Preimage and Collision Attacks on MD2

Lars R. Knudsen[1] and John E. Mathiassen[2]

[1] Department of Mathematics, Technical University of Denmark
[2] Department of Informatics, University of Bergen, Norway

Abstract. This paper contains several attacks on the hash function MD2 which has a hash code size of 128 bits. At Asiacrypt 2004 Muller presents the first known preimage attack on MD2. The time complexity of the attack is about 2^{104} and the preimages consist always of 128 blocks. We present a preimage attack of complexity about 2^{97} with the further advantage that the preimages are of variable lengths. Moreover we are always able to find many preimages for one given hash value. Also we introduce many new collisions for the MD2 compression function, which lead to the first known (pseudo) collisions for the full MD2 (including the checksum), but where the initial values differ. Finally we present a pseudo preimage attack of complexity 2^{95} but where the preimages can have any desired lengths.

1 Introduction

A hash function is a function that takes an arbitrary long input, and produces a fixed length output. The output is often called a fingerprint of the input. A cryptographic hash function needs to satisfy certain security criteria in order to be called a secure hash function. Let

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

denote a hash function, whose output is of length n bits. A cryptographic hash function should be resistant against the following attacks:

- **Collision:** Find x and x' such that $x \neq x'$ and $H(x) = H(x')$.
- **2nd preimage:** Given x and $y = H(x)$ find $x' \neq x$ such that $H(x') = y$.
- **Preimage:** Given $y = H(x)$, find x' such that $H(x') = y$.

Typically one requires that there must not exist attacks of these three types which are better than brute-force methods. Thus, to find a collision should not have a lower complexity than about $2^{n/2}$ and it should not be possible to find preimages in time less than 2^n.

It is common to construct hash functions from iterating a so-called a compression function

$$h : \{0,1\}^n \times \{0,1\}^l \rightarrow \{0,1\}^n,$$

which compresses a fixed number of bits. Here the output of one application of this function, h_i, of length n is called a chaining variable and is used as an input in the next iteration together with the next message block m_{i+1} of length l. If the design of a hash function follows the principles of Merkle and Damgård [4,1], it can be shown that a collision for the hash function H implies a collision for the compression function h. Thus, if one can design a secure compression function, then one can also design a secure hash function. Still, the first step towards finding weaknesses in the hash function may be to find weaknesses in the compression function. The first chaining variable in an iterated hash function is often called the IV (initial value) and this is often fixed. Attacks on hash functions where the attacker is able to choose or change the IV are called pseudo attacks. Must popular hash functions are using an iterative compression function and a fixed IV. Examples are MD4, MD5, SHA-1, and RIPEMD-160.

The organisation of this paper is as follows. Section 2 presents the MD2 hash function. Section 3 presents some collision attacks on the compression function where many details are included in an appendix. Section 4 presents several attacks on MD2 (including the checksum). They are a pseudo collision attack, several preimage attacks, as well as a pseudo preimage attack. As far as we are informed the complexities of all these attacks are the lowest known today. Below is a summary of all known results on MD2, where an asterisk (*) indicates that the attack is new.

	Collision	Preimage	Comments
Compression function	2^8[6]	2^{73} [5]	
Hash function (pseudo)	2^{16} (*)	2^{95} (*)	Arbitrary length messages
Hash function	-	2^{104}[5]	Message length 128 blocks
Hash function	-	$2^{97.6}$-2^{112} (*)	Message length 44-128 blocks

2 The MD2

The MD2 hash algorithm is designed by Ron Rivest and published in 1988[2,3]. It is a function $H : GF(256)^* \rightarrow GF(256)^{16}$, which takes an arbitrary number of bytes $GF(256)$ and outputs a string of 16 bytes $GF(256)^{16}$. The function consists of iterations of a compression function $h : GF(256)^{16} \times GF(256)^{16} \rightarrow GF(256)^{16}$, $h_i = h(h_{i-1}, m_i)$, where the input in the ith iteration is the ith message block m_i and the chaining variable h_{i-1}. The message m to be hashed is appended with some padding bytes and a checksum c before it is processed: $m||p||c = m_1||m_2|| \cdots ||m_{t+1}$, where $|m_i| = 128$ for $i = 1, 2, \ldots, t+1$. At least one byte and at most 16 bytes of m_t are padded. Let b be the length of the message in bytes, and $i \equiv b \bmod 16$, $i \in \{0, 1, \ldots, 15\}$, then $d = 16 - i$ (represented in a byte) is added to the message d times. There is at least one byte padding, so if the length is $b \equiv 15 \bmod 16$, then $d = 1$ the byte $p = 1$ is appended the message. If the message length in bytes is 0 modulo 16 , then $d = 16$ and the byte sequence $p = 16| \cdots |16$ of length 16 bytes is added to the message, so that the length of the message still is 0 modulo 16.

