), (0,1,1,1,1,1)\}$.

Given a boolean function BF_i and local triple $ABC \in \mathcal{L}$ we can then determine the set of possible output differences $O_{BF_i,ABC}$:

$$O_{BF_i,ABC} = \{BF_i(x',y',z') - BF_i(x,y,z) | (x,y,z,x',y',z') \in I_{ABC}\}$$

We can then immediately determine the set $\mathcal{S}_{BF_i,g}$ of *condition solutions* for BF_i, i.e., local triples ABC that have output difference set g:

$$\mathcal{S}_{BF_i,g} = \{ ABC \in \mathcal{L} \mid O_{BF_i,ABC} = g \}$$

We are interested only in sets $g = \{0\}, g = \{+1\}, g = \{-1\}$ and for bit position 31 also $g = \{-1, +1\}$.

And we can determine the restriction $S_{BF_i,ABC,g}$ of condition solutions DEF for BF_i that are compatible with ABC (i.e., $I_{DEF} \subseteq I_{ABC}$) that have output difference set g (i.e., $O_{BF_i,DEF} = g$):

$$\mathcal{S}_{BF_i,ABC,g} = \{ DEF \in \mathcal{S}_{BF_i,g} \mid I_{DEF} \subseteq I_{ABC} \}$$

We are only interested in the solutions that maximize freedoms, i.e., maximize $|I_{DEF}|$.
However, as there may be many equivalent representations (e.g., (000) ≡ (^^0) ≡ (M^0)),
we will determine the canonical optimal solution $c_{BF_i,ABC,g} = DEF$ for each set $\mathcal{S}_{BF_i,ABC,g}$ that firstly maximizes $|I_{DEF}|$, secondly minimizes the number of double-backwards conditions (M#?) and thirdly minizes the number of single-backwards conditions (!^).

The Sufficient Condition Table for a given BF_i and for each $ABC \in \mathcal{L}$, lists the possible output differences and their canonical optimal solutions.
E.g., consider the following partial Sufficient Condition Table for the first round:

$BF_0 = \cdots = BF_{15}$	$g = \{0\}$	$g = \{+1\}$	$g = \{-1\}$	$g = \{-1,+1\}$
ABC	$c_{BF_0,ABC,g}$	$c_{BF_0,ABC,g}$	$c_{BF_0,ABC,g}$	$c_{BF_0,ABC,g}$
XXX	X!X	X++	X--	X^X
+..	+^.	+10	+01	+!.
.--	n/a	n/a	.--	.--

Note that due to the backwards direction of the sufficient conditions, one needs to derive the sufficient conditions in a backwards direction to ensure that in every step you are working with local condition triples.

10.5.4. Message modification

As mentioned before there is a bijective mapping between W_i and Q_{i+1}, and a bijective mapping between W_i and Q_{i-3}.
Both imply that given $Q_{i-3}, Q_{i-2}, Q_{i-1}, Q_i, Q_{i+1}$ there is a unique value for W_i determined by:

$$F_i = BF_i(Q_i, Q_{i-1}, Q_{i-2})$$
$$R_i = Q_{i+1} - Q_i$$
$$T_i = RL(R, 32 - RC_i)$$
$$W_i = T_i - Q_{i-3} - F_i - AC_i$$

Hence, given$(Q_{-3}, Q_0, Q_{-1}, Q_{-2}) = CV_{in}$ satisfying the sufficient conditions on $Q_{-3}, \ldots, Q_0$, one can trivially find a message block that satisfies the sufficient conditions up to Q_{16}:

- Choose $Q_1, \ldots, Q_{16}$ that satisfy the sufficient conditions
- Determine $W_0, \ldots, W_{15}$ and thus $m_0, \ldots, m_{15}$ via the above relation

To satisfy conditions on $Q_1, \ldots, Q_{16}$ requires thus only to compute 16 steps or 1/4-th of a compression function call.

