The MD2 Hash Function is Not One-Way

Frédéric Muller

DCSSI Crypto Lab 51, Boulevard de Latour-Maubourg
75700 Paris 07 SP France
Frederic.Muller@sgdn.pm.gouv.fr

Abstract. MD2 is an early hash function developed by Ron Rivest for RSA Security, that produces message digests of 128 bits. In this paper, we show that MD2 does not reach the ideal security level of 2^{128}. We describe preimage attacks against the underlying compression function, the best of which has complexity of 2^{73}. As a result, the full MD2 hash can be attacked in preimage with complexity of 2^{104}.

1 Introduction

Cryptographic hash functions are an important primitive used in various situations. The main fields of applications are message authentication codes, digital signatures, and therefore certificates. Hash functions are also used as a building tool in many protocols and advanced constructions.

By definition, a hash function H is a function mapping an input message m of arbitrary length to an output h of fixed length (typically this length ranges from 128 to 512 bits)

$$h = H(m)$$

The main properties expected from a cryptographic hash function are :

- **collision resistance** : it should be hard to find two inputs m and m' that map to the same output by H.
- **second preimage resistance** : for a given m, it should be hard to find a second input m' such that m and m' map to the same output by H.
- **preimage resistance** : for a given challenge h, it should be hard to find an input m which maps to h by H.

More can be found on the theory of hash functions in [9, 10]. Most of the hash functions used in practice belong to the so-called "MD family". This family of hash functions was initially developed by Ron Rivest for RSA Security. The first proposal was MD2 [7], an early, non-conventional, byte-oriented design. It was quickly followed by MD4 [11] and MD5 [12], two hash functions with a more modern, 32-bit-oriented design. Despite not being collision-resistant [3], MD4 has inspired most modern hash functions designs, like the RIPEMD family or the SHA family. Over the last years, the effort on attacking hash functions has mostly concerned collision resistance [2–4, 15], since this property is essential for many applications. However, few results have been reported regarding (second) preimage attacks for these hash functions (see [5, 9]).

In this paper, we focus on the MD2 hash function [7]. Despite being the oldest hash functions from its family, and despite using an old-fashioned architecture, MD2 is still used in several contexts. For instance, if we look at the recent PKCS #1 v2.1, a cryptographic standard from RSA Security [17], the MD2 hash is still given as an example of one-way, collision-resistant hash function, while MD4 has been removed, presumably because of Dobbertin's collision attack [3]. In addition, it is precised that "MD2 (is) recommended only for compatibility with existing applications based on PKCS #1 v1.5". The underlying explanation is that the use of MD2 was highly encouraged in the previous version from 1993 [16] where MD2 was recommended as a "conservative design". This confidence in MD2 is not surprising because, despite being quite inefficient and based on an older design philosophy, MD2 has surprisingly well resisted to cryptanalysis. The only attack known is a collision attack against the compression function [14]. This attacks works with the correct IV, however it no longer works when a checksum is appended to the message, as imposed in the specifications [7]. For the full hash function, no attack is known.

Consequently MD2 still appears in various applications and even some proposed standards [1]. However, the crucial security point regarding MD2 is now its use in public-key infrastructures. Many certificates have been generated with RSA-MD2 in the past and many of them are still widely used (like Verisign certificates for instance). Actually, anyone can easily verify that recent versions of Windows are delivered with those MD2 certificates. Therefore millions of users are probably using MD2-based certificates on a regular basis. The security of certificates is a particular problem. Indeed, collision attacks do not threat the security of the scheme, because the input of the signature primitive (typically the usual primitive used with MD2 is the RSA signature) is fixed. An attacker needs to find a collision between two inputs of MD2, one of them being the data part of the certificate. If he succeeds, he will manage to forge a new valid certificate. Hence what is required here is exactly second preimage resistance of MD2. This is an important motivation to analyze the security of MD2 regarding preimage and second preimage attacks, which is the focus of this paper. We obtained interesting new results and theoretical attacks. Since our best attack against MD2 is more efficient than a naive guessing attack in 2^{128}, MD2 can no longer be considered a secure one-way hash function.

First, we describe briefly the MD2 algorithm. Then, we focus on the compression function and describe several attacks. The best is a pseudo-preimage attack with complexity 2^{73}. Finally, we show how to turn these attacks into an attack for the full hash, which is not straightforward because of the checksum bytes.

