Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1

Kazumaro Aoki and Yu Sasaki

NTT, 3-9-11 Midoricho, Musashino-shi, Tokyo 180-8585 Japan

Abstract. Preimage resistance of several hash functions has already been broken by the meet-in-the-middle attacks and they utilize a property that their message schedules consist of only permutations of message words. It is unclear whether this type of attacks is applicable to a hash function whose message schedule does not consist of permutations of message words. This paper proposes new attacks against reduced SHA-0 and SHA-1 hash functions by analyzing a message schedule that does not consist of permutations but linear combinations of message words. The newly developed cryptanalytic techniques enable the meet-in-the-middle attack to be applied to reduced SHA-0 and SHA-1 hash functions. The attacks find preimages of SHA-0 and SHA-1 in $2^{156.6}$ and $2^{159.3}$ compression function computations up to 52 and 48 steps, respectively, compared to the brute-force attack, which requires 2^{160} compression function computations. The previous best attacks find preimages up to 49 and 44 steps, respectively.

keywords: SHA-0, SHA-1, meet-in-the-middle, one-way, preimage

1 Introduction

After the breakthrough described in Wang's study [14], much attention has been paid to the security of MD4-like hash functions such as MD4 [7], MD5 [8], HAVAL [15], and SHA family [13]. First, attention was focused on collision resistance, and the more recently, attention has been focused on preimage resistance. Preimage resistance is more important than collision resistance because the security of many applications employing hash functions are based on preimage resistance, and breaking preimage resistance implies breaking collision resistance of the practical hash functions[1], and, therefore, we should focus more attention on preimage resistance. Saarinen showed a preimage attack on new FORK-256 [4] in 2007 [9], and in 2008, Leurent showed that a preimage of MD4 can be computed to the complexity of $2^{100.51}$ MD4 computations [5]. In these attack, the meet-in-the-middle technique helps to compute the preimage. After this, the meet-in-the-middle attack is directly used to compute a (second) preimage of hash functions [2, 1, 10, 12, 11], and the meet-in-the-middle technique seems to be a very powerful tool to compute a preimage.

[1] Collision resistance implies preimage resistance for hash functions with uniformly random output. Note that collision resistance does not always imply preimage resistance. Construction of such artificial hash functions is explained in [6, Note 9.20].

SHA-1 is a widely used hash function, and its security assessment is very important. Actually, many public key encryption and signature schemes use SHA-1 as a random oracle, and the SSL/TLS protocol uses SHA-1 for many purposes. In Crypto 2008, [3] showed the first preimage attacks against reduced SHA-0 and SHA-1. Its authors use "reversing the inversion problem" and attacked SHA-0 and SHA-1 up to 49 and 44 steps, respectively. On the other hand, the resistance of SHA-0 and SHA-1 against the meet-in-the-middle technique is an interesting problem to be studied. However, the previous preimage attacks using the meet-in-the-middle technique are only applied to hash functions whose message schedule consists of permutations of the message words, while the message schedules of SHA-0 and SHA-1 are in more complicated form, that is, linear transformation of message words. Moreover, the techniques developed in [3] seems not to be able to be applied to the framework of the meet-in-the-middle attack.

