Preimage Attacks on 3-Pass HAVAL and Step-Reduced MD5*

Jean-Philippe Aumasson[1,**], Willi Meier[1,***], and Florian Mendel[2]

[1] FHNW, Windisch, Switzerland
[2] IAIK, Graz University of Technology, Graz, Austria

Abstract. This paper presents preimage attacks on the hash functions 3-pass HAVAL and step-reduced MD5. Introduced in 1992 and 1991 respectively, these functions underwent severe collision attacks, but no preimage attack. We describe two preimage attacks on the compression function of 3-pass HAVAL. The attacks have a complexity of about 2^{224} compression function evaluations instead of 2^{256}. We present several preimage attacks on the MD5 compression function that invert up to 47 steps (out of 64) within 2^{96} trials instead of 2^{128}. Although our attacks are not practical, they show that the security margin of 3-pass HAVAL and step-reduced MD5 with respect to preimage attacks is not as high as expected.

Keywords: cryptanalysis, hash function, preimage attack.

1 Introduction

A cryptographic hash function h maps a message M of arbitrary length to a fixed-length hash value H and has to fulfill the following security requirements:

- *Collision resistance:* it is infeasible to find two messages M and $M^\star$, with $M^\star \neq M$, such that $h(M) = h(M^\star)$.
- *Second preimage resistance:* for a given message M, it is infeasible to find a second message $M^\star \neq M$ such that $h(M) = h(M^\star)$.
- *Preimage resistance:* for a given hash value H, it is infeasible to find a message M such that $h(M) = H$.

The resistance of a hash function to collision and (second) preimage attacks depends in the first place on the length n of the hash value. Regardless of how a hash function is designed, an adversary will always be able to find preimages or second preimages after trying out about 2^n different messages. Finding collisions requires a much smaller number of trials: about $2^{n/2}$ due to the birthday paradox. A function is said to achieve *ideal security* if these bounds are guaranteed.

* The work in this paper was supported in part by the Austrian Science Fund (FWF), project no. P19863.
** Supported by the Swiss National Science Foundation, project no. 113329.
*** Supported by GEBERT RÜF STIFTUNG, project no. GRS-069/07.

R. Avanzi, L. Keliher, and F. Sica (Eds.): SAC 2008, LNCS 5381, pp. 120–135, 2009.

Recent cryptanalytic results on hash functions mainly focus on collision attacks but only few results with respect to preimages have been published to date. In this article, we analyze the preimage resistance of the hash functions MD5 and HAVAL. Both are iterated hash functions based on the Merkle-Damgård design principle. MD4 and MD5 both underwent critical collision attacks [4, 7, 8, 17, 18, 19], and hence should not be used anymore. But in practice MD5 is still widespread and remains secure for applications that do not require collision resistance. While three preimage attacks on MD4 are known [3, 5, 6], the picture is different for MD5: using a SAT-solver De et al. [3] inverted 26 (out of 64) steps of MD5, and no analytical attack is known to date. Idem for HAVAL: while several collision attacks [7, 13, 20, 21] and even a second preimage attack [9] were published, no preimage attack is known.

Independent Work. Sasaki and Aoki discovered preimage attacks on round-reordered and step-reduced MD5 [14]: their best attack with original round-ordering inverts 44 steps of the compression function within 2^{96} trials, starting at the step 3 and ending at step 46. They subsequently improved this result in a paper presented at this workshop [15].

Our Contribution. First, we invert the compression function of MD5 reduced to 45 steps by using a meet-in-the-middle approach. The attack makes about 2^{100} compression function evaluations and needs negligible memory. Second, we exploit special properties of the permutations used in the compression function to extend this attack to 47 steps (out of 64). The attack has a complexity of 2^{96} compressions and memory requirements of 2^{36} bytes. Third, we extend the attacks on the compression function to the hash function by using a meet-in-the-middle and tree-based approach. With this method we can construct preimages for MD5 reduced to 45 and 47 steps with a complexity of about 2^{106} and 2^{102} compression function evaluations and memory requirements of 2^{39} bytes.

Similar strategies can be applied to the compression function of HAVAL. We can invert the compression function of 3-pass HAVAL with a complexity of about 2^{224} compression function evaluations and memory requirements of 2^{69} bytes. We can turn the attack on the compression function into a preimage attack on the hash function with a complexity of about 2^{230} compression function evaluations and memory requirements of 2^{70} bytes.

Outline. The article is structured as follows. §2 presents two methods to invert to compression function of MD5 reduced to 45 and 47 steps. We use the same methods to invert the compression function of 3-pass HAVAL in §3. In §4, we show how the attacks on the compression function of MD5 and HAVAL can be extended to preimage attacks on the hash function, and §5 concludes.

2 Preimage Attacks on Step-Reduced MD5

This section presents two techniques to invert the MD5 compression function. The first attack on 45 steps is based on a standard meet-in-the-middle (MITM)

and requires about 2^{100} trials. The second attack inverts up to 47 steps, and exploits special properties of the message ordering. Combined with a MITM, we construct a preimage attack with complexity about 2^{96} trials. But prior to that, we provide a brief description of MD5 and illustrate the basic idea of our attacks over 32 steps.

2.1 Short Description of MD5

The MD5 compression function takes as input a 512-bit message block and a 128-bit chain value and outputs another 128-bit chain value.

