J. Cryptol. (2010) 23: 72–90
DOI: 10.1007/s00145-009-9054-1

Journal of
CRYPTOLOGY

Cryptanalysis of MD2

Lars R. Knudsen
Department of Mathematics, Technical University of Denmark, Matematiktorvet 303S, 2800 Kgs. Lyngby, Denmark
lars@ramkilde.com

John Erik Mathiassen
Avenir AS, Bergen, Norway
john.erik.mathiassen@gmail.com

Frédéric Muller
HSBC, Paris, France
frederic.muller@m4x.org

Søren S. Thomsen
Department of Mathematics, Technical University of Denmark, Matematiktorvet 303S, 2800 Kgs. Lyngby, Denmark
crypto@znoren.dk

Communicated by Bart Preneel

Received 27 August 2007 and revised 8 October 2009
Online publication 28 October 2009

Abstract. This paper considers the hash function MD2 which was developed by Ron Rivest in 1989. Despite its age, MD2 has withstood cryptanalytic attacks until recently. This paper contains the state-of-the-art cryptanalytic results on MD2, in particular collision and preimage attacks on the full hash function, the latter having complexity 2^{73}, which should be compared to a brute-force attack of complexity 2^{128}.

Key words. Cryptographic hash function, MD2, Collision attack, Preimage attack.

1. Introduction

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

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

denote a hash function whose output is of length n bits. A cryptographic hash function should be resistant to 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 $H(x') = y$.

A collision for any hash function can be found by a birthday attack with complexity around $2^{n/2}$. Preimages and 2nd preimages can be found by a brute force search with complexity 2^n. Typically, one considers a hash function secure only if no attacks better than these are known.

The Merkle–Damgård construction [2,11] is a typical method of constructing hash functions. This method works as follows. Given a so-called compression function $f : \{0, 1\}^n \times \{0, 1\}^b \to \{0, 1\}^n$ and an initial n-bit value h_0, the message m is split into a number of b-bit message blocks $m_1, m_2, \ldots, m_t$. Then, for every i from 1 to t, one computes

$$h_i \leftarrow f(h_{i-1}, m_i), \tag{1}$$

and finally $H(m) = h_t$ is returned as output. Examples of hash functions based on the Merkle–Damgård construction are MD4 [17], MD5 [18], SHA-1 [13], and many others.

MD2 [5] is an example of a hash function which does not follow the Merkle–Damgård principle. It was developed in 1989 by Ron Rivest. It was soon deemed to be too slow; since it is based on operations on words of only 8 bits, and inherently serial, significant speed improvements on the more common processors operating with larger word sizes are not possible. MD2 deviates from Merkle–Damgård-based hash functions in that a second state, the so-called checksum, is computed from the message, and this checksum is subsequently appended to the message as an additional message block. This feature is quite unique to MD2. Another feature which separates MD2 from immediate successors such as MD4, MD5, and SHA-1 is the use of an S-box.

MD2 was soon superseded by other designs, not because it had been broken, but because its performance could not compete with the more modern designs. Although, as we describe in the following, there is a number of successful cryptanalytic results on MD2, it is still used in practise and is part of several (de facto) standards, e.g., PKCS #1 v2.1: RSA Cryptography Standard [20]. As an example, VeriSign has issued a class 3 root certificate (expiring in 2028) using MD2 in the signature algorithm [22].

The first analysis of MD2 was done by Preneel [15], who noted that after 16 rounds (out of 18), not all hash values are possible. Rogier and Chauvaud [19] described a collision attack on a simplified variant of MD2 where the checksum block is omitted. The first attack against the full MD2 hash function was a preimage attack published by Muller [12] in 2004. This attack was improved by Knudsen and Mathiassen in [6], where they also generalised and found further use of the collision attack by Rogier and Chauvaud.

In this paper we present the current state of the art of cryptanalysis of MD2. Hence, we describe known attacks and also present new attacks and improvements. Among the new attacks, there are the first collision attack breaking the birthday bound on the full MD2 hash function and also an improved preimage attack on the full hash function, having complexity about 2^{73}. See Table 1 for a summary of the best known attacks on MD2.

The paper is organised as follows. In Sect. 2 we introduce the MD2 hash function and mention an important observation in Sect. 3. In Sect. 4 we describe the known collision

Table 1. A summary of the best known attacks on MD2 and their *approximate* complexities.

Attack type	Note	Reference	Time complexity	Memory complexity
Collision	No checksum	[19]	2^{12}	(negligible)
Collision	Full hash	This paper	$2^{63.3}$	2^{52}
(2nd) preimage	Full hash	This paper	2^{73}	2^{72}

attacks of Rogier and Chauvaud [19] and Knudsen and Mathiassen [6], and we also present a new collision attack on the MD2 compression function giving the attacker more freedom than in the known attacks, and which leads to a collision attack on the full hash function. In Sect. 5 we describe known preimage attacks by Muller [12] on the MD2 compression function, one of which can be improved and extended to a new preimage attack on the full hash function. Finally, in Sect. 6 we conclude.

2. Description of MD2

MD2 takes messages of any length and returns a 128-bit hash. The message is padded so that its length becomes a multiple of 16 in bytes. Padding is described in Sect. 2.1. The message is then split into t blocks $m_1, m_2, \dots, m_t$ of 16 bytes each, and a 16-byte checksum block c is computed from the padded message. c is appended to the message as the $(t+1)$th message block. The $t+1$ blocks are then processed sequentially: starting from the initial state (denoted h_0 in the following) which is the all zero 16-byte string, every message block updates the state, and the state after m_{t+1} has been processed is the output of the hash function. Hence, once the checksum block has been appended to the message, MD2 can be seen as following the Merkle–Damgård principle (1) on the resulting message. We now give the relevant details of the MD2 hash function. Since the internals of the checksum function are irrelevant to the attacks presented in this paper, a detailed description is postponed to Appendix A.2. We would like to mention here, however, that the checksum function takes two inputs, the current checksum and a message block, and produces a new checksum. The checksum function is invertible, i.e., given two of the three values, the third can be easily computed.

2.1. *Padding*

If the original message consists of r bytes, then d bytes, each having the value d, are appended to the message, where d is the integer between 1 and 16 such that $r+d$ is a multiple of 16. Hence, all messages are padded, even if r is itself a multiple of 16. This padding rule ensures that there is a one-to-one relationship between the original message and the padded message.

