MD4 is Not One-Way

Gaëtan Leurent

École Normale Supérieure – Département d'Informatique,
45 rue d'Ulm, 75230 Paris Cedex 05, France
Gaetan.Leurent@ens.fr

Abstract. MD4 is a hash function introduced by Rivest in 1990. It is still used in some contexts, and the most commonly used hash functions (MD5, SHA-1, SHA-2) are based on the design principles of MD4. MD4 has been extensively studied and very efficient collision attacks are known, but it is still believed to be a one-way function.
In this paper we show a partial pseudo-preimage attack on the compression function of MD4, using some ideas from previous cryptanalysis of MD4. We can choose 64 bits of the output for the cost of 2^{32} compression function computations (the remaining bits are randomly chosen by the preimage algorithm).
This gives a preimage attack on the compression function of MD4 with complexity 2^{96}, and we extend it to an attack on the full MD4 with complexity 2^{102}. As far as we know this is the first preimage attack on a member of the MD4 family.
Key words: MD4, hash function, cryptanalysis, preimage, one-way.

1 Introduction

Hash functions are fundamental cryptographic primitives used in many constructions and protocols. A hash function takes a bitstring of arbitrary length as input, and outputs a digest, a small bitstring of fixed length n.

$$F : \{0,1\}^* \mapsto \{0,1\}^n$$

When used in a cryptographic context, we expect a hash function to behave somewhat like a random oracle. The digest is used as a kind of fingerprint: it can be used to test whether two documents are equal, but should neither reveal any other information about the input nor be malleable. More concretely, we ask a cryptographic hash function to resist three major attacks:

Collision: Given F, find $M_1 \neq M_2$ s.t. $F(M_1) = F(M_2)$.
Second-preimage: Given F and M_1, find $M_2 \neq M_1$ s.t. $F(M_1) = F(M_2)$.
Preimage: Given F and $\overline{H}$, find M s.t. $F(M) = \overline{H}$.

Due to the birthday paradox, we have a generic collision attack with complexity $2^{n/2}$, while brute force preimage or second-preimage attacks have complexity 2^n: this defines the security requirements of a n-bit hash function. Collision resistance is the strongest notion, so most constructions use a collision resistant hash function, and most cryptanalysis target collision attack. For a more formal definition of these properties, and the relations between them, see [18,17].

Unfortunately, many currently used hash functions have been broken by collision attacks: MD4 [5,21,19] (the best attack has complexity 2^1), MD5 [23,10] (best attack: 2^{23}), and SHA-1 [22,13] (best attack: 2^{60}). These functions are now considered unsafe but in practice very few constructions or protocols are really affected.

In this paper we consider preimage resistance, which is a weaker security notion and is still believed to hold for these hash functions. In particular MD4 is broken by collision attacks since 1996 but it is still used in some applications where speed is important and/or a one-way function is needed but collision resistance is not important:

- to "encrypt" passwords in Windows NT and later (as the NTLM hash);
- for password derivation in the S/KEY one time password system [8];
- to compare file blocks in the incremental file transfer program rsync;
- for file identification and integrity in the eDonkey peer-to-peer network.

S/Key and rsync even use a truncated MD4 and rely on the partial one-wayness of MD4.

Preimage attacks are rather rare in the world of hash function cryptanalysis; the most notable example is the preimage attack against MD2 by Muller [15], later improved by Knudsen and Mathiassen [11] which has a complexity of 2^{97}. A preimage attack has much more impact than a collision attack: it can be used to fool integrity checks, to forge signatures using only known messages, to break "encrypted" password files,... Moreover, when the hash function follows the Merkle-Damgård paradigm (this is the case for MD4) we can add any chosen prefix: given a message M and a target hash value $\overline{H}$, we can actually compute N such that $\mathrm{MD4}(M||N) = \overline{H}$. For instance, this can be used to create a malicious software package with a given signature when trailing garbage is allowed (*eg.* this is the case with zip, gzip, and bzip2 files).

1.1 Our results

Our main result is a preimage attack against MD4 with complexity 2^{102}. This attack uses messages of 18 blocks or slightly more (more precisely 9151 bits, about one kilobyte), and we can add any chosen prefix.

This is based on a partial pseudo-preimage attack on the compression function: we can choose 64 bits of the output (the other bits being randomly chosen by the preimage algorithm) and 32 bits of the input for the cost of 2^{32} compression function (brute force would require 2^{64}).

Our attack uses many ideas from previous cryptanalysis of MD4 [5,20,6,21]. We consider MD4 as a system of equation, we use some kind of differential path and use the Boolean functions to absorb some differences, we fix many values of the internal state using some particularities of the message expansion.

1.2 Related work

MD4 has been introduced as a cryptographic hash function by Rivest [16], in 1990 and many cryptanalytic effort has been devoted to study its security. The design principles of MD4 are used in MD5 and the SHA family, which are the most widely used hash function today. Any result about MD4 is interesting by itself, and also gives some insight to the security level of the other members of the MD4 family.

Shortly after the introduction of MD4, collision attacks were found on reduced variants of MD4: den Boer and Bosselaers [4] found an attack against the last two rounds, and Merkle had an unpublished attack against the first two rounds. Another attack against the first two rounds was later found by Vaudenay [20]. The first collision attack against the full MD4 is due to Dobbertin [5] in 1996. More recently, Wang *et. al.* found a very efficient collision attack on MD4 [21], which was later improved by Sasaki *et. al.* [19] and only costs 2 compression functions. Due to all these attacks MD4 is no longer used as a collision-resistant hash function.

The main result concerning the one-wayness of MD4 is due to Dobbertin [6]. He showed that if the last round of MD4 is removed, preimages can be found in the resulting hash function with a complexity of 2^{32} compression function calls. This work was studied and improved using SAT solvers by De *et al.* [2]. They managed to invert up to 2 rounds and 7 steps of MD4. To the best of our knowledge, no preimage attack has been found on the full MD4 with three rounds.

Recently, Yu *et al.* found a kind of second-preimage attack on MD4 [24]. However this kind of attack is not what we usually call a second-preimage attack because it only works for a small subset of the message space. This attack has a complexity of one compression function, but it works only with probability 2^{-56} and cannot be repeated when it fails. If we want to build an attack that works for any message out of this, we will use a brute-force search when the attack fails: it will have a workload of 1 with probability 2^{-56}, and a workload of $1 + 2^{128}$ with probability $1 - 2^{-56}$; the average workload is still extremely close to 2^{128}. More interestingly, we can use this with long messages: if the message is made of 2^{63} blocks (there is not limitation to the size of the message in MD4, as opposed to SHA-1), we will be able to find a second-preimage for at least one of the blocks with a probability of $1 - \exp(-2^{63-56}) \approx 1 - 2^{-184}$. Thus, the cases where we have to run a brute-force search become negligible, and the average workload is just the time needed to test each block until a good one is found, so we expect it to be 2^{56}.