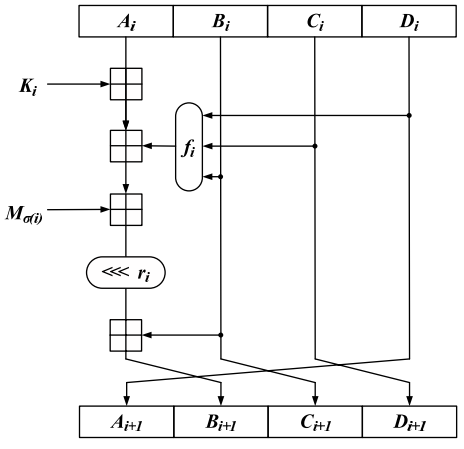

Fig. 1. The step function of MD5

The input chain value $H_0 \dots H_3$ is first copied into registers $A_0 \dots D_0$:

$$(A_0, B_0, C_0, D_0) \leftarrow (H_0, H_1, H_2, H_3). \tag{1}$$

This inner state is then transformed by a series of 64 steps and the output is

$$(H_0^\star, H_1^\star, H_2^\star, H_3^\star) = (A_{64} + A_0, B_{64} + B_0, C_{64} + C_0, D_{64} + D_0). \tag{2}$$

where $A_{64} \dots D_{64}$ are defined by the recursion below:

$$\begin{aligned} A_i &= D_{i-1} \\ B_i &= B_{i-1} + (A_{i-1} + f_i(B_{i-1}, C_{i-1}, D_{i-1}) + M_{\sigma(i)} + K_i) \lll r_i \\ C_i &= B_{i-1} \\ D_i &= C_{i-1} \end{aligned} \tag{3}$$

The K_i's and r_i's are predefined constants and $\sigma(i)$'s are in Table 1. The function f_i is defined as

$$\begin{aligned} f_i(B, C, D) &= (B \wedge C) \vee (\neg B \wedge D) && \text{if } \ 0 < i \leq 16 \\ f_i(B, C, D) &= (D \wedge B) \vee (\neg D \wedge C) && \text{if } 16 < i \leq 32 \\ f_i(B, C, D) &= B \oplus C \oplus D && \text{if } 32 < i \leq 48 \\ f_i(B, C, D) &= C \oplus (B \vee \neg D) && \text{if } 48 < i \leq 64 \end{aligned} \tag{4}$$

Table 1. Values of $\sigma(i)$ in MD5 for $i = 1, \ldots, 64$ (we boldface the M_2 key inputs used in the attacks on 32 and 47 steps, and the M_6 and M_9 key inputs used in the attack on 45 steps)

Step index i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Message word $\sigma(i)$	0	1	**2**	3	4	5	6	7	8	9	10	11	12	13	14	15
Step index i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Message word $\sigma(i)$	1	6	11	0	5	10	15	4	9	14	3	8	13	**2**	7	12
Step index i	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Message word $\sigma(i)$	5	8	11	14	1	4	7	10	13	0	3	**6**	**9**	12	15	**2**
Step index i	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Message word $\sigma(i)$	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9

Fig. 1 gives a schematic view of the step function, and [12] gives a complete specification.

Fact 1. *At step i only B_i is a really new value, the others are just shifted as in a feedback shift register. Hence for $i = 0, \ldots, 60$ we have $B_i = C_{i+1} = D_{i+2} = A_{i+3}$.*

Fact 2. *The step function is* invertible*, i.e. from $A_i \ldots D_i$ and $M_{\sigma(i)}$ we can always compute $A_{i-1} \ldots D_{i-1}$. Removing the feedforward by $H_0 \ldots H_3$ in Eq.* (2) *would thus make the compression function trivially invertible.*

2.2 Preimage Attack on 32 Steps

This attack computes preimages for the 32-step compression function within about 2^{96} trials (instead of 2^{128}). It introduces two tricks used in the 45- and 47-step attacks: absorption of changes in C_0 and exploitation of the ordering of the message words.

Key Facts. Observe in Table 1 that M_2 is only input at the very beginning and the very end of 32-step MD5, namely at steps 3 and 30. Hence, if we could pick a message and freely modify M_2 such that B_3 stays unchanged, we would be able to "choose" $B_{30} = C_{31} = D_{32}$ (cf. Fact 1). A key observation is that the function f_i can either preserve or absorb an input difference: indeed for $0 < i \leq 16$ and any C and D we have

$$f_i(\texttt{0x00000000}, C, D) = (0 \wedge C) \vee (\texttt{0xffffffff} \wedge D) = D \tag{5}$$

$$f_i(\texttt{0xffffffff}, 0, D) = (\texttt{0xffffffff} \wedge 0) \vee (0 \wedge D) = 0 \tag{6}$$

These properties will be used to "absorb" a change in $C_0 = D_1 = A_2$ at steps 1 and 2. More precisely, we need that $B_0 = 0$ to absorb the changes of C_0 at step 1. And to absorb the change in $D_1 = C_0$ we need that $B_1 = \texttt{0xffffffff}$. We can now sketch the attack:

1. pick a chain value $H_0 \dots H_3 = A_0 \dots D_0$ (with certain constraints)
2. pick a message $M_0 \dots M_{15}$ (with certain constraints)
3. modify M_2 to choose $B_{30} = C_{31} = D_{32}$
4. modify $H_2 = C_0$ such that the change in M_2 doesn't alter subsequent $A_i \dots D_i$

Our strategy is inspired from Leurent's MD4 inversion [6]; the main difference is that [6] exploits absorption in the second round, whereas we use it in the early steps.