2 The MD2 Hash Function

2.1 Generalities

The MD2 Message-Digest algorithm was developed in 1989 by Ron Rivest. The actual specifications can be found in RFC 1319 [7]. This algorithm belongs, together with MD4 and MD5, to the family of hash functions developed by Ron Rivest for RSA Security. However, compared to the other algorithms of the family (and to most actual hash functions), MD2 has several interesting particularities

- MD2 is a **byte-oriented** hash function. Indeed all instructions handle 8 bits of data. While this was useful for old architectures, today's processors can manipulate words of (at least) 32 bits. Consequently all modern hash functions use 32-bit instructions. This is the case of MD4, MD5 and also for the hash functions of the RIPEMD and SHA families.
- MD2 uses a **checksum** of 128 bits computed from the whole message and appended as the last input block of the compression function. Hence MD2 does not follow the Merkle-Damgård construction, contrarily to most actual hash functions. Consequently classical results [9] on how to turn collisions on the compression function to collisions for the whole hash function do not apply here. This is the reason why the collision attack described in [14] does not extend to the full MD2 hash.
- the compression function of MD2 has a different architecture from most modern hash functions. Indeed it does not look like a block cipher. Instead, a fixed "scrambling" function is iterated on a 384 bits long internal state. The initial state is derived linearly from a message block of length 128 bits and an intermediate hash of 128 bits. The final state is truncated to 128 bits. This function uses simple instructions like XOR and a nonlinear S-box.

Therefore MD2 is a very early design of hash function and differs significantly from modern hash functions. In terms of efficiency, it compares quite bad to its challengers (mostly because of the byte-oriented structure).

2.2 Description of MD2

In this section, we describe more precisely the mechanisms used by the MD2 hash function (see [7] for the full specifications). The general description of MD2 is found in Figure 1.

All blocks manipulated have length 128 bits. We refer to the blocks of the message by $M_0, \ldots, M_n$. The first step of MD2 is to append a padding to the initial message, then to compute a checksum block (that we call C). This increases the length of the message by 1 block. Finally the compression function (referred to as F) is applied iteratively to produce the hash value. If we call H_i the i-th intermediate hash,

$$H_{i+1} = F(H_i, M_i)$$

The IV of the hash function is H_0 and is set by default to 0.

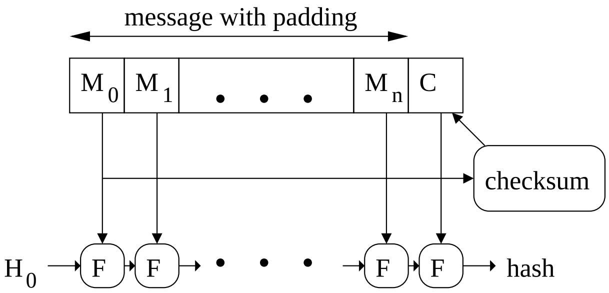

Fig. 1. The MD2 Hash Function

The compression function A precise representation of the compression function F is given in Figure 2. Each box in this figure contains one byte. F is decomposed into 3 matrices - denoted by A, B and C - with 16 columns and 19 rows each. The first row of each matrix is initialized respectively with H_i, M_i and $H_i \oplus M_i$. Then the rows of each matrix are computed recursively from top to bottom. The last rows of B and C are not used. The '+' symbol denotes addition modulo 256.

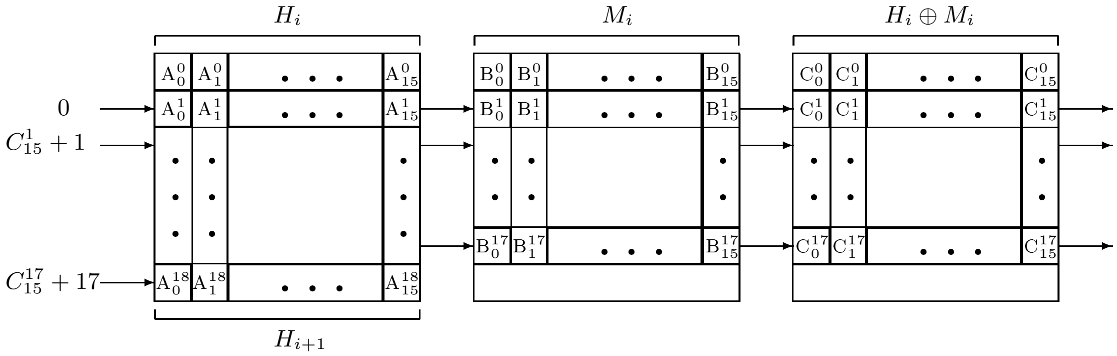

Fig. 2. The Compression Function of MD2

The computations are based on a function ϕ from 16 bits to 8 bits. In the case of the matrix A, this can be described by the equations :

$$\begin{aligned} A_i^t &= \phi(A_i^{t-1}, A_{i-1}^t) \\ &= A_i^{t-1} \oplus \mathrm{S}(A_{i-1}^t) \end{aligned}$$

where S is a fixed S-box of size 8 bits. Equations for matrices B and C are exactly the same. This function ϕ is represented in Figure 3. For the particular case $i = 0$, a byte extracted from another matrix is used instead of A^t_{i-1} (see Figure 2).