Another related work due to Kelsey and Schneier [9] introduced a generic second-preimage attack against iterated hash functions using long messages. This is a nice result showing the limitations of the Merkle-Damgård paradigm, but an attack on messages of 2^{64} bits is not really practical. Our attack typically uses messages of 20 blocks (about 1 kilobyte in total).

1.3 Description of MD4

MD4 is an iterated hash function following the Merkle-Damgård paradigm. The message is padded and cut into blocks of k bits (with $k = 512$ for MD4), and the digest is computed by iterating a compression function cF, starting with an initial value IV.

$$\begin{gathered} cF : \{0,1\}^{n+k} \mapsto \{0,1\}^n \\ h_0 = IV, \qquad h_{i+1} = cF(h_i, M_i) \\ F(M_0, M_1, ...M_{p-1}) = h_p \end{gathered}$$

The padding of MD4 uses the MD strengthening: it is designed to be invertible, and includes the size of the message. The message is first padded with a single `1` bit followed by a variable number of `0`'s, so that the size of the message is congruent to 448 modulo 512. This first step adds between 1 and 512 bits to the message. Then the last 64 bits are filled with the size of the original message modulo 2^{64}. Note that MD4 can hash any bitstring: it is not restricted to hash bytes, and there is no limit to the size of the message.

We will use the following definitions for attacks on the compression function cF:

Pseudo-Preimage: Given cF and $\overline{H}$, find IV, M s.t. $cF(IV, M) = \overline{H}$.
Preimage: Given cF, IV and $\overline{H}$, find M s.t. $cF(IV, M) = \overline{H}$.

The compression function of MD4 is an unbalanced Feistel ladder with an internal state of four 32-bit registers. It is made of 48 steps, where each step updates one of these registers. The 48 steps are divided into 3 rounds of 16 steps; each round reads the 16 message words in a different order (this is a very simple message expansion). To better describe our attack, we will assign the name Q_i to the value computed in the step i: we now have 48 internal state variables, and each one is computed from the 4 preceding ones (we use Q_{-4} to Q_{-1} to denote the IV):

Step update:	$Q_i = (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_{\pi(i)} \boxplus k_i) \lll s_i$
Input:	$Q_{-4} \;\|\|\; Q_{-1} \;\|\|\; Q_{-2} \;\|\|\; Q_{-3}$
Output:	$Q_{-4} \boxplus Q_{44} \;\|\|\; Q_{-1} \boxplus Q_{47} \;\|\|\; Q_{-2} \boxplus Q_{46} \;\|\|\; Q_{-3} \boxplus Q_{45}$

First round:	$0 \le i < 16$	$\Phi_i = \mathrm{IF}$	$k_i = K_0 = 0$
Second round:	$16 \le i < 32$	$\Phi_i = \mathrm{MAJ}$	$k_i = K_1 =$ `0x5a827999`
Third round:	$32 \le i < 48$	$\Phi_i = \mathrm{XOR}$	$k_i = K_2 =$ `0x6ed9eba1`

π(0...15):	0,	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11,	12,	13,	14,	15
π(16...31):	0,	4,	8,	12,	1,	5,	9,	13,	2,	6,	10,	14,	3,	7,	11,	15
π(32...47):	0,	8,	4,	12,	2,	10,	6,	14,	1,	9,	5,	13,	3,	11,	7,	15

We will use the fact that we can fix some values of the internal state instead of only fixing values of the message words: since the step update function is invertible, when Q_{i-1}, Q_{i-2} and Q_{i-3} are known, we can compute any one of Q_i, Q_{i-4} or m_i from the two others (see Algorithm 1 for explicit formulas).

Algorithm 1 Step functions

---

function MD4StepForward(i)
    $Q_i \leftarrow (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_{\pi(i)} \boxplus k_i) \lll s_i$
end function
function MD4StepBackward(i)
    $Q_{i-4} \leftarrow (Q_i \ggg s_i) \boxminus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxminus m_{\pi(i)} \boxminus k_i$
end function
function MD4StepMessage(i)
    $m_{\pi(i)} \leftarrow (Q_i \ggg s_i) \boxminus Q_{i-4} \boxminus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxminus k_i$
end function

In our attack we consider MD4 as a big system of equations over the variables Q_{-4} to Q_{47} and m_0 to m_{15} (we consider only words, and never look at individual bits); we have 48 step update equations, and 4 equations for the output value. We consider the input chaining variables IV_0, IV_1, IV_2, IV_3 as free: this makes the attack on the compression function easier (it's a pseudo-preimage attack), but we will have some work to do to turn it into a preimage attack on the full hash function. We use $\overline{X}$ to denote the desired value of a variable X, which is given as an input to our attack. The system we are trying to solve can be written as:

$$\begin{cases} Q_i = (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_{\pi(i)} \boxplus k_i) \lll s_i \quad \text{for } i \in \{0...47\} \\ H_0 = Q_{-4} \boxplus Q_{44} = \overline{H}_0 \\ H_1 = Q_{-3} \boxplus Q_{45} = \overline{H}_1 \\ H_2 = Q_{-2} \boxplus Q_{46} = \overline{H}_2 \\ H_3 = Q_{-1} \boxplus Q_{47} = \overline{H}_3 \end{cases}$$

To make the equations more readable, we will use a grey background to show the variables whose value has already been chosen in a previous step of the algorithm.

We will use $x^{[k]}$ to represent the $(k+1)$-th bit of x, that is $x^{[k]} = (x \ggg k) \bmod 2$ (note that we count bits and steps starting from 0). We also use $\mathbb{0}$ and $\mathbb{1}$ to denote the 32-bit words whose bits are all zeroes and all ones (*i.e.* $\mathbb{0} = \texttt{0x00000000}$ and $\mathbb{1} = \texttt{0xffffffff}$).

1.4 Road map

First we will describe our attack against the compression function cMD4. Then we will show how to extend it to an attack against the full MD4, with a better complexity than the generic attack.