On conducting the meet-in-the-middle attack, first we partition steps for SHA-0 (or SHA-1) into two *chunks*. A chunk comprises consecutive steps of the corresponding hash function and includes at least one *neutral word*, which appears in the chunk and does not appear in the other chunk. So, steps in a chunk can be executed using the neutral word for the chunk, and does not require the neutral word for the other chunk. Although finding neutral words is important in this scenario, the previous meet-in-the-middle attack only are applied to hash functions such as MD4, which has a simple message schedule, that is, consisting only of permutations of message words. So, a message word itself can be regarded as a neutral word, and it is very easy to find chunks of long steps. For example, [10] can compute a (second) preimage of HAVAL-5 up to 151 steps. To apply the same strategy to SHA-0 or SHA-1, we face the first difficulty which is that we cannot find any neutral words for long steps, because SHA-0 and SHA-1 adopt the linear transformation of a message word as message schedule, and the linear transformation spreads the effect of a message word to many steps and prevents finding neutral words. To solve this problem, we seek chunks that the rank of their matrix representation is not full, and we regard the kernel generators of linear transformations of each chunk as neutral words. This seems to be a good idea, in fact, if message words included in the kernel generators of the first and the second chunks are different, each kernel can be computed independently, and thus, the meet-in-the-middle attack can be performed. However, we face the second difficulty that the kernel generators for two chunks may share the same message word, and how to determine the value of the neutral word in each chunk is unclear. To overcome the second problem, we convert a message schedule by multiplying by a regular matrix, so that converting a message schedule using a matrix results in converted kernel generators for two chunks becoming unit vectors. That is, we can choose converted message words as neutral words, and this enables us to apply the existing meet-in-the-middle attack to SHA-0 and SHA-1.

This paper presents a new analysis method for a linear message schedule, which enables us to utilize the meet-in-the-middle technique to compute a preimage effectively. We then apply this technique to SHA-0 and SHA-1. The newly

Table 1. Preimage Attacks Against SHA-0 and SHA-1

	Attack Type	[3] # of Steps	[3] Complexity Time	[3] Complexity Memory	Current Results # of Steps	Current Results Complexity Time	Current Results Complexity Memory
SHA-0	pseudo-preimage	50	2^{158}	2^{25}	52	$2^{151.2}$	2^{15}
					52	$2^{152.2}$	negligible
	preimage	49	2^{159}	2^{25}	52	$2^{156.6}$	2^{15}
					52	$2^{157.1}$	negligible
SHA-1	pseudo-preimage	45	2^{157}	2^{20}	48	$2^{156.7}$	2^{40}
					48	$2^{157.7}$	negligible
	preimage	44	2^{157}	2^{20}	48	$2^{159.3}$	2^{40}
					48	$2^{159.8}$	negligible

The unit of time complexity is one compression function computation, and the unit of memory complexity is a few times of the hash length which is 160 bits.

developed analysis is a generalization of the previously reported analysis for MD5 and other hash functions [11, 10, 12]. The technique with the detailed analysis of step functions can find a preimage of reduced SHA-0 and SHA-1 faster than the brute-force attack up to 52 and 48 steps, respectively (out of 80 steps), which are the best results so far. Table 1 summarizes the preimage attacks against SHA-0 and SHA-1. We also note the complexity of memoryless attack in Table 1.

2 Preliminaries

2.1 Specification of SHA-0 and SHA-1

This paper focuses on SHA-b ($b = 0$ or 1). This section shows the specifications of SHA-b used in this paper. For more details, please refer to the original specifications [13].

SHA-b adopts the Merkle-Damgård structure [6, Algorithm 9.25]. The message string is first padded to be a 512-bit multiple, and divided into 512-bit blocks, $(M_0, M_1, \ldots, M_{m-1})$ ($M_i \in \{0,1\}^{512}$). The compression function inputs a 512-bit message string and 160-bit chaining variable. The message blocks are input to the iterative use of compression function CF to compute hash value H_m.

$$H_0 \leftarrow \mathrm{IV}, \qquad H_{i+1} \leftarrow \mathrm{CF}(H_i, M_i) \quad (i = 0, 1, \ldots, m-1)$$

where IV is the constant defined in the specification.

The compression function is based on the Davies-Meyer mode [6, Algorithm 9.42]. Let $^{\lll x}$ denote the x-bit left rotation. First, the message block is expanded using the message schedule algorithm.

$$\begin{cases} w_j \leftarrow m_j, & (\ 0 \le j < 16) \\ w_j \leftarrow (w_{j-3} \oplus w_{j-8} \oplus w_{j-14} \oplus w_{j-16})^{\lll b}, & (16 \le j < 80) \end{cases} \tag{1}$$

where $(m_0, m_1, \ldots, m_{15}) \leftarrow M_i$ ($m_j \in \{0,1\}^{32}$). Hereafter, we call a 32-bit string a *word*. Then, the step functions are applied.