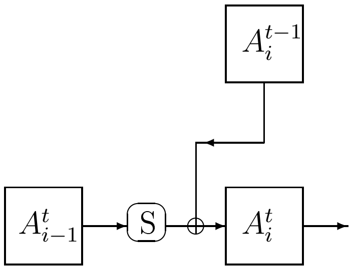

Fig. 3. The ϕ function

The checksum function The checksum C is computed from the blocks of message by iterating a non-linear checksum function, that we call G. Details on G are not relevant for our attacks. Basically G uses only basic operations like XOR and the S-box S. At a high-level, the following equations describe this mechanism :

$$\mathrm{IC}_0 = 0$$
$$\mathrm{IC}_{i+1} = G(\mathrm{IC}_i, M_i)$$

G is a complicated function, however it is straightforward to compute the intermediate checksum IC_i from IC_{i+1} and M_i. The final value IC_{n+1} is the appended checksum C. A precise description of G is available in [7].

3 Collision Attacks against MD2

The only known cryptanalytic result against MD2 is the paper by Rogier and Chauvaud [14]. In this paper, collisions on the compression function of F are described. This attack works very well because the IV used in MD2 is O (although a variant is proposed for other IV's with an increased complexity). Details of this attack are not essential here. The key idea is to use the symmetry between matrices B and C when $H_i = 0$. (the first rows are equal in this case). Unfortunately collisions cannot be extended to the full MD2 because of the checksum bytes.

Although collision attacks may be of interest in many contexts, there are several arguments why researching efficient collision attacks for MD2 is no longer a major concern.

- First, one has to take into account the dimension of MD2. The produced hashed values have length only 128 bits. Therefore birthday paradox attacks have complexity of the order of 2^{64}. This is not a satisfying level of security for modern applications. As an example, the MD5 hash function (whose output have also a length of 128 bits) is actually the subject of a distributed attack to find collisions [8]. It is clear that the interest of finding complicated shortcut attacks diminishes when efficient attacks using a large computational power are possible [18].
- Secondly, MD2 is no longer widely used in practice. For instance, in MAC or signatures, the collision resistance of a hash function is generally a requirement, but MD2 is no longer recommended for such applications. However, as we mentioned previously, MD2 is still used in some certificates. In this context, collision resistance is not really a concern but preimage and second preimage resistance are required.

4 Preimage Attacks against MD2 Compression Function

A large variety of definitions for preimage and second preimage attacks exist in the literature, depending on what is a fixed challenge for the attacker and what can be freely chosen. A classical reference is [9], however a new classification of these notions has been recently given in [13].

In this section, we focus only on (preimage) attacks against the compression function of MD2. It is well known that these attacks can generally be extended to attacks against the whole hash (see [9]).

4.1 Three Scenarios

According to the previous notations, the compression function F operates by :

$$H_{i+1} = F(H_i, M_i)$$

where the H_i's are intermediate hash values and M_i is a message block (see Section 2.2). Basically we can consider 3 attack scenarios at this point :

1. H_{i+1} and H_i are given and the attacker must find an appropriate M_i.
2. H_{i+1} and M_i are given and the attacker must find an appropriate H_i.
3. H_{i+1} is given and the attacker must find appropriate H_i and M_i.

Any of these attacks may be of interest to attack the whole hash. Obviously, the 1st and 2nd attack are very similar because the roles of H_i and M_i in F are almost symmetric.

These 3 attack scenarios have received different names in the literature. Recently the names "aPre" ("a" stands for "always"), "ePre" ("e" stands for "everywhere") and "Pre" have been given to these 3 notions [13]. In [9], the terminology of "preimage resistance" and "pseudo-preimage resistance" is used. In the following sections, we envisage each scenario separately and propose new attacks.

4.2 Attacking Scenario 1

In this scenario, we suppose that H_i and H_{i+1} are a fixed challenge and our goal is to find an appropriate M_i such that

$$H_{i+1} = F(H_i, M_i)$$

First, we notice that a solution does not necessarily exist. Indeed all variables have length 128 bits, so in average only one solution M_i is expected, but there is no guarantee. We propose an attack that recovers all solutions corresponding to a given challenge (H_i, H_{i+1}). Basically our attack is a sophisticated combination of exhaustive search and meet-in-the-middle attacks. It proceeds with two distinct steps. In the following, we call $(m_0, \ldots, m_{15})$ the 16 bytes of M_i.

First Step The first step of the attack is to derive all possible information from the challenge (H_i, H_{i+1}). These two objects are stored at the first and last row of matrix A (see Figure 4 where dashed cells correspond to the known bytes).