2 Pseudo-Preimage on the Compression Function

The general idea of the attack comes from a simple observation: we know that MD4 is very sensitive to differential attacks — there is a collision attack that costs less than 2 calls to the compression function [19]. However, in a preimage attack, we are looking for a single message and differential tools seem unsuitable for this task. Our idea is to start with an *initial message* (IV, M) with very specific properties, and the digest cMD4(IV, M) of this message. Now we will use differential paths to create a family of *related messages* $(IV^{(i)}, M^{(i)})$ so that the computation of cMD4$(IV^{(i)}, M^{(i)})$ differs from the computation of cMD4(IV, M) in a controlled way. This will allow us to choose a particular $(IV^{(i_0)}, M^{(i_0)})$ so that some bits of cMD4$(IV^{(i_0)}, M^{(i_0)})$ agree with a target value. Alternatively, we could try to mount a second preimage attack using differential tools, as done in [24], but this will probably only work for a small fraction of the message space, because differential paths *à la* Wang impose many constraints on the message.

Following this idea, we look for a set of constraints on the initial message and a way to derive the related messages. We managed to easily select a related message that agrees with 32 bits of the target hash. We can compute 2^{32} good related messages for a cost of 2^{32} compression function calls: our attack has an amortized cost of 1. When we run it 2^{32} times, we will test 32 more bits of the target hash, and we expect to find a partial pseudo-preimage on 64 bits. Similarly, to find a full pseudo-preimage we expect to repeat it 2^{96} times.

2.1 The Initial Message

The key part is to choose a set of constraints that allow to place many differential paths in MD4, in order to have as many related messages as possible. Instead of looking at single bits, we will consider the 32-bit words of MD4, and try to have 32 paths at once. We will use some properties of the Boolean functions in MD4, and some properties of the message expansion.

The Boolean functions used in the first and second round of MD4 have the nice property to be able to absorb some difference. This key property was used in early cryptanalysis of MD4 [20,6] and is the starting point of the construction of differential paths [7] *à la* Wang. Notably, we have the following properties, where $\mathbb{C}$ is a constant, and x is variable:

Absorb 1$^{\text{st}}$ input	Absorb 2$^{\text{nd}}$ input	Absorb 3$^{\text{rd}}$ input
$\text{IF}(x, \mathbb{C}, \mathbb{C}) = \mathbb{C}$	$\text{IF}(\mathbb{0}, x, \mathbb{C}) = \mathbb{C}$	$\text{IF}(\mathbb{1}, \mathbb{C}, x) = \mathbb{C}$
$\text{MAJ}(x, \mathbb{C}, \mathbb{C}) = \mathbb{C}$	$\text{MAJ}(\mathbb{C}, x, \mathbb{C}) = \mathbb{C}$	$\text{MAJ}(\mathbb{C}, \mathbb{C}, x) = \mathbb{C}$

This can be used to let one message word be free in the first round or in the second round, as shown by Figure 1. If we fix some internal state variables, a change of m_i will only change Q_i, Q_{i+4}, ... Q_{i+4k} and can be corrected by a change in m_{i+4k}. Additionally, we are free to change m_{i+4},m_{i+8},... $m_{i+4(k-1)}$ without breaking any extra internal state.

To choose the step i_0 where we introduce the difference, and i_1 where we cancel it, we will look at the message expansion. We want i_0 and i_1 to be quite far away so as to skip a big part of the compression function and to maximize the number of free message words in between, but we do not want to use the same message word twice between i_0 and i_1. This leave us with 3 possibilities:

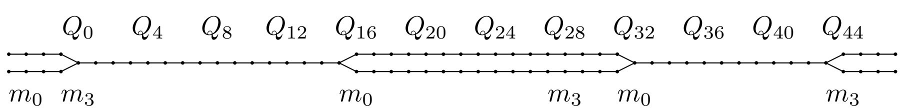
1st round: m_4 is free		2nd round: m_{20} is free	
$Q_2 = \mathbb{1}$		$Q_{18} = \mathbb{C}$	
$Q_3 = \mathbb{1}$		$Q_{19} = \mathbb{C}$	
Q_4		Q_{20}	
$Q_5 = \mathbb{0}$	$\mathrm{IF}(Q_4, Q_3, Q_2) = \mathbb{1}$	$Q_{21} = \mathbb{C}$	$\mathrm{MAJ}(Q_{20}, Q_{19}, Q_{18}) = \mathbb{C}$
$Q_6 = \mathbb{1}$	$\mathrm{IF}(Q_5, Q_4, Q_3) = \mathbb{1}$	$Q_{22} = \mathbb{C}$	$\mathrm{MAJ}(Q_{21}, Q_{20}, Q_{19}) = \mathbb{C}$
$Q_7 = \mathbb{1}$	$\mathrm{IF}(Q_6, Q_5, Q_4) = \mathbb{0}$	$Q_{23} = \mathbb{C}$	$\mathrm{MAJ}(Q_{22}, Q_{21}, Q_{20}) = \mathbb{C}$
Q_8	$\mathrm{IF}(Q_7, Q_6, Q_5) = \mathbb{1}$	Q_{24}	$\mathrm{MAJ}(Q_{23}, Q_{22}, Q_{21}) = \mathbb{C}$
$Q_9 = \mathbb{0}$	$\mathrm{IF}(Q_8, Q_7, Q_6) = \mathbb{1}$	$Q_{25} = \mathbb{C}$	$\mathrm{MAJ}(Q_{24}, Q_{23}, Q_{22}) = \mathbb{C}$
$Q_{10} = \mathbb{1}$	$\mathrm{IF}(Q_9, Q_8, Q_7) = \mathbb{1}$	$Q_{26} = \mathbb{C}$	$\mathrm{MAJ}(Q_{25}, Q_{24}, Q_{23}) = \mathbb{C}$
$Q_{11} = \mathbb{1}$	$\dots$	$Q_{27} = \mathbb{C}$	$\dots$

Fig. 1. Absorption of a difference. The step update function is:
$Q_i = (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_{\pi(i)} \boxplus k_i) \lll s_i$

- $(i_0, i_1) = (\ 0, 16)$: m_0 , m_4 , m_8 and m_{12} are free
- $(i_0, i_1) = (15, 31)$: m_{15}, m_{12}, m_{13} and m_{14} are free
- $(i_0, i_1) = (16, 32)$: m_0 , m_1 , m_2 and m_3 are free

Note that Vaudenay [20] and Dobbertin [6] used the same idea in their attacks, with $(i_0, i_1) = (15, 31)$. Here we choose $(i_0, i_1) = (16, 32)$ because the free message words are used in the very beginning of the first round. To fix the first round, we will correct a modification of the free message words using the IV, and this correction will only involve the first 4 steps of the compression function.