A more careful consideration of the message word order over the second round (steps 16,…,31) allows us to easily fulfill sufficient conditions up to Q_{19}:

- $W_{16} = m_1 = W_1$ can be changed by choosing different Q_1, Q_2 that satisfy their sufficient bitconditions. This will also change m_0, but that message word is used later on in step 19. Thus we can easily satisfy conditions on Q_{17}.
- $W_{17} = m_6 = W_6$ can be changed by choosing different Q_3, Q_4, Q_5, Q_6, Q_7 that satisfy their sufficient bitconditions. This will also change m_2 (step 29), m_3 (step 26), m_4 (step 23), m_5 (step 20), m_7 (step 30), m_8 (step 27), m_9 (step 24) and m_{10} (step 21). Note that changes can be made to the Q_i's directly and we only have to compute m_i's on-the-go once we need them to compute steps in the second round. Thus we can easily satisfy conditions on Q_{18}.
- $W_{18} = m_{11} = W_{11}$ can be changed by choosing different $Q_8, Q_9, Q_{10}, Q_{11}, Q_{12}$ that satisfy their sufficient bitconditions. This will also change m_7 (step 30), m_8 (step 27), m_9 (step 24), m_{10} (step 21), m_{12} (step 31), m_{13} (step 28), m_{14} (step 25) and m_{15} (step 22): we can easily satisfy conditions on Q_{19}.

To find a dBB-collision requires on average 16 + 2*2 + 2*2 = 24 steps (which is 3/8-th compression function) to satisfy conditions up to Q_{18} (e.g. Q_{17} requires two attempts, each attempt costs 2 steps: compute step 1 to find W_1, compute step 16 to find Q_{17}.).
There are 13+16=29 conditions on $Q_{19}, \ldots, Q_{64}$ remaining and we can expect to need to try 2^{29} different values for $(Q_8, .., Q_{12})$ that satisfy their conditions to find a message block that leads to a dBB-collision.
This has a cost of $O(3/8 + 2^{29})$.

Fortunately, we can do even better using advanced message modification that exploits additional conditions to obtain easy modifications beyond Q_{19} without invalidating previously satisfied conditions.
As an example, let's look at the best advanced message modification for MD5:

> We're going to change Q_9 into $\widehat{Q_9}$ that in turn affects steps 8, 9, 10, 11 and 12:
> $m_8 = W_8$ is used in step 27, $m_9 = W_9$ in step 24, $m_{10} = W_{10}$ in step 21, $m_{11} = W_{11}$ in step 17 and $m_{12} = W_{12}$ in step 31.
> We'd like to change Q_{25} in step 24 but without invalidating previously satisfied conditions.
> However, if m_{10} and/or m_{11} is changed, then we change also Q_{18}, … and thereby invalidate their conditions.
>
> Let's take a closer look how m_{10} and m_{11} are changed exactly:
> $m_{10} = RL(Q_{11} - Q_{10}, 32 - RC_{10}) - Q_7 - BF_{10}(Q_{10}, \widehat{Q_9}, Q_8) - AC_{10}$
> $m_{11} = RL(Q_{12} - Q_{11}, 32 - RC_{11}) - Q_8 - BF_{11}(Q_{11}, Q_{10}, \widehat{Q_9}) - AC_{11}$
> Clearly m_{10} and m_{11} are only changed if the boolean function output changes.
> As $BF_{10}(b,c,d) = BF_{11}(b,c,d) = (b \wedge c) \vee (\bar{b} \wedge d)$ operates bitwise and for each bit position $j = 0, \dots, 31$ $b[j]$ selects whether the output is $c[j]$ ($b[j] = 1$) or $d[j]$ ($b[j] = 0$), we find that :
> $BF_{10}(0, \widehat{Q_9}[j], Q_8[j]) = Q_8[j] = BF_{10}(0, Q_9[j], Q_8[j])$
> $BF_{11}(1, Q_{10}[j], \widehat{Q_9}[j]) = Q_{10}[j] = BF_{11}(1, Q_{10}[j], Q_9[j])$
>
> For every bit $Q_9[j]$ of Q_9 that is entirely free (not involved in any sufficient conditions) such that conditions $Q_{10}[j] = 0$ and $Q_{11}[j] = 1$ do not form a contradiction with the set of sufficient conditions,
> we can add the conditions $Q_{10}[j] = 0$ and $Q_{11}[j] = 1$ to the set of sufficient conditions and thereby use the following advanced message modification at step 24 when all sufficient conditions up to Q_{24} are satisfied:
> $\widehat{Q_9} = Q_9 \oplus 0^{31-j}\ 1\ 0^j)$
> $\widehat{m_9} = RL(Q_{10} - \widehat{Q_9}, 32 - RC_9) - Q_6 - BF_9(\widehat{Q_9}, Q_8, Q_7) - AC_9$
> $\widehat{Q_{25}} = Q_{24} + RL(Q_{21} + BF_{24}(Q_{24}, Q_{23}, Q_{22}) + \widehat{m_9} + AC_{24}, RC_{24})$
>
> That means that if t bits fulfill the above requirements and given one solution for the sufficient conditions up to Q_{24}, we can generate 2^t different solutions that satisfy the sufficient conditions up to Q_{24}.