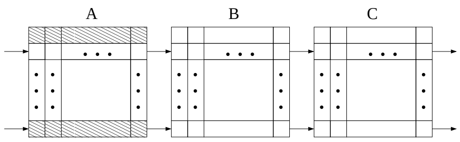

Fig. 4. Initial knowledge when H_{i+1} and H_i are fixed

Because of the structure of ϕ (this function is used to compute the contents of the matrices, see Section 2.2), more information can be derived directly from the challenge. For instance, when A_{i-1}^t and A_i^t are known, we can obtain A_i^{t-1} since :

$$\begin{aligned} A_i^t &= \phi(A_i^{t-1}, A_{i-1}^t) \\ &= A_i^{t-1} \oplus \mathrm{S}(A_{i-1}^t) \end{aligned}$$

In Figure 5, we represented by dashed boxes the large portion of A that can be directly derived this way. The second row is known because the byte introduced on the left hand side is known and always equal to 0.

In addition, if we **guess the byte introduced on the left hand side** of the 3rd row in A (*i.e.* $C_{15}^1 + 1$), then we can derive the **full content** of matrix A by similar considerations. In particular the bytes A_{15}^i's are known, and also the bytes C_{15}^i's for $i > 0$.

Second Step Then, the second step of the attack is to perform a meet-in-the-middle attack on the matrices B and C to find an appropriate value of M_i.

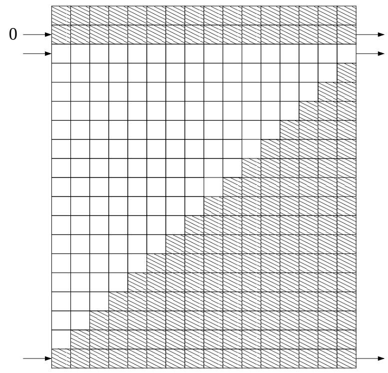

Fig. 5. Known values in the matrix A

Basically at this point, we know what enters on the left hand side of B and what exits on the right hand side of C. Hence, we apply the following "meet-in-the-middle" algorithm :

- Guess the 4 bytes $(B^1_{15}, \ldots, B^4_{15})$
  - for all values of the 8 bytes $(m_0, \ldots, m_7)$,
    - compute the 4 bytes $(B^1_7, \ldots, B^4_7)$ (this is possible because the sequence of A^i_{15}'s is known)
    - compute the 4 bytes $(C^1_7, \ldots, C^4_7)$ (this is possible because H_i is known)
    - store these $4+4=8$ bytes in a table T_1
  - sort T_1 (which has 2^{64} entries of 64 bits each)
  - Repeat the same process with $(m_8, \ldots, m_{15})$ to obtain a table T_2 that contains also the bytes $(B^1_7, \ldots, B^4_7, C^1_7, \ldots, C^4_7)$.
  - Find all collisions between T_1 and T_2. This can be done efficiently by computing the joint product $T = T_1 \bowtie T_2$ (see [19]) with complexity of the order of 2^{64}
  - The resulting table T contains on average 2^{64} candidate values for $M_i = (m_0, \ldots, m_{15})$
  - Loop over all these candidates to find all valid M_i's

One can also refer to Figure 6 for the general philosophy of this attack. Dashed boxes represent the 8 bytes stored in tables T_1 and T_2, where we look for collisions.

Analysis In this attack, there are two outside loops. A loop of size 2^8 comes from the First Step of the attack (we need to guess one byte in order to find the full content of A). Besides an outside loop of length 2^{32} is required in the "meet-in-the-middle" algorithm. Inside these loops we need to create and to sort the tables T_1 and T_2. Those are tables with 2^{64} entrees, sorted using a key of 64

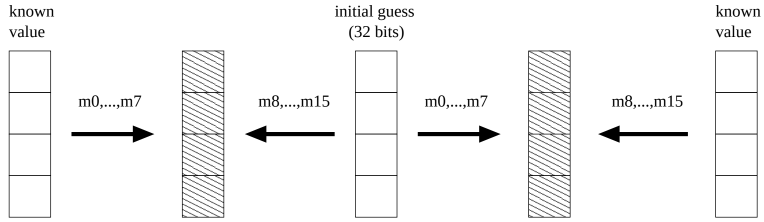

Fig. 6. The general philosophy of the attack

bits. Sorting the tables can be done efficiently with an appropriate "bucket-sort" algorithm so the cost is above 2^{64} instructions. Creating the tables has also a cost of the order of 2^{64} instructions. Since these two operations are performed twice (once for T_1 and once for T_2), the complexity is of the order of

$$\text{Complexity } = 2^8 \times 2^{32} \times (4 \times 2^{64}) = 2^{106}$$

basic instructions. This corresponds approximatively to 2^{95} applications of the compression function (a quick estimation shows that about 2^{11} instructions are needed for the compression function).

This should be compared to the complexity of an exhaustive search to find a preimage which would cost 2^{128} applications of the compression function. However, our attack requires about 2^{71} bits of memory. High memory requirements are known to increase the "real" cost of attacks [20]. Nevertheless this complexity is of the order of $2^{3n/4}$ while 2^n would be expected for a good compression function on n bits. An improved attack is also proposed in Appendix A to reduce these memory requirements. Further improvements have been investigated but no attack with complexity below $2^{3n/4}$ was found.