Next a checksum block $m_{t+1} = c = c_0 \mid c_1 \mid \cdots \mid c_{15}$ is appended to the message. The checksum [Algorithm 1] is generated processing every byte of the

Algorithm 1 Algorithm to compute the checksum $c = c_0||c_1||\cdots||c_{15}$

for $j = 0, 1, \ldots, 15$
  $c_j = 0$
for $i = 1$ to t do
  for $j = 0$ to 15 do
    $c_j = s(c_{j-1 \bmod 16} \oplus m_{i,j}) \oplus c_j$
  end /*for i*/
end /*for j*/

Algorithm 2 The compression function in MD2, where the output is the 16 first bytes of $h_{i,1} \mid h_{i,2} \mid \cdots \mid h_{i,16}| \cdots |h_{i,48}$.

for $j = 1$ to 16 do
  $h_{i,j} = h_{i-i,j}$
  $h_{i,16+j} = m_{i,j}$
  $h_{i,32+j} = h_{i-i,j} \oplus m_{i,j}$
t=0
for $r = 1$ to 18 do
  for $j = 1$ to 48 do
    $t = h_{i,j} = s(t) \oplus h_{i,j}$
  end /*for j*/
  $t = r - 1$modulo 256
end /*for r*/

message one block at the time, starting at the first block. The checksum is initialized to 0, $c_i = 0$ for $i = 0, 1, \ldots, 15$. Then for all t message blocks, m_i for $i = 1, 2, \ldots, t$, process all 16 bytes of that block and the checksum $j = 0, 1, \ldots, 15$ by the function $c_j = s(c_{(j-1)} \oplus m_{i,j}) \oplus c_j$ where $m_{i,j}$ is the j'th byte of the i'th block of the message and where $s : \{0,1\}^8 \rightarrow \{0,1\}^8$ is a bijective mapping, which is also used in the compression function. The details of s are not important for the results in this paper. The hash function is iterated in the following way:

- $h_0 = iv = 0$
- $h_i = h(h_{i-1}, m_i)$ for $i = 1, 2, \ldots, t+1$
- $H(m) = h_{t+1}$

The compression function [Algorithm 2] of MD2 takes two inputs of each 128 bits, cf., earlier, and consists of an 18-round iterative process, where a vector of the 48 bytes constructed from $h_{i-1}||m_i||h_{i-1} \oplus m_i$ and denoted

$$h_i = h_{i,1}||h_{i,2}||\cdots||h_{i,48}$$

is repeatedly processed from left to right through the use of the same round function consisting of simple byte exclusive-ors and the eight-bit bijective mapping $s()$, also used in the checksum calculation.

3 Attacks on the Compression Function

In [6] a collision attack on the compression function of MD2 is given. Recall that this function computes $h_i = h(h_{i-1}, m_i)$. Rogier and Chavaud give 141 collisions for the compression function where for all collisions h_{i-1} is fixed to the value zero. Note that the IV of MD2 as stated in [2] is zero. We found some variations of this attack. First of all we found that the collision attack extends and it is possible to find many more collisions of this form. We implemented one improvement and found 32,784 collisions, all with $h_{i-1} = 0$. This attack takes very little time. Also we found that it is possible to find so-called multi-collisions for the compression function, that is, a set of different m_is all with same output in the compression function and all with $h_{i-1} = 0$. With a complexity of about 2^{72} one expects a multiple collision of eight messages.

Another variation of Rogier and Chavauds attack is to fix m_i to zero and find different values of h_{i-1} leading to identical outputs of the compression function and yet another variation is to fix $m_i \oplus h_{i-1}$. These variants are similar to the above original one, although the complexities are slightly higher. [6] also consider cases where only a subset of the bytes of h_{i-1} are zeros. We show similar results for the variations. The details of the variant where $h_{i-1} = 0$ are descibed in Appendix B. The details of the other variants are described in an extended version of the paper available upon request.

In the next section we shall use some of the improvements and variations of the attacks on the compression function.

4 Attacks on the MD2 Hash Function

4.1 A Pseudo Collision Attack on MD2

In Section 3 we mention a collision attack on the compression function where $m_i = m'_i = 0$ and $h_{i-1} \neq h'_{i-1}$, but where $h_i = h'_i$. Using this attack we are able to find collision for MD2 (including the checksum) but using different IVs. We have found 130 such collisions in 2 seconds on a single PC, and can find $\approx 2^{15}$ such collisions in about 512 seconds (under 9 minutes) with that property. For any such collision $h_{i-1} \neq h'_{i-1}$, thus if two different IV values of MD2 are chosen to be $IV = h_{i-1}$ and $IV' = h'_{i-1}$ then one can find collisions for all of MD2 for a message using two different IVs.