We now have a very good differential path with m_0 and m_3: if we consider the set of 2^{32} pairs that keep Q_{32} constant, their effect on the final hash only involves the first 4 steps and the last 4 steps of MD4. The other free variables will be used to simplify the equations so as to make this path easier to use. Schematically, the differential path looks like this:



We can now choose what will be fixed by the initial message and what will be free for the related messages. The message words m_4 to m_{15} will be fixed by the initial message, while m_0 to m_3 are part of the related message. The internal state variables $Q_{14}, Q_{15}, Q_{17}, Q_{18}, Q_{19}, Q_{21}, ...Q_{30}$ need to be equal and will be part of the initial message. Q_{13} is in the initial message because it is fixed by the step 17, and similarly Q_{31} is fixed by step 31. We will add Q_{12} in the initial message to fix the internal state of the first round. See Figure 2 for a graphical representation.

To select an initial message, we choose random values for $\mathbb{C}$, Q_{12}, Q_{13} and m_{15}; this allows us to compute Q_{31} and m_4 to m_{14} in the second round, and Q_0 to Q_{11} in the first round. Thus we can build 2^{128} different initial messages.

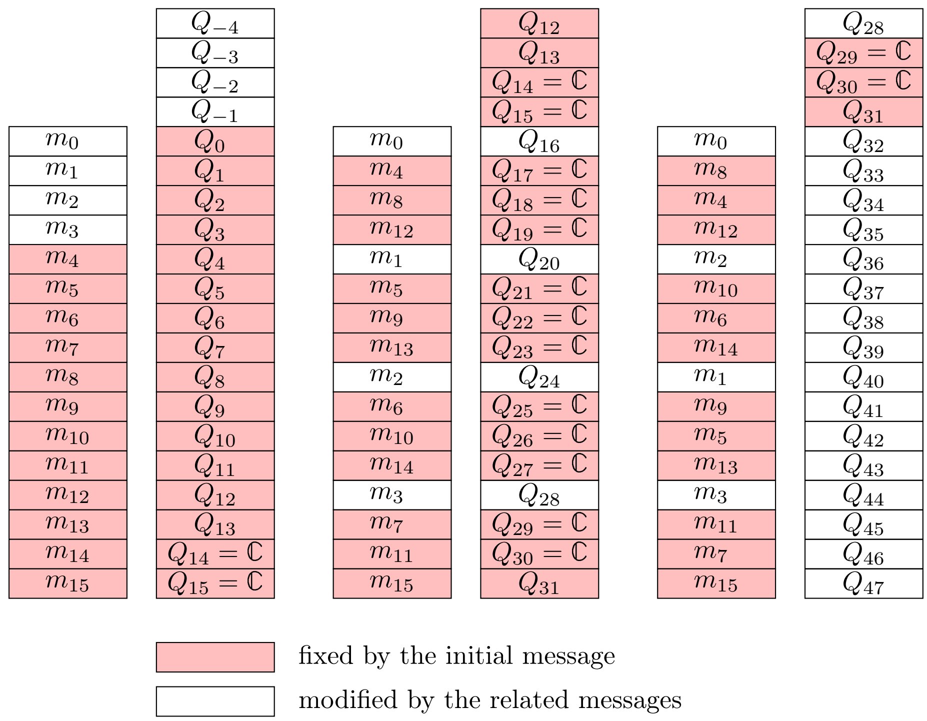


Fig. 2. Initial message and related message

Each initial message have 2^{128} related messages (by choosing the value of m_0, m_1, m_2, m_3).

2.2 The Related Messages

When an initial message is fixed, we have to choose m_0, m_1, m_2 and m_3 in a way that will give us some control on the hash value. We will first isolate the third round from the second round, by choosing the value of Q_{32}. Then the choice of m_2, m_1 and m_3 will give the final state $Q_{44}, ...Q_{47}$, and m_0 will be chosen last: we expect that one value of m_0 will be consistent with the choice of Q_{32}. Note that since m_0 is used in step 0, we can compute H_1, H_2 and H_3 without knowing m_0. Thus, we can choose a good value of m_2, m_1 and m_3 by looking only at the last round and the first 4 steps of the first round, and compute m_0 later in order to correct the second round.

We now study the first steps and the last steps. Our goal is to find efficiently a value of m_2, m_1, and m_3 that solves some of the equations. The message words m_2 and m_1 are used quite far for the last steps, so it will be hard to study how they affect the final state $Q_{44}, ...Q_{47}$: we will only use m_3 to control the hash, while m_2 and m_1 will be used to simplify the equations.

First Steps. We assume that an initial message has been chosen. Let us first study the initial steps of MD4:

$$\begin{aligned}
Q_0 &= (Q_{-4} \boxplus \mathrm{IF}(Q_{-1}, Q_{-2}, Q_{-3}) \boxplus m_0) \lll 3 && (1)\\
Q_1 &= (Q_{-3} \boxplus \mathrm{IF}(Q_0, Q_{-1}, Q_{-2}) \boxplus m_1) \lll 7 && (2)\\
Q_2 &= (Q_{-2} \boxplus \mathrm{IF}(Q_1, Q_0, Q_{-1}) \boxplus m_2) \lll 11 && (3)\\
Q_3 &= (Q_{-1} \boxplus \mathrm{IF}(Q_2, Q_1, Q_0) \boxplus m_3) \lll 19 && (4)
\end{aligned}$$

Equation (4) shows that Q_{-1} is completely determined by m_3. Additionally, we will ask that $Q_1 = \mathbb{1}$: this simplify Equation (3) and make Q_{-2} completely determined by m_2, independently of Q_{-1}:

$$Q_2 = (Q_{-2} \boxplus Q_0 \boxplus m_2) \lll 11 \qquad (3')$$

Last Steps. Let us now study the final steps of MD4. We will assume that a value has been chosen for Q_{32}, m_2, m_1: we can now compute Q_{32} to Q_{43} in the third round, and Q_{-2} by equation (3'). This gives us $Q_{46} = \overline{H}_2 - Q_{-2}$.