2.2. *The Compression Function*

In the following we denote by x^b the bth byte of a string x. The compression function $f : \{0,1\}^{128} \times \{0,1\}^{128} \to \{0,1\}^{128}$ works as follows. Given 16-byte strings h_{in} (called the *chaining input*) and m (the *message block*), let

$$A_0 \leftarrow h_{\text{in}},$$

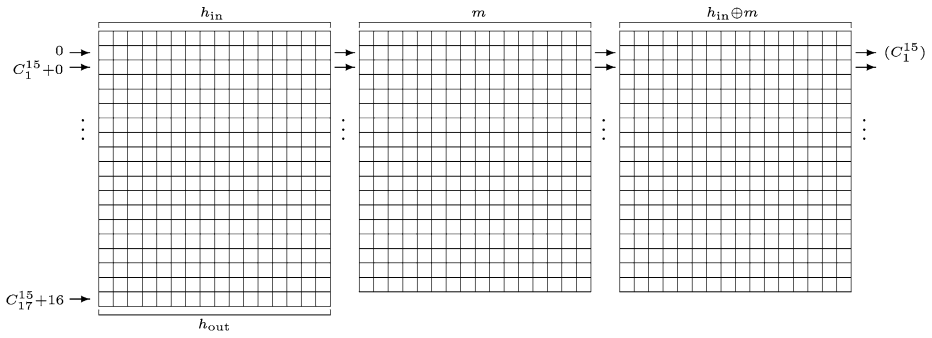


Fig. 1. The MD2 compression function. The '0' entering from the left in the first computed row is a constant. In the rows below, the value that enters from the left is the sum of the last byte computed in the row above, and a round constant which is $i-1$ in row i.

$$B_0 \leftarrow m,$$
$$C_0 \leftarrow h_{\text{in}} \oplus m$$

($\oplus$ is the exclusive or (xor) operator). The concatenation $A_i \| B_i \| C_i$ may be viewed as a single 48-byte entity X_i.

To generate the output of the compression function, do the following:

```
1. $L \leftarrow 0$
2. For increasing $i$ from 1 to 18 do
   (a) For increasing $j$ from 0 to 47 do
       i. $X_i^j \leftarrow S(L) \oplus X_{i-1}^j$
       ii. $L \leftarrow X_i^j$
   (b) $L \leftarrow L + i - 1$
3. Output $A_{18}$, i.e., the first 16 bytes of $X_{18}$.
```

Here, S is a nonlinear bijective function (an S-box) from $\{0,1\}^8$ to $\{0,1\}^8$, the specification of which can be found in Appendix A.1.

See also Fig. 1. This view of the compression function is instructive when studying the attacks presented in this paper. We shall often refer to the three rectangular structures as rectangles A, B, and C, since row i of, e.g., A is exactly A_i. Note that since there are 832 bytes in total in the three rectangles A, B, and C, we shall often estimate that computing one byte corresponds to $1/832$ compression function evaluations. This will also be used as an estimate for a simple operation such as an xor.

3. Observations on the Compression Function

Since X_i^j is computed as the xor of X_{i-1}^j and a bijective function of X_i^{j-1} (for $i > 0$ and $j > 0$), a cryptanalyst may instead compute either of the three values from the two others in one of the following ways:

$$X_i^j = X_{i-1}^j \oplus S\left(X_i^{j-1}\right), \tag{2}$$

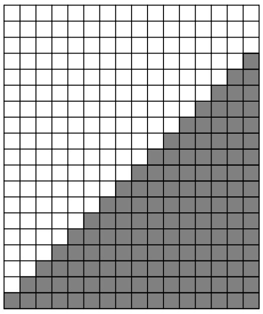


Fig. 2. The shaded values are the values of the rectangle A that may be computed if the output of the compression function, corresponding to the last row of A, is known.

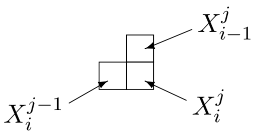
$$X_{i-1}^{j} = X_i^j \oplus S\big(X_i^{j-1}\big), \tag{3}$$

$$X_i^{j-1} = S^{-1}\big(X_i^j \oplus X_{i-1}^j\big). \tag{4}$$

For $i = 0$, X_i^j is not computed but is rather part of the input. $X_1^0 = A_1^0$ is computed from $X_0^0 = A_0^0 = h_{\text{in}}^0$ only.

For $j = 0$, a similar relationship exists, i.e., X_i^0 $(= A_i^0)$ is computed from X_{i-1}^0 and X_{i-1}^{47} $(= C_{i-1}^{15})$.

Comparing with Fig. 1, this means that any "triangle" of the form



is completely determined by two of the squares. As an example, if the output of the compression function, which corresponds to the last row of A, is known, then by repeatedly applying (3) above, one may compute the entire lower right triangle as indicated in Fig. 2. In the following, we shall generally indicate known values as shaded squares of the rectangles A, B, and C.

4. Collision Attacks

In this section we describe a number of collision attacks on MD2. The first attack by Rogier and Chauvaud [19] finds collisions of the MD2 *compression* function and requires the chaining input to be the all-zero 128-bit string (which is also the specified initial value of the hash function). Because of the checksum block, this attack does not constitute an attack on the MD2 *hash* function. The second attack we describe is a generalisation of the first attack, where more collisions are found, and hence it improves the applicability of the first attack. For instance, it enables the construction of multicollisions, which are sets $\{m_1, m_2, \ldots, m_r\}$, $r \geq 2$, of messages all having the same hash,

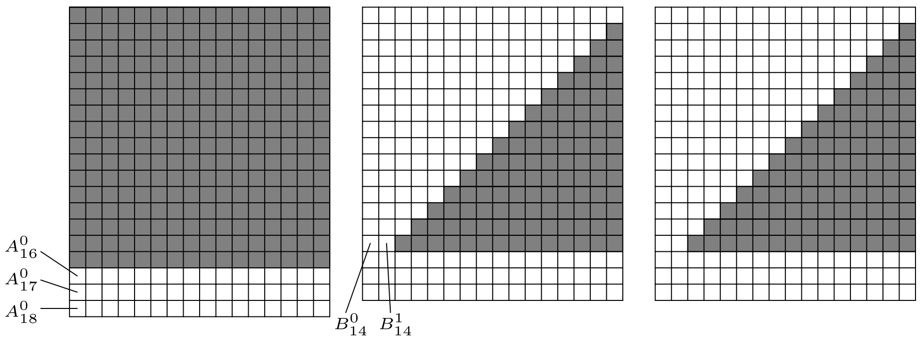


Fig. 3. Known values (shaded) in the collision attack.

however, these are still only attacks on the compression function. The last collision attack we describe is an attack on the MD2 compression function where any chaining input may be used. This attack can be extended into an attack on the MD2 hash function, which is slightly faster than the birthday attack.

4.1. *A Collision Attack on the Compression Function*

The first collision attack [19] requires that $h_{\text{in}} = 0$, i.e., $A_0^j = 0$ for $0 \leq j < 16$. This also means that $B_0 = C_0$, and the idea of the attack is to have

$$A_i^{15} = B_i^{15} = C_i^{15} \quad \text{for } 1 \leq i \leq 14. \tag{5}$$

In other words, the last columns of A, B, and C should be identical down to row 14, inclusive. With this condition and the condition on h_{in}, the entire rectangle A down to row 15 can be computed. Also, given part of the last column of B and C, we can compute the lower right triangle of B and C from row 1 down to row 14. See Fig. 3. Note that all these values are independent of the message when (5) holds and that the lower right triangle contains the same values in all three rectangles.