- Find a pair $(h_0, m_1) \neq (h'_0, m_1)$ where $m_1 = 0$ such that $h(h_0, m_1) = h(h'_0, m_1)$.
- Set $IV = h_0$ and $IV' = h'_0$.
- Choose message blocks $m_2|m_3|, \ldots, |m_t$.
- Then clearly $H(IV, m) = H(IV', m)$, where $m = m_1|m_2|m_3|, \ldots, |m_t$.

Notice that the checksums for both hashes are identical since the message blocks are identical, and therefore we have pseudo collision for MD2.

Let us now consider a situation where such collisions could become practical. Imagine a scenario where Alice and Bob use a digital signature system using a hash function. Imagine that they are signing the same message m many times, e.g., "Alice owes Bob 100 US$". In order to avoid that the same message gives an identical signature, Alice suggests to use a time-stamp, but Bob convinces her that instead he shall send Alice a fresh random hash-IV (e.g., a nonce) to be used in every new signature. Alice agrees to this, however demands that the IV Bob chooses should be run through the hash function first. And so, they agree on the following protocol.

- Bob chooses a random IV
- Alice calculates $r = h(IV, 0)$, creates the hash as usual by $h = H(r, m)$, and signs the hash value, $sign(h)$.

Assuming that the digital signature scheme and the hash function are secure, it seems hard for Bob to cheat. In every new signature a different IV is used, so Bob cannot play the replay attack. However using MD2 in this protocol is a problem since Bob is able to find many collisions of the type $h(IV, 0) = h(IV', 0)$, and hence he is able to reuse the signature and message together with other IVs.

4.2 The Preimage Attack

In [5] F. Muller presents the first known preimage attack on MD2 faster than a brute-force attack. The attack is divided into two parts: in the first part one finds many preimages of the compression function and in the second part one finds those preimages which conform with the checksum function. Note that for most iterated hash functions a preimage attack of the compression function immediately gives at least a pseudo preimage on the hash function, but this is not true for MD2 because of the additional checksum block which is appended to the messages. [5] lists three different attacks on the compression function:

1. Given h_i and h_{i-1}, find a message m_i such that $h_i = h(h_{i-1}, m_i)$. The complexity is 2^{95}.
2. Given h_i and m_i, find a value h_{i-i} such that $h_i = h(h_{i-1}, m_i)$. The complexity is 2^{95}.
3. Given h_i, find a value h_{i-i} and a message m_i such that $h_i = h(h_{i-1}, m_i)$. The complexity is 2^{73}.

Here one unit in the complexity measures is the time to run the compression function once. All these attacks are expected to give one solution, but there might also be zero or several solutions. Assuming that the compression function is a random function, the probability that there is no solution is $(1 - 2^{-128})^{2^{128}}$, and the probability that there are at least w solutions is:

$$p_w \approx 1 - \sum_{i=0}^{w-1} \left[\binom{2^{128}}{i} 2^{-128i} \cdot (1 - 2^{-128})^{2^{128}-i} \right] \approx 1 - (\sum_{i=0}^{w-1} \frac{1}{i!})e^{-1}.$$

The first attack above can be used to find also preimages for (all of) MD2[5]. With $h_0 = 0$ and $h = h_{128}$ the attack is as follows, where h_0 is given and i is initialised to 1:

1. Choose a random value of h_i.
2. If more than 2 solutions of m_i satisfying $h_i = h(h_{i-1}, m_i)$ is found: Increase i by 1. If $i < 128$: Goto step 1.
3. If no more than 2 solutions of m_{128} satisfying $h_{128} = h(h_{127}, m_{128})$ is found: Set i to 127 and goto step 1.

This gives 128 consecutive pairs (h_{i-1}, h_i) for which there are at least 2 different values of m_i such that $h_i = h(h_{i-1}, m_i)$. Consequently there are at least 2^{128} different messages m (of 128 blocks) such that $h = H(m)$, and therefore one of these messages is expected to conform with the checksum $m_{128} = c$. Let $c[i]$ denote the checksum after i iterations (i message blocks). Using the birthday attack on the checksum function has a complexity of about 2^{64}:

- Compute 2^{64} values of $c[64]$ by iterating the checksum function through 2^{64} possible values of the blocks $m_1, m_2, \ldots, m_{64}$.
- Compute 2^{64} values of $c[64]$ by calculating the checksum backwards through 2^{64} possible values of the blocks $m_{65}, m_{66}, \ldots, m_{128} = c$.
- Search for a collision between elements in the two lists.