$$p_0 \leftarrow H_i, \quad p_{j+1} \leftarrow R_j(p_j, w_j) \text{ (for } j = 0, 1, \ldots, 79), \quad H_{i+1} \leftarrow H_i + p_{80}, \tag{2}$$

where "+" denotes the wordwise addition. Step function R_j is defined as given hereafter:

$$\begin{cases} a_{j+1} \leftarrow a_j^{\lll 5} + f_j(b_j, c_j, d_j) + e_j + w_j + k_j \\ b_{j+1} \leftarrow a_j,\ c_{j+1} \leftarrow b_j^{\lll 30},\ d_{j+1} \leftarrow c_j,\ e_{j+1} \leftarrow d_j \end{cases}$$

where $(a_j, b_j, c_j, d_j, e_j) = p_j$, f_j is a bitwise function, and k_j is a constant specified by the specification.

Note that the difference between SHA-0 and SHA-1 is only the existence of the rotation in Eq.(1).

2.2 Converting pseudo-preimage attack to preimage attack

We call (H_i, M_i) a *pseudo-preimage* of the compression function, where the given H_{i+1} satisfies $H_{i+1} = \mathrm{CF}(H_i, M_i)$. Hereafter, we use the computational unit as one computation of the compression function.

[6, Fact 9.99] gives an algorithm for converting a pseudo-preimage attack to a preimage attack for the Merkle-Damgård construction. A preimage can be computed in $2^{1+(x+n)/2}$ with one more block message, where the hash value is n-bit long, when a pseudo-preimage can be computed in 2^x.

Note that the attacks [5, 3] generalize this conversion, tree and graph based approaches. Their conversions require to fix some part of hash value and pseudo-preimage in the pseudo-preimage attack, and to generate this combination at very small cost. Unfortunately, our attack described later cannot satisfy this condition. So, we cannot use tree and graph based approaches with our attacks.

2.3 Meet-in-the-middle attack

This section describes the basic strategy of the preimage attack using the meet-in-the-middle attack proposed in [1].

Assume that the message length with padding is equal to one block. The hash value is computed by $H_1 = \mathrm{CF}(\mathrm{IV}, M_0)$. Focusing on Eq.(2) reduced to s steps, we assume that some t, u, and v exist with the following conditions.

$$\begin{cases} w_j\ (0 \le j < t) \text{ is independent of } m_v \\ w_j\ (t \le j < s) \text{ is independent of } m_u \end{cases} \tag{3}$$

We can construct the following algorithm.

0. Choose m_j $(j \in \{0, 1, \ldots, 15\} \backslash \{u, v\})$ arbitrary.
1. For all $m_u \in \{0, 1\}^{32}$, compute $p_t \leftarrow R_{t-1}(R_{t-2}(\cdots R_0(\mathrm{IV}, w_0) \cdots, w_{t-2}), w_{t-1})$ and store (m_u, p_t) in a table.
2. For all $m_v \in \{0, 1\}^{32}$, compute $p_t \leftarrow R_t^{-1}(R_{t+1}^{-1}(\cdots R_{s-1}^{-1}(p_s, w_{s-1}) \cdots, w_{t+1}), w_t)$, where $p_s \leftarrow H_1 - \mathrm{IV}$ and "−" denotes the wordwise subtraction. If one of the p_ts has a match in the table generated in 1, M_0 $(= (m_0, m_1, \ldots, m_{15}))$ is a preimage of the hash function.

The complexity of the above algorithm is about 2^{32}, and the success probability is about $2^{-160+64}$. Thus, to iterate the above algorithm 2^{160-64} times, we expect to find a preimage with high probability. The time complexity of the attack is 2^{160-32} and the memory complexity is about 6×2^{32} words.

Hereafter, we call such m_u and m_v *neutral words*, and call consecutive steps $j \in [0, t)$ and $j \in [t, s)$ *chunks*. In this meet-in-the-middle attack, how to find two chunks with a neutral word is important. Section 3 describes how to find this that satisfies Condition (3) with given w_i $(i = 0, 1, \ldots, s-1)$.