If from the above requirements only $Q_{10}[j] = 0$ cannot be fulfilled, one can instead use conditions $Q_{10}[j] = 1$ which will force a change in m_{10} and therefore one can use such bits j for advanced message modification at step 21 (as $W_{21} = m_{10}$).
To reduce time spent on steps before step 21, one can optimize the number of bits to use for advanced message modification at step 21 even if they could also be used on step 24.

It's even possible to use advanced message modification dynamically, i.e., add no condition $Q_{10}[j] = 0$ or $Q_{10}[j] = 1$, but randomly select values for those bits.
Then one would dynamically use bits j for which $Q_{10}[j] = 1$ at step 21, and bits j for which $Q_{10}[j] = 0$ at step 24.

This type of advanced message modification is also called a *tunnel*, and there are 6 different tunnels that change a single bit in some Q_i in the first round.
Two can be used at step 20, two can be used at step 21, the other two are used at step 23 and step 24.
We've already seen one for step 24 and one for step 21 whose auxiliary conditions contradict each other,
meaning that for each bit position one cannot use all 6 tunnels.
We leave it to the reader to find the other tunnels.

Now we can use message modification at steps 16, 17, 18, 20, 21, 23 and 24.
To find a dBB-collision now requires 16 + 2*2 + 2*2 + 4*4 + 2*2 + 4*4 + 2*2 = 64 steps (1 compression function) to find a solution up to Q_{24} (up to step 23).
We can then multiply solutions using the above tunnel to find 2^{23} solutions up to Q_{24}, which is enough to expect a dBB-collision.
Note that using early-stop, i.e., stop as soon as a sufficient condition is not satisfied, we calculate on average about 4 steps from step 24 over the 2^{23} attempts.
In total, the expected complexity is thus $O\left(1 + 2^{23}(4/64)\right) = O(2^{19})$, which is significantly better than the straightforward attack with 2^{49} complexity.
This really shows the advantage that we have full knowledge over all intermediate computations instead of only knowing the plaintext and ciphertext as with blockcipher cryptanalysis.

10.5.6. A collision attack against MD5

To complete Den Boer and Bosselaers attack against MD5's compression function to an attack on MD5 itself, one needs to find a message block pair (B, B') such that $f(IV, B') - f(IV, B) = (2^{31}, 2^{31}, 2^{31}, 2^{31})$ and such that $CV_1 = f(IV, B)$ satisfies the sufficient conditions of dBB's attack on $Q_0, Q_{-1}, Q_{-2}, Q_{-3}$.

With the techniques above we can find such a message block pair if we have an exact differential path that starts with $CV_0 = CV_0' = IV$ and ends with $\delta Q_{61} = \delta Q_{62} = \delta Q_{63} = \delta Q_{64} = 2^{31}$.
Naturally, the differential path should try to use the 2^{31}-type differential steps for as many steps as possible.
A good message block difference for this can be found by reasoning from the last step towards the beginning.
So we start with $\delta Q_{61} = \delta Q_{62} = \delta Q_{63} = \delta Q_{64} = 2^{31}$ and go backwards.
Note that we know that in round 4 $\delta F_i = 2^{31}$ with probability $1/2$ and $\delta F_i = 0$ otherwise.
Since $\delta W_i = 2^{31}$ and $\delta F_i = 0$ also result in $\delta Q_{i-3} = \delta T_i - \delta F_i - \delta W_i = 2^{31}$ (given that $\delta R_i = \delta Q_{i+1} - \delta Q_i = 2^{31} - 2^{31} = 0$ and thus $\delta T_i = 0$), in round 4 the success probability for a 2^{31}-type differential step is $1/2$ both for $\delta W_i = 0$ and for $\delta W_i = 2^{31}$.
Hence, if we limit ourselves to $\delta m_i \in \{0, 2^{31}\}$, then round 4 is the same as for dBB's attack.