At this point, row 14 of B is determined except for the two values B_{14}^0 and B_{14}^1. By selecting these, given the last column of A, we can compute the entire rectangle B down to row 14, and hence m is determined (since it equals B_0). Since A is determined down to row 15, two messages collide if they collide on the values A_{16}^0, A_{17}^0, and A_{18}^0. From the 2^{16} different values of m arising from different choices of B_{14}^0 and B_{14}^1, we can form $2^{32}/2$ pairs, so we expect about 2^7 of them to collide on the 24 bits, assuming that these are uniformly distributed.

If one is interested in only a single collision, this can be found in time about 2^{12}, since the attack is in effect a birthday attack on 24 bits. Memory requirements are negligible.

4.2. *More Collisions and Multicollisions*

The attack just described can be generalised as follows. In the attack we may choose to impose the requirement $A_i^{15} = B_i^{15} = C_i^{15}$ for $1 \leq i \leq k$, where k is some fixed value that we are free to choose as long as $1 \leq k \leq 14$. This would have the effect

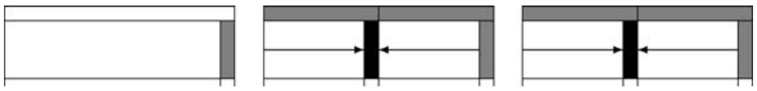


Fig. 4. The idea of the collision attack with arbitrary chaining input. Shaded values are chosen, or computed from the chaining input and the chosen values. The blackened columns are computed from the left and from the right, as indicated by the arrows. These are the "meet-in-the-middle" columns.

that we can vary the $16-k$ bytes $B_k^0, \ldots, B_k^{15-k}$, giving us more collisions as k is decreased. The messages now have to collide on the $17-k$ bytes $A_{k+2}^0, \ldots, A_{18}^0$, and the complexity of finding all collisions is $2^{8(16-k)}$. The expected number of collisions is $2^{2\cdot 8(16-k)-1}/2^{8(17-k)} = 2^{8(15-k)-1}$.

To find an r-way multicollision, i.e., r message blocks that all have the same hash, we may reduce k above. For every set of r messages, the probability that all r messages have the same hash is $2^{-8(17-k)(r-1)}$. The number of different subsets of r messages out of a set of $2^{8(16-k)}$ messages is

$$\binom{2^{8(16-k)}}{r} \approx 2^{8(16-k)r}/r!.$$

Multiplying this number by the probability of an r-way multicollision per subset of r messages, we get the expected number

$$2^{8(16-k)r}/r! \cdot 2^{-8(17-k)(r-1)} = 2^{8(17-k-r)}/r!$$

of r-way multicollisions.

Hence, if one is looking for, e.g., four messages all having the same hash, by choosing $k = 12$ there is a very high probability that one succeeds. The complexity of this attack with $k = 12$ is about 2^{32}.

4.3. *A Collision Attack with Arbitrary Chaining Input*

We now describe a collision attack on the MD2 compression function which does not impose any restrictions on the chaining input and which leads to an attack on the full MD2 hash function. The attack is largely inspired by the preimage attack described in Sect. 5.1.2, which was discovered first. We describe the two attacks independently.

Given a chaining input h_{in}, we look for messages that collide starting from h_{in}. We choose $C_1^{15}, C_2^{15}, \ldots, C_k^{15}$ arbitrarily, for some k, $1 \le k < 18$. This will enable us to compute A_1 to A_{k+1}, because h_{in} is known. We also choose $B_1^{15}, \ldots, B_k^{15}$ arbitrarily. The attack proceeds as follows, where we are free to choose $\ell \in [0, 64]$ (see Fig. 4 for an illustration of the idea of the attack).

1. For 2^ℓ different values of $m^0, \ldots, m^7$, compute B_j^7 and C_j^7 for $1 \le j \le k$, in the forward direction (i.e., using (2)). Store the message bytes in table T_1, indexed by the $2k$ bytes computed. In the case of collisions on the index, several messages can be stored in, e.g., a linked list.
2. For 2^ℓ different values of $m^8, \ldots, m^{15}$, compute B_j^7 and C_j^7 for $1 \le j \le k$, in the backward direction (using (3)). Store the message bytes in table T_2, indexed by the $2k$ bytes computed. Again, collisions must be handled properly.

3. Find (in linear time) all matches between indices of the two tables. Store the corresponding complete message blocks in table T. The expected number of messages in T is $2^{2\ell-16k}$, since there are $2^{2\ell}$ pairs of message halves, and each pair constitutes a match on the $2k$ bytes with probability about 2^{-16k}.
4. Find all colliding messages in T.

Two messages in T collide with probability about $2^{-8(17-k)}$, since they must collide in $17-k$ bytes. Hence, the size of T must be at least $2^{4(17-k)}$ for a collision to occur with good probability. This means that for a given (integer) value of k, we would choose ℓ such that

$$2\ell - 16k = 4(17-k) \quad \Longleftrightarrow \quad \ell = 6k + 34. \tag{6}$$

The complexity of the attack is dominated by the construction of T_1 and T_2, and the subsequent construction of and search through T. T_1 and T_2 are constructed using $2 \cdot 2^{\ell}$ computations of the $2k$ bytes in column 7 of B and C. Each computation (on average) corresponds to about $2k$ 8-bit xors, since given, e.g., $m^0, \ldots, m^6$, one may precompute part of column 6 and loop through all values of m^7. Then, only a single xor is required to compute each byte in column 7. On a 32-bit machine, 4 xors can be done in one operation. Hence, each computation may be estimated to be equivalent to about $2/832 < 2^{-8}$ compression function evaluations. This depends on k, but for values up to $k = 6$, this estimate seems reasonable. Hence, the construction of T_1 and T_2 takes time equivalent to about $2^{\ell-7}$ compression function evaluations. The subsequent search through the two lists takes a similar amount of time, so the complexity of this first part is about $2^{\ell-6}$.

Finding collisions in T requires about the same number of compression function evaluations as the size of T, which is $2^{2\ell-16k}$. Hence, the total complexity of the attack is about $2^{\ell-6} + 2^{2\ell-16k}$. Using (6) to eliminate ℓ, we get the complexity $2^{6k+28} + 2^{-4k+68}$, which we want to minimise, observing that k must be an integer. The optimal value is $k = 4$, yielding $\ell = 58$, and a total complexity of around 2^{53}. This complexity is well below the complexity of a standard birthday attack. However, the memory requirements are about 2^{ℓ} message blocks. These can be reduced; see the subsection below.

The probability of success for the attack is estimated to be about 0.39. For a better success probability, ℓ may be increased; with, e.g., $\ell = 58.6$, the success probability is estimated to be about 0.93, and the complexity of the attack is about 2^{54}. We shall use this complexity estimate below.

4.3.1. *Memory Requirements*

The memory requirements of the meet-in-the-middle part can be reduced significantly by using a memoryless meet-in-the-middle search, as described by Quisquater and Delescaille [16] and also by van Oorschot and Wiener [21]. This technique can also be used to find multiple solutions for the meet-in-the-middle problem, as needed in the attack described above. Using this method, memory requirements can be reduced to a negligible quantity, but the time complexity slightly increases.