Description of the Attack. Suppose we seek a preimage of $\tilde{H} = \tilde{H}_0 \dots \tilde{H}_3$. The algorithm below first sets $B_0 = 0$ and $B_1 = \texttt{0xffffffff}$, to guarantee that a change in C_0 will only affect A_2. Then, from an arbitrarily chosen message, Algorithm 1 modifies M_2 in order to "meet in the middle". Finally, C_0 corrects the change in M_2, and this new value of C_0 does not affect the initial steps of the function.

Algorithm 1. Preimage attack on 32-step MD5

1. set $B_0 = 0$ and A_0, C_0, D_0 to arbitrary values
2. **repeat**
3. pick M_0 such that $B_1 = \texttt{0xffffffff}$
4. pick arbitrary values for $M_1 \dots M_{15}$
5. compute $A_{30} \dots D_{30}$
6. modify M_2 to get $B_{30} = D_{32} = \tilde{H} - D_0$
7. correct C_0 to keep B_3 unchanged
8. compute the final hash value $H^\star = H_0^\star \dots H_3^\star$
9. **if** $H^\star = \tilde{H}$ **then**
10. **return** $A_0 \dots D_0$ and $M_0 \dots M_{15}$

Algorithm 1 makes about 2^{96} trials by choosing 32 bits in the 128-bit image and bruteforcing the 96 remaining bits. (We denote $H^\star = H_0^\star \dots H_3^\star$ a final hash value, so our goal is to have in the end $H^\star = \tilde{H}$.)

Correctness of the Attack. We now explain in details why the attack works. First, the operation at line 3 of our algorithm is feasible because it corresponds to setting

$$M_0 = \texttt{0xffffffff} - A_0 - D_0 - K_0. \tag{7}$$

Then right after line 4 we have for any choice of C_0:

1. $f_1(B_0, C_0, D_0) = f_1(0, C_0, D_0) = D_0$
2. $f_2(B_1, C_1, D_1) = f_2(\texttt{0xffffffff}, C_1 = B_0, D_1) = 0$

In other words, the first two steps are *independent of* C_0. This will allow us to modify $C_0 = D_1 = A_2$—to correct a change in M_2—without altering $A_i \dots D_i$ between steps 4 and 30.

Now, at line 6 we set

$$M_2 = (\tilde{H}_3 - D_0 - B_{29}) \ggg 9 - G(B_{29}, C_{29}, D_{29}) - A_{29} - K_{30} \tag{8}$$

With this new value of M_2 we get in the end $H_3^\star = \tilde{H}_3$.

Finally we "correct" this change by setting

$$C_0 = (B_3 - B_2) \ggg r_3 - f_3(B_2, C_2, D_2) - M_2 - K_2. \tag{9}$$

With this new value of $C_0 = A_2$ we keep the same B_3 as with the original choice of M_2.

We can thus choose the output value $H_3^\star$ by modifying M_2 and "correcting" C_0. However, $H_0^\star$, $H_1^\star$ and $H_2^\star$ are random for the attacker. Hence, 96 bits have to be bruteforced to invert the 32-step function. This gives a total cost of 2^{96} trials.

We experimentally verified the correctness of our algorithm by searching for inputs that give $H_2^\star = H_3^\star = 0$ (see Appendix A).

2.3 Preimage Attack on 45 Steps

We present here an attack that computes 45-step preimages within 2^{100} trials and negligible memory. This combines a MITM with a conditional linear approximation of the step function. In short, the attack is based on the fact that M_2 appears at the very beginning and that M_6 and M_9 appear at the very end of 45-step MD5. Another key observation is that M_2 is used only once in the first 25 steps, and M_6 and M_9 are used only once after step 25. Algorithm 2 describes the attack for finding a preimage of $\tilde{H}_0 \dots \tilde{H}_3$.

Correctness of the Attack. First, we use again (at line 1) the trick to absorb the modification of C_0, necessary to keep the forward stage unchanged with the new value of M_2. Then, observe that

- between steps 25 and 45, M_6 and M_9 are input at steps 44 and 45 (cf. Table 1)
- at line 7 we use values of M_6 and M_9 distinct from the ones used in the forward stage (line 5)

Hence, by setting M_6 and M_9 to the values chosen the matching L entry, we would expect different values of $B_{44} = C_{45}$ and B_{45} than the (zero) ones used for the backward computation. Recall (cf. line 1) that we need $A_{45} = 0$, $B_{45} = \tilde{H}_1$, $D_{45} = 0$, hence the values of C_{45} will not matter; we would however expect a random B_{45} from the new values of M_6 and M_9.

The trick used here is that the condition imposed on M_6 and M_9 at line 5 implies that the new B_{45} equals the original $H_1^\star = \tilde{H}_1$ with probability 2^{-4} instead of 2^{-32} for random values (see below). The attack thus succeeds to find a 96-bit preimage when the MITM succeeds *and* $B_{45} = \tilde{H}_1$, that is with probability $2^{-64} \times 2^{-4} = 2^{-68}$. Storage for 2^{68} bytes is required for the MITM. For full (128-bit preimage) we bruteforce the 32 remaining bits thus the costs grows to 2^{100} trials.

