Meet-in-the-Middle Preimage Attacks Revisited
New Results on MD5 and HAVAL

Yu Sasaki[1], Wataru Komatsubara[2], Yasuhide Sakai[2], Lei Wang[3], Mitsugu Iwamoto[2],
Kazuo Sakiyama[2] and Kazuo Ohta[2]
[1]*NTT Secure Platform Laboratories, Tokyo, Japan*
[2]*The University of Electro-Communications, Tokyo, Japan*
[3]*Nanyang Technological University, Nanyang, Singapore*

Keywords: MD5, HAVAL, Hash Function, Preimage Attack, Meet-in-the-Middle, Local-collision, Initial Structure.

Abstract: In this paper, we revisit previous meet-in-the-middle preimage attacks on hash functions. We firstly present a technical improvement for the existing local-collision and initial-structure techniques. With applying some equivalent transformation, we can significantly reduce the memory requirement from the original proposals. We then revisit the previous preimage attacks on MD5 and HAVAL with recent techniques. Consequently, we can improve the memory complexity of the previous preimage attack on full MD5 from 2^{45} to 2^{13} and on full 4-pass HAVAL from 2^{64} to 2^{32}. Moreover, we extend the preimage attack on 5-pass HAVAL from 151 steps to 158 steps, and present the first preimage attack with a single block message for 3-pass HAVAL.

1 INTRODUCTION

Cryptographic hash functions are one of the most fundamental primitives for cryptography, which compress an input message of arbitrary length into a fixed-size hash value. For a hash function $\mathcal{H}$, the preimage resistance is one of the most important security notions, which means that for a given hash value y, finding x such that $\mathcal{H}(x) = y$ must be computationally hard. When the hash value size is N bits, $\mathcal{H}$ must resist any preimage attack with a complexity less than 2^N computations. In fact, in the SHA-3 competition conducted by NIST (NIST, 2007), submitted algorithms were required to provide the preimage resistance up to 2^N computations.

Most of hash functions in practice adopt the narrow-pipe Merkle-Damgård domain extension algorithm. In this scheme, an N-bit initial value IV$(= H_0)$ is defined and the input message M is divided into several message blocks $M = M_0 \| M_1 \| \cdots \| M_{\ell-1}$. The hash value of M is computed by iteratively updating IV with a fixed-input size compression function $H_{i+1} = \mathrm{CF}(H_i, M_i)$ for $0 \leq i \leq \ell - 1$. It is widely known that preimages for the compression function CF, which are also called *pseudo-preimages*, can be converted to preimages for the hash function $\mathcal{H}$ with a multi-block message (Menezes et al., 1997, Fact9.99).

Based on this property, many researches have been conducted in order to find preimages on the compression function with a complexity less than 2^N. Leurent presented the first successful preimage attack on MD4 (Leurent, 2008). Then, Aoki and Sasaki presented the framework of the meet-in-the-middle preimage attack on the compression function (Aoki and Sasaki, 2009). The basic idea is separating the compression function into two independent subfunctions called *forward chunk* and *backward chunk* so that a part of input message bits for the forward chunk labeled as M_F (resp. M_B for the backward chunk) never impacts to the computation of the backward chunk (resp. forward chunk), respectively. In this paper, M_F and M_B are called *free bits*. One of the core ideas in (Aoki and Sasaki, 2009) is the *splice-and-cut* technique, which regards the first and last steps of the compression function as consecutive steps. The technique significantly enlarges the choices of how to separate the compression function into two independent chunks. However, as a side-effect, the attack only generates pseudo-preimages and thus using the conversion algorithm becomes necessary, which means that generated preimages are always longer than 1 block.

Many other techniques have been studied for the meet-in-the-middle preimage attack. In this paper, we focus on the *local-collision* (Sasaki and Aoki, 2008) and *initial-structure* (Sasaki and Aoki, 2009) techniques which make the attack possible even if the

Sasaki Y., Komatsubara W., Sakai Y., Wang L., Iwamoto M., Sakiyama K. and Ohta K..
Meet-in-the-Middle Preimage Attacks Revisited - New Results on MD5 and HAVAL.
DOI: 10.5220/0004521101110122
In *Proceedings of the 10th International Conference on Security and Cryptography* (SECRYPT-2013), pages 111-122
ISBN: 978-989-8565-73-0

beginning of two chunks contain free bits for both chunks. As a side-effect, the memory requirement becomes large. In fact, the previous preimage attack on MD5 and 4-pass HAVAL requires a memory to store about 2^{45} words and 2^{64} words, respectively, which is infeasible or very hard to implement. Note that the original definition of the initial-structure is very conceptual, and a part of the initial-structure was later formalized as biclique in (Bogdanov et al., 2011). However, the biclique only can deal with deterministic events, and thus cannot be applied to the previous attacks on MD5 and HAVAL which utilize probabilistic events for their initial-structure or local-collision techniques. We believe that investigating improvement of the local-collision and initial-structure techniques are useful.

MD5 is a 128-bit hash function designed by Rivest in 1992 (Rivest, 1992), and HAVAL is a variable-output-size hash function designed by Zheng *el al.* in 1992 (Zheng et al., 1993). In this paper, we focus on the 256-bit output for HAVAL, which makes the computation structure be a narrow-pipe Merkle-Damgård. The previous results on MD5 and HAVAL are summarized in Table 1. The previous preimage attacks on full MD5 and full 4-pass HAVAL require a large amount of memory. The previous preimage attack on 5-pass HAVAL does not reach the full (160) steps. All of the previous attacks on MD5 and HAVAL cannot generate preimages which fit within 1-block.

Motivation for Short Preimages

To apply preimage attacks for protocols, the size of preimages may be a critical issue. For example, let us consider the following authentication protocol;

1. *A (secret) key string denoted by M, say 1024 bits, is generated and given to a user.*
2. *The user stores M in his own device such as a smart card.*
3. *The user registers the hash value of the key string denoted by $\mathcal{H}(M)$ to the database. The database stores $\mathcal{H}(M)$ rather than M in order to protect the original key string even if the data in the database is leaked.*
4. *Every time he accesses to the system, the user inputs the key string M via his device and the system computes its hash value and compares it with the stored $\mathcal{H}(M)$.*

If the preimage resistance of $\mathcal{H}$ is broken, an attacker can recover the key string[1] from the stored digest. In such a protocol, usually the (maximum) length of the key string M is defined by the system. This indicates that if the length of generated preimages by the attacker is very long, they do not give any impact. In other words, evaluating the minimum length of preimages is an important work. In fact, the preimage attack on full MD5 only generates preimages of more than 2^{32} message blocks, which seem hard to give impact to protocols in practice.

[1]The recovered string may be different from the original key string M but has the same effect for this protocol.

Our Contributions

In this paper, we present several improvements for the meet-in-the-middle preimage attacks on MD5 and HAVAL. The results are summarized in Table 1. We improve the memory requirement for full MD5 from 2^{45} to 2^{13} and for full 4-pass HAVAL from 2^{64} to 2^{32}. We then extend the number of attacked steps for 5-pass HAVAL from 151 to 158 steps, and present the first preimage attack with a single block message for 3-pass HAVAL.