However, still about $2^{4(17-k)}$ (which is 2^{52} with $k = 4$) solutions to the meet-in-the-middle problem must be found, and the corresponding message blocks must be stored so that collisions among them can be found. With fewer solutions, the collision still occurs with some (lower) probability. Hence, a way to reduce memory requirements of the second part also is to repeat the attack several times with a smaller value of ℓ. However, this is not an effective way of reducing memory requirements; reducing them by a factor K leads to roughly a factor K^2 increase in time complexity.

We might add that the time complexity of a cryptanalytic attack can never (asymptotically) be below the memory complexity, since the memory needs to be written and (usually) read. Accurately estimating the time needed for a memory access compared to the time needed for a compression function evaluation is, however, close to impossible and depends on many factors such as the type of storage used. Assuming a one-to-one relationship between the two seems to be the best (conservative) estimate.

4.3.2. *Collisions for the Full MD2*

The collision attack on the compression function can be extended to attack the full MD2 hash function. This extension has a time complexity (in terms of compression function evaluations) slightly below that required by the birthday attack on MD2, but the memory requirements are high, and therefore, in practise, the attack may not be faster than a birthday attack. We describe the attack nonetheless.

The extension of the collision attack entails two complications: (1) padding for the message must be correct, and (2) the checksum block must be correct. Condition (1) is easily fulfilled; simply append to every message an additional message block of 16 bytes each with the value 16. Condition (2), on the other hand, is not as easily dealt with.

By Joux's method [4], using 65 collisions with chosen chaining input, we can construct a 2^{65}-way multicollision of messages of 65 blocks each. With probability about $p_1 = 1 - 1/e^2$, one of the pairs also collides on the checksum block. To find this pair, we need to compute (and store) the 2^{65} checksums. Since updating the checksum with one message block takes about $16/832 \approx 2^{-5.7}$ times the time of one compression function evaluation, computing the 2^{65} checksums takes time about $(2^{65} + 2^{64} + \cdots + 2) \cdot 2^{-5.7} \approx 2^{60.3}$ in terms of compression function evaluations. The memory requirements are 2^{65} checksums (along with some bits identifying the combination of message blocks). Finding the $65 \approx 2^6$ collisions takes time about $2^{6+54} = 2^{60}$ using the collision attack described above. Hence, the total complexity of the attack is about $2^{60.3} + 2^{60} \approx 2^{61.2}$. Taking into account the probability of finding the collision in the checksum block, the complexity may be estimated to $2^{61.2}/p_1 \approx 2^{61.4}$. As mentioned above, accessing the memory required to store the checksums may take longer than this estimate. The birthday attack on MD2 is expected to take time about $2^{65.5}$, because hashing a message always requires at least two compression function calls due to the checksum block.

To reduce memory requirements, a collision in the checksum can be found using a cycle-finding method [1,3,14]. (Note, however, that the memory requirements for finding the collisions in the compression function are about 2^{59} hash values, or 2^{52} in the memory-reduced case.) As an example trade-off, consider the following attack: Precompute all 2^{49} checksums of the messages that can be constructed from the first 49 message blocks. Use the cycle-finding algorithm to iterate through the full checksums, requiring the computation of $16 = 2^4$ "atomic" checksums for each, hence requiring the time of about $2^{-1.7}$ compression function evaluations. Assuming that $2^{64.6}$ checksums are needed before the algorithm starts repeating (estimate taken from Nivasch [14]), this method takes time about $2^{62.9}$. The complexity of finding the 2^{65}-way multi-collision should be added, yielding a total time complexity of about $2^{63.3}$. Memory requirements are now dominated by the collision attack on the compression function, hence about 2^{52} in the memory-reduced case.

5. Preimage Attacks

A preimage attack on the compression function of a hash function in Merkle–Damgård mode can often be extended to a preimage attack on the hash function. This is true in particular if the attack works for any chaining input to the compression function. In MD2, however, the checksum block makes things more complicated. We start off with a description of three types of preimage attack on the compression function, and then we describe how these can be extended to cover the full MD2 hash function. We note here that any type of preimage attack, whether on the compression function or on the hash function, when carried out by the brute force method, has complexity about 2^{128} in terms of compression function evaluations.

5.1. *Preimage Attacks on the MD2 Compression Function*

When discussing preimage attacks on a compression function f taking two inputs, we need to make the following distinction between three different types of preimage attack:

Type 1: Given h_{out}, find h_{in} and m such that $f(h_{\text{in}}, m) = h_{\text{out}}$.
Type 2: Given h_{out} and h_{in}, find m such that $f(h_{\text{in}}, m) = h_{\text{out}}$.
Type 3: Given h_{out} and m, find h_{in} such that $f(h_{\text{in}}, m) = h_{\text{out}}$.

Type 1 is sometimes called a pseudo-preimage attack on the compression function. We now describe how to carry out these attacks on the MD2 compression function.

5.1.1. *A Type 1 Attack*

In a type 1 attack, the attacker is free to choose the chaining input h_{in}. Hence, a type 1 attack will always be at least as efficient as a type 2 or 3 attack. In the case of MD2, the type 1 attack can be carried out more efficiently than the two others. Let the target output be h_{out}.

Given h_{out}, we can compute the lower right triangle as indicated in Fig. 5a. By arbitrarily fixing A_1^{15} and A_2^{15}, we may complete a further two "diagonals" in A, see Fig. 5b. We note that fixing A_1^{15} and A_2^{15} introduces a condition on h_{in}: one byte-degree of freedom is lost because A_1^{15} depends only on h_{in}. A_2^{15}, on the other hand, depends also on the message block, so fixing this byte introduces no condition on h_{in}.

When h_{out}, A_1^{15} and A_2^{15} are fixed, the rectangle B does not depend on h_{in} but only depends on the message block m.

On the other hand, using just the chaining input h_{in}, we can compute all of A. By fixing the last, say, k bytes of every message block, we can furthermore, independently of the remaining bytes in the message blocks, compute a large part of C and part of the last column of B. See Fig. 6, which shows how much of B and C can be computed when $k = 6$ is assumed. The blackened bytes play a certain role in the attack. With any given k, we can compute $k + 1$ bytes of the last column of B. The idea of the attack is

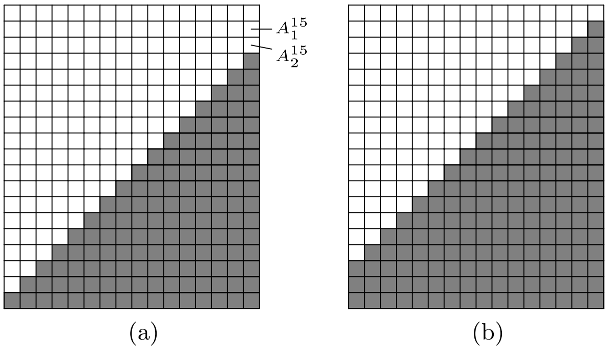


Fig. 5. The type 1 preimage attack. This figure shows the values of A that can be computed (**a**) from h_{out} and (**b**) from h_{out}, A_1^{15} and A_2^{15}.