4.3 Attacking Scenario 2

In the second scenario, the message block M_i is fixed and we search an appropriate H_i. Attacking this scenario is very similar to attacking scenario 1 because there is an important symmetry in the compression function.

In the previous attack we managed to reconstruct the content of A from the initial challenge, and then applied a "meet-in-the-middle" attack to B and C. In Scenario 2, we can reconstruct the content of B from the challenge (M_i, H_{i+1}) and then attack by the middle the matrices A and C. Details of this attack are not very helpful to break the full MD2 hash, so we decided not to explore further this scenario.

4.4 Attacking Scenario 3

Finally, we suppose that only H_{i+1} is fixed, and the problem is to find any pair (H_i, M_i) solution of the equation

$$H_{i+1} = F(H_i, M_i)$$

This type of attack is often referred to as a **pseudo-preimage attack** on the compression function [9]. Of course, it is easier to find such a solution because we have more degrees of freedom. Therefore we wish to find an attack with complexity better than the previous 2^{95}. In this section, we describe an attack with complexity of the order of 2^{73} against this scenario.

The Attack First, one should notice that many solutions exist to this problem. Indeed, we expect

$$\frac{2^{128} \times 2^{128}}{2^{128}} = 2^{128}$$

solutions in average. Therefore it is reasonable to impose some additional constraints.

Like for the previous attacks, we first derive all possible information from the given challenge (H_{i+1} here). In addition, we impose the constraint that $A^1_{15} = A^2_{15} = c$, where c is some constant, say $c = 0$ for instance. Figure 7 represents the resulting known values in the matrix A.

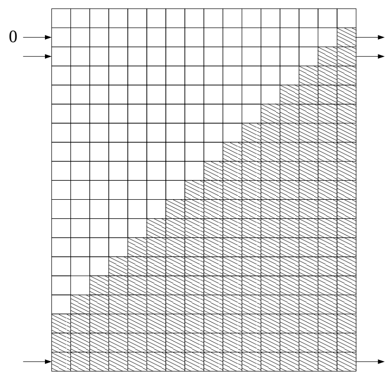

Fig. 7. Known values in the matrix A

We observe that the complete rightmost column of A is known, which helps when considering the behavior of matrix B. At this point, a 6 bytes constant $(k_0, \ldots, k_5)$ is chosen at random. Then we apply the following algorithm :

– Pick 2^{72} messages M_i of the form

$$M_i = (m_0, \ldots, m_9, k_0, \ldots, k_5)$$

where the m_i's are chosen at random. It is straightforward to compute the matrix B for each M_i since the rightmost column of A is known. Hence we build a table T (with 2^{72} entries) where we store the rightmost column of B, i.e. the B^i_{15}'s.

– Pick 2^{64} intermediate hashes H_i of the form

$$H_i = (h_0, \ldots, h_9, k_0, \ldots, k_5)$$

where the h_i's are chosen at random[1]. It is straightforward to compute the complete matrix A for each H_i. Therefore all values C_{15}^i for $i > 0$ are also known. Besides

$$H_i \oplus M_i = (*, \ldots, *, k_0 \oplus k_0, \ldots, k_5 \oplus k_5)$$

thus the 6 rightmost boxes of the first row of C are known and equal to 0. Hence a lot of information about C can be derived (see Figure 8). By the

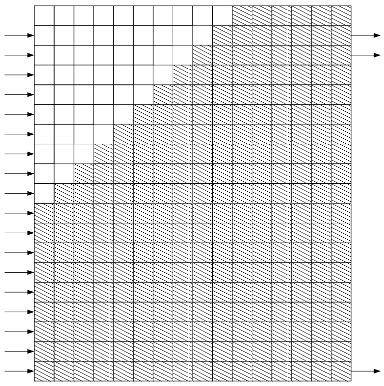

Fig. 8. Known values in the matrix C

way, the bytes B_{15}^i for $11 \leq i \leq 17$ are also known at this point. We store these elements in a table T'.

The final step of the attack is to find collisions on the objects of 56 bits

$$(B_{15}^{11}, \ldots, B_{15}^{17})$$

that have been computed by two different means and stored in tables T and T'. Using the birthday paradox, we expect 2^{80} collisions because

$$|T| \times |T'| \times 2^{-56} = 2^{72} \times 2^{64} \times 2^{-56} = 2^{80}$$

All these collisions can be found efficiently by computing $T \bowtie T'$ (see [19]). Each collision corresponds to some pair (H_i, M_i). In order for this pair to solve the initial problem, we need an additional equality between

[1] Actually there is an extra constraint, that $\phi(A_0^0) = A_0^1$. Thus only 1 out of 256 values of H_i are valid. Once $(h_1, \ldots, h_9)$ are chosen, the value of h_0 is fully determined. This induce no extra cost but must be taken into account when choosing the H_i's

– the bytes $(B_{15}^1, \ldots, B_{15}^{10})$ stored in table T
– the value of the same bytes obtained when we fill up all the content of matrix C (which is possible for each candidate since $H_i \oplus M_i$ is now known).