2.4 Auxiliary techniques with the meet-in-the-middle attack

This section describes the techniques proposed in [1, 11] that can be used with the algorithm described in Section 2.3. These techniques improve the complexity and increase the number of steps that can be attacked by the attack described in Section 2.3.

Splice-and-cut. The meet-in-the-middle attack in Section 2.3 starts to compute input p_0 in step 0 and output p_s in step $s-1$. Considering the final addition in the Davies-Meyer mode in Eq.(2), we regard that the final and the first steps are consecutive. Thus, we can determine that the meet-in-the-middle attack starts with any step and matches with any step. We call this technique *splice-and-cut* [1]. Note that this technique will produce a *pseudo*-preimage, because IV cannot be controlled by an attacker, though we can compute a preimage using Section 2.2.

Partial-matching and partial-fixing. The step function R_j in SHA-b does not update all words in p_j. In fact, all words in p_j match p_{j+1} except one word. This fact enables us not to fix matching-step t in Section 2.3, and Condition (3) changes from the chunk partition of $[0, t)$ and $[t, s)$ to that of $[0, t)$ and $[t + c, s)$, where $c \leq 4$. This may increase the number of steps that a preimage can be computed because we may be able to include the neutral words m_u and m_v in the steps $[t, t + c)$. This loosens the conditions based on which the neutral words are selected and how the chunks are selected. We call this technique *partial-matching* [1].

Moreover, we can choose a larger c than that for partial-matching, by fixing partial bits in m_u and/or m_v, since the partial bits in p_j depending on m_u or m_v $(j \in [t, t + c))$ can be computed. We call this technique *partial-fixing* [1]. In the case of SHA-b, with manual attempts, c seems to be chosen up to ≈ 15. An example of partial-matching with partial-fixing is provided in a later section.

Initial structure. In the partial-matching or partial-fixing technique, we can ignore several steps regarding neutral words for the matching-part in the meet-in-the-middle attack to choose the appropriate chunks. Similarly, we can ignore several steps for the starting-part in the meet-in-the-middle attack. A preliminary version of the technique is introduced in [2], and it can be considered as a local collision [10] similar to existing collision attacks. Using the local collision technique, neutral words should be chosen at the edges of the starting-part. After computing the matching-part, we should confirm that the values of the neutral words satisfies the condition of the local collision. This condition is satisfied with probability 2^{-32}, and we lose the advantage to use the meet-in-the-middle attack. To solve the problem, [2] chooses additional neutral words from a chaining variable. Anyway, the condition for the neutral words is very restrictive for the local collision technique.

A variant of the local collision was introduced in [12] and generalized to the *initial structure* [11]. As opposed to [2], [11] introduced the *efficient consistency check* technique for the initial structure and it can also be used for the local collision technique. In regard to the technique in [2], the consistency for local collisions is satisfied randomly after matching the meet-in-the-middle attack, while the efficient consistency check satisfies the consistency for the initial structure at the same time as the meet-in-the-middle attack by adding a word for the table used by the meet-in-the-middle-attack.

Similar to partial-fixing, we can ignore d steps regarding neutral words for the starting part in the meet-in-the-middle attack. How to construct an initial structure is still somewhat ambiguous. With several manual attempts, it seems possible to construct d-step initial structures up to ≈ 4 for the case of SHA-b. An example of the initial structure is provided in a later section.

Summary. Considering the meet-in-the-middle attack, we can use all of the techniques described above: splice-and-cut, partial-matching and -fixing, and initial structure. Figure 1 shows how to partition the steps in SHA-b with these techniques in an abstract model.