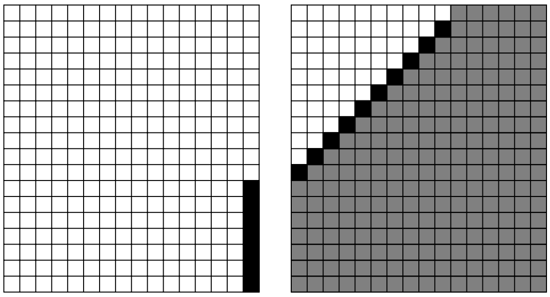


Fig. 6. The type 1 preimage attack. The shaded and blackened values are the values of B and C that can be computed once the chaining input and the last 6 bytes of the message block are fixed.

now to compute B for many different values of the message block m, but where the last k bytes of m are the same for all these message blocks. Similarly, we compute A and the part of B and C seen in Fig. 6 for many different chaining inputs h_{in}, using the fact that we know the last k bytes of the message block. We then look for a collision on the blackened bytes of B in Fig. 6. For each collision, we check if the message block and the chaining input also match on the remaining blackened bytes (those in C) in Fig. 6. If they match, then we have found a chaining input h_{in} and a message block m such that $f(h_{\text{in}}, m) = h_{\text{out}}$.

An algorithmic version of the attack, with any value of k from 0 to 16, follows.

1. We are given the target chaining output h_{out}. Fix A_1^{15} and A_2^{15} arbitrarily and compute the part of A that is shaded in Fig. 5.
2. For each of the $2^{8(16-k)}$ values of the message block for which the last k bytes are fixed to some arbitrary values, compute B and store the last column in the table T_1.
3. Choose $2^{8(k+1)}$ chaining inputs such that A_1^{15}, as chosen in Step 1, is reached. In other words, choose the first 15 bytes of each input and compute the required

value of the 16th byte. For each of these inputs, compute all of A and the parts of B and C that are shaded in Fig. 6. Store the blackened bytes in the table T_2.

4. Find collisions in T_1 and T_2 on the last $k+1$ bytes of the last column of B, i.e., the blackened bytes of B in Fig. 6. Since T_1 contains $2^{8(16-k)}$ values, and T_2 contains $2^{8(k+1)}$ values, and the collision must occur in $k+1$ bytes, we expect to find about $2^{8(16-k)+8(k+1)-8(k+1)} = 2^{8(16-k)}$ collisions.
5. For a collision from the previous step to correspond to a preimage, the chaining input/message pair must also agree in the remaining $16-k$ blackened bytes in Fig. 6. For each collision, this happens with probability about $2^{-8(16-k)}$. Hence, we expect to find one preimage with this method. Since we stored the last column of B in T_1, and all blackened bytes in T_2, we only need to do a few computations for each collision to check if there is a match.

Let us find the optimal value of k. Step 1 is only performed once and hence does not contribute to the complexity of the attack. In Step 2 we compute B $2^{8(16-k)}$ times. Since computing B corresponds to evaluating about one third of the compression function, Step 2 has complexity below $2^{8(16-k)-1}$. In Step 3 we compute about half of A, most of C, and $k+1$ bytes of B, $2^{8(k+1)}$ times. The complexity is equivalent to about $2^{8(k+1)-1}$ evaluations of the compression function. In Step 4 we find collisions between T_1 and T_2. If the tables are sorted, which can be done as they are formed, the expected complexity of finding the collisions is about $\max(2^{8(16-k)}, 2^{8(k+1)})/832$, assuming that a comparison on the $k+1$ bytes on average requires an amount of work similar to computing one byte in the compression function (recall that 832 bytes need to be computed in the compression function). In Step 5 we need to check each collision to see if the collision extends to the remaining blackened bytes. There are about $2^{8(16-k)}$ collisions, and we need to compute on average about $16-k$ bytes to check if there is a match (in most cases, there will be no match already on the byte C_1^{15-k}, i.e., the top blackened byte in Fig. 6). Hence, the expected complexity of Step 5 is about $2^{8(16-k)} \cdot (16-k)/832$.

To sum up, Steps 2, 3, and 5 are expected to dominate the total complexity, so we try to minimise these complexities. Having $16-k = k+1$ would give Steps 2 and 3 similar complexity. This would result in k being a noninteger, which complicates the attack. With $k=7$ we get a total complexity of about $2^{71} + 2^{63} + 2^{72} \cdot 9/832 \approx 2^{71}$. With $k=8$ the complexity is about $2^{63} + 2^{71} + 2^{64} \cdot 8/832 \approx 2^{71}$. The memory requirements and the number of memory accesses are about 2^{72} in both cases; therefore, we might "round up" the time complexity also to 2^{72}.

We would like to point out that there are two important differences between the description of the attack given here and that of Muller [12]:

– In Muller [12], k is chosen to be 6, but the work done in Steps 2 and 3 is scaled differently. One does not really need to choose $2^{8(16-k)}$ message blocks and $2^{8(k+1)}$ chaining inputs, as described above, but one cannot choose more message blocks than this number, and the product of the two numbers must be at least 2^{136} to get a preimage with good probability. The resulting complexity in Muller [12] is dominated by Step 5 and is therefore about $2^{80} \cdot 10/832 \approx 2^{73.6}$.
– In Muller [12], the last $k=6$ bytes of the chaining input are also fixed. This is not necessary, since C can be (and needs to be) computed from scratch for every chaining input. The reason is that the first column of A depends on all bytes of the

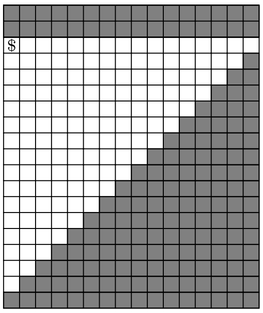


Fig. 7. The type 2 preimage attack. The shaded bytes are the bytes of A that can be computed from h_{in} and h_{out}. By fixing the value marked '\$', the rest of A can be computed.

chaining input, and therefore C does as well. The fact that only a single byte of the chaining input is fixed (by A_1^{15}) leads to an improvement of the preimage attack on the full MD2 hash function, as we shall see in Sect. 5.2.

5.1.2. *A Type 2 Attack*

In a type 2 attack on the MD2 compression function, a solution is not guaranteed to exist, but on the average one message m solves $f(h_{\text{in}}, m) = h_{\text{out}}$, where h_{in} and h_{out} are given. We now describe such an attack. The attack follows the same principles as the collision attack of Sect. 4.3.

Since h_{in} and h_{out} are given, we can compute a large part of A as shown in Fig. 7. As shown, by fixing a single byte of A, all of A can be computed. With h_{in} and h_{out} given, most likely only a single value of the byte A_2^0, indicated with a '\$' in Fig. 7, will produce a preimage. With the first and last column of A known, we know what enters the first column of B and what leaves the last column of C.