Algorithm 2. Preimage attack on 45-step MD5

1. set $A_0 = \tilde{H}_0$, $B_0 = 0$, $D_0 = \tilde{H}_3$
   (We thus need $A_{45} = 0$, $B_{45} = H_1^\star$, $D_{45} = 0$. Note that we'll have $f_{45}(B_{44}, C_{44}, D_{44}) = f_{45}(C_{45}, D_{45}, A_{45}) = C_{45}$.)
2. **repeat**
3. pick M_0 such that $B_1 =$ `0xffffffff`
4. set arbitrary values to the remaining M_i's except M_6 and M_9
5. **for** all 2^{64} choices of C_0 and (M_6, M_9) such that

$$M_9 = -((M_6 \lll 19) + (M_6 \lll 23))$$

   (Here 23 coincides with r_{44} and $19 = r_{44} - r_{45}$)
6. compute $A_{25} \dots D_{25}$, store it in a list L
7. **for** $M_6 = M_9 = 0$ and all 2^{64} choices of C_{45} and M_2
8. compute $A_{25} \dots D_{25}$
9. **if** this $A_{25} \dots D_{25}$ matches an entry in L **then**
10. correct C_0 to keep B_3 unchanged
11. **return** $A_0 \dots D_0$ and $M_0 \dots M_{15}$

   (Here the message contains the M_2, M_6, M_9 corresponding to the matching entries)

Reducing the Memory Requirements. By using a cycle-finding algorithm (as for instance [16,11]) the memory requirements of the meet-in-the-middle step of the attack can be significantly reduced. Hence, we can find a preimage for 45-step MD5 with a complexity of about 2^{100} and negligible memory requirements.

On the Choice of M_6 and M_9. We explain here why the condition

$$M_9 = -(M_6 \lll 19 + M_6 \lll 23) \tag{10}$$

gives $B_{45} = \tilde{H}_1$ with high probability.

Consider the last two steps (44 and 45): because $A_{45} = D_{45} = 0$ we have $C_{44} = D_{44} = 0$ and $B_{43} = C_{43} = 0$. Hence we have

$$f_i(B, C, D) = B \oplus C \oplus D = B + C + D \tag{11}$$

in these two steps.

Note that A_{43} and D_{43} depend on the C_{45} used for the backward computation. Now we can compute B_{44} and B_{45} (note $r_{44} = 23, r_{45} = 9$)

$$B_{44} = (A_{43} + D_{43} + K_{43} + M_6) \lll 23 \tag{12}$$

$$B_{45} = (A_{44} + B_{44} + K_{44} + M_9) \lll \ \ 4 + B_{44} \tag{13}$$

For simplicity we rewrite

$$B_{44} = (X + M_6) \lll 23 \tag{14}$$

$$B_{45} = ((Y + B_{44} + M_9) \lll 4) + B_{44} \tag{15}$$

Now we can express B_{45}:

$$B_{45} = ((Y + ((X + M_6) \lll 23) + M_9) \lll 4) + ((X + M_6) \lll 23) \quad (16)$$

Since (cf. line 7 of the algorithm) we chose $(M_6, M_9) = (0, 0)$ this simplifies to

$$B_{45} = ((Y + (X \lll 23)) \lll 4) + (X \lll 23) \quad (17)$$

Consider now the case $M_9 = -(M_6 \lll 19 + M_6 \lll 23)$; Eq. (16) becomes:

$$\begin{aligned} B_{45} = ((Y + ((X + M_6) \lll 23) - ((M_6 \lll 19) + (M_6 \lll 23))) \lll 4) \\ +((X + M_6) \lll 23) \end{aligned} \quad (18)$$

We will simplify this equation by using the generic approximation:

$$(A + B) \lll k = A \lll k + B \lll k \quad (19)$$

Daum showed [2, §4.1.3] that Eq. (19) holds with probability about 2^{-2} for random A and B. We first use this approximation to replace $(X + M_6) \lll 23$ by

$$(X \lll 23) + (M_6 \lll 23). \quad (20)$$

Thus Eq. (18) yields

$$\begin{aligned} B_{45} = ((Y + (X \lll 23) - (M_6 \lll 19)) \lll 4) + (X \lll 23) \\ +(M_6 \lll 23) \end{aligned} \quad (21)$$

Finally we approximate $(Y + (X \lll 23) - (M_6 \lll 19)) \lll 4$ by

$$((Y + (X \lll 23)) \lll 4) - ((M_6 \lll 19) \lll 4) \quad (22)$$

and Eq. (21) becomes

$$B_{45} = ((Y + X \lll 23) \lll 4) + (X \lll 23) \quad (23)$$

Note that this is the same equation as for $(M_6, M_9) = (0, 0)$ in Eq. (17). Hence, we get the correct value in B_{45} with a probability of 2^{-4}, since we used two approximations[1].

Delayed-Start Attack. This attack strategy can be applied to invert the 47 steps from step 16 to 62, using M_6 in place of M_2, and the pair (M_4, M_{11}) instead of (M_6, M_9).

2.4 Preimage Attack on 47 Steps

In the following we will show how to construct a preimage for the compression function of 47-step MD5 with a complexity of about 2^{96}. This attack combines the 32-step attack with a meet-in-the-middle (MITM) strategy. The latter is made possible by the invertibility of the step function.

[1] The exact probability is $2^{-3.9097}$ according to Daum's formulas.

The attack on 47-step MD5 can be summarized as follows:

1. set initial state variable to absorb a change in C_0, as in the 32-step attack
2. compute $A_{29} \dots D_{29}$ for all 2^{32} choices of C_0 and save the result in a list L
3. compute $A_{30} \dots D_{30}$ for all 2^{32} choices of C_{47} and "meet in the middle" by finding a matching entry in L

Algorithm 3 describes the attack more formally.