Hence a little extra processing is required to find a real solution and a condition on 80 bits must be verified. However, we have 2^{80} candidates from the joint product of T and T' so one "real" solution should be found among them. The probability of failure (*i.e.* that no solution exists) can be roughly approximated to $\frac{1}{e} \simeq 0.368$. Otherwise, we can pick a little more candidates for M_i and H_i or choose other constants.

Analysis The bottleneck in the previous attack is the time spent analyzing each of the 2^{80} candidates (H_i, M_i). However, using an "early-abort" strategy, most candidates can be eliminated after the first check for the value B_{15}^1. Therefore, only half a row of matrix C must be computed in average. To compute the compression function, $3 \times 18 = 54$ rows are computed. So we have a speedup by a factor

$$2 \times 54 \simeq 2^{6.75}$$

compared to a full computation of F.

Therefore this pseudo-preimage attack has complexity of about $2^{73.25}$ computations of the compression function, and requires about 2^{78} bits of memory. This is much faster than the expected value of 2^{128}. All attacks against the compression function are summarized in Table 1.

Attack	Fixed Challenge	Variable	Time	Memory
Simple	H_{i+1} and H_i	M_i	2^{95}	2^{71}
Improved	H_{i+1} and H_i	M_i	2^{95}	2^{38}
Pseudo-Preimage	H_{i+1}	H_i and M_i	2^{73}	2^{78}

Table 1. Summary of the attacks against the compression function

5 Preimage Attacks for the Full MD2 Hash

The objective of a preimage attack is, for a given challenge h, to find a message m such that hashing m with MD2 gives h :

$$\mathrm{MD2}(m) = h$$

Classical techniques exist to turn attacks against the compression function into attacks against the full hash. However they apply to classical iterated hash functions, like those based on the Merkle-Damgård paradigm. The use of an additional checksum in MD2 make things slightly more complicated.

5.1 Attacking MD2 without the checksum

If we omit the checksum, it is straightforward to apply the previous attacks directly to MD2. For instance, the attack described in Section 4.2 is immediately useful. Indeed, for a given (H_i, H_{i+1}), we are able to find M_i such that :

$$H_{i+1} = F(H_i, M_i)$$

faster than exhaustive search. If we take $H_i = 0$ (*i.e.* the IV of the MD2 specifications) and $H_{i+1} = h$ (the target value), the message of 1 block $m = M_i$ basically solves the preimage problem (some extra work might be necessary to ensure the padding is correct). Anyway, this clearly no longer works when the checksum block is appended at the end.

Preimage attacks against the full hash can also be found based on a pseudo-preimage attack (like the one described in Section 4.4, with complexity 2^{73}). For instance, a general meet-in-the-middle technique is :

- Pick 2^{100} random values of the first block of message M_1, and store all intermediate hashes H_1 in a table T_1.
- Apply 2^{28} times the pseudo-preimage attack and, for each solution (H_2, M_2), store the intermediate hash H_2 in a table T_2.
- Search for a collision between some H_1 in table T_1 and some H_2 in table T_2. The corresponding message $m = (M_1, M_2)$ is a solution.

Since $2^{100} \times 2^{28} = 2^{128}$, a collision is indeed expected. Hence this attack builds a solution m of length two blocks and has complexity of the order of 2^{101}, which is faster than exhaustive search. However when the checksum is used, this input message is likely to be invalid. Indeed, we need a collision on the intermediate hash values and the intermediate checksums simultaneously.

5.2 A Chaining Attack

The principle of chaining attacks is to iterate an attack against the compression function, while chaining the intermediate variables used in each attack. Here, we first choose at random a sequence of intermediate hashes of the form :

$$0 = H_0, H_1, \ldots, H_{127}, H_{128} = h$$

For each pair (H_i, H_{i+1}), we apply the attack of Section 4.2 to find all solutions of :

$$H_{i+1} = F(H_i, M_i)$$

A constraint we add is that at least two solutions M_i and M_i' must be found, for all i. Assuming F is a random function, this should happen with a reasonable probability (called p). It can roughly be approximated by 1 minus the probability to have exactly 0 or 1 solution :

$$\begin{aligned} p &\simeq 1 - (1 - 2^{-128})^{2^{128}-1} - (1 - 2^{-128})^{2^{128}} \\ &\simeq 1 - 2\, e^{-1} \\ &\simeq 0.264 \end{aligned}$$

If there are less than 2 solutions, we throw away H_{i+1} and pick another value. In average, we need to apply $128 \times p^{-1} \simeq 2^9$ times the attack of Section 4.2 to find an appropriate pair of solutions (M_i, M'_i) for all i.