Regarding MD5 and 4-pass HAVAL, we present a technical improvement, which significantly reduces the memory requirement of the previous local-collision and initial-structure techniques. We explain our idea based on the previous application to HAVAL. See its illustration in Figure 1. Let m_j and m_{j+8} be

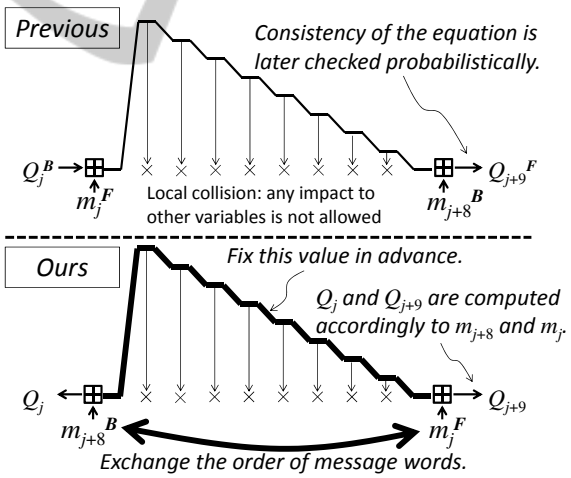

Figure 1: Memoryless local-collision technique. The superscripts 'B' and 'F' represent the free bits for the backward chunk and forward chunk, respectively. The size of free bits is $2n$ for the previous work while only n for ours.

message words used in steps j and $j+8$, respectively. Then, the chaining variable after step $j+8$ labeled as Q_{j+9} is represented by $Q_{j+9} = Q_j + m_j + m_{j+8}$. The attack uses m_j and m_{j+8} as free bits for the forward chunk and backward chunk, respectively. To make the two chunks independent, Q_{j+9} (resp. Q_j) must be computed independently of m_{j+8} (resp. m_j). However, Q_{j+9} is dependent of both of m_j and m_{j+8}. To

Table 1: Summary of preimage attacks on MD5 and HAVAL. Our improved points are emphasized.

Target	#Steps	Minimum length of preimages	Complexity	Memory (words)	Reference
MD5	64 (full)	2^{33} blocks	$2^{123.4}$	2^{45}	(Sasaki and Aoki, 2009)
	64 (full)	2^{33} blocks	$2^{123.4}$	$\underline{\mathbf{2^{13}}}$	**Ours**
3-pass HAVAL	96 (Full)	2 blocks	2^{230}	2^{64}	(Aumasson et al., 2009)
	96 (Full)	2 blocks	2^{225}	2^{64}	(Sasaki and Aoki, 2008)
	96 (Full)	**1 block**	2^{244}	2^{15}	**Ours**
4-pass HAVAL	128 (Full)	2 blocks	2^{241}	2^{64}	(Sasaki and Aoki, 2008)
	128 (Full)	2 blocks	2^{242}	$\underline{\mathbf{2^{32}}}$	**Ours**
5-pass HAVAL	151	2 blocks	2^{241}	2^{64}	(Sasaki and Aoki, 2008)
	158	2 blocks	2^{254}	2^{9}	**Ours**

solve the problem, the previous work also regards the Q_j and Q_{j+9} as free bits for the backward and forward chunks, respectively, and later check the consistency of the equation $Q_{j+9} = Q_j + m_j + m_{j+8}$ probabilistically. In details, the forward chunk and backward chunk are computed depending on $2n$ free bits of (m_j, Q_{j+9}) and (m_{j+8}, Q_j), respectively, where n represents the word size. The approach requires a memory to store 2^{2n} values to perform the meet-in-the-middle attack with $2n$ free bits. In this paper, as shown in Figure 1, we observe that the order of the addition with m_j and m_{j+8} can be exchanged as long as the local collision is formed properly. We fix the value of $Q_j + m_{j+8}$ and $Q_{j+9} - m_j$ denoted by the bold line in the bottom of Figure 1 to some value x, say $x = 0$. Then, Q_{j+9} can be computed solely dependent on m_j by $x + m_j$ and Q_j can be computed solely dependent on m_{j+8} by $x - m_{j+8}$. Due to this effort, the forward chunk and the backward chunk can be computed depending on only n free bits of m_j and m_{j+8} respectively, with always satisfying the relation of these 4 variables. This reduces the memory requirement to the square root of the previous work, *i.e.*, from 2^{2n} to 2^n, while the time complexity keeps the same as the previous work.

Regarding 5-pass and 3-pass HAVAL, we evaluate their structures in details with the recent techniques. Specifically, we analyze the initial-structure and partial computation during the matching part in a bit-wise level rather than a word-wise level which was done by the previous work. We search for new choices of free bits taking into account these technical advancements, and find better ones.

The rest of the paper is organized as follows. In Section 2, we describe MD5 and HAVAL. In Section 3, we present the low-memory local-collision technique and apply it to 4-pass HAVAL. In Section 4, we present our attack on MD5. In Section 5, we present our attacks on 5-pass and 3-pass HAVAL. Finally, we conclude this paper in Section 6.

2 SPECIFICATIONS

2.1 Specification of MD5

MD5 (Rivest, 1992) is a 128-bit hash function that adopts the narrow-pipe Merkle-Damgård domain extension. First, an input message M is padded to be a multiple of 512 bits, and then divided into 512-bit message blocks $M_0 \| M_1 \| \cdots \| M_{\ell-1}$. H_0 is set to the initial value IV defined in the specification, and $H_{i+1} \leftarrow \mathrm{md5}(H_i, M_i)$ is computed for $i = 0, 1, \ldots, \ell - 1$, where md5: $\{0,1\}^{128} \times \{0,1\}^{512} \rightarrow \{0,1\}^{128}$ is the compression function of MD5. Finally, H_n is output as a hash value of M.

The compression function takes H_i and M_i as input, and outputs H_{i+1}. M_i is divided into 32-bit message words $m_0 \| m_1 \| \cdots \| m_{15}$, and a 128-bit value p_0 is set to H_i. Then, $p_{j+1} \leftarrow R_j(p_j, m_{\pi(j)})$ is computed for $j = 0, 1, \ldots, 63$, where R_j is the step function for step j explained later. Finally, $p_{64} + H_i$ is output as H_{i+1}.

The step function is shown in Figure 2. Let Q_j be a 32-bit value satisfying $p_j = (Q_{j-3} \| Q_j \| Q_{j-1} \| Q_{j-2})$. The step function $R_j(p_j, m_{\pi(j)})$ first computes $Q_{j+1} \leftarrow Q_j + (Q_{j-3} + \Phi_j(Q_j, Q_{j-1}, Q_{j-2}) + m_{\pi(j)} + k_j) \lll s_j$, and then output p_{j+1} as $p_{j+1} \leftarrow (Q_{j-2}, Q_{j+1}, Q_j, Q_{j-1})$. Here, Φ_j, k_j, and $\lll s_j$ are the bitwise Boolean function, constant value, and left rotation defined in the specification, respectively. $\pi(j)$ is an MD5 message expansion. Refer to (Rivest, 1992) for details. In Table 2, we give the specification of $\pi(j)$, which is important for our paper.

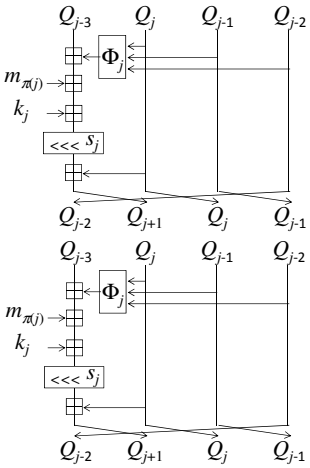

Figure 2: MD5 step function for step j.

Table 2: Message expansion of MD5. The number in the i-th row ($0 \le i \le 3$) and the j-th column ($0 \le j \le 15$) represents the message-word index for step $16 \cdot i + j$.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	6	11	0	5	10	15	4	9	14	3	8	13	2	7	12
5	8	11	14	1	4	7	10	13	0	3	6	9	12	15	2
0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9

2.2 Specification of HAVAL

HAVAL adopts the narrow-pipe Merkle-Damgård structure, which uses a 256-bit chaining variable and a 1024-bit message block to compute a compression function. The number of steps in the compression function is chosen from either 96, 128, or 160, where the corresponding algorithms are called 3-pass HAVAL, 4-pass HAVAL, and 5-pass HAVAL, respectively. Due to the similarity to MD5, we omit the description of the domain extension.