Algorithm 3. Preimage attack on 47-step MD5

1. set $B_0 = 0$ and A_0, C_0, D_0 to arbitrary values
2. **repeat**
3. pick M_0 such that $B_1 =$ `0xffffffff`
4. pick arbitrary values for $M_1 \dots M_{15}$
5. **for** all 2^{32} choices of C_0
6. compute $A_{29} \dots D_{29}$, store it in a list L
7. set $A_{47} = \tilde{H}_0 - A_0$, $B_{47} = \tilde{H}_1 - B_0$, $D_{47} = \tilde{H}_3 - D_0$
8. **for** all 2^{32} choices of C_{47}
9. compute (backwards) $A_{30} \dots D_{30}$
10. **if** L contains an entry $A_{30} = D_{29}, C_{30} = B_{29}, D_{30} = C_{29}$ **then**
11. modify M_2 to have

$$B_{30} = ((A_{29} + f(B_{29}, C_{29}, D_{29}) + M_2 + K_{29}) \lll 9) + B_{29}$$

12. correct C_0 to keep B_3 unchanged
13. compute the final hash value $H_0^\star \dots H_3^\star$
14. **return** $A_0 \dots D_0$ and $M_1 \dots M_{15}$

Again this attack essentially exploits the "absorption" of 32 bits during the early steps to save a 2^{32} complexity factor. Note that when the MITM succeeds, i.e. when the line 10 predicate holds, we only have a 96-bit preimage because $H_2^\star = C_{47} + C_0$ is random. This is because both C_0 and C_{47} are random for the attacker.

Each **repeat** loop hence succeeds in finding a 96-bit preimage with probability 2^{-32}, and costs 2^{32} trials. This is respectively because

1. we have $2^{32} \times 2^{32} = 2^{64}$ candidate pairs that each match with probability 2^{-96}
2. the cost of the two **for** loops amounts to 2^{32} computations of the compression function

The total cost for finding a 128-bit preimage is thus $2^{32} \times 2^{32} \times 2^{32} = 2^{96}$, with a required storage of 2^{36} bytes (64 Gb) for the MITM. This allows us to find preimages on the 47-step MD5 compression function 2^{32} times faster than bruteforce. However it doesn't directly give a preimage attack for the hash function because the initial value is here partially random, whereas in the hash function it is fixed.

3 Preimage Attacks on 3-Pass HAVAL

HAVAL was proposed with either 3, 4, or 5 passes, i.e. 96, 128, or 160 steps. It has message blocks and hash values twice as large as MD5, i.e. 1024 bits (32 words) and 256 bits (8 words) respectively. In the following, we present two methods to invert the compression function of 3-pass HAVAL. Both attacks have a complexity of about 2^{224} compression function evaluations. Like in the attacks on step-reduced MD5, we combine a generic MITM with weaknesses in the design of the compression function. In detail, we exploit the properties of the Boolean functions to absorb differences in its input and special properties of the message ordering in 3-pass HAVAL. But before describing the attacks, we give a short description of 3-pass HAVAL.

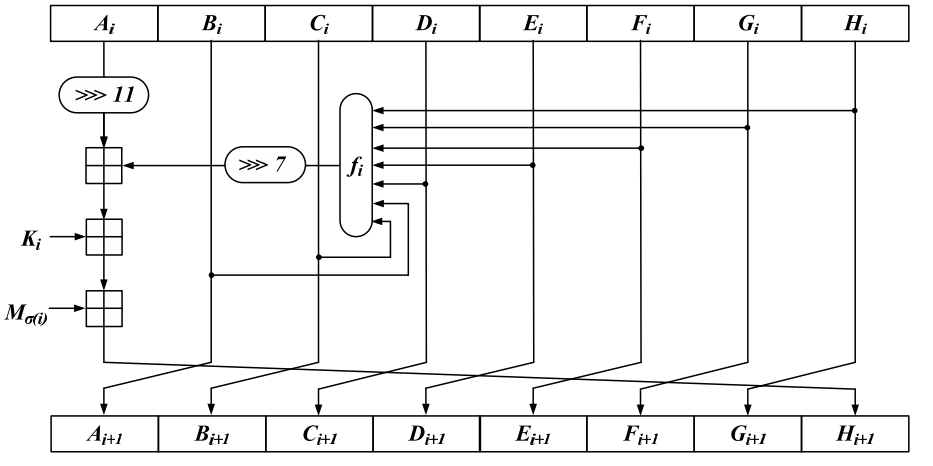

Fig. 2. The step function of HAVAL

3.1 Short Description of 3-Pass HAVAL

The structure of HAVAL is similar to that of MD5: registers $A_0, B_0, \ldots, G_0, H_0$ are initialized to the input chain values and finally the function returns

$$(H_0^\star, \ldots, H_7^\star) = (A_{96} + A_0, B_{96} + B_0, \ldots, G_{96} + G_0, H_{96} + H_0) \tag{24}$$

after 96 steps that set

$$\begin{aligned} A_i &= B_{i-1}, \\ B_i &= C_{i-1} \\ &\ldots \quad \ldots \\ G_i &= H_{i-1} \\ H_i &= A_{i-1} \ggg 11 + f_i(B_{i-1}, C_{i-1}, D_{i-1}, E_{i-1}, F_{i-1}, G_{i-1}, H_{i-1}) \ggg 7 + K_i + M_{\sigma(i)} \end{aligned} \tag{25}$$