The attack proceeds as follows, where $A_2^0 = 0$ to begin with (the reader may again refer to Fig. 4).

1. Compute all of A.
2. For every possible value of the four bytes B_1^{15}, B_2^{15}, B_3^{15}, and B_4^{15}, do the following:
   (a) For every value of the first 8 message bytes, compute rows 1–4 of columns 0–7 of both B and C. Store the message bytes in table T_1 indexed by the value of the 8 bytes of B and C. T_1 consists of 2^{64} messages. In the case of a collision on the index value, resolve this by using, e.g., a linked list.
   (b) For every value of the last 8 message bytes, compute rows 1–4 of columns 7–15 of both B and C (in the backward direction). Store the message bytes in table T_2 indexed by the value of the 8 bytes of B and C. T_2 also consists of 2^{64} values, and collisions must be taken care of.
   (c) Find all matches between indices of tables T_1 and T_2. Produce corresponding message blocks by all possible combinations of message halves stored at colliding indices. Since there are 2^{128} possible messages and 2^{64} possible indices, about 2^{64} candidate messages are expected.

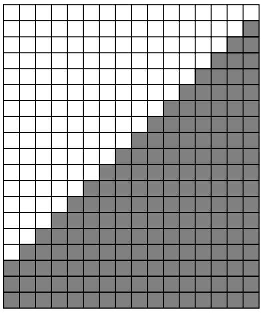


Fig. 8. The type 3 preimage attack. Known values in A (shaded) after having guessed two bytes, e.g., A_1^{15} and A_2^{15}. All values in B are known at this point.

(d) For every candidate message, check if it is a proper preimage, and if it is, add it to list M.
(e) Increment A_2^0 by 1. If $A_2^0 = 256$, stop. Otherwise, go to Step 1.

At the end M contains all preimages for the given values of h_{in} and h_{out}. The expected number of such preimages is 1.

The complexity of the attack must be evaluated. First, there is an outer loop counting over the 2^8 possibilities of A_2^0. Second, there is another loop inside the first, which counts over the four bytes B_i^{15}, $1 \le i \le 4$. For these, there are 2^{32} possibilities. For each of these, in total 2^{40} values, we produce two lists of 2^{64} elements each by computing four rows of B and C. However, as described in Sect. 4.3, by not doing the same work more than once, we may compute the 8 bytes by (on average) 2 32-bit xors, and hence we estimate the workload to be equivalent to 2^{-8} compression function evaluations. In total, producing the two lists requires an estimated time about 2^{57} in terms of compression function calls. Once the lists are produced, collisions can be found in linear time since the lists are sorted. Hence, we estimate that this step takes about the same time as producing the two lists, i.e., 2^{57}. Finally, we must search the about 2^{64} candidates for messages that produce the right A_{18}. On average, about 32 bytes of the full state X must be computed for each candidate, and hence we may estimate that this step corresponds to $2^{64} \cdot 32/832 \approx 2^{59.6}$ evaluations of the compression function. This should be added to the $2 \cdot 2^{57}$ and subsequently multiplied by 2^{40}, yielding a total complexity of about 2^{100}, not counting memory accesses, which amount to about 2^{105}. The memory requirements are about 2^{64} message blocks.

5.1.3. *A Type 3 Attack*

The type 3 attack on the MD2 compression function is similar to the type 2 attack. In this case B_0 is known, but A_0 is not. From A_{18}, however, the lower right triangle of A can be computed, and by guessing two bytes, e.g., A_1^{15} and A_2^{15}, all of B can be computed. See Fig. 8, where the known bytes in A (after guessing two bytes) are shaded.

Now one may perform a meet-in-the-middle attack as the one in the type 2 attack, only the four bytes in rows 1–4 of the last column of C are now guessed, and the meet-in-the-middle columns are columns 7 of C and A. The complexity of the attack after guessing is the same, but in this type 3 attack we need to guess one more byte, increasing the complexity to about 2^{108} (2^{113} memory accesses).

5.2. *Preimage Attack on the Full MD2*

The preimage attacks on the MD2 compression function can be extended to cover the full MD2 hash function, as described in this section. In the previous papers [6,12] the type 2 preimage attack from above was used to produce a preimage on the hash function. However, given the additional freedom in the choice of h_{in} in the type 1 attack, compared to the description of this attack in Muller [12], the type 1 attack can in fact be used to produce preimages for the full hash function more efficiently.

The idea is to compute the $2^{8(k+1)}$ chaining inputs from a given initial value, instead of choosing them directly. Say we are given an initial value h_{iv}. Then we may choose message blocks m_0^i, $0 \le i < 2^{8(k+2)}$ and compute $h_1^i = f(h_{\text{iv}}, m_0^i)$ for each i. Since only about 1 in 256 of these will produce the value of A_1^{15} that we choose in the beginning, there will be about $2^{8(k+1)}$ valid chaining inputs. This is the number required in the type 1 attack, as described in Sect. 5.1.

We still have not taken the checksum into account. We shall postpone this a little longer and determine the complexity of the attack when the chaining inputs are computed rather than chosen directly. The only difference is in Step 3, where the complexity now is about $2^{8(k+2)}$ instead of $2^{8(k+1)-1}$. With $k = 7$ the total complexity is about $2^{71} + 2^{72} + 2^{72} \cdot 9/832 \approx 2^{72.6}$.

The attack described provides two messages blocks, say m_0 and m_1, such that $f(f(h_{\text{iv}}, m_0), m_1) = h_{\text{T}}$ for some given target chaining output h_{T} and a given initial chaining value h_{iv}. The probability that m_1 is the checksum of m_0, as required, is only about 2^{-128}. However, if we have 2^{128} preimages of h_{T}, then with good probability, one of them will have m_1 as its checksum. We do not have to carry out the attack 2^{128} times, though. Instead, we do the following (see also Fig. 9).

1. Given a target hash value h_{T} and the initial value h_0 of MD2, produce a 2^{128}-way multicollision by the method of Joux [4], starting from h_0. Let the common chaining output of all these 2^{128} messages be h_{128}.
2. With $h_{\text{iv}} = h_{128}$, carry out the preimage attack described above on the compression function. Say this preimage attack produces two message blocks m_0 and m_1 such that $f(h_{\text{iv}}, m_0) = h_{129}$ and $f(h_{129}, m_1) = h_{\text{T}}$.
3. Find the checksum state C^* such that $g(C^*, m_0) = m_1$, where g is the checksum function of MD2. This step requires negligible time, since inverting the checksum function is as efficient as computing it in the forward direction.
4. Perform a meet-in-the-middle search through the 2^{128}-way multicollision to find a message in this multicollision which has checksum C^*. This search is done by computing the 2^{64} checksums of the first 64 blocks of the messages in the multicollision, and also the 2^{64} checksums reached by inverting the checksum backwards through the last 64 blocks of the messages in the multicollision. One then searches for a collision among these 2×2^{64} values. Let the message found by this meet-in-the-middle search be M.
5. A preimage of h_{T} is $m^* = M \| m_0$. m_1 is the checksum of m^*. It is a simple matter to ensure, in Step 2, that m_0 is properly padded.