3 Analysis of linear message schedule

The message schedule of SHA-b is different from that for MD4 and MD5, which were already attacked [5, 11], and is essentially linear for w_j $(j \geq 16)$ from Eq.(1). Similarly, HAS-160 adopts a linear message schedule, but most part of the message schedule is only permutations of message words. In fact, only one fifth of w_js are essentially linear, and this linear w_js are only XOR of 4 message words. Thus, for example, the case of $w_{16} = m_{12} \oplus m_{13} \oplus m_{14} \oplus m_{15}$ is regarded such that all of $m_{12}, m_{13}, m_{14}, m_{15}$ are used in this step in the attack [12]. While, on the message schedule of SHA-b, w_j $(0 \leq j < 16)$ is equal to m_j and seems simple, but w_j $(j \geq 20)$ depends on almost half the number of m_js since w_j $(j \geq 16)$ is computed using Eq.(1). So, it seems that we can compute a preimage up to ≈ 39 steps $(= 20 + 15 + 4)$ faster than the brute-force attack under the same strategy in [12], and it seems difficult to increase the number of steps that can be attacked. This section presents a way to address this problem, that is, the following section finds the chunks that satisfy Condition (3) and detect the neutral words in the chunks.

3.1 Kernel and neutral words

This section describes how to partition steps into chunks and find neutral words for SHA-0. For SHA-1, the same approach can be applied by considering bits instead of words.

The expanded message, w_j, is computed using Eq.(1), and its matrix representation is given hereafter: $[w_0\ w_1\ \cdots\ w_{79}]^T = WM^T$, where $M = [m_0\ m_1\ \cdots\ m_{15}]$ and W is represented in Figure 3. Consider that SHA-0 is reduced to s

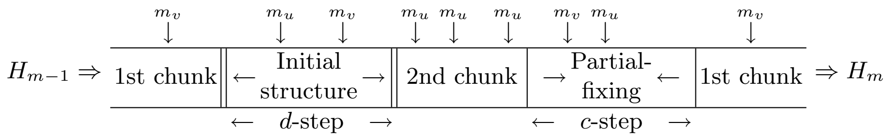


Fig. 1. A chunk partition with initial structure and partial-fixing technique

steps and the steps are partitioned into the following two chunks.

$$\begin{cases} [w_0\ w_1\ \cdots\ w_{t-1}]^T = W_1 M^T \\ [w_t\ w_{t+1}\ \cdots\ w_{s-1}]^T = W_2 M^T \end{cases} \tag{4}$$

We assume that

$$\begin{cases} \operatorname{rank} W_1 < 16 \\ \operatorname{rank} W_2 < 16 \end{cases} \tag{5}$$

holds. So, there exists the following non-trivial kernels.

$$\begin{cases} \ker W_1 = \langle k_1^{(0)}, k_1^{(1)}, \ldots, k_1^{(\kappa_1-1)} \rangle \\ \ker W_2 = \langle k_2^{(0)}, k_2^{(1)}, \ldots, k_2^{(\kappa_2-1)} \rangle \end{cases}, \tag{6}$$

where κ_1 and κ_2 denote the dimension of the corresponding kernel. Let $K_1 = [k_1^{(0)}\ k_1^{(1)}\ \cdots\ k_1^{(\kappa_1-1)}]$ and $K_2 = [k_2^{(0)}\ k_2^{(1)}\ \cdots\ k_2^{(\kappa_2-1)}]$. We regard the message words corresponding to the vectors in K_1 and K_2 as neutral words for the opposite chunk. Consider the following as an example. $\kappa_1 = \kappa_2 = 1$ and

$$\begin{cases} k_1 = [1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T \\ k_2 = [0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T \end{cases}.$$

Since k_1 and k_2 are in the kernel of W_1 and W_2, $W_1 k_1 = 0$ and $W_2 k_2 = 0$ holds. That is, m_0 can be used as a neutral word for the second chunk with $m_0 = m_2 = m_3$ to see '1' in k_1, and m_1 can be used as a neutral word for the first chunk with $m_1 = m_4$ to see '1' in k_2. Similarly, we can choose neutral words whenever the representation of the generating vectors does not share the same position of '1's. However, the strategy does not always work in a straightforward manner. We notice the case that the generating vectors share the '1' at the same position. In this case, we cannot independently determine the value of neutral words for each chunk. We can solve this problem using a sophisticated linear transformation by substituting M with M', where $M^T = RM'^T$ with regular matrix R. Once we find M', we can easy to recover the preimage M by multiplying the matrix R.