Let haval : $\{0,1\}^{256} \times \{0,1\}^{1024} \to \{0,1\}^{256}$ be the compression function of HAVAL. First, M_i is divided into 32-bit message words $m_0 \| m_1 \| \cdots \| m_{31}$ and a 256-bit value p_0 is set to H_i. Then, p_j is iteratively updated with the step function $p_{j+1} \leftarrow R_j(p_j, m_{\pi(j)})$ for $j = 0, 1, \ldots, r$, where $r = 32x - 1$ for x-pass HAVAL. Finally, $p_{r+1} + H_i$ is output as H_{i+1}.

R_j is the step function for Step j, which is depicted in Figure. 3. Q_j is a 32-bit value satisfying $p_j = (Q_{j-7} \| Q_{j-6} \| \cdots \| Q_j)$. R_j for x-pass HAVAL is defined as follows:

$$\begin{cases} T_j = f_j \circ \phi_{x,j}(Q_{j-6}, Q_{j-5}, \cdots, Q_j), \\ Q_{j+1} = (Q_{j-7} \ggg 11) + (T_j \ggg 7) + m_{\pi(j)} + k_{x,j}, \\ R_j(p_j, m_{\pi(j)}) = (Q_{j-6} \| Q_{j-5} \| \ldots \| Q_j \| Q_{j+1}), \end{cases}$$

where f_j is a bitwise Boolean function defined in Table 3, $\phi_{x,j}$ is a word-wise permutation defined in Table 4, π_j is a message expansion function defined in Table 5, $\ggg s$ is an s-bit right rotation, and $k_{x,j}$ is a constant defined in the specification.

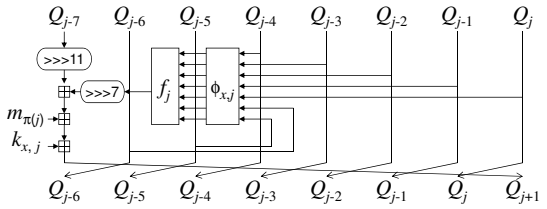

Figure 3: HAVAL step function for step j

Table 3: Boolean functions of HAVAL. $x_a x_b$ represents bitwise AND operation.

j	$f_j(x_6, x_5, \ldots, x_0)$
0–31	$x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_1 \oplus x_0$
32–63	$x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_1x_2 \oplus x_1x_4 \oplus x_2x_6 \oplus$ $x_3x_5 \oplus x_4x_5 \oplus x_0x_2 \oplus x_0$
64–95	$x_1x_2x_3 \oplus x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_3 \oplus x_0$
96–127	$x_1x_2x_3 \oplus x_2x_4x_5 \oplus x_3x_4x_6 \oplus x_1x_4 \oplus x_2x_6 \oplus$ $x_3x_4 \oplus x_3x_5 \oplus x_3x_6 \oplus x_4x_5 \oplus x_4x_6 \oplus x_0x_4 \oplus x_0$
128–159	$x_1x_4 \oplus x_2x_5 \oplus x_3x_6 \oplus x_0x_1x_2x_3 \oplus x_0x_5 \oplus x_0$

Table 4: Wordwise rotations of HAVAL.

Input	x_6	x_5	x_4	x_3	x_2	x_1	x_0
$\phi_{3,1}$	x_1	x_0	x_3	x_5	x_6	x_2	x_4
$\phi_{3,2}$	x_4	x_2	x_1	x_0	x_5	x_3	x_6
$\phi_{3,3}$	x_6	x_1	x_2	x_3	x_4	x_5	x_0
$\phi_{4,1}$	x_2	x_6	x_1	x_4	x_5	x_3	x_0
$\phi_{4,2}$	x_3	x_5	x_2	x_0	x_1	x_6	x_4
$\phi_{4,3}$	x_1	x_4	x_3	x_6	x_0	x_2	x_5
$\phi_{4,4}$	x_6	x_4	x_0	x_5	x_2	x_1	x_3
$\phi_{5,1}$	x_3	x_4	x_1	x_0	x_5	x_2	x_6
$\phi_{5,2}$	x_6	x_2	x_1	x_0	x_3	x_4	x_5
$\phi_{5,3}$	x_2	x_6	x_0	x_4	x_3	x_1	x_5
$\phi_{5,4}$	x_1	x_5	x_3	x_2	x_0	x_4	x_6
$\phi_{5,5}$	x_2	x_5	x_0	x_6	x_4	x_3	x_1

3 MEMORY EFFICIENT LOCAL-COLLISION TECHNIQUE

In this section, we explain that the local-collision technique (Sasaki and Aoki, 2008) can be performed with lower amount of memory with keeping the same time complexity, and then apply this improvement to the previous preimage attack on 4-pass HAVAL.

3.1 Previous Attack on 4-pass HAVAL

The previous attack separates the 128-step compression function into two independent chunks as shown in Table 6. All bits of the message words m_{24} and m_5 are chosen as the free bits for the forward chunk

Table 5: Message expansion of HAVAL. The number in the i-th row ($0 \le i \le 4$) and the j-th column ($0 \le j \le 31$) represents the message-word index for step $32 \cdot i + j$.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
5	14	26	18	11	28	7	16	0	23	20	22	1	10	4	8	30	3	21	9	17	24	29	6	19	12	15	13	2	25	31	27
19	9	4	20	28	17	8	22	29	14	25	12	24	30	16	26	31	15	7	3	1	0	18	27	13	6	21	10	23	11	5	2
24	4	0	14	2	7	28	23	26	6	30	20	18	25	19	3	22	11	31	21	8	27	12	9	1	29	5	15	17	10	16	13
27	3	21	26	17	11	20	29	19	0	12	7	13	8	31	10	5	9	14	30	18	6	28	24	2	23	16	22	4	1	25	15

Table 6: Message word distribution for 4-pass HAVAL.

Step	0	1	2	3	4	5	6	7	⋯	20	21	22	23	24	25	26	27	28	29	30	31
index	0	1	2	3	4	⑤	6	7	⋯	20	21	22	23	㉔	25	26	27	28	29	30	31
	backward chunk												←	local-collision							
Step	32	33	34	⋯	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
index	⑤	14	26	⋯	30	3	21	9	17	㉔	29	6	19	12	15	13	2	25	31	27
		→	forward chunk																		
Step	64	65	66	67	68	69	70	71	72	73	74	75	76	77	⋯	90	91	92	93	94	95
index	19	9	4	20	28	17	8	22	29	14	25	12	㉔	30	⋯	21	10	23	11	⑤	2
	forward chunk																			skip	
Step	96	97	98	⋯	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	
index	㉔	4	0	⋯	22	11	31	21	8	27	12	9	1	29	⑤	15	17	10	16	13	
	backward chunk																				

and the backward chunk, respectively, which means that m_{24} never appears during the backward chunk and m_5 never appears during the forward chunk. In Table 6, m_{24} and m_5 are stressed by circles. Note that each message word appears exactly once in every 32 steps. Besides two chunks, Table 6 contains two other steps called "local-collision" and "skip". For the local-collision part, as we will explain the mechanism later, the attack can be performed as if the message-word positions of m_{24} in step 24 and m_5 in step 32 were swapped. This enables the attacker to avoid the dependency between two chunks caused by these message words. For the skip part, the attacker cannot continue the independent computations (from neither direction) due to the use of the free bits for the opposite chunk. However, even if they cannot be computed, the match of the results from two chunks can be partially confirmed. We refer to the original paper (Sasaki and Aoki, 2008) for the details. Briefly speaking, the forward chunk fully computes the 256-bit value p_{94} and the backward chunk fully computes the 256-bit value p_{97}. Because the step function of HAVAL updates only 32 bits of the internal state in each step, the forward chunk still contains 224-bit information of p_{95}, 192-bit information of p_{96}, and 160-bit information of p_{97}. Hence, the attacker can match the 160-bit information of p_{97} from both chunks.