The expected number of collisions in this last step is 1. The overall complexity of this attack is as follows. The probability of finding at least two solutions in the attack on the compression function is approximately $p_2 = 1 - 2e^{-1}$, and for each of the steps in the algorithm we expect p_2^{-1} repeats. So the total complexity is $128 \cdot p_2^{-1} \cdot 2^{95} \approx 2^{104}$. The padding bytes have not been considered in this attack, but it is strightforward to ensure that the preimages have correct padding without increasing the complexity of the attack[5]. One drawback of this preimage attack is that the messages always consist of 128 blocks. It is left as an open question in [5] to find preimages with fewer blocks. In the next section we give an improvement in complexity of the above attack as well as variants where the messages have fewer than 128 blocks.

4.3 Improvement of the Preimage Attack

First we give a preimage attack also with 128 blocks in the messages but with a lower complexity. We are given $h_0 = 0$ and $h = h_{128}$ and proceed as follows:

1. Given $h_0 = 0$; use the collision attack from Section 3 (see also Appendix B) to find h_1 and a collision for $u \geq 4$ different values of m_1 satisfying $h_1 = h(h_0, m_1)$.
2. Let $h_{127} = h_1$, and use the preimage attack to try to find $v \geq 1$ values of m_{128} such that $h_{128} = h(h_{127}, m_{128})$. If there are no solutions, use another collision from step 1.
3. Let $h_2 = h_1$ and find $w \geq 2$ values of m_2 such that $h_2 = h(h_1, m_2)$. If there are no solutions, repeat step 2 using another collision from step 1.
4. Set $h_i = h_1$ for $i = 3, \ldots, 126$.

This is a situation where $h_0 = 0, h_1 = h_2 = \cdots = h_{127}, h_{128} = h$, and the use of the birthday attack on the checksum is expected to give 1 solution. The first

Table 1. Complexities of the preimage attack for different message lengths, where in each case one solution is expected.

$w \geq$	message length	complexity
2	128	$2^{97.6}$
3	80	$2^{99.3}$
4	64	$2^{101.4}$
5	55	$2^{103.8}$
6	50	$2^{106.4}$
7	46	$2^{109.2}$
8	43	$2^{112.2}$

step has a relative small complexity as discussed before, but we might be forced to repeat steps 2 and 3. The probability of a solution in step 2 is approximately $p_1 = 0.63$, and the probability in the third step is approximately $p_2 = 0.26$. Total complexity of the attack is then

$$p_1^{-1} \cdot p_2^{-1} \cdot 2^{95} \leq 2^{97.6}.$$

There are possible ways to shorten the number of blocks in the preimages, but at the expense of higher complexity. If we require that $w \geq 3$ in step 3, we expect a slightly higher complexity, but the number of blocks in the preimages would drop to approximately $log_3 2^{128}$. Table 1 shows the complexities and lengths of the preimages for different lower bounds of w. As an example, it is possible to lower the number of blocks in the preimages to 55 instead of 128, by requiring $w \geq 5$ in which case the complexity is $\leq 2^{104}$.

It is also possible to get more preimages without increasing the total (time) complexity. Since we use a preimage where $h_{i-1} = h_i$, the possible length of the chain in the middle can be arbitrarily long, however the length is limited by the complexity of the collision attack of the checksum. One example is an attack where the messages are of length 191 and where $w \geq 2$. This gives a memory and computational complexity of 2^{95} in the birthday attack on the checksum, and it is expected to give 2^{62} collisions and thereby 2^{62} possible preimages, but total running time of the attack is unchanged.

4.4 A Pseudo Preimage Attack on MD2

In this section we present a pseudo preimage attack on MD2 which has better complexity than the preimage attack, and where the messages can be (almost) as short or as long as we desire. This attack uses two attacks from [5] on the compression function having complexities 2^{73} and 2^{95} respectively.

Initially a hash value h is given, and we are able to find a message m and an IV which give us the desired hash value $h = H(IV, m)$. First use the method of finding pseudo preimages h_t and m_{t+1} of $h_{t+1} = h$ in the compression function. Remember that the last message block m_{t+1} is the checksum block, and we might repeat this preimage attack to find the second last message block, which also contains the padding bytes. Due to the high degree of freedom in the attack on the compression function, it is possible to choose between 1 and 16 suitable padding bytes in this message block m_t, but it is sufficient to choose the last byte of m_t equal to 1, and the attack still gives us m_t and h_{t-1} with complexity 2^{73}.