$$\begin{aligned}
Q_{44} &= (Q_{40} \boxplus \mathrm{XOR}(Q_{43}, Q_{42}, Q_{41}) \boxplus m_3 \boxplus K_2) \lll 3 && (5)\\
Q_{45} &= (Q_{41} \boxplus \mathrm{XOR}(Q_{44}, Q_{43}, Q_{42}) \boxplus m_{11} \boxplus K_2) \lll 9 && (6)\\
Q_{46} &= (Q_{42} \boxplus \mathrm{XOR}(Q_{45}, Q_{44}, Q_{43}) \boxplus m_7 \boxplus K_2) \lll 11 && (7)\\
Q_{47} &= (Q_{43} \boxplus \mathrm{XOR}(Q_{46}, Q_{45}, Q_{44}) \boxplus m_{15} \boxplus K_2) \lll 15 && (8)
\end{aligned}$$

Here we see that (7) gives the value $Q_{44} \oplus Q_{45}$. Moreover, we will ask that $Q_{41} \boxplus m_{11} \boxplus K_2 = \mathbb{0}$ so as to simplify (6). We let V be $Q_{42} \oplus Q_{43} \oplus Q_{44} \oplus Q_{45}$, which is a known constant, and equation (6) becomes:

$$\begin{aligned}
Q_{45} &= \mathrm{XOR}(Q_{44}, Q_{43}, Q_{42}) \lll 9 &&\\
Q_{45} &= (Q_{45} \oplus V) \lll 9 && (6')
\end{aligned}$$

This last equation is actually a system of linear equations over the bits of Q_{45}; it is easy to check whether it is satisfiable, and to compute the solutions (see Appendix B for an optimization). From Q_{45}, we compute Q_{44} by (6) and m_3 by (5), and we know that this particular choice of m_3 will give the right value for Q_{46}, and we will have $H_2 = \overline{H}_2$.

Simplifications. We have introduced two extra conditions to simplify the equations:

$$\begin{aligned}
Q_1 &= \mathbb{1} && (C_1)\\
Q_{41} \boxplus m_{11} \boxplus K_2 &= \mathbb{0}\,. && (C_2)
\end{aligned}$$

Algorithm 2 Partial Pseudo Preimage

Input: $\overline{H}_0, \overline{H}_2, \overline{IV}_2$
Output: M, IV st. $H_0 = \overline{H}_0$, $H_2 = \overline{H}_2$ and $IV_2 = \overline{IV}_2$
Running Time: 2^{32}

loop ▷ *Expect 1 iteration*
  Choose an initial message with $Q_1 = \mathbb{1}$ ▷ 2^{96} *possibilities*
  for all Q_{32} **do** ▷ 2^{32} *iterations*
    Compute Q_{33}, Q_{34}, Q_{35}.
    Choose m_2 so that $Q_{-2} = \overline{IV}_2$. ▷ Q_{-2} *is* IV_2
    Compute $Q_{36}, Q_{37}, Q_{38}, Q_{39}$.
    Choose m_1 so that $Q_{41} = -m_{11} - K_2$.
    Compute $Q_{40}, Q_{41}, Q_{42}, Q_{43}$.
    Choose m_3 so that $Q_{46} = \overline{H}_2 \boxminus Q_{-2}$. ▷ $Q_{46} \boxplus Q_{-2}$ *is* H_2
    Compute $Q_{44}, Q_{45}, Q_{46}, Q_{47}$, and Q_{-1}, Q_{-2}, Q_{-3}.
    Choose m_0 so that $Q_{-4} = \overline{H}_0 \boxminus Q_{44}$. ▷ $Q_{44} \boxplus Q_{-4}$ *is* H_0
    if m_0 matches Q_{32} **then** ▷ *OK with prob.* 2^{-32}
      return
    end if
  end for
end loop

(C_1) can only be satisfied statistically, by computing about 2^{32} initial messages and keeping the good ones, but this cost can be amortized over the many choices of Q_{32}, m_2, and m_1. On the other hand, (C_2) can be satisfied by choosing an appropriate m_1 when Q_{32} and m_2 have been chosen:

$$Q_{40} = (Q_{36} \boxplus \mathrm{XOR}(Q_{39}, Q_{38}, Q_{37}) \boxplus m_1 \boxplus K_2) \lll 3 \tag{9}$$
$$Q_{41} = (Q_{37} \boxplus \mathrm{XOR}(Q_{40}, Q_{39}, Q_{38}) \boxplus m_9 \boxplus K_2) \lll 9 \tag{10}$$

The choice of Q_{41} gives Q_{40} by (10), which gives m_1 by (9). Conversely, with this choice of m_1 (C_2) will be satisfied. Every initial message can now be used with 2^{64} choices of Q_{32} and m_2, so we still have some extra degree of freedom: we can use the freedom of m_2 to choose the value of Q_{-2}. In the end we can choose both Q_{-2} and Q_{46} (hence H_2) for an amortized cost of one compression function.

Partial pseudo-preimage. When we put this all together, we can compute m_3 so that $H_2 = \overline{H}_2$ for an amortized cost of 1 compression function, and we can also choose IV_2. The full algorithm, given in Algorithm 2, is a partial pseudo-preimage attack, which is 2^{32} times more efficient than exhaustive search. It should be repeated about 2^{64} times to find a full pseudo-preimage, and we have enough different initial messages for that. Note that Algorithm 2 finds pseudo-preimages on (H_0, H_2), but if we change a little bit the end of the algorithm we can have pseudo-preimages on (H_1, H_2) or (H_2, H_3) just as easily. See Appendix A for an example of a partial pseudo-preimage.

3 Preimage of the full MD4

To extend this attack to the full MD4, we will use an idea similar to the unbalanced meet-in-the-middle attack of Lai and Massey [12]. We compute many pseudo-preimages of $\overline{H}$, we hash many random messages, and we use the birthday paradox to meet in the middle. If we have a pseudo-preimage attack with complexity 2^s, the generic attack uses the pseudo-preimage attack $2^{(n-s)/2}$ times starting from the target digest $\overline{H}$ (we assume there is no problem with the padding), and hashes $2^{(n+s)/2}$ random blocks, starting from the standard IV. Thanks to the birthday paradox, we expect one match. This gives a preimage attack with complexity $2^{1+(n+s)/2}$. In our case this would be 2^{113} (we have $s = 96$), but we will show how to use some specific properties of our pseudo-preimage attack to build a preimage attack with complexity 2^{102}.