The local collision part lies between step 24 and step 32, where the input to step 24 is $p_{24} = (Q_{17}\|Q_{18}\|\cdots\|Q_{24})$ and m_{24}, and step 32 outputs $p_{33} = (Q_{26}\|Q_{27}\|\cdots\|Q_{33})$ by using m_5. The main idea of this technique is to fix the value of seven 32-bit variables $Q_{18}\|\cdots\|Q_{24}$ so that the value of Q_{25}, which is the updated value in step 24, does not give any influence in the subsequent seven steps. This is achieved by using the absorption property of f_j, which makes T_{25} to T_{32} constant irrespective of the value of Q_{25}. Then, Q_{33}, which is the updated value in step 33, becomes solely dependent of the value of Q_{25}, and is expressed as follows:

$$\begin{aligned} Q_{33} &= Q_{25} \ggg 11 + T_{33} + k_{33} + m_5 \\ &= (Q_{17} \ggg 11 + c_{24} + m_{24}) \ggg 11 + c_{33} + m_5, \quad (1) \end{aligned}$$

where c_j is the constant value denoted by $T_j + k_j$. Because the equation includes the free bits for the both chunks, the independent computation cannot be performed. To solve this problem, the previous work firstly ignored the relationship in Eq. 1 (32-bit constraint), and regard all bits of m_{24} and Q_{33} as the free bits for the forward chunk, and all bits of m_5 and Q_{17} as the free bits for the backward chunk. Because each chunk has 64 free bits, the attack can be faster than the brute force attack by a factor of 2^{64} at this stage, while the memory requirement is about 2^{64} words to store the results of at least one chunk. After the match of two independent computations are found, the attacker checks if Eq. 1 is satisfied or not. The probability that Eq. 1 is satisfied is 2^{-32}. In the end, the previous work finds pseudo-preimages faster than the brute force attack by a factor of 2^{32}, which is $2^{256-32} = 2^{224}$. Finally, the pseudo-preimage attack is converted to a preimage attack with the complexity of 2^{241}.

3.2 Idea of Our Improvement

We show that the local-collision technique can be performed only with a memory requirement of 2^{32}

words. The sketch of the idea is already given in Figure 1. For simplicity, let us omit the cyclic shift in Eq. 1 and $c_{24} = c_{33} = 0$. Then, the equation becomes

$$Q_{33} = Q_{17} + m_{24} + m_5.$$

Due to the property of the local-collision, the value of $Q_{17} + m_{24}$ only impacts to Q_{33} but never impacts to other variables. Therefore, the order of the addition can be swapped. Then, we write the equation as

$$Q_{33} - m_{24} = Q_{17} + m_5.$$

Suppose that $Q_{33} - m_{24} = x$ and $Q_{17} + m_5 = x$ for some x. Here, we fix the value of x to any value, say $x = 0$. Then, for any value of m_5, which is the free bits for the backward chunk, we can compute the corresponding Q_{17} by $x - m_5$. Similarly, for any value of m_{24}, we can compute the corresponding Q_{33} by $x + m_{24}$. In the end, each chunk only contains 32 free bits with the relationship among $Q_{17}, m_{24}, Q_{33}, m_5$ always satisfied. This achieves the meet-in-the-middle attack faster than the brute force attack by a factor of 2^{32} only with a memory requirement of 2^{32} words. Our approach has the same efficiency as the previous work but has a lower memory requirement.

3.3 Application to 4-pass HAVAL

We then extend our idea for the exact form of Eq. 1. Due to the cyclic shift, exchanging the positions of m_{24} and m_5 is not straight-forward. For simplicity, we label $Q_{24} \ggg 11 + c_{24}$ as Q'_{24} and $Q_{33} - c_{33}$ by Q'_{33}. Then, Eq. 1 becomes as follows:

$$Q'_{33} = (Q'_{24} + m_{24}) \ggg 11 + m_5.$$

If the following relation

$$(Q'_{24} + m_{24}) \ggg 11 = (Q'_{24} \ggg 11) + (m_{24} \ggg 11) \quad (2)$$

is satisfied, then we can derive

$$Q'_{33} - (m_{24} \ggg 11) = (Q'_{24} \ggg 11) + m_5.$$

Suppose that $Q'_{33} - (m_{24} \ggg 11) = (Q'_{24} \ggg 11) + m_5 = x$ for some x. Then, whenever we choose m_{24}, Q'_{33} (and thus Q_{33} as well) can be computed independently of m_5 by $x + (m_{24} \ggg 11)$. The same is applied for (Q'_{24}, m_5).

Unfortunately, Eq. 2 is not always satisfied. Hence, we experimentally verify the success probability of this equation. The strategy is counting up how many times Eq. 2 is satisfied for all possibilities of (Q'_{24}, m_{24}). However, this is infeasible due to the too large space (2^{64} values). Therefore, we execute a small experiment with reducing the word-size to 16 bits. The exact code for the experiment written by the C language is given in Appendix. From the experiment, we obtain that the success probability of Eq. 2 is about 2^{-2}. Intuitively, Eq. 2 is satisfied when

Table 7: Message word distribution for full MD5.

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

1. a carry does not occur from bit position 11 to 12.
2. a carry does not occur from bit position 31.

Due to the two conditions, the success probability is about 2^{-2}.

In the end, we perform the meet-in-the-middle part by assuming that Eq. 2 is always satisfied. We then later check the consistency of Eq. 2 only for the matched pairs, which is satisfied with 2^{-2}. Hence, our attack achieves the memory requirement of 2^{32} words, but loses the advantage of the time complexity by a factor of 2^2. Thus the time complexity of the pseudo-preimage attack is $2^{256-(32-2)=2^{226}}$, and this is converted to a preimage attack with the time complexity of $2^{(256+226)/2+1} = 2^{242}$.

4 IMPROVED ATTACK ON MD5

With the similar technique used for 4-pass HAVAL, we can reduce the memory requirement for the preimage attack on full MD5.

4.1 Previous Attack on MD5

The previous attack separates the 64-step compression function into two independent chunks as shown in Table 7. m_{14} and m_6 are chosen as the free bits for the forward chunk and the backward chunk, respectively. 4 steps between steps 14 and 17 are called "initial structure", where its role is the same as the local-collision technique, namely, the attack can be performed as if the message-word positions of m_{14} in step 14 and m_6 in step 17 were swapped.