Then, we have 2^{128} possible messages that are solution of the preimage problem for MD2 with challenge h (there are 2 possible blocks of message for all i). Among them, one of the message is likely to satisfy the checksum constraint, *i.e.* its last block should be the checksum of the 127 previous blocks. To find this message, a simple meet-in-the-middle attack applies :

- Compute the 2^{64} intermediate checksums IC_{64} by testing the two possible blocks of message at all positions $i, 0 \leq i \leq 63$.
- Compute the 2^{64} intermediate checksums IC_{64} by inverting the checksum function G, starting for both values M_{127} and M'_{127}, and for all blocks of message at positions $i, 64 \leq i < 127$.
- Search for a collision between these 2 lists of 2^{64} elements

This technique is similar to the one used in [6]. The resulting attack against the full hash is only marginally slower than the attack against the compression function, since the deterioration corresponds to a factor 2^9. Therefore it will cost about $2^{95} \times 2^9 = 2^{104}$ applications of the compression function. In addition, a memory of 2^{71} bits is required (or 2^{38} using the improved algorithm of Appendix A). This is much faster than a naive exhaustive search.

6 Second Preimage Attacks

A second preimage attack consists, on the challenge of a message m, to provide a second message m' which gives the same MD2 hash :

$$\mathrm{MD2}(m) = \mathrm{MD2}(m')$$

The resistance of MD2 against this type of attack is critical for the security of existing certificates. Indeed a certificate generally consists in a data part m and a signature of the data part. To compute this signature, a hash of the data part is generally computed. If an attacker is able to replace m with an other data part m' mapping to the same hash, he is able to forge a new certificate.

If we omit the checksum blocks for MD2, it is straightforward to find a second preimage, based on the previous attacks. For any of the intermediate steps

$$H_{i+1} = F(H_i, M_i)$$

in the original message m, we apply the attack described in Section 4.2. With probability $p \simeq 0.26$, another message block M'_i, mapping H_i to H_{i+1} is found. Then we can simply substitute M'_i to M_i to forge a new certificate.

Unfortunately, when the checksum is used, this attack no longer works because the checksum is altered by the previous substitution. Therefore the last block of message is no longer valid.

We could not find a dedicated second preimage attack against the full MD2, including the checksum bytes. An attack is still possible by applying a preimage attack on $h = \text{MD2}(m)$. The result m' is a preimage of h and is very likely to be different from m. Unfortunately m' is very constrained :

- its length is at least 128 blocks (including the checksum block), so the message m' is of length > 2 Kbytes. Some variants of the attack can increase this message length but it is not possible to reduce it. This is slightly larger than a typical certificate, however a trade-off between the size of the forged certificate and the probability of success could also be envisaged.
- at least 128 blocks in the forged certificates are random and therefore cannot be chosen by the attacker.

All together, it seems difficult for the moment to forge new certificates that respect the required format. However we are not far from it and we think it is an interesting topic for further research. We encourage a deeper analysis of the MD2 hash function whose security, especially regarding (second) preimage attacks is important for many existing certificates.

7 Conclusion

In this paper, we described preimage and pseudo-preimage attacks against the compression function of MD2, the best of which has complexity 2^{73}. The resulting attack against the full hash (including the checksum) costs about 2^{104} applications of the compression function. As a consequence, MD2 can no longer be considered a secure one-way hash function.

These results are also very interesting from a theoretical point of view, because preimage attacks against hash functions are quite rare. Most of the research in recent years has focused on finding collisions for hash functions.

References

1. D. Balenson. RFC 1423 - Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers, february 1993. RSA Laboratories.
2. F. Chabaud and A. Joux. Differential Collisions in SHA-0. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lectures Notes in Computer Science*, pages 56–71. Springer, 1998.
3. H. Dobbertin. Cryptanalysis of MD4. In D. Gollmann, editor, *Fast Software Encryption – 1996*, volume 1039 of *Lectures Notes in Computer Science*, pages 53–69. Springer, 1996.
4. H. Dobbertin. The Status of MD5 after a Recent Attack. *CryptoBytes*, 2(2):1–6, 1996.
5. H. Dobbertin. The First Two Rounds of MD4 are Not One-Way. In S. Vaudenay, editor, *Fast Software Encryption – 1998*, volume 1372 of *Lectures Notes in Computer Science*, pages 284–292. Springer, 1998.
6. A. Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In *Advances in Cryptology – CRYPTO'04*, To appear.