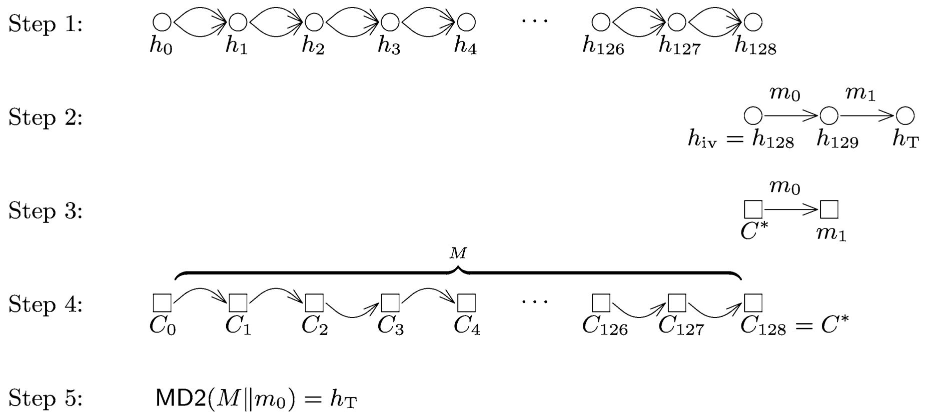

Step 1:
h_0 h_1 h_2 h_3 h_4 $\cdots$ h_{126} h_{127} h_{128}
Step 2:
m_0 m_1
$h_{\text{iv}} = h_{128}$ h_{129} h_{T}
Step 3:
m_0
C^* m_1
M
Step 4:
C_0 C_1 C_2 C_3 C_4 $\cdots$ C_{126} C_{127} $C_{128} = C^*$
Step 5:
$\mathsf{MD2}(M \| m_0) = h_{\text{T}}$

Fig. 9. An illustration of the 5 steps of the preimage attack on the full MD2 hash function. Circles represent hash (chaining) states, and squares represent checksum states.

(We note that the technique used in Step 4 to find the message having the right checksum has been used before in preimage attacks on other hash functions [7–10].) Step 1 requires finding 128 collisions in the MD2 compression function. By the birthday method, this has complexity about $2^{64+7} = 2^{71}$. By using the collision attack of Sect. 4.3, the complexity is only about 2^{60}. Step 2 has complexity about $2^{72.6}$, as found above. Step 3 has negligible complexity, and Step 4 requires about 2^{66} evaluations of the checksum function, when the tree structure of the multicollisions is taken into account. Hence, the complexity is about $2^{60.3}$ in terms of compression function evaluations. Altogether, Step 2 dominates the attack with respect to complexity, and hence the total complexity is about $2^{72.6}$. If collisions are found by a birthday attack in Step 1, then the total complexity is about 2^{73}. As in the type 1 preimage attack on the MD2 compression function, memory requirements are about 2^{72}. The meet-in-the-middle attack in Step 4 can be carried out using memoryless techniques [16,21].

The preimage is of length 129 message blocks. If we can find a shorter 2^{128}-way multicollision, then we can find a shorter preimage. We now describe how the type 1 attack on the compression function can be used to produce short multicollisions faster than by brute force. Then we explain how to use these multicollisions to produce shorter preimages for the full hash function.

5.2.1. *Multicollisions*

By using more chaining inputs in the type 1 attack, we can find several preimages. For instance, we may choose 2^{80} message blocks m_0^i, used to produce chaining inputs for the type 1 attack. This would result in about 2^{72} valid chaining inputs. With $k = 7$ we get expected 2^8 preimages. The complexity is about 2^{80} in terms of compression function evaluations. Notice that this attack can be carried out using any initial chaining value h_{in} and any target chaining value h_{T}. The result is a set of 2^8 two-block messages all mapping h_{in} to h_{T}, and hence we have a 2^8-way multi-collision. This method can be generalised; a 2^t-way multicollision can be constructed in time about 2^{72+t} for t from 0 to 56.

Table 2. Complexity of the preimage attack with different preimage lengths.

Preimage length (blocks)	Complexity
129	$2^{72.6}$
63	2^{81}
31	2^{84}
15	2^{91}
7	2^{106}

5.2.2. *Shorter Preimages*

Let $h_{32} = h_\mathrm{T}$ be the target hash, and choose h_{30} arbitrarily. Then find a 2^8-way multicollision mapping h_{30} to h_{32}. Repeat this procedure, finding 2^8-way multicollisions mapping h_{2i} to $h_{2(i+1)}$ for decreasing i from 14 down to 0. h_{2i} may be chosen arbitrarily in each step, except when $i = 0$: h_0 must be chosen as the initial value of MD2.

By this procedure we obtain $(2^8)^{16} = 2^{128}$ preimages of h_T, and we expect one of them to have the right checksum. The complexity is about $16 \times 2^{80} = 2^{84}$, and the preimages are of length $2 \cdot 16 - 1 = 31$ blocks (the last block being the checksum). Other combinations of preimage lengths and complexities are possible; see Table 2 for examples.

5.3. *2nd Preimages*

The preimage attack is also a 2nd preimage attack, since it is trivial to ensure that the preimage obtained is different from some given preimage. Hence, 2nd preimages of length 129 blocks can be found in time about 2^{73}.

6. Conclusion

This paper describes both known and previously unpublished cryptanalysis of the cryptographic hash function MD2. The cryptanalysis includes a number of collision and preimage attacks on the MD2 compression function, and also methods of extending these to the full MD2 hash function. The attacks on the full hash function constitute (in the case of preimage attacks: significant) improvements over the previous best known attacks.

The attacks prove that MD2 is neither one-way nor collision resistant, and hence we recommend that the use of MD2 in any application relying on its cryptographic properties be phased out.

Acknowledgements

We would like to thank the referees for very helpful comments and Martin Clausen for making us aware of the use of MD2 in VeriSign certificates.