One of the most significant techniques of their attack is the construction of the initial-structure. They analyzed the step function in details and separated those 4 steps so that the forward computation with m_{14} and the backward computation with m_6 are independent each other. The construction is very complicated and thus we refer to the original paper (Sasaki

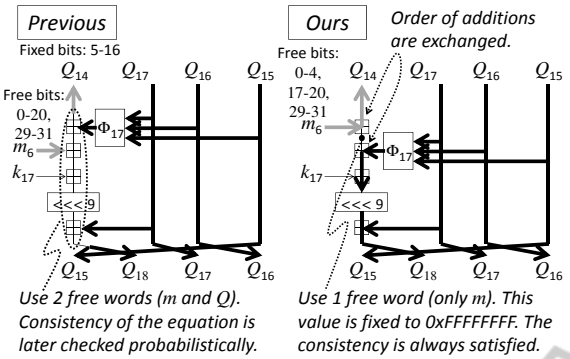

Figure 4: Construction of the initial-structure for step 17.

and Aoki, 2009) for the details. Here, we only explain the related part, which is the last step of the initial-structure, *i.e.*, step 17. The left-hand side in Figure 4 shows the computation in step 17 by the previous work. Before step 17, the free bits of m_{14} in step 14 propagates to Q_{15}, Q_{16}, Q_{17}, while in step 17, the free bits of m_6 in step 17 will propagate to Q_{14}. In Figure 4, the forward computation depending of m_{14} is drawn by black bold lines and the backward computation depending of m_6 is drawn by grey bold lines. Because the equation

$$Q_{18} = (Q_{14} + \Phi_{17} + m_6 + k_{17}) \lll 9 + Q_{17} \qquad (3)$$

depends on the both chunks, Q_{18} and Q_{14} cannot be computed independently. To solve this problem, the previous work regards two variables (20 bits in Q_{14} and 24 bits in m_6) as free bits for the backward chunk and two variables (32 bits in Q_{18} and 12 bits in m_{14}) as free bits for the forward chunk. After the match of the meet-in-the-middle part, the consistency of the 32-bit relation in Eq. 3 is checked with a probability of 2^{-32}. Note that the free bits of m_6 are bit positions 0–20 and 29–31, while bit positions 5–16 of Q_{14} must be fixed to some value in order to construct the initial-structure properly.

In the end, the improved factor for the time complexity is about 2^{44} due to the meet-in-the-middle part but it loses the advantage by a factor of 2^{32} with the consistency check. According to (Sasaki and Aoki, 2009), the complexity of the pseudo-preimage attack is $2^{116.86}$, and this is converted to a preimage attack with a complexity of $2^{123.4}$. The memory requirement is about 2^{44} for the meet-in-the-middle part. If all the details are taken into account, the memory requirement is 2^{45} as claimed in (Sasaki and Aoki, 2009).

4.2 Our Improvement

With exchanging the order of additions in step 17, we can avoid the probabilistic consistency check, which enables us to perform the attack only with one free variable and thus the memory requirement is significantly reduced. The details are given in the right-hand side of Figure 4. We fix the value of $Q_{14} + m_6$ to 0xFFFFFFFF in advance. Then, for any value of the free bits for the backward chunk m_6, we can compute Q_{14} by $\texttt{0xFFFFFFFF} - m_6 = \texttt{0xFFFFFFFF} + 1 + \overline{m_6} = \overline{m_6}$, where $\overline{m_6}$ is the bit complement of m_6. Therefore, for a given m_6, we can derive the corresponding Q_{14} bit-by-bit, without the carry effect. In the forward chunk, for any value of the free bits of m_{14} and the corresponding Q_{15}, Q_{16}, Q_{17}, we can compute Q_{18} by $(\texttt{0xFFFFFFFF} + \Phi_{17} + k_{17}) \lll 9 + Q_{17}$.

We also need to ensure that free bits of m_6 do not break the fixed bits of Q_{14}. We simply reduce the number of free bits of m_6: use only bit positions 0-4, 17-20, and 29-31, in total 12 bits. The other bits are fixed. Then, the free bits of m_6 never impact to the fixed bits of Q_{14}, and the free bits of Q_{14} can be set to any value by choosing the corresponding bits of m_6.

For the entire attack, we compute the forward chunk with 12 free bits in m_{14} and compute the backward chunk with 12 free bits in m_6. The memory requirement is now reduced to about 2^{12} message words. With all the details, the memory requirement becomes 2^{13} message words with the same reason as the previous work. The other part of the attack is exactly the same as the previous work. Thus the time complexity does not change, which is $2^{116.86}$ for pseudo-preimages and $2^{123.4}$ for preimages.

5 NEW ATTACKS ON 3-PASS HAVAL AND 5-PASS HAVAL

In this section, we extend the previous preimage attack on 5-pass HAVAL from 151 steps to 158 steps, and present the first 1-block preimage attack on 3-pass HAVAL. We evaluate the step function in details with the recent techniques. Specifically, we analyze the initial-structure and the matching part in a bit-wise level, and search for new choices of free bits.

5.1 158-step Attack on 5-pass HAVAL

The chunk separation is shown in Table 8. We choose m_{25} as free bits for the forward chunk and m_5 for the backward chunk. Steps 25 to 32 are forming the initial-structure and steps 94 to 109 are the matching part with some partial computation.

5.1.1 Construction of the Initial-structure

We locate the initial-structure between steps 25 and 32. Namely, we need to guarantee that p_{33} is inde-

Table 8: Message word distribution for 5-pass HAVAL reduced to 158 steps.

Step	0	1	2	3	4	5	6	7	8	9	⋯	23	24	25	26	27	28	29	30	31
index	0	1	2	3	4	⑤	6	7	8	9	⋯	23	24	⑮	26	27	28	29	30	31
	backward chunk													initial-structure						
Step	32	33	34	35	36	37	38	39	40	41	42	43	44	45	⋯	59	60	61	62	63
index	⑤	14	26	18	11	28	7	16	0	23	20	22	1	10	⋯	13	2	⑮	31	27
	forward chunk																			
Step	64	65	66	67	68	69	70	71	72	73	74	75	76	77	⋯	91	92	93	94	95
index	19	9	4	20	28	17	8	22	29	14	⑮	12	24	30	⋯	10	23	11	⑤	2
	forward chunk																		skip	
Step	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	⋯	124	125	126	127
index	24	4	0	14	2	7	28	23	26	6	30	20	18	⑮	19	⋯	1	29	⑤	15
	skip														backward chunk					
Step	128	⋯	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
index	27	⋯	31	10	⑤	9	14	30	18	6	28	24	2	23	16	22	4	1	⑮	15
	backward chunk																		excluded	

pendent of the free bits $m_{\pi(32)}$ and p_{25} is independent of the free bits $m_{\pi(25)}$. Firstly, we set bit positions 31–25 of $m_{\pi(25)}$ and bit positions 31–22 and 5–0 of $m_{\pi(32)}$ as free bits. The free bits of $m_{\pi(25)}$ impacts to bit positions 31–25 of Q_{26} and the free bits of $m_{\pi(32)}$ impacts to bit positions 16–1 of Q_{25}. Note that Q_{25} is computed by $(x - m_{\pi(32)}) \lll 11$. We can avoid the carry by setting $x =$ 0xFFFFFFFF so that $x - m_{\pi(32)} = x + 1 + \overline{m_{\pi(32)}} = \overline{m_{\pi(32)}}$. Secondly, we fix other chaining variables so that the change of the free bits does not propagate through the f_j function. How chaining variables are fixed is shown in Table 9 and Fig. 5. In the following, we explain the details of the computation step by step.

In Table 9, **0**, **1**, **C**, $\mathbf{C}_i$, X, x, and Y denote 0x00000000, 0xFFFFFFFF, $(-f_{25} \lll 4) - (k_{5,25} \lll 11)$, a fixed value, free bits for the forward chunk, a value dependent on X, and free bits for the backward chunk, respectively. In Fig. 5, numbers written in a small bold font denote the value of each variable. The notation a^b represents that the one-bit value a continues for b bits. For example, $X^7 0^{25}$ means that bit positions 24–0 are set to 0 and 31–25 are set to free bits for the forward chunk. We also use notations ^{For} and ^{Back} to denote free bits for the forward and backward chunks, respectively, Black and gray bold lines represent data lines depending on the free bits for the forward and backward chunks, respectively. Dotted lines represent data lines that are fixed to absorb the impact of the free bits via f_j. Narrow lines represent data lines that are always fixed irrespective of the value of the free bits.