Next we need to have at least one more message block in our preimage to make the checksum consistent with the (given) initial value $c[0] = 0$, (recall that $c[i]$ denotes the checksum after i iterations (i message blocks). A potential problem with the checksum could be to fit the two fixed ends $c[0] = 0$ and $c[t] = m_{t+1}$. However it turns out to be easy to "glue" two consecutive checksum values $c[i-1]$ and $c[i]$ together by choosing an appropriate value m_i. Notice that it is also possible to calculate the checksum $c[i] = c(c[i-1], m_i)$ backwards by inverting the function, $c[i-1] = c^{-1}(c[i], m_i)$. Now suppose we have found the message values m_2 and the checksum, we compute $c[2]$ and then $c[1]$ by going backwards. We now "glue" $c[0]$ and $c[1]$ together by finding the appropriate m_1. To get a preimage of two blocks we set $h_1 = h_{t-1}$ and $m_1 = m_{t-1}$, and use another pseudo preimage attack from [5], having complexity 2^{95}, to find $IV = h_0$. Using the MD2 hash function on the IV and a message m will now give the required hash $h = H(IV, m)$. The total complexity in this situation where the message length is two, is 2^{95}.

For a required message length t, and given $h_{t+1} = h$ the algorithm is as follows:

- Find h_t and $m_{t+1}(= c)$ such that $h_{t+1} = h(h_t, m_{t+1})$.
- Find h_{t-1} and m_t (included valid padding byte), such that $h_t = h(h_{t-1}, m_t)$.
- Repeat the preimage attack $t - 2$ times to find h_1 and m_2.
- Find $c[1]$ by calculating the checksum backwards by using m_i for $i = 2, 3, \ldots, t + 1$
- Use special property in the checksum algorithm to find m_1 such that $c[1] = c(0, m_1)$.
- Use the other pseudo preimage attack[5] to find $IV = h_0$ given h_1 and m_1.

The complexity of three first steps of the attack is $t \cdot 2^{73}$ and the last step has complexity 2^{95}. The other parts of the algorithm have relatively small complexity and the total complexity of the attack is 2^{95} as long as $t \leq 2^{21}$. The message length could be as small as $t = 2$.

5 Conclusion

In this paper some new attacks on the hash function MD2 were presented. First some extended collision attacks on the compression function were given. Using one of these attacks it was shown to be possible to mount a pseudo collision on the MD2, which is the first known attack of its kind faster than the trivial attacks. The paper also presented the best known preimage attack on MD2 which is an improvment of a factor of 80 compared to existing attacks. Also, it was shown that the lengths of the preimages can be made smaller than in previous attacks, where the lengths were fixed and relatively high. Moreover it was shown that it is possible to extend the attack such that many preimages are found.

References

1. I.B. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology: CRYPTO'89, Lecture Notes in Computer Science 435*, pages 416–427. Springer Verlag, 1990.
2. B. Kaliski. The MD2 message-digest algorithm. Request for Comments (RFC) 1319, Internet Activities Board, Internet Privacy Task Force, April 1992. Available from `http://www.faqs.org/rfcs/rfc1319.html`.
3. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.
4. R. Merkle. One way hash functions and DES. In G. Brassard, editor, *Advances in Cryptology - CRYPTO'89, Lecture Notes in Computer Science 435*, pages 428–446. Springer Verlag, 1990.
5. F. Muller. The MD2 hash function is not one-way. In P.J. Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, LNCS 3329*, pages 214–229. Springer Verlag, 2004.
6. N. Rogier and P. Chauvaud. MD2 is not secure without the checksum byte. In *Designs, Codes and Cryptography, 12*, pages 245–251, 1997.

A Properties of the MD2 Compression Function

In order to be able to describe the attacks it is convenient to describe the compression function and its intermediate states in a 19×49-matrix

$$T = (T_{i,j})_{j=0,1,\ldots,48}^{i=0,1,\ldots,18},$$

which is also shown in Figure 1, where the first row is made from h_{i-1}, m_i and $h_{i-1} \oplus m_i$. The first element $T_{0,0}$ is never used, but $(T_{0,j})_{j=1,2,\ldots,48} = h_{i-1} \mid m_i \mid h_{i-1} \oplus m_{i-1}$.

Next the rows of the matrix is processed in an iterative manner:

- $T_{1,0} = 0$
- $T_{i,0} = T_{i-1,48} + i - 2 \bmod 256$ for $i = 2, 3, \ldots, 18$ (but not for $i = 1$)
- $T_{i,j} = T_{i-1,j} \oplus s(T_{i,j-1})$ for $i = 1, 2, \ldots, 18$ and $j = 1, 2, \ldots, 48$
- $h_i = (T_{18,j})_{j=1,2,\ldots,16}$

After this procedure the matrix contains all the states of the compression matrix. As we shall see, it is sometimes advantageous in a cryptanalytic approach to try and compute the values in the matrix in a different order than the above line by line approach. To help us do this, we have derived five computing rules directly from the algorithm. The three first rules are shown in Figure 2. The two remaining are just the dependencies between the first and last columns of T. The rules are:

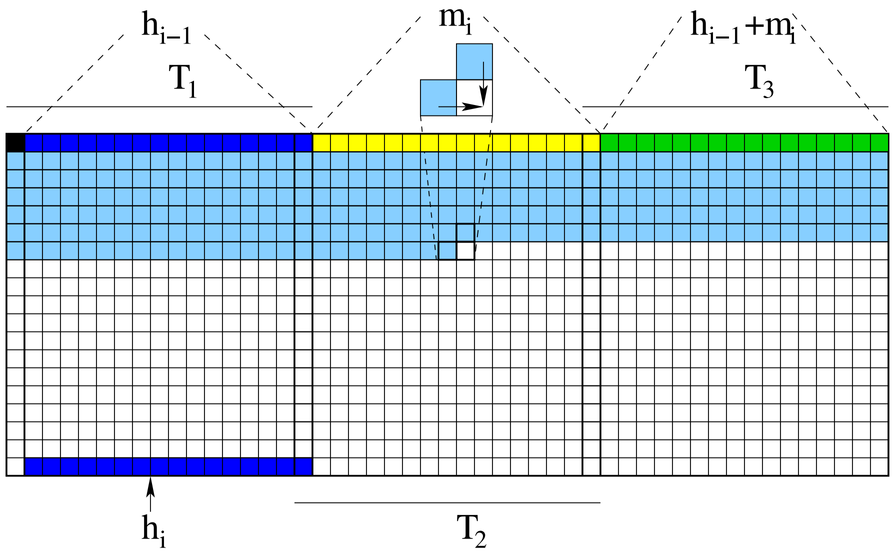

Fig. 1. The MD2 compression function calculation shown as a matrix T. It also shows how the submatrices T_1, T_2 and T_3 are defined, and one line at the time is computed from left to right. The 16 rightmost bytes of the last line of T_1 (the dark area in the last line) contains $h_i = h(h_{i-1}, m_i)$ when the matrix is completed.

1. $T_{i,j} = T_{i-1,j} \oplus s(T_{i,j-1})$, where $i = 1, 2, \ldots, 18$ and $j = 1, 2, \ldots, 48$.
2. $T_{i-1,j} = T_{i,j} \oplus s(T_{i,j-1})$, where $i = 1, 2, \ldots, 18$ and $j = 1, 2, \ldots, 48$.
3. $T_{i,j-1} = s^{-1}(T_{i,j} \oplus T_{i-1,j})$, where $i = 1, 2, \ldots, 18$ and $j = 1, 2, \ldots, 48$.
4. $T_{i,0} = T_{i-1,48} + (i-2) \bmod 256$, where $i = 2, 3, \ldots, 18$.
5. $T_{i-1,48} = T_{i,0} - (i-2) \bmod 256$, where $i = 2, 3, \ldots, 18$.

The three first rules give us five properties from [6] also shown in Figure 3 and Figure 4.

Property 1: Let $k < m$ and $l < n$. If the elements $(T_{k,j})_{j=l,l+1,\ldots,n}$ from row k and $(T_{i,l})^{i=k,k+1,\ldots,m}$ from column l are known the submatrix $(T_{i,j})^{i=k,k+1,\ldots,m}_{j=l,l+1,\ldots,n}$ is uniquely determined using rule 1 (Figure 3).

Property 2: Let $k < m$ and $l < n$. If the elements $(T_{k,j})_{j=l,l+1,\ldots,n}$ from row k and $(T_{i,n})^{i=k,k+1,\ldots,m}$ from column n are known the matrix $(T_{i,j})^{i=k,k+1,\ldots,m}_{j=l,l+1,\ldots,n}$ is uniquely determined using rule 3 (Figure 3).

Property 3: Let $k < m$ and $l < n$. If the elements $(T_{m,j})_{j=l,l+1,\ldots,n}$ from row m and $(T_{i,l})^{i=k,k+1,\ldots,m}$ from column l are known the matrix $(T_{i,j})^{i=k,k+1,\ldots,m}_{j=l,l+1,\ldots,n}$ is uniquely determined using rule 2 (Figure 3).

Property 4: Let $l < n$ and $k < m$, such that $m - k = n - l$. If the elements $(T_{i,n})^{i=k,k+1,\ldots,m}$ from column n are known then half the square matrix $(T_{i,j})^{i=k,k+1,\ldots,m}_{j=l,l+1,\ldots,n}$ is uniquely determined under the diagonal $(T_{i,j})^{i=k,k+1,\ldots,m}_{j=n+k-i,(n+k-i)+1,\ldots,n}$ using rule 3 (Figure 4).