Appendix A. MD2 Details

Some details of the MD2 compression function are omitted in the paper but given here.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	29	2e	43	c9	a2	d8	7c	01	3d	36	54	a1	ec	f0	06	13
1	62	a7	05	f3	c0	c7	73	8c	98	93	2b	d9	bc	4c	82	ca
2	1e	9b	57	3c	fd	d4	e0	16	67	42	6f	18	8a	17	e5	12
3	be	4e	c4	d6	da	9e	de	49	a0	fb	f5	8e	bb	2f	ee	7a
4	a9	68	79	91	15	b2	07	3f	94	c2	10	89	0b	22	5f	21
5	80	7f	5d	9a	5a	90	32	27	35	3e	cc	e7	bf	f7	97	03
6	ff	19	30	b3	48	a5	b5	d1	d7	5e	92	2a	ac	56	aa	c6
7	4f	b8	38	d2	96	a4	7d	b6	76	fc	6b	e2	9c	74	04	f1
8	45	9d	70	59	64	71	87	20	86	5b	cf	65	e6	2d	a8	02
9	1b	60	25	ad	ae	b0	b9	f6	1c	46	61	69	34	40	7e	0f
a	55	47	a3	23	dd	51	af	3a	c3	5c	f9	ce	ba	c5	ea	26
b	2c	53	0d	6e	85	28	84	09	d3	df	cd	f4	41	81	4d	52
c	6a	dc	37	c8	6c	c1	ab	fa	24	e1	7b	08	0c	bd	b1	4a
d	78	88	95	8b	e3	63	e8	6d	e9	cb	d5	fe	3b	00	1d	39
e	f2	ef	b7	0e	66	58	d0	e4	a6	77	72	f8	eb	75	4b	0a
f	31	44	50	b4	8f	ed	1f	1a	db	99	8d	33	9f	11	83	14

Fig. 10. The MD2 S-box.

A.1. *The S-box*

The MD2 S-box is defined as follows. View the input as a two-digit hexadecimal value. In Fig. 10, find the first input digit in the first column and find the second input digit in the first row. Where the row and the column meet, find the output of S (in hexadecimal). This S-box is derived from the digits of the fractional part of π.

A.2. *The Checksum Function*

The checksum function of MD2 operates with a 128-bit (16-byte) state (initially all bytes are zero), which is updated by a message block of the same size. Let D denote the state, and D^i be the ith byte of the state. Let m be the message block with ith byte m^i. The state D is updated by m as follows.

```
1. $L \leftarrow D^{15}$
2. For increasing $i$ from 0 to 15 do
    (a) $D^i \leftarrow D^i \oplus S(L \oplus m^i)$
    (b) $L \leftarrow D^i$
```

The final value of the state D is the checksum of the message and will be processed as a final message block. The reader may note the similarity between the checksum function and the compression function of MD2.

References

[1] R.P. Brent, An improved Monte Carlo factorization algorithm. *BIT*, **20**(2), 176–184 (1980)

[2] I. Damgård, A design principle for hash functions, in *Advances in Cryptology—CRYPTO '89, Proceedings*, ed. by G. Brassard. Lecture Notes in Computer Science, vol. 435 (Springer, Berlin, 1990), pp. 416–427

[3] R.W. Floyd, Nondeterministic algorithms. *J. Assoc. Comput. Mach.* **14**(4), 636–644 (1967)

[4] A. Joux, Multicollisions in iterated hash functions. Application to cascaded constructions, in *Advances in Cryptology—CRYPTO 2004, Proceedings*, ed. by M.K. Franklin. Lecture Notes in Computer Science, vol. 3152 (Springer, Berlin, 2004), pp. 306–316

[5] B.S. Kaliski Jr., The MD2 Message-Digest Algorithm, April 1992. Network Working Group, Request for Comments: 1319

[6] L.R. Knudsen, J.E. Mathiassen, Preimage and collision attacks on MD2, in *Fast Software Encryption 2005, Proceedings*, eds. by H. Gilbert, H. Handschuh. Lecture Notes in Computer Science, vol. 3557 (Springer, Berlin, 2005), pp. 255–267

[7] X. Lai, J.L. Massey, Hash functions based on block ciphers, in *Advances in Cryptology—EUROCRYPT '92, Proceedings*, ed. by R.A. Rueppel. Lecture Notes in Computer Science, vol. 658 (Springer, Berlin, 1993), pp. 55–70

[8] F. Mendel, V. Rijmen, Weaknesses in the HAS-V compression function, in *International Conference on Information Security and Cryptology (ICISC) 2007, Proceedings*, ed. by K.-H. Nam, G. Rhee. Lecture Notes in Computer Science, vol. 4817 (Springer, Berlin, 2007), pp. 335–345

[9] F. Mendel, N. Pramstaller, C. Rechberger, A (second) preimage attack on the GOST hash function, in *Fast Software Encryption 2008, Proceedings*, ed. by K. Nyberg. Lecture Notes in Computer Science, vol. 5086 (Springer, Berlin, 2008), pp. 224–234

[10] F. Mendel, N. Pramstaller, C. Rechberger, M. Kontak, J. Szmidt, Cryptanalysis of the GOST hash function, in *Advances in Cryptology—CRYPTO 2008, Proceedings*, ed. by D. Wagner. Lecture Notes in Computer Science, vol. 5157 (Springer, Berlin, 2008), pp. 162–178

[11] R.C. Merkle, One way hash functions and DES, in *Advances in Cryptology—CRYPTO '89, Proceedings*, ed. by G. Brassard. Lecture Notes in Computer Science, vol. 435 (Springer, Berlin, 1990), pp. 428–446

[12] F. Muller, The MD2 hash function is not one-way, in *Advances in Cryptology—ASIACRYPT 2004, Proceedings*, ed. by P.J. Lee. Lecture Notes in Computer Science, vol. 3329 (Springer, Berlin, 2004), pp. 214–229

[13] National Institute of Standards and Technology. FIPS PUB 180-2, Secure Hash Standard, 1 August 2002

[14] G. Nivasch, Cycle detection using a stack. *Inf. Process. Lett.* **90**(3), 135–140 (2004)

[15] B. Preneel, Analysis and design of cryptographic hash functions. PhD thesis, Katholieke Universiteit Leuven, January 1993

[16] J.-J. Quisquater, J.-P. Delescaille, How easy is collision search. New results and applications to DES, in *Advances in Cryptology—CRYPTO '89, Proceedings*, ed. by G. Brassard. Lecture Notes in Computer Science, vol. 435 (Springer, Berlin, 1990), pp. 408–413

[17] R.L. Rivest, The MD4 message digest algorithm, in *Advances in Cryptology—CRYPTO '90, Proceedings*, eds. by A. Menezes, S.A. Vanstone. Lecture Notes in Computer Science, vol. 537 (Springer, Berlin, 1991), pp. 303–311

[18] R.L. Rivest, The MD5 Message-Digest Algorithm, April 1992. Network Working Group, Request For Comments: 1321

[19] N. Rogier, P. Chauvaud, MD2 is not secure without the checksum byte. *Des. Codes Cryptogr.* **12**(3), 245–251 (1997)

[20] RSA Laboratories, PKCS #1: RSA Cryptography Standard (Version 2.1, June 14, 2002). Available: http://www.rsa.com/rsalabs/node.asp?id=2125 [2009/1/28]

[21] P.C. van Oorschot, M.J. Wiener, Parallel collision search with cryptanalytic applications. *J. Cryptol.* **12**(1), 1–28 (1999)

[22] Verisign, Inc. Status Responder Certificate. Class 3 Public Primary Certification Authority. Serial number: 70:BA:E4:1D:10:D9:29:34:B6:38:CA:7B:03:CC:BA:BF. Issued 1996/01/29, expires 2028/08/02. http://www.verisign.com/repository/root.html#c3pca [2009/08/17]