Now we explain how to construct the initial-structure step by step. In step 26, the f_j function is $Q_{24}Q_{25} \oplus Q_{21}Q_{22} \oplus Q_{26}Q_{23} \oplus Q_{20}Q_{24} \oplus Q_{20}$. To make the output independent of the free bits in Q_{25}, we fix the corresponding bits of Q_{24} to 0. Similarly, to make the output independent of Q_{26}, we fix Q_{23} to 0. Therefore, the output of f_{26} is always constant even if Q_{25} and Q_{26} change.

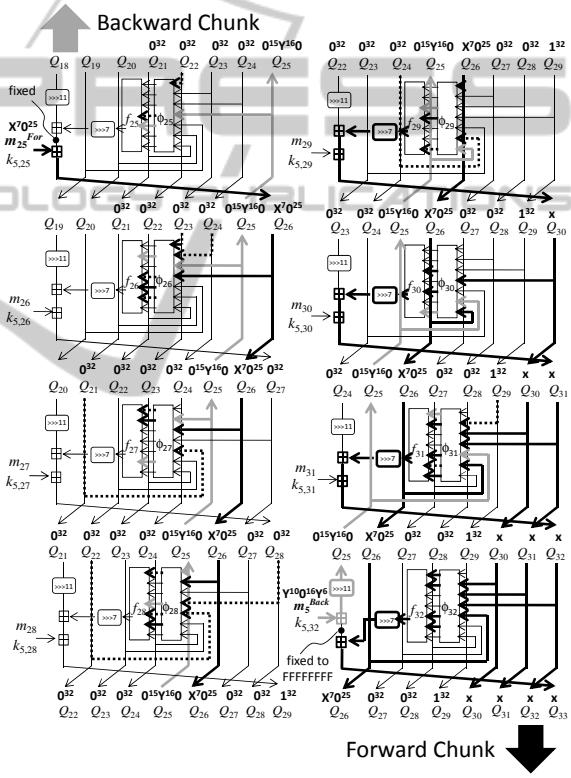

Figure 5: Initial-structure for 5-Pass HAVAL.

Similarly, in steps 27 and 28, Q_{25} and Q_{26} are absorbed by fixing the other chaining variables as shown in Table 9. Note that free bits for the forward chunk denoted by X can give impact to p_{33}. Therefore, in steps 29, 30, and 31, we do not set conditions to absorb the impact from X, and we only consider absorbing the impact from Q_{25}.

Finally, p_{33} is computed independently of $m_{\pi(32)}$ and p_{25} is computed independently of $m_{\pi(25)}$.

Table 9: Fixed values in the initial-structure for 5-pass HAVAL reduced to 158 steps

step	Conditions		$m_{\pi(j)}$	Q_{j-7}	Q_{j-6}	Q_{j-5}	Q_{j-4}	Q_{j-3}	Q_{j-2}	Q_{j-1}	Q_j
j	to absorb Q_{25}	to absorb Q_{26}									
25	$Q_{22}=\mathbf{0}$		$\textcircled{m}_{25}$	$\mathbf{C}$	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	0^7Y^{25}
26	$Q_{24}=\mathbf{0}$	$Q_{23}=\mathbf{0}$	m_{26}	$\mathbf{C}_1$	$\mathbf{C}_2$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	0^7Y^{25}	X^70^{25}
27	$Q_{21}^{24-0}=Q_{26}^{24-0}$	$Q_{25}=\mathbf{0}$	m_{27}	$\mathbf{C}_2$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	0^7Y^{25}	X^70^{25}	$\mathbf{0}$
28	$Q_{28}=\mathbf{0}$	$Q_{22}=Q_{27}$	m_{28}	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	0^7Y^{25}	X^70^{25}	$\mathbf{0}$	$\mathbf{0}$
29	$Q_{24}=\mathbf{0}$		m_{29}	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	0^7Y^{25}	X^70^{25}	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$
30	$Q_{26}=\mathbf{0}$		m_{30}	$\mathbf{0}$	$\mathbf{0}$	0^7Y^{25}	X^70^{25}	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$	x
31	$Q_{29}=\mathbf{1}$		m_{31}	$\mathbf{0}$	0^7Y^{25}	X^70^{25}	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$	x	x
32			$\textcircled{m}_{5}$	0^7Y^{25}	X^70^{25}	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$	x	x	x
33				X^70^{25}	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{1}$	x	x	x	x

5.1.2 Computations for the Skipped Part

Next we explain how to partially compute steps 94–96 and 109–104. The details are shown in Fig 6 and Fig 7. In step 94 for example, the equation of Q_{95} is as follows: $Q_{95} = ((Q_{87} \ggg 11) + (f_{94}(\phi_{94}(Q_{88}, Q_{89}, \ldots, Q_{94})) \ggg 7) + k_{5,94}) + m_5$. Bit positions 21–6 of m_5 are fixed values, so we can compute bit positions 21–6 of Q_{95}. First, we compute the sum of $Q_{87} \ggg 11$, $k_{5,94}$ and the output of f_{94}. Then, we compute the addition with m_5 to obtain the value of Q_{95}^{21-6}, where the superscript represents the computable bit positions. Because bit positions 5–0 of m_5 are unknown, there are two possible carry patterns from bit position 5 to 6. We consider both carry patterns and proceed the attack for both of them. The carry information should be stored so that we can later check the correctness of the carry assumption. We use variables C_a^{For} and C_a^{Back} which store the assumed value for the carry. Note that each carry assumption costs only one bit of memory but the computational complexity increases.

In step 95 for example, the equation of Q_{96} is as follows: $Q_{96} = (Q_{88} \ggg 11) + (f_{95}(\phi_{95}(Q_{89}, Q_{90}, \ldots, Q_{95})) \ggg 7) + m_{\pi(95)} + k_{5,95}$. Q_{95}, whose 16 bits are fixed, is used in the f_j function, thus we can compute 16 bits of Q_{96}. Let the value after the right cyclic shift by 7 bits be v, and then, we uniquely obtain $v^{31,14-0}$. Here, we do not use v^{31}, because holding this value will not contribute to the matching part. Finally we compute the addition of $Q_{88} \ggg 11$, v^{14-0}, $m_{\pi(95)}$, and $K_{5,95}$, and then we uniquely obtain the value of Q_{96}^{14-0}.

Similarly, we partially compute the step function in steps 96, 109, 108, 107, 106, 105, and 104, to obtain bit positions 31 and 7–0 of Q_{97} from both chunks. Finally, the match at bit positions 31 and 7–0 of Q_{97}, in total 9 bits, can be performed.

5.1.3 Attack Procedure

1. Set chaining variables and message words but the free bits in order to satisfy the initial structure and several constraints for the padding string.
2. For bit positions 31–25 of m_{25}, in total 7 free bits, compute the forward chunk with guessing two unknown carry bits for the skipped part. Store 9 bits (bit positions 31 and 0–7) of Q_{97} for each guess. We obtain $2^7 \cdot 2^2 = 2^9$ candidates.
3. For bit positions 31–22 and 5–0 of m_5, in total 16 free bits, do as follows.
   (a) Compute the backward chunk with guessing five unknown carry bits for the skipped part to obtain 9 bits (bit positions 31 and 0–7) of Q_{97}. We obtain $2^{16} \cdot 2^5 = 2^{21}$ candidates.
   (b) Check whether or not the 9 bits of Q_{97} match.
   (c) If 9 bits match, check whether the other bits of p_{97} match and all of the carry assumptions are correct.
   (d) If correct, the corresponding (p_0, M) is a pseudo-preimage. Otherwise, repeat this procedure with choosing other values for randomly fixed message words in phase 1.