In round 3 we are not so fortunate: any difference $\delta W_i \neq 0$ implies $\delta Q_{i-3} \neq 2^{31}$.
But we are still free to choose where to put a difference, and at the beginning of round 3 is the best that we can do.
We could choose $\delta m_5 = 2^{31}$, but instead $\delta m_8 = 2^{31}$ in the end turns out to lead to a lower complexity.
Working backwards it remains a very sparse path down to step 23.
From the chaining value we can work forwards and there are no differences up to step 8.
To create a full differential path that completes the partial paths over steps 0,...,8 and steps 23,...,63, we can use the tools from Project HashClash that use a meet in the middle approach (see http://marc-stevens.nl/research/papers/MTh%20Marc%20Stevens%20-%20On%20Collisions%20for%20MD5.pdf).
(i.e., the tools generate many path from above and below and tries to connect them in the middle over steps, say, 13,14,15,16.)

```
    bit 31                                  0  Pr[δR|δT]
Q-3:    |........ ........ ........ ........|
Q-2:    |........ ........ ........ ........|
Q-1:    |........ ........ ........ ........|
Q0:     |........ ........ ........ ........| ok p=1
Q1:     |........ ........ ........ ........| ok p=1
Q2:     |........ ........ ........ ........| ok p=1
Q3:     |........ ........ ........ ........| ok p=1
Q4:     |........ ........ ........ ........| ok p=1
Q5:     |........ ........ ........ ........| ok p=1
Q6:     |........ ........ ........ ........| ok p=1
Q7:     |........ ........ ........ .1......| ok p=1
Q8:     |........ .....0.. ........ .00.....| ok p=1
Q9:     |........ 1....0.. ........ 1+1.....| ok p=1
Q10:    |...0...0 1....-.. .......0 0+......| ok p=1
Q11:    |.100..01 +.1..-.. 1000.000 +-......| ok p=1
Q12:    |.11-..1+ -00^^-01 0111^10- 0+0..1.^| ok p=1
Q13:    |.--+..-0 +1-+++10 +---++++ +-1.10^+| ok p=1
Q14:    |!0+10.0- ++1011+- ---100-- 0+-.1+-0| ok p=1
Q15:    |.1100.11 0011-001 1.100101 +010-000| ok p=0.885742
Q16:    |..0.-..0 1+..0.01 1^0...01 .111.0-.| ok p=0.279297
Q17:    |........ .-1.+.1. ........ ^.--^...| ok p=0.895508
Q18:    |....^... .10...+. 0....10. ......^.| ok p=1
Q19:    |.....0.0 .1+.^... 0....11. ..^^....| ok p=0.943359
Q20:    |.....1.1 ......^. -....++. ........| ok p=0.993164
Q21:    |.....-.+ ..^..... ........ ....0...| ok p=0.987305
Q22:    |0.0..... ........ ^....^^. ....1...| ok p=1
Q23:    |1.1..^.^ ........ ........ ....+...| ok p=0.506836
Q24:    |-.-..... ........ ........ ........| ok p=0.629883
Q25:    |........ ......0. ........ ....^...| ok p=1
Q26:    |^.^..... ......1. ........ ........| ok p=1
Q27:    |........ ......+. ........ ........| ok p=0.868164
Q28:    |........ ........ ........ ........| ok p=1
Q29:    |0....... ......^. ........ ........| ok p=1
Q30:    |........ ........ ........ ........| ok p=0.483398
Q31:    |-....... ........ ........ ........| ok p=1
Q32:    |X....... ........ ........ ........| ok p=1
Q33:    |X....... ........ ........ ........| ok p=1
Q34:    |X....... ........ ........ ........| ok p=1
Q35:    |X....... ........ ........ ........| ok p=1
Q36:    |X....... ........ ........ ........| ok p=1
Q37:    |X....... ........ ........ ........| ok p=1
Q38:    |X....... ........ ........ ........| ok p=1
Q39:    |X....... ........ ........ ........| ok p=1
```

```
Q40:    |X....... ........ ........ ........| ok p=1
Q41:    |X....... ........ ........ ........| ok p=1
Q42:    |X....... ........ ........ ........| ok p=1
Q43:    |X....... ........ ........ ........| ok p=1
Q44:    |X....... ........ ........ ........| ok p=1
Q45:    |X....... ........ ........ ........| ok p=1
Q46:    |X....... ........ ........ ........| ok p=1
Q47:    |X....... ........ ........ ........| ok p=1
Q48:    |M....... ........ ........ ........| ok p=1
Q49:    |M....... ........ ........ ........| ok p=1
Q50:    |M....... ........ ........ ........| ok p=1
Q51:    |M....... ........ ........ ........| ok p=1
Q52:    |M....... ........ ........ ........| ok p=1
Q53:    |M....... ........ ........ ........| ok p=1
Q54:    |M....... ........ ........ ........| ok p=1
Q55:    |M....... ........ ........ ........| ok p=1
Q56:    |#....... ........ ........ ........| ok p=1
Q57:    |M....... ........ ........ ........| ok p=1
Q58:    |M....... ........ ........ ........| ok p=1
Q59:    |M....... ........ ........ ........| ok p=1
Q60:    |M....... ........ ........ ........| ok p=1
Q61:    |M....... ........ ........ ........| ok p=1
Q62:    |M....... ........ ........ ........| ok p=1
Q63:    |M....... ........ ........ ........| ok p=1
Q64:    |X....... ........ ........ ........|
```