Table 2. Values of $\sigma(i)$ in 3-pass HAVAL for $i = 1, \ldots, 96$ (we boldface the key inputs of M_5 and M_6)

Step index i	1	**2**	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Message word $\sigma(i)$	0	1	2	3	4	**5**	**6**	7	8	9	10	11	12	13	14	15
Step index i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Message word $\sigma(i)$	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Step index i	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Message word $\sigma(i)$	**5**	14	26	18	11	28	7	16	0	23	20	22	1	10	4	8
Step index i	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Message word $\sigma(i)$	30	3	21	9	17	24	29	**6**	19	12	15	13	**2**	25	31	27
Step index i	65	66	67	68	69	70	77	72	73	74	75	76	77	78	79	80
Message word $\sigma(i)$	19	9	4	20	28	17	8	22	29	14	25	12	24	30	16	26
Step index i	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Message word $\sigma(i)$	31	15	7	3	1	0	18	27	13	**6**	21	10	23	11	**5**	**2**

We thus have $H_i = G_{i+1} = F_{i+2} = E_{i+3} = D_{i+4} = C_{i+5} = B_{i+6} = A_{i+7}$ for $i = 0 \ldots 89$. Like in MD5 the step function is invertible, and uses step-specific constants, Boolean functions f_i, and message words $M_{\sigma(i)}$. The step functions are defined as (with e.g. $BC = (B \wedge C)$):

$$\begin{aligned} f_i(B, C, \ldots, H) &= FE \oplus BH \oplus CG \oplus DF \oplus D && \text{if } 0 < i \leq 32 \\ f_i(B, C, \ldots, H) &= ECH \oplus CGH \oplus CE \oplus EG \oplus CD \oplus FH \\ &\quad \oplus GF \oplus BC \oplus B && \text{if } 32 < i \leq 64 \\ f_i(B, C, \ldots, H) &= CDE \oplus CF \oplus DG \oplus EB \oplus EH \oplus H && \text{if } 64 < i \leq 96 \end{aligned} \tag{26}$$

The $\sigma(i)$'s are in Table 2. See [22] or [20] for a complete specification.

3.2 Preimage Attack A

Suppose we seek a preimage of $\tilde{H}_0 \ldots \tilde{H}_7$ with an arbitrary value for $\tilde{H}_6$; that is, we only want a 224-bit preimage. In the attack below we exploit the properties of the Boolean function f_i to absorb a difference in the input, and combine it with a MITM to improve on bruteforce search. Algorithm 4 describes the attack in detail. In the end the computed image $H^\star$ is the same as the image sought $\tilde{H}$ except (with probability $1 - 2^{-32}$) for $H_6^\star = G_{96} + G_0$. Here M_5 and M_6 are used as "neutral words", respectively in the second and the first part of the attack; the change in G_0 will correct the change in M_6, while being absorbed during the first six steps. Furthermore, if the MITM condition at line 8 is satisfied then we directly get a 224-bit preimage, because at line 6 we choose $A_{96} \ldots F_{96} H_{96}$.

Indeed we have 2^{64} candidates for $A_{48}, \ldots, H_{48}$ resulting from the forward computation and 2^{64} candidates resulting from the backward computation, so we'll find a match and thus a partial preimage with probability 2^{-128}. Hence, by repeating the attack 2^{128} times we'll find a 224-bit preimage with about $2^{128} \times 2^{64} = 2^{192}$ compression function evaluations. We need storage for 2^{69} bytes to perform the MITM. Note that a full (256-bit) preimage is obtained by bruteforcing the 32 remaining bits, increasing the cost to 2^{224} trials.

Algorithm 4. Preimage attack A on 3-pass HAVAL

1. set $C_0 = 0$, $D_0 = \tilde{H}_3 - \texttt{0xffffffff}$, $E_0 = F_0$, $H_0 = 0$, and arbitrary $A_0 B_0 G_0$
   (We need to assume $D_{96} =$ `0xffffffff` for our attack to work)
2. **repeat**
3. choose an arbitrary message for which $H_1 =$ `0xffffffff` and $H_3 = H_5 = 0$
   (This guarantees that differences in G_0 will be absorbed in the first 6 rounds)
4. **for** all 2^{64} choices of G_0 and M_5
   (A difference in M_5 only changes G_{96} after step 48)
5. compute $A_{48} \dots H_{48}$ and store it in a list L.
6. set $A_{96} = \tilde{H}_0 - A_0, \dots, H_{96} = \tilde{H}_7 - H_0$
7. **for** all 2^{64} choices of G_{96} and M_6
8. compute $A_{48} \dots H_{48}$ by going backwards
9. **if** this $A_{48} \dots H_{48}$ matches an entry in L **then**
10. correct G_0 such that $A_7 \dots H_7$ remains unchanged
11. **return** $A_0 \dots H_0$ and $M_0 \dots M_{31}$

3.3 Preimage Attack B

This attack exploits the fact that M_2 appears at the very beginning in the first pass and at the very end in the last pass. By combining this with absorption of the Boolean function in the early steps (similarly to our attack on 47-step MD5), we can construct a 192-bit preimage within about 2^{160} trials. By repeating the attack about 2^{64} times we can construct a preimage for the compression function with complexity of about 2^{224} instead of the expected 2^{256} compression function evaluations. Algorithm 5 computes a preimage of $\tilde{H}_0 \dots \tilde{H}_7$ where all $\tilde{H}_i$'s are fixed but $\tilde{H}_2$ and $\tilde{H}_6$ (i.e. a 192-bit preimage):