5.1.4 Complexity Evaluation

Let the complexity of 1 step be $\frac{1}{158}$ compression function computations. Roughly speaking, the computation for the forward chunk requires $2^9 \cdot \frac{68}{158}$ compression function computations and the memory requirement is about 2^9 state values. The computation for the backward chunk requires $2^{21} \cdot \frac{81}{158}$ compression function computations. Hence, the time complexity is about 2^{20} compression function computations.

The complexity can be improved by considering the attack details. For example, the unknown carry

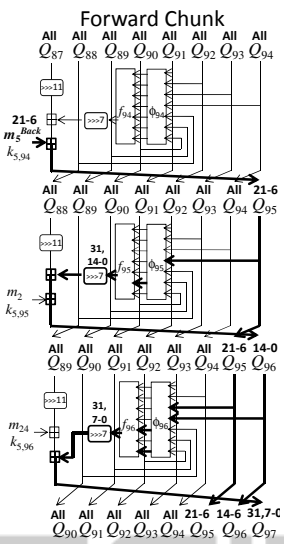

Figure 6: Partial-computation for the matching part in the forward chunk.

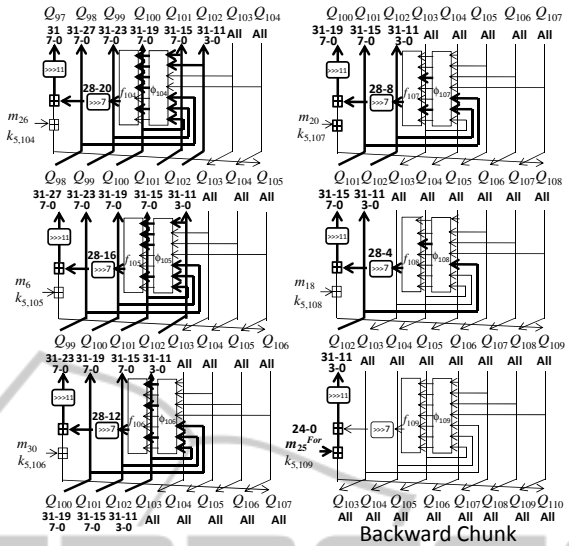

Figure 7: Partial-computation for the matching part in the backward chunk.

bits occur in the very last of the computation for each chunk. Hence, the complexity of the forward chunk can be $2^7 \cdot \frac{66}{158} + 2^7 \cdot 2 \cdot \frac{1}{158} + 2^7 \cdot 2^2 \frac{1}{158}$ rather than $2^9 \cdot \frac{68}{158}$. By considering such an optimization,the sum of the complexity for the forward and backward chunks becomes $2^{14} \cdot \frac{1169}{158} < 2^{17}$. This means that we can obtain $2^{7+16-9} = 2^{14}$ pairs where 9 bits match with a complexity of 2^{17}. Therefore, by repeating the above procedure 2^{233} times, we expect to obtain a pseudo-preimage. Finally, the complexity of finding a pseudo-preimage of 5-pass HAVAL is $2^{17+233} = 2^{250}$, and this is converted to a preimage with a complexity of 2^{254}. In the attack procedure, the dominant memory complexity is for the forward chunk, which requires about 2^9 internal state values.

5.2 1-block Attack on 3-pass HAVAL

The overall strategy is the same as the 1-block preimage attack on MD4 (Aoki and Sasaki, 2009). To generate 1-block preimages, we must fix p_0 to the original HAVAL's *IV*. To achieve this, we use two free variables in the backward chunk, *i.e.*, we cancel the impact of changing one free variable by changing the other variable so that p_0 can be a fixed value. Our chunk separation is shown in Table 10, where two message words (m_{19}, m_{27}) are the free bits for the backward chunk and m_{28} is the free bits for the forward chunk.

To cancel the impact of changing m_{27} with m_{19} during the backward computation, we need to guarantee that m_{19} and m_{27} form a local-collision in steps 19-27. Therefore we fix chaining variables so that the change of a chaining variable directly affected by m_{19} and m_{27} does not propagate through the Boolean functions. How chaining variables are fixed is shown in Table 11, where $*$ denotes a flexible value which depends on m_{19} and m_{27}. The Boolean function f_j for steps 19 to 27 has the absorption property. For example for $j = 20$, the Boolean function is $Q_{18}Q_{17} \oplus Q_{14}Q_{20} \oplus Q_{15}Q_{19} \oplus Q_{16}Q_{18} \oplus Q_{16}$. Q_{20} is a flexible value, so Q_{14} is fixed to **0** to make the output of f_{20} independent of Q_{20}. Similarly, in other steps, the impact of Q_{20} is absorbed by fixing other chaining variables.

In this attack, similarly to the attack on 5-pass HAVAL, we consider the unknown carry effect when we compute the skipped part for steps 56 to 68. The strategy for the partial computation is basically the same as the one for 5-pass HAVAL, because rotation numbers are identical in all steps. Due to the page limitation, we omit the details. The attack procedure is as follows.

Attack Procedure

1. Fix the chaining variables and message words for steps 19 to 27 to form a local collision as shown in Table 11. Then, fix m_{29}, m_{30}, and m_{31} to satisfy the padding string for a 1-block message.
2. Randomly determine the message words m_i, where $i \in \{8, 9, \ldots, 18, 27\}$, and compute the step function from step 19 to 8. After this step, p_8 is computed.
3. Set p_0 to *IV*. With a given p_0 and p_8, compute the message words $m_i (i \in \{0, 1, \ldots, 7\})$ that connect these values. This is done by inverting the step function with respect to m_i.

Table 10: Message word distribution for 3-pass HAVAL.

Step	0	1	2	3	⋯	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
index	0	1	2	3	⋯	16	17	18	⑲	20	21	22	23	24	25	26	㉗	㉘	29	30	31
	fixed								local collision									forward			
Step	32	33	34	35	36	37	38	⋯		52	53	54	55	56	57	58	59	60	61	62	63
index	5	14	26	18	11	㉘	7	⋯		17	24	29	6	⑲	12	15	13	2	25	31	㉗
	forward chunk													skip							
Step	64	65	66	67	68	69	70	71	⋯	84	85	86	87	88	89	90	91	92	93	94	95
index	⑲	9	4	20	㉘	17	8	22	⋯	1	0	18	㉗	13	6	21	10	23	11	5	2
	skip					backward chunk								fixed							

Table 11: Fixed values for the local-collision in 3-pass HAVAL.