3.1 The Padding

First, we need to handle the padding in the last block. When looking for a padded message of b blocks, we will use a message length of $512b - 65$ bits. The last block is correctly padded if and only if $m_{15} = 0$, $m_{14} = 512b - 65$, and $m_{13}^{[0]} = \mathtt{1}$. The condition on m_{15} is easy to satisfy since we can choose m_{15} in the initial message, and on the other hand m_{14} depends only on $\mathbb{C}$:

$$
\begin{aligned}
Q_{27} &= (Q_{23} \boxplus \mathrm{MAJ}(Q_{26}, Q_{25}, Q_{24}) \boxplus m_{14} \boxplus k_{27}) \lll 13 \\
\mathbb{C} &= (\mathbb{C} \boxplus \mathbb{C} \boxplus m_{14} \boxplus K_1) \lll 13 \\
m_{14} &= \mathbb{C} \ggg 13 \boxminus \mathbb{C} \boxminus \mathbb{C} \boxminus K_1
\end{aligned}
$$

Thus, we just run an exhaustive search over $\mathbb{C}$, and we expect to find one value that gives the correct m_{14}. Similarly, we have $m_{13} = \mathbb{C} \ggg 13 \boxminus \mathbb{C} \boxminus \mathbb{C} \boxminus K_1 = m_{14}$, so the condition $m_{13}^{[0]} = 1$ will be satisfied.

Note that we can not set a size that is a multiple of 8 this way since $m_{14} = m_{13}$ is used both as the padding and as the length. If we really need to use a message made of bytes and not of bits, we can build a second-preimage attack by reusing the last block of the original message (and keeping the same padding).

When searching for the last block, we only have 2^{32} initial messages available. We will not chose the value of IV_2, but keep m_2 as a degree of freedom. Each initial message can be used to compute 2^{64} related messages with $H_2 = \overline{H}_2$. There is probability of $1 - 1/e \approx 0.63$ that at least one of these messages will give the full correct hash, so we might have to repeat this a few times. The extra freedom will come from the message length: if we change the number of blocks b, this gives us a new m_{14} and a new $\mathbb{C}$, and we can try again to find a padding block.

We start with $b = 34$ or $b = 18$ (see next section), and increase b until we find a padding block. Note that some values of b will give no suitable value of $\mathbb{C}$, but this is not a problem. Additionally, if one wants to choose a prefix for the preimage attack, one just has to start with a bigger b.

3.2 Layered Hash Tree

To improve over the basic meet-in-the-middle attack, we will use an extra property of our pseudo-preimage attack on the compression function: we need a workload of 2^s (in our case, $s = 96$) to find a pseudo-preimage of a single target value, but if have a set of k target values (with some extra conditions on the set), we can find a pseudo-preimage of *one of them* in time $2^s/k$. This is because our pseudo-preimage attack is based on the repetition of a partial pseudo-preimage attack, where the remaining bits are random. Thus, we can find $2k$ pseudo-preimages in time 2^{s+1}, and if we can also make sure that the pseudo-preimage set satisfy the extra conditions, we can iterate this operation. We will start with a set $\mathcal{H}_0$ of size 1, and after $n - s$ iterations we have a set $\mathcal{H}_{n-s}$ of size 2^{n-s}, which we use for the unbalanced meet-in-the-middle. The resulting structure is shown in figure 3. In the end, we will find a preimage in time $2(n - s)2^s + 2^s$, using a memory of size $\mathcal{O}(2^{n-s})$.

A similar idea based on multi-target pseudo-preimage was used by Mendel and Rijmen to attack HAS-V [14]. In that attack, they could run a multi-target pseudo-preimage attack on a set of size 2^s (this is not possible in our case), and this result in an attack with time complexity 2^{s+1} and a memory requirement of $\mathcal{O}(2^s)$.

Our attack against MD4 can be used as a multi-target pseudo-preimage attack following Algorithm 3, if the target set $\mathcal{H}$ satisfies the following extra properties:

- $\{\overline{H}_2 : \overline{H} \in \mathcal{H}\}$ is a singleton: a single m_3 can be used for the whole set;
- $|\mathcal{H}| \leq 2^{64}$: the loop of line 11 is negligible.

Since our algorithm allows us to choose the value of $\overline{IV}_2$ in the pseudo-preimages, we can build the pseudo-preimage set so that the extra conditions are still satisfied.

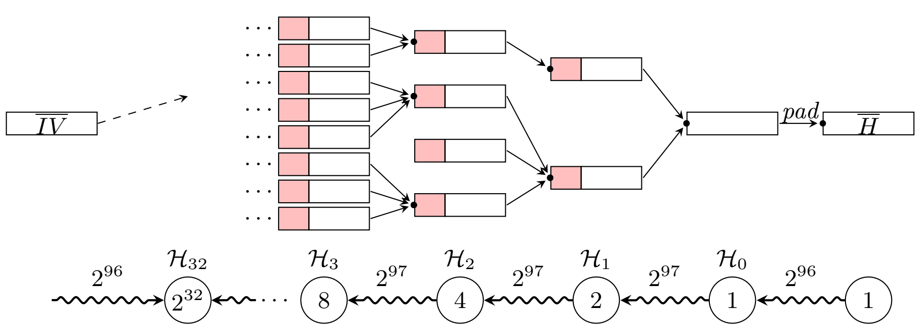


Fig. 3. Preimage attack on the full MD4 with a layered hash tree.

The tree costs $32 \times 2^{97} = 2^{102}$ to build, and gives 2^{32} pseudo-preimages of $\overline{H}$; this is significantly better than the generic attack which find 2^t pseudo-

Algorithm 3 Multi-Target Pseudo Preimage

Input: $\overline{IV}_2$, $\overline{H}_2$ and a target set $\mathcal{H}$ st. $\forall \overline{X} \in \mathcal{H}, \overline{X}_2 = \overline{H}_2$.
Output: preimage set $\mathcal{I}$ st. $\forall (IV, M) \in \mathcal{I},\ F(IV, M) \in \mathcal{H}$ and $IV_2 = \overline{IV}_2$.
Running Time: 2^{97}