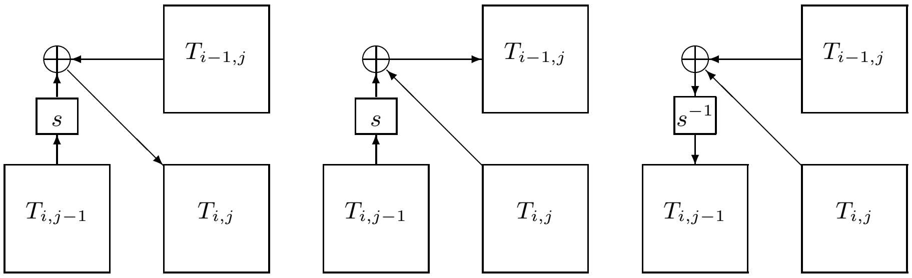

Fig. 2. The dependency of an element $T_{i,j}$ in the matrix T. These three figures show these three dependencies $T_{i,j} = T_{i-1,j} \oplus s(T_{i,j-1})$, $T_{i-1,j} = T_{i,j} \oplus s(T_{i,j-1})$ and $T_{i,j-1} = s^{-1}(T_{i,j} \oplus T_{i-1,j})$ respectively.

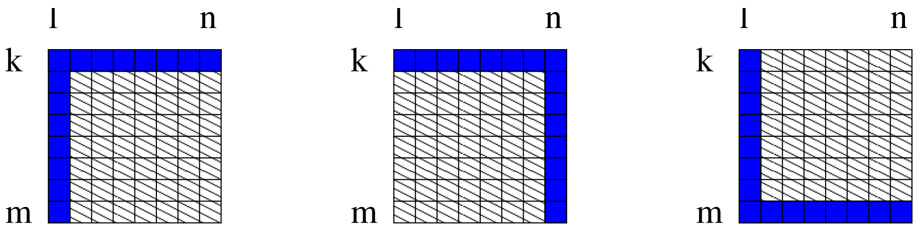

Fig. 3. The figure from left to right shows the Properties 1, 2 and 3 respectively. If the dark areas are known the rest of the matrix is uniquely defined.

Property 5: Let $k < m$ and $l < n$, such that $n - l = m - k$. If the elements $(T_{m,j})_{j=l,l+1,\ldots,n}$ from row m is known then half the square matrix $(T_{i,j})_{j=l,l+1,\ldots,n}^{i=k,k+1,\ldots,m}$ is uniquely determined under the diagonal $(T_{i,j})_{j=n+k-i,(n+k-i)+1,\ldots,n}^{i=k,k+1,\ldots,m}$ using rule 2 (Figure 4).

Observe that the Properties 4 and 5 are similar and define exactly the same triangle, and that the Properties 1, 2 and 3 define the same rectangle. In the attacks of the compression function it is useful to denote the leftmost 17, the middle 17 and the rightmost 17 columns of the matrix T by (the matrices) T_1,

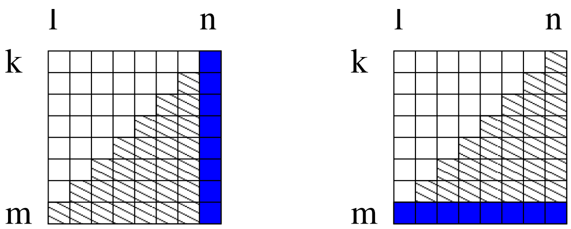

Fig. 4. Illustration of the Properties 4 and 5. If the bottom row or the rightmost column is known, the shaded triangle is uniquely defined.

T_2, respectively T_3 as shown in Figure 1. Notice that the first and last column of T_2 overlap with the last column of T_1 and the first column of T_3.

B Collision Attacks on the Compression Function of MD2

B.1 Collision Attack where $h_{i-1} = 0$

The first part of this section is from [6] with our extensions at the end. We shall