Algorithm 5. Preimage attack B on 3-pass HAVAL

1. set $A_0 = \tilde{H}_0$, $B_0 = \tilde{H}_1$, $D_0 = \tilde{H}_3$, $E_0 = \tilde{H}_4$, $F_0 = \tilde{H}_5$, $G_0 = 0$.
   (To get a 192-bit preimage we thus need $A_{96} = B_{96} = 0$, $D_{96} = E_{96} = F_{96} = 0$, $G_{96} = \tilde{H}_7$)
2. **repeat**
3. pick an arbitrary message for which the state variable $H_1 = 0$.
   (This guarantees that a change in C_0 will only affect A_2)
4. **for** all 2^{64} choices of C_0 and H_0
5. compute $A_{60} \dots H_{60}$ and store it in a list L.
6. **for** all 2^{64} choices of C_{96} and H_{96}
7. compute $A_{61} \dots H_{61}$
8. **if** L contains a tuple such that $A_{61} = B_{60}, \dots, G_{61} = H_{60}$ **then**
9. modify M_2 to have

$$H_{61} = (A_{60} \ggg 11) + (f_{61}(\dots) \ggg 7) + M_2 + K_{61}$$

10. correct C_0 and H_{96} accordingly
11. **return** $A_0 \dots H_0$ and $M_0 \dots M_{15}$

The MITM will succeed (line 8 of Algorithm 5) with probability $2^{-96} = 2^{64} \times 2^{64}/2^{224}$, hence $2^{96} \times 2^{64} = 2^{160}$ trials are required to get a 192-bit preimage (and storage 2^{69} bytes). A full (256-bit) preimage is obtained by bruteforcing the 64 remaining bits, which increases the cost to 2^{224} trials.

4 Extension to the Hash Functions

In this section, we will show how to extend the preimage attacks on the compression of step-reduced MD5 and 3-pass HAVAL to the hash function. The extension of the attacks to the hash function is constrained by the padding rule and the the predefined IV. The padding rule of MD5 and HAVAL forces the last bits of the message to encode its length. Thus a preimage attack should find messages that match this constraint. In our attacks we have no restrictions on the last message words and hence the padding rule is no problem; in each of the attacks proposed, we shall simply choose the end of the message to be of the form $100\cdots0\langle\ell\rangle$, where $\langle\ell\rangle$ represents the bitlength of the original message (without the padding bits).

However, the IV of our preimages for the compression function is different from the fixed one; e.g. in the attack on MD5 reduced to 47-steps we require $B_0 = 0$, and get a random value for C_0. There are several methods to turn our attacks into preimage attacks starting from the predefined IV, as described in the next two sections; the general idea will be to find many preimages (with partially random initial value) and to find many images of the fixed IV, and then combine them to "bridge the gap" between the IV and the image.

4.1 Basic Meet-in-the-Middle

Suppose we want a preimage of H. This attack sets a parameter $0 < x < n$, and first computes 2^x preimages $(\tilde{H}_i, \tilde{M}_i)$, $i = 0, \ldots, 2^x - 1$, that is, such that $f(\tilde{H}_i, \tilde{M}_) = H$; the $\tilde{M}_i$'s are chosen to have convenient padding bits. Then the attack computes 2^{n-x} random images $H_j = f(\mathrm{IV}, M_j)$, $j = 0, \ldots, 2^{n-x} - 1$, for random M_i's and the IV specified for the function. Finally we find a pair (i, j) such that $\tilde{H}_i = H_j$, and return the message $M = M_j \| \tilde{M}_i$ as a preimage of H. Because there's in total 2^n pairs (i, j), the attack will work with high probability.

For reduced-step MD5 with the optimal x we compute forward 2^{112} random chain values and compute backward 2^{16} preimages within $2^{96} \times 2^{16} = 2^{112}$ trials. The total cost of the 47-step preimage attack is thus about 2^{113} trials and memory for a preimage attack. For 3-pass HAVAL we compute forward 2^{240} chain values and backward 2^{16} preimages within $2^{224} \times 2^{16} = 2^{240}$ trials. The total cost is 2^{241} trials plus memory for a preimage attack.

4.2 Tree Approach

This attack is an improved version of the meet-in-the-middle above. It is based on the finding of multi-target preimages, and the construction of a tree whose root is the target image. This is exactly the technique described in [6], (a similar

approach was published before by Mendel and Rijmen in [10]). To summarize, we proceed in two stages

1. Backward stage: use a tree-based technique to compute a set S of multi-block preimages
2. Forward stage: compute images of random message blocks with the predefined IV until one lies in S

For MD5 the forward stage costs 2^{96} trials and the backward stages costs $32 \times 2^{97} = 2^{102}$ trials to compute 32-block preimages, plus storage for 2^{33} message blocks (i.e. 2^{39} bytes). Applied to 3-pass HAVAL we get a preimage attack that makes 2^{230} trials and needs 2^{71} bytes of storage.

5 Conclusion

We presented the first preimage attacks for the hash functions 3-pass HAVAL and step-reduced MD5: we described several preimage attacks on the MD5 compression function that invert up to 47 (out of 64) steps within 2^{96} compression function evaluations, instead of the expected 2^{128}, and two preimage attacks on the 3-pass HAVAL compression function that cost 2^{224} compression function evaluations instead of 2^{256}. We extended our best attacks to the hash functions (with padding and fixed IV) for a cost of 2^{230} and 2^{102} trials, respectively. Although these attacks are not practical (notably due to large memory requirements), they show that the security margin of 3-pass HAVAL and step-reduced MD5 with respect to preimage attacks is not as high as expected.