step j	Condition to set off Q_{20}	$m_{\pi(j)}$	Q_{j-7}	Q_{j-6}	Q_{j-5}	Q_{j-4}	Q_{j-3}	Q_{j-2}	Q_{j-1}	Q_j
19		⑲	$\mathbf{C}_0$	$\mathbf{C}_1$	**0**	$\mathbf{C}_2$	**0**	$\mathbf{C}_3$	$\mathbf{C}_4$	**0**
20	$Q_{14} = \mathbf{0}$	m_{20}	$\mathbf{C}_1$	**0**	$\mathbf{C}_2$	**0**	$\mathbf{C}_3$	$\mathbf{C}_4$	**0**	$*$
21	$Q_{16} = \mathbf{0}$	m_{21}	**0**	$\mathbf{C}_2$	**0**	$\mathbf{C}_3$	$\mathbf{C}_4$	**0**	$*$	**0**
22	$Q_{18} = Q_{19} = \mathbf{0}$	m_{22}	$\mathbf{C}_2$	**0**	$\mathbf{C}_3$	$\mathbf{C}_4$	**0**	$*$	**0**	**1**
23	$Q_{21} = \mathbf{0}$	m_{23}	**0**	$\mathbf{C}_3$	$\mathbf{C}_4$	**0**	$*$	**0**	**1**	$\mathbf{C}_5$
24	$Q_{22} = \mathbf{1}$	m_{24}	$\mathbf{C}_3$	$\mathbf{C}_4$	**0**	$*$	**0**	**1**	$\mathbf{C}_5$	**0**
25	$Q_{24} = \mathbf{0}$	m_{25}	$\mathbf{C}_4$	**0**	$*$	**0**	**1**	$\mathbf{C}_5$	**0**	$\mathbf{C}_6$
26	$Q_{26} = \mathbf{0}$	m_{26}	**0**	$*$	**0**	**1**	$\mathbf{C}_5$	**0**	$\mathbf{C}_6$	**0**
27		㉗	$*$	**0**	**1**	$\mathbf{C}_5$	**0**	$\mathbf{C}_6$	**0**	$\mathbf{C}_7$
28		m_{28}	**0**	**1**	$\mathbf{C}_5$	**0**	$\mathbf{C}_6$	**0**	$\mathbf{C}_7$	$\mathbf{C}_8$

4. Compute the backward chunk until step 88. Note that phase 1 to phase 4 are independent of the free bits for both chunks.
5. For all possible values of bit positions 31–20 of m_{19} in total 12 free bits, do as follows.
   (a) Compute m_{27} accordingly so that the impact of m_{19} is canceled.
   (b) Compute the backward chunk with guessing 3 unknown carry bits. As a result, we store $2^{12} \cdot 2^3 = 2^{15}$ candidates of bit positions 31–25 and 5–0, in total 13 bits of Q_{59}.
6. For all possible values of bit positions 31–23 and 5–0 of m_{28} in total 15 free bits,
   (a) Compute the forward chunk with guessing 1 unknown carry bit. As a result, we obtain $2^{15} \cdot 2^1 = 2^{16}$ candidates of bit positions 31–25 and 5–0, in total 13 bits of Q_{59}.
   (b) Check whether both Q_{59} computed from the forward and backward chunks match in the 13 bits.
   (c) If 13 bits match, check whether all of the other internal state bits match and the carry assumptions are correct. If correct, the corresponding M is a preimage.
   (d) If a preimage is not found, repeat this procedure with choosing other values for randomly fixed message words.

The complexity of this attack is as follows. Steps 1 to 4 are independent of the free bits for both chunks. The complexity is negligible compared to the other steps. Step 5 requires the complexity of $2^{12} \cdot 2^3 \cdot \frac{23}{96}$, and provides 2^{15} items in the table. Step 6a requires the complexity of $2^{15} \cdot 2 \cdot \frac{31}{96}$, and provides 2^{16} candidates. After step 6b $2^{18} (= 2^{15} \cdot 2^{16} \cdot 2^{-13})$ pairs will remain. In step 6c, for the remaining 2^{18} candidates, we compute a few steps, and check the correctness of the carry assumption for 4 bits. This requires about $2^{18} \cdot \frac{8}{96}$ steps, which is less than 2^{15}. After that, $2^{14} (= 2^{18} \cdot 2^{-4})$ pairs will remain.

If all the complicated details are considered, the sum of the above complexity becomes $2^{14} \cdot \frac{119}{96}$, which is less than 2^{15}. This means that we can obtain 2^{14} pairs where 13 bits match with a complexity of 2^{15}. Therefore, by repeating the above procedure 2^{229} times, we expect to obtain a preimage. Finally, the complexity of finding a 1-block preimage of 3-pass HAVAL is 2^{244}.

In the attack procedure, the dominant memory complexity is for Step 5, which requires 2^{15} candidates to be stored. Therefore the memory complexity of our attack is about 2^{15} states.

6 CONCLUSIONS

In this paper, we showed that the memory requirement for the previous local-collision technique can be significantly reduced. We then applied our observation to the previous preimage attacks on MD5

and HAVAL. Consequently, we improved the memory complexity of the previous preimage attack on full MD5 from 2^{45} to 2^{13} and on full 4-pass HAVAL from 2^{64} to 2^{32}. Moreover, we extended the preimage attack on 5-pass HAVAL from 151 steps to 158 steps, and presented the first preimage attack with a single block message for 3-pass HAVAL.

REFERENCES

Aoki, K. and Sasaki, Y. (2009). Preimage attacks on one-block MD4, 63-step MD5 and more. In Avanzi, R. M., Keliher, L., and Sica, F., editors, *Selected Areas in Cryptography SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119, Berlin, Heidelberg, New York. Springer-Verlag.

Aumasson, J.-P., Meier, W., and Mendel, F. (2009). Preimage attacks on 3-pass HAVAL and step-reduced MD5. In Avanzi, R. M., Keliher, L., and Sica, F., editors, *Selected Areas in Cryptography SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 120–135, Berlin, Heidelberg, New York. Springer-Verlag.

Bogdanov, A., Khovratovich, D., and Rechberger, C. (2011). Biclique cryptanalysis of the full AES. In Lee, D. H. and Wang, X., editors, *Advances in Cryptology — ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371, Berlin, Heidelberg, New York. Springer-Verlag.

Leurent, G. (2008). MD4 is not one-way. In Nyberg, K., editor, *Fast Software Encryption (FSE 2008)*, volume 5086 of *Lecture Notes in Computer Science*, pages 412–428, Berlin, Heidelberg, New York. Springer-Verlag.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. (1997). *Handbook of applied cryptography*. CRC Press.

NIST (2007). *Federal Register /Vol. 72, No. 212/Friday, November 2, 2007/Notices*. http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.

Rivest, R. L. (1992). *Request for Comments 1321: The MD5 Message Digest Algorithm*. The Internet Engineering Task Force. http://www.ietf.org/rfc/rfc1321.txt.

Sasaki, Y. and Aoki, K. (2008). Preimage attacks on 3, 4, and 5-pass HAVAL. In Pieprzyk, J. P., editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271, Berlin, Heidelberg, New York. Springer-Verlag.

Sasaki, Y. and Aoki, K. (2009). Finding preimages in full MD5 faster than exhaustive search. In Joux, A., editor, *Advances in Cryptology — EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152, Berlin, Heidelberg, New York. Springer-Verlag.

Zheng, Y., Pieprzyk, J., and Seberry, J. (1993). HAVAL — one-way hashing algorithm with variable length of output. In Seberry, J. and Zheng, Y., editors, *Advances in Cryptology — AUSCRYPT'92*, volume 718 of *Lecture Notes in Computer Science*, pages 83–104. Springer-Verlag, Berlin, Heidelberg, New York.

APPENDIX

The below is the code for the experiment. It returns the following result (in a hexadecimal form).

#Success: 42084000

From this result, we obtain the success probability of $\texttt{0x42084000}/\texttt{0xFFFFFFFF} \approx 2^{-2}$.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

typedef unsigned int  UINT32;

UINT32 Q24,m24,temp1,temp2,count=0;
UINT32 rrotate(UINT32 x, int i);

int main(){
   for(Q24=0;Q24<=0xFFFF;Q24++){
      for(m24=0;m24<=0xFFFF;m24++){
         temp1=rrotate(((Q24+m24)&0xFFFF),11);
         temp2=((rrotate(Q24,11)+
                  rrotate(m24,11))&0xFFFF);
         if(temp1==temp2){
            count++;
         }
      }
   }
   printf("#Success: %08x\n",count);
   return(0);
}

UINT32 rrotate(UINT32 x, int i){
   return (x<<(16-i)|x>>i)&0xFFFF;
}
```