7. B. Kaliski. RFC 1319 - The MD2 Message-Digest Algorithm, april 1992. RSA Laboratories.
8. MD5CRK, a new distributed computing project. See `http://www.md5crk.com/`.
9. A. Menezes, P. van 0orschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
10. B. Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit Leuven, 1993.
11. R. Rivest. The MD4 Message Digest Algorithm. In A. Menezes and S. Vanstone, editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lectures Notes in Computer Science*, pages 303–311. Springer, 1991.
12. R. Rivest. RFC 1321 - The MD5 Message-Digest Algorithm, april 1992. RSA Laboratories.
13. P. Rogaway and T. Shrimpton. Cryptographic Hash Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In B. Roy and W. Meier, editors, *Fast Software Encryption – 2004*, pages 349–366, 2004. Pre-proceedings Version.
14. N. Rogier and P. Chauvaud. MD2 Is not Secure without the Checksum Byte. *Designs, Codes and Cryptography*, 12(3):245–251, november 1997. An early version of this paper was presented at the 2nd SAC Workshop in 1995.
15. B. Van Rompay, A. Biryukov, and B. Preneel. Cryptanalysis of 3-Pass HAVAL. In C. Laih, editor, *Advances in Cryptology – ASIACRYPT'03*, volume 2894 of *Lectures Notes in Computer Science*, pages 228–245. Springer, 2003.
16. RSA Laboratories. PKCS #1 v1.5 : RSA Encryption Standard, 1993. Available at `http://www.rsalabs.com/pkcs/pkcs-1`.
17. RSA Laboratories. PKCS #1 v2.1 : RSA Encryption Standard, 2002. Available at `http://www.rsalabs.com/pkcs/pkcs-1`.
18. P. van Oorschot and M. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12(1):1–28, 1999.
19. D. Wagner. A Generalized Birthday Problem. In M. Yung, editor, *Advances in Cryptology – Crypto'02*, volume 2442 of *Lectures Notes in Computer Science*, pages 288–303. Springer, 2002. Extended Abstract.
20. A. Wiemers. The Full Cost of Cryptanalytic Attacks. *Journal of Cryptology*, 17(2):105–124, March 2004.

A A memory-efficient Attack

The attack described in Section 4.2 is much faster than an exhaustive search, however the large memory requirements make it highly unpractical and probably contributes to under-estimate the "real" complexity. Here, we propose an improved attack regarding the data complexity.

The general idea of the attack of Section 4.2 is to split the target M_i in two halves $(m_0, \ldots, m_7)$ and $(m_8, \ldots, m_{15})$ of 64 bits each. The improved attack consists in **splitting M_i in 4 parts instead of** 2 using the following algorithm :

– Guess the 6 bytes $\{(B_7^1, B_7^2), (B_{15}^1, B_{15}^2), (C_7^1, C_7^2)\}$
  - guess the 4 bytes $m_0, \ldots, m_3$
    - ∗ compute and store in table T_1 the bytes $B_3^1, B_3^2, C_3^1, C_3^2$
  - guess the 4 bytes $m_4, \ldots, m_7$

  * $*$ compute and store in table T_2 the bytes $B_3^1, B_3^2, C_3^1, C_3^2$
- guess the 4 bytes $m_8, \ldots, m_{11}$
  * $*$ compute and store in table T_3 the bytes $B_{11}^1, B_{11}^2, C_{11}^1, C_{11}^2$
- guess the 4 bytes $m_{12}, \ldots, m_{15}$
  * $*$ compute and store in table T_4 the bytes $B_{11}^1, B_{11}^2, C_{11}^1, C_{11}^2$
- Compute the joint product $T = T_1 \bowtie T_2$ of size 2^{32}. It contains candidate values for $(m_0, \ldots, m_7)$.
- Compute the joint product $T' = T_3 \bowtie T_4$ of size 2^{32}. It contains candidate values for $(m_8, \ldots, m_{15})$.
- Guess 2 additional bytes B_{15}^3 and B_{15}^4
  * $*$ For each element of T compute the 4 bytes $B_7^3, B_7^4, C_7^3, C_7^4$
  * $*$ Compute similarly these 4 bytes for each element of T'
  * $*$ Search for a collision in the two resulting lists.
- This results in a list of 2^{32} candidates for $(m_0, \ldots, m_{15})$.

This slightly more complex attack has complexity of the order of

$$2^8 \times 2^{48} \times 2^{16} \times 2^{32} \simeq 2^{104}$$

instructions, like previously. However the largest tables we handle have 2^{32} entries of 32 bits. The philosophy of this improved attack is described in Figure 9.

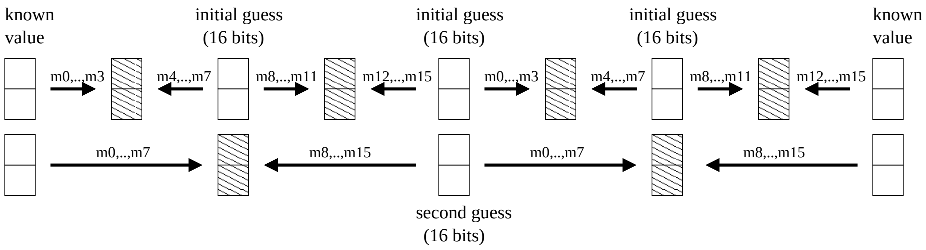

Fig. 9. The general philosophy of the improved attack