Differential path using $\delta m_8 = 2^{31}$ described using sufficient conditions.
Generated in about 10 minutes using the tools from Project HashClash:
https://marc-stevens.nl/p/hashclash/
See the challenges on how to use these tools to construct such paths.

Given such a differential path, we can now construct a collision of MD5:

1. Given chaining value $CV_i = CV_i'$ (e.g., IV)
2. Use the simple and advanced message modification to find a message block pair (B_1, B_1') that satisfies the above differential path
3. We have $CV_{i+1} = f(CV_i, B_1)$ and $CV_{i+1}' = f(CV_i', B_1') = CV_{i+1} + (2^{31}, 2^{31}, 2^{31}, 2^{31})$
4. Find a dBB-collision $f(CV_{i+1}, B_2) = f(CV_{i+1}', B_2)$.
5. return message pair $B_1||B_2$ and $B_1'||B_2$.

The complexity to find the message block pair (B_1, B_1') using advanced message modification can be estimated as:

- Steps 0-15: 16 steps
- Step 16: has about 10 conditions (8 conditions on Q_{17} and rotation probability $\approx 1/4$)
  expected cost: $2^{10} \cdot 2$ steps
- Step 17: has about 8 conditions: expected cost: $2^8 \cdot 2$ steps
- Step 18-19: has about 16 conditions on Q_{19} and Q_{20}: expected cost $2^{16} \cdot 4$ steps
- Step 20: has about 4 conditions: expected cost $2^4 \cdot 2$ steps
- Step 21-22: has about 11 conditions: expected cost: $2^{11} \cdot 4$ steps
- Step 23: has about 3 conditions (2 conditions on Q_{24} and rotation probability $\approx 1/2$): expected cost: $2^3 \cdot 2$ steps
- Step 24: has about 27 conditions (25 conditions on $Q_{25}, \ldots, Q_{64}$ and rotation probability $\approx 1/4$ over steps 24,...,63)
  on average we compute a bit less than 4 steps, so expected cost is about $2^{25} \cdot 4$ steps
- $2^{27} + 2^4 + 2^{13} + 2^5 + 2^{18} + 2^9 + 2^{11} + 16 =$

Total complexity is dominated by $2^{25} \cdot 4$ steps $\approx 2^{23}$ compression function call equivalent.
The total attack takes time equivalent to about $2^{19} + 2^{23} \approx 2^{23.09}$ compression function calls, which takes about 2 seconds on a single modern CPU core.