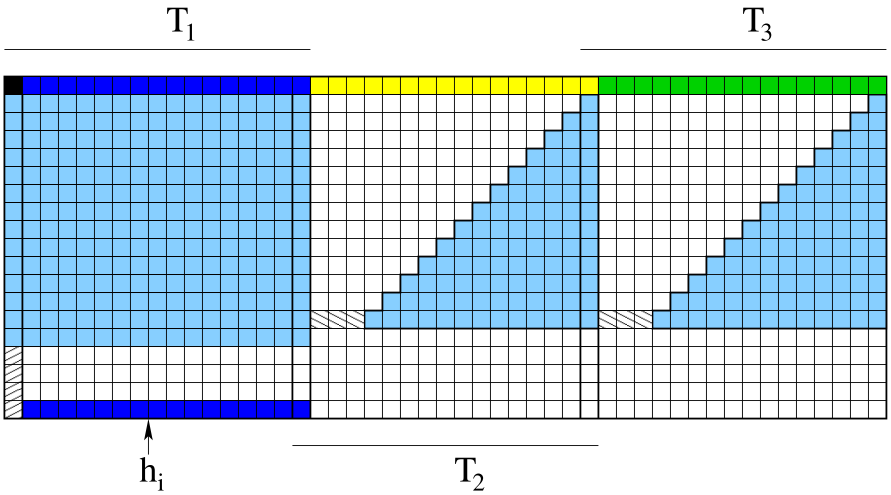

Fig. 5. The figure shows the collision attack on the compression function where $h_{i-1} = 0$. The dark areas are processed line by line.

consider a special case where $h_{i-1} = 0$ and as a consequence $m_i = h_{i-1} \oplus m_i$ and the first rows of T_2 and T_3 are equal. Since the first row of T_1 and the first element in row 1 are known (defined to be 0), we are able to calculate row 1 of T_1. Now we try to find values of m_i such that the 13 first rows of T_2 and T_3 are equal, and in order to be equal the leftmost columns of T_2 and T_3 must be equal and the rightmost columns of T_2 and T_3 must be equal. Since the rightmost column of T_2 coincide with the leftmost column of T_3, the four of them must be equal in order for the matrices to be equal. Having the rightmost element $(T_1)_{1,16}$ in the first row of T_1, we know that we must have:

$$(T_1)_{1,16} = (T_2)_{1,0} = (T_3)_{1,0} = (T_2)_{1,16} = (T_3)_{1,16} = T_{1,48}$$

and if we know $T_{1,48}$ we know that $T_{2,0} = T_{1,48} + 0 \bmod 256$, so it is simple to complete row 2 of T_1. We continue until row k:

$$(T_1)_{i,16} = (T_2)_{i,16} = (T_3)_{i,16} \text{ for } i = 1, 2, \ldots, k$$

and calculate row $k+1$ of T_1

The k values in the right column of T_2 and T_3 are now known and we might complete a triangle in the rows $1, 2, \ldots, k$ of these two matrices according to property 2, shown in Figure 5. The figure shows the situation where 13 rows ($k = 13$) are preprocessed and the triangles are completed, and there are 3 remaining bytes to be chosen to complete row 13 of T_2 and T_3. The 2^{24} possible choices of these bytes will determine 2^{24} different first rows $m_i = h_{i-1} \oplus m_i$ (property 3) and will complete row 13 in both of these matrices, and since the first 14 rows of T_1 is already fixed we have a multi collision in:

$$((T_1)_{i,0})_{i=1,2,\ldots,14}$$

containing $(2^8)^3$ different messages m_i. It remains to find collisions among these in the last 4 rows of column 0:

$$((T_1)_{i,0})_{i=15,16,17,18}$$

and equal values in row 0 and column 0 of T_1 give an equal matrix by property 1, and we also have collisions in 16 bytes of the last row of T_1, which is the chaining variable h_i. The expected number of collisions in this case is approximately

$$(((2^8)^3)^2/2)/((2^8)^4) = 2^{15} = 32768$$

in theory, and we found 32784 collisions in practice. In [6] $k = 14$ and 2 bytes are varied, and the expected number of collisions were 128 and in practice there were 141 collisions, but to decrease k to get more collisions is not mentioned explicitly in the paper.

In general we would expect

$$(((2^8)^{16-k})^2/2)/((2^8)^{18-(k+1)}) = 2^{8(15-k)-1}$$

collisions, only depending on the choice of k. The memory and computational complexity is proportional to the number of bytes varied: $2^{8(16-k)}$.

In the preimage attack described earlier in this paper it is advantageous to use this attack when $h_0 = 0$ and to get collisions in m_1. It is possible to get more than 2 different m_1 such that all of them give the same output h_1, and if so we have a multiple collision. If we look for a d-tuple collision and we are able to vary $b = 16 - k$ bytes in the first phase of the attack, we expect

$$\binom{2^{8b}}{d} / 2^{8(b+1)(d-1)} \approx 2^{8(b+1-d)}/d!$$

d-tuple collisions. If $b = 9$ and $d = 8$ we expect $\approx 2^{0,7} \geq 1$ multiple collisions of size 8, and the complexity is approximately 2^{72}.

There are similar attacks on the compression function where $m_i = 0$ or where $h_{i-1} \oplus m_i = 0$. For these two attacks and the one where $h_{i-1} = 0$ there are generalizations which are described in detail in an extended version of the paper available at `http://www.ii.uib.no/~johnm/publications/md2-procExtended.pdf`