while $|\mathcal{I}| < 2|\mathcal{H}|$ **do** ▷ *Expect* 2^{65} *iterations*
  Choose an initial message with $Q_1 = \mathbb{1}$ ▷ 2^{96} *possibilities*
  for all Q_{32} **do** ▷ 2^{32} *iterations*
    Compute Q_{33}, Q_{34}, Q_{35}.
    Choose m_2 so that $Q_{-2} = \overline{IV}_2$. ▷ Q_{-2} *is* IV_2
    Compute $Q_{36}, Q_{37}, Q_{38}, Q_{39}$.
    Choose m_1 so that $Q_{41} = -m_{11} - K_2$.
    Compute $Q_{40}, Q_{41}, Q_{42}, Q_{43}$.
    Choose m_3 so that $Q_{46} = \overline{H}_2 \boxminus Q_{-2}$. ▷ $Q_{46} \boxplus Q_{-2}$ *is* H_2
    Compute $Q_{44}, Q_{45}, Q_{46}, Q_{47}$, and Q_{-1}, Q_{-2}, Q_{-3}.
    for all $\overline{H} \in \mathcal{H}$ st. $H_3 = \overline{H}_3$, $H_4 = \overline{H}_4$ **do** ▷ *Expect* $2^{-64}|\mathcal{H}|$ *values*
      Choose m_0 so that $Q_{-4} = \overline{H}_0 \boxminus Q_{44}$. ▷ $Q_{44} \boxplus Q_{-4}$ *is* H_0
      if m_0 matches Q_{32} **then** ▷ *OK with prob.* 2^{-32}
        Add the solution to $\mathcal{I}$
      end if
    end for
  end for
end while

preimages in time 2^{96+t}. In the forward step, we compute the hashes of 2^{96} random messages, and we expect to find a match thanks to the birthday paradox. The full preimage search has time complexity 2^{102}, and require a memory of about 2^{33} message blocks to store the tree (we do not have to store the 2^{96} hashes in the forward step).

Tweaking the tree. We can tweak the parameters of the tree so as to slightly improve the attack. First, instead of doubling the size of the set at each iteration, we can triple it (the length of the preimage starts from 23, and the cost of the attack is about $2^{101.92}$) or quadruple it (the length of the preimage starts from 18, and the cost of the attack is about 2^{102}).

We can also replace the layered tree by another structure. We start with a set of 1 target value, and every time we find a pseudo-preimage of one element of the set, we add it to the set. The first pseudo-preimage will cost 2^{96}, the second one $2^{96}/2$, then $2^{96}/3$ and so on... the set will have size 2^{32} after an expected workload of:

$$2^{96} \sum_{k=1}^{2^{32}} \frac{1}{k} \leq 2^{96}(\ln 2^{32} + 1) \leq 2^{100.54}.$$

In this case, we do not control the length of the preimage, so we will use an expendable message [3,9] in the forward step.

Conclusion

Our attack on MD4 is still theoretical due to the high complexity, but it shows that MD4 is even weaker than we thought. Our attack relies on the absorption property of some of the Boolean functions, and exploits the message expansion. It is the first preimage attack on the full MD4 and it is much less efficient than Dobbertin's attack on a two round version [6].

We did not find any direct application of the attack on the compression function, but constructions relying on the partial one-wayness of cMD4 should be carefully analysed: our attack might be practical depending on the exact assumptions made on cMD4.

This attack reduces the security margin of other members of the MD4, but it is not a direct threat. The features introduced in later members of the family make the attack unsuitable:

- The rounds function of MD5, SHA-1, and SHA-2 have a much better diffusion that MD4 due to the summation of Q_{i-1} to compute Q_i (we can not absorb a difference);
- The number of rounds is more important;
- The message expansion in the SHA family is much harder to control.

Acknowledgement

Part of this work is supported by the Commission of the European Communities through the IST program under contract IST-2002-507932 ECRYPT, and by the French government through the Saphir RNRT project.

References

1. Cramer, R., ed.: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005)
2. De, D., Kumarasubramanian, A., Venkatesan, R.: Inversion Attacks on Secure Hash Functions Using SAT Solvers. In Marques-Silva, J., Sakallah, K.A., eds.: SAT. Volume 4501 of Lecture Notes in Computer Science., Springer (2007) 377–382
3. Dean, R.D.: Formal Aspects of Mobile Code Security. PhD thesis, Princeton University (January 1999)
4. den Boer, B., Bosselaers, A.: An Attack on the Last Two Rounds of MD4. In Feigenbaum, J., ed.: CRYPTO. Volume 576 of Lecture Notes in Computer Science., Springer (1991) 194–203
5. Dobbertin, H.: Cryptanalysis of MD4. J. Cryptology **11**(4) (1998) 253–271
6. Dobbertin, H.: The First Two Rounds of MD4 are Not One-Way. In Vaudenay, S., ed.: Fast Software Encryption. Volume 1372 of Lecture Notes in Computer Science., Springer (1998) 284–292

7. Fouque, P.A., Leurent, G., Nguyen, P.: Automatic Search of Differential Path in MD4. ECRYPT Hash Worshop – Cryptology ePrint Archive, Report 2007/206 (2007) `http://eprint.iacr.org/`.
8. Haller, N.: The S/KEY One-Time Password System. RFC 1760 (Informational) (February 1995)
9. Kelsey, J., Schneier, B.: Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. [1] 474–490
10. Klima, V.: Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105 (2006) `http://eprint.iacr.org/`.
11. Knudsen, L.R., Mathiassen, J.E.: Preimage and Collision Attacks on MD2. In Gilbert, H., Handschuh, H., eds.: FSE. Volume 3557 of Lecture Notes in Computer Science., Springer (2005) 255–267
12. Lai, X., Massey, J.L.: Hash Function Based on Block Ciphers. In: EUROCRYPT. (1992) 55–70
13. Mendel, F., Rechberger, C., Rijmen, V.: Update on SHA-1. Presented at the rump session of CRYPTO '07 `http://rump2007.cr.yp.to/`.
14. Mendel, F., Rijmen, V.: Weaknesses in the HAS-V Compression Function. In Nam, K.H., Rhee, G., eds.: ICISC. Volume 4817 of Lecture Notes in Computer Science., Springer (2007) 335–345
15. Muller, F.: The MD2 Hash Function Is Not One-Way. In Lee, P.J., ed.: ASIACRYPT. Volume 3329 of Lecture Notes in Computer Science., Springer (2004) 214–229
16. Rivest, R.L.: The MD4 Message Digest Algorithm. In Menezes, A., Vanstone, S.A., eds.: CRYPTO. Volume 537 of Lecture Notes in Computer Science., Springer (1990) 303–311
17. Rogaway, P.: Formalizing Human Ignorance. In Nguyen, P.Q., ed.: VIETCRYPT. Volume 4341 of Lecture Notes in Computer Science., Springer (2006) 211–228
18. Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Roy, B.K., Meier, W., eds.: FSE. Volume 3017 of Lecture Notes in Computer Science., Springer (2004) 371–388
19. Sasaki, Y., Wang, L., Ohta, K., Kunihiro, N.: New Message Difference for MD4. In Biryukov, A., ed.: FSE. Volume 4593 of Lecture Notes in Computer Science., Springer (2007) 329–348
20. Vaudenay, S.: On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In Preneel, B., ed.: Fast Software Encryption. Volume 1008 of Lecture Notes in Computer Science., Springer (1994) 286–297
21. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. [1] 1–18
22. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
23. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. [1] 19–35
24. Yu, H., Wang, G., Zhang, G., Wang, X.: The Second-Preimage Attack on MD4. In Desmedt, Y., Wang, H., Mu, Y., Li, Y., eds.: CANS. Volume 3810 of Lecture Notes in Computer Science., Springer (2005) 1–12

A A Partial Pseudo-Preimage of MD4

Here is an example of a partial pseudo-preimage of MD4. We ran our algorithm with $\overline{H}_1 = 0, \overline{H}_2 = 0$, and $\overline{IV}_2 = 0$. It would cost 2^{64} hash evaluations to find this message by exhaustive search, but our algorithm finds it in about 20 minutes on a desktop computer.