Acknowledgments

We would like to thank Kazumaro Aoki and Yu Sasaki for communicating us theirs results on MD5 and making helpful comments.

References

1. Cramer, R. (ed.): Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Aarhus, Denmark, May 22-26, 2005. LNCS, vol. 3494, pp. 22–26. Springer, Heidelberg (2005)
2. Daum, M.: Cryptanalysis of Hash Functions of the MD4-Family. PhD thesis, Ruhr Universität Bochum (2005)
3. De, D., Kumarasubramanian, A., Venkatesan, R.: Inversion attacks on secure hash functions using SAT solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 377–382. Springer, Heidelberg (2007)
4. den Boer, B., Bosselaers, A.: Collisions for the compression function of MD-5. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 293–304. Springer, Heidelberg (1994)

5. Dobbertin, H.: The first two rounds of MD4 are not one-way. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 284–292. Springer, Heidelberg (1998)
6. Leurent, G.: MD4 is not one-way. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 412–428. Springer, Heidelberg (2008)
7. Kim, J.-S., Biryukov, A., Preneel, B., Lee, S.-J.: On the security of encryption modes of MD4, MD5 and HAVAL. In: Qing, S., Mao, W., López, J., Wang, G. (eds.) ICICS 2005. LNCS, vol. 3783, pp. 147–158. Springer, Heidelberg (2005)
8. Klima, V.: Tunnels in hash functions: MD5 collisions within a minute. Cryptology ePrint Archive, Report 2006/105 (2006), `http://eprint.iacr.org/`
9. Lee, E., Kim, J., Chang, D., Sung, J., Hong, S.: Second preimage attack on 3-pass HAVAL and partial key-recovery attacks on NMAC/HMAC-3-pass HAVAL (to appear) (2008)
10. Mendel, F., Rijmen, V.: Weaknesses in the HAS-V compression function. In: Nam, K.-H., Rhee, G. (eds.) ICISC 2007. LNCS, vol. 4817, pp. 335–345. Springer, Heidelberg (2007)
11. Quisquater, J.-J., Delescaille, J.-P.: How easy is collision search? Application to DES. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 429–434. Springer, Heidelberg (1990)
12. Rivest, R.: RFC 1321 - The MD5 Message-Digest Algorithm (1992)
13. Van Rompay, B., Biryukov, A., Preneel, B., Vandewalle, J.: Cryptanalysis of 3-pass HAVAL. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 228–245. Springer, Heidelberg (2003)
14. Sasaki, Y., Aoki, K.: Preimage attacks on step-reduced MD5. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 282–296. Springer, Heidelberg (2008)
15. Sasaki, Y., Aoki, K.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009)
16. Sedgewick, R., Szymanski, T.G., Yao, A.C.-C.: The complexity of finding cycles in periodic functions. SIAM Journal of Computing 11(2), 376–390 (1982)
17. Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 1–22. Springer, Heidelberg (2007)
18. X. Wang, X. Lai, D. Feng, H. Chen, X. Yu.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer [1], pp. 1–18 (2005)
19. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer [1], pp. 19–35 (2005)
20. Yoshida, H., Biryukov, A., De Cannière, C., Lano, J., Preneel, B.: Non-randomness of the full 4 and 5-pass HAVAL. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 324–336. Springer, Heidelberg (2005)
21. Yu, H., Wang, X., Yun, A., Park, S.: Cryptanalysis of the full HAVAL with 4 and 5 passes. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 89–110. Springer, Heidelberg (2006)
22. Zheng, Y., Pieprzyk, J., Seberry, J.: HAVAL - a one-way hashing algorithm with variable length of output. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 83–104. Springer, Heidelberg (1993)

A Partial Preimage for 32-Step MD5

With the IV

$$H_0 = \texttt{0x67452301} \qquad H_2 = \texttt{0x382ca539}$$
$$H_1 = \texttt{0x00000000} \qquad H_3 = \texttt{0x10325476}$$

and the message

$$\begin{array}{llll} M_0 = \texttt{0xb11de410} & M_4 = \texttt{0x792a351e} & M_8 = \texttt{0x6d32a030} & M_{12} = \texttt{0x1dd5ec6d} \\ M_1 = \texttt{0x5c0cd1ec} & M_5 = \texttt{0x420582b7} & M_9 = \texttt{0x16b2e752} & M_{13} = \texttt{0x4794f768} \\ M_2 = \texttt{0xd7d35ac7} & M_6 = \texttt{0x77v8de3d} & M_{10} = \texttt{0x3b70c422} & M_{14} = \texttt{0x04fef18f} \\ M_3 = \texttt{0x5704c13b} & M_7 = \texttt{0x2476b43b} & M_{11} = \texttt{0x685cb2aa} & M_{15} = \texttt{0x00000000} \end{array}$$

we get the image

$$H_0^\star = \texttt{0xb4df93c9} \qquad H_2^\star = \texttt{0x00000000}$$
$$H_1^\star = \texttt{0x3348e3f2} \qquad H_3^\star = \texttt{0x00000000}$$

This was found in fewer than five minutes on our 2.4 GHz Core 2 Duo, whereas brute force would take about 2^{64} trials (thousands of years on the same computer).