Let the unit vector be $\mathbf{e}_i = [0\ \cdots\ \overset{i}{\breve{1}}\ \cdots\ 0]^T$ and j-dimensional identity matrix be E_j. In the following, we construct regular matrix R such that

$$\begin{cases} W_1 R \mathbf{e}_i \quad\ = 0 \quad \text{for } i = 0, 1, \ldots, \kappa_1 - 1 \\ W_2 R \mathbf{e}_{i+\kappa_1} = 0 \quad \text{for } i = 0, 1, \ldots, \kappa_2 - 1 \end{cases}. \tag{7}$$

If such a matrix is constructed, we have

$$[w_0\ w_1\ \cdots\ w_{s-1}]^T = \begin{bmatrix} W_1 \\ \hline W_2 \end{bmatrix} M^T = \left(\begin{bmatrix} W_1 \\ \hline W_2 \end{bmatrix} R \right) (R^{-1} M^T).$$

Let $W'_1 = W_1 R$, $W'_2 = W_2 R$, $M'^T = R^{-1} M^T$, and $M' = [m'_0\ m'_1\ \cdots\ m'_{15}]$, and we have

- $\ker W'_1 = \langle \mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{\kappa_1 - 1} \rangle$, and $\ker W'_2 = \langle \mathbf{e}_{\kappa_1}, \mathbf{e}_{\kappa_1+1}, \ldots, \mathbf{e}_{\kappa_1+\kappa_2-1} \rangle$.
- $m'_0, m'_1, \ldots, m'_{\kappa_1-1}$ are neutral words for the second chunk, and $m'_{\kappa_1}, m'_{\kappa_1+1}, \ldots, m'_{\kappa_1+\kappa_2-1}$ are neutral words for the first chunk.

Thus, we can perform the meet-in-the-middle attack described in Section 2.3 by adjusting a recovered preimage M' with $M^T \leftarrow RM'^T$. The rest of this section describes how to construct R.

Assume $\mathrm{rank}[K_1\ K_2] = \kappa_1 + \kappa_2$[2]. We can choose $\kappa_1 + \kappa_2$ independent row vectors in $[K_1\ K_2]$, and there is regular matrix T that collects these independent row vectors and can be constructed from E_{16} by swapping corresponding rows, H, at the top by swapping rows, and regular matrices B and S are defied as follows.

$$\left[\begin{array}{c} H \\ \hline \\ * \\ \\ \end{array}\right] = T[K_1\ K_2],\ B = \left[\begin{array}{c|c} H^{-1} & 0 \\ \hline 0 & \\ & E_{16-\kappa_1-\kappa_2} \end{array}\right],\ S = \left[\begin{array}{c|c} BT[K_1\ K_2] & 0 \\ \cline{2-2} & \\ & E_{16-\kappa_1-\kappa_2} \end{array}\right].$$

Note that the top $\kappa_1 + \kappa_2$ rows of $BT[K_1\ K_2]$ is $E_{\kappa_1+\kappa_2}$. Then, $R = T^{-1}B^{-1}S$ satisfies $k_1^{(i)} = R\mathbf{e}_i \quad (\text{for } 0 \le i < \kappa_1)$ and $k_2^{(i)} = R\mathbf{e}_{i+\kappa_1} \quad (\text{for } 0 \le i < \kappa_2)$. So, Eq.(7) holds.

3.2 Notes on auxiliary techniques

Both the splice-and-cut and partial-matching techniques described in Section 2.4 can be used in the same way. Note, we generate pseudo-preimages in the same way as Section 2.4, because the splice-and-cut technique cannot specify IV.

We can apply the partial-fixing and initial structure techniques described in Section 2.4 to SHA-b in a similar way. However, careful analysis is required, since the message schedule of SHA-b sometimes produces XOR of several message words in one step.

[12] applies the partial-fixing technique