IV			
72 fa 31 aa	a0 6e 27 95	00 00 00 00	13 c9 dc ce
Message block			
8e 34 9e ad	6c 36 1e 1c	21 b7 0e bd	14 1e 98 d9
79 67 c3 19	d7 3c 6a 19	d7 3c 6a 19	d7 3c 6a 19
14 61 85 33	14 61 85 33	14 61 85 33	14 61 85 33
58 13 27 05	58 13 27 05	58 13 27 05	17 46 57 27
MD4			
34 5e 59 ae	c5 6a 3b 8e	00 00 00 00	00 00 00 00

The internal state variables for this message are given by:

Q_{-4}=0xaa31fa72	Q_{-3}=0xcedcc913	Q_{-2}=0x00000000	Q_{-1}=0x95276ea0
Q_0 =0x1545809d	Q_1 =0xffffffff	Q_2 =0xa1bdf692	Q_3 =0x1a9925ec
Q_4 =0xa8473548	Q_5 =0x92125811	Q_6 =0x9b4aaa1d	Q_7 =0x00a73054
Q_8 =0x6ef7f38b	Q_9 =0xa3789cab	Q_{10} =0x3de0878e	Q_{11} =0x2f9cbd24
Q_{12} =0x0ffc6391	Q_{13} =0x1e2a88f4	Q_{14} =0x1e83b396	Q_{15} =0x1e83b396
Q_{16} =0xb5062a71	Q_{17} =0x1e83b396	Q_{18} =0x1e83b396	Q_{19} =0x1e83b396
Q_{20} =0x51547062	Q_{21} =0x1e83b396	Q_{22} =0x1e83b396	Q_{23} =0x1e83b396
Q_{24} =0x3b4aa594	Q_{25} =0x1e83b396	Q_{26} =0x1e83b396	Q_{27} =0x1e83b396
Q_{28} =0x6f4786bc	Q_{29} =0x1e83b396	Q_{30} =0x1e83b396	Q_{31} =0x24db97dc
Q_{32} =0x84d9f63d	Q_{33} =0xc9a584fe	Q_{34} =0x475e7886	Q_{35} =0x508d517f
Q_{36} =0x79ca3034	Q_{37} =0x3bd701b4	Q_{38} =0x980fef11	Q_{39} =0x9784cf50
Q_{40} =0xc8f3a1b1	Q_{41} =0x5da0b34b	Q_{42} =0x5fa99919	Q_{43} =0x2d166b40
Q_{44} =0x042763c2	Q_{45} =0x312336ed	Q_{46} =0x00000000	Q_{47} =0xf913fc25

Note that MD4 uses a little-endian convention to convert a sequence of bytes to a sequence of words, and that the order of the words in the IV and in the hash is not the same as in the internal state.

B Solving the equation $x = (x \oplus V) \lll 9$

In Section 2.2 we find that Q_{45} has to be the solution of the following equation:

$$x = (x \oplus V) \lll 9. \tag{11}$$

where V is a constant that depends on the choices made on the previous steps of the algorithm. We have to solve this equation 2^{96} times for each pseudo-preimage, so we want to solve it very efficiently (it should cost less that one evaluation of cMD4).

We can write this equation as a linear system over the bits of x and V:

$$(11) \iff \begin{cases} x^{[\ 0]} = x^{[23]} \oplus V^{[23]} \\ x^{[\ 1]} = x^{[24]} \oplus V^{[24]} \\ \ldots \\ x^{[31]} = x^{[22]} \oplus V^{[22]} \end{cases}$$

And we can express each bit of x as a function of $x^{[0]}$ and V:

$$\iff \begin{cases} x^{[\ 9]} = x^{[0]} \oplus V^{[0]} \\ x^{[18]} = x^{[0]} \oplus V^{[0]} \oplus V^{[9]} \\ \ldots \\ x^{[23]} = x^{[0]} \oplus V^{[0]} \oplus V^{[9]} \oplus V^{[18]} \oplus V^{[27]} \cdots \oplus V^{[14]} \\ x^{[\ 0]} = x^{[0]} \oplus V^{[0]} \oplus V^{[9]} \oplus V^{[18]} \oplus V^{[27]} \cdots \oplus V^{[14]} \oplus V^{[23]} \end{cases}$$

The system is consistent if and only if the last equation holds, *i.e.* $\bigoplus_{i=0}^{31} V^{[i]} = 0$. In this case we have a first solution x_0 given by

$$\begin{cases} x_0^{[\ 0]} = 0 \\ x_0^{[\ 9]} = V^{[0]} \\ x_0^{[18]} = V^{[0]} \oplus V^{[9]} \\ \ldots \\ x_0^{[23]} = V^{[0]} \oplus V^{[9]} \oplus V^{[18]} \oplus V^{[27]} \cdots \oplus V^{[14]} \end{cases}$$

and a second solution $x_1 = x_0 \oplus \mathbb{1}$. Note that the expression of the bits of x_0 is linear in the bits of V: $x_0 = \varphi(V)$. We will split V into 4 bytes, $V = V_3||V_2||V_1||V_0$, and we precompute 4 tables (each one contains 256 words):

$$T_0[u] = \varphi(0||0||0||u) \quad T_1[u] = \varphi(0||0||u||0) \quad T_2[u] = \varphi(0||u||0||0) \quad T_3[u] = \varphi(u||0||0||0)$$

Then we have $x_0 = \varphi(V) = T_0[V_0] \oplus T_1[V_1] \oplus T_2[V_2] \oplus T_3[V_3]$. We can solve the equation, with only:

- the computation of the parity of the hamming weight of V;
- 4 table look-ups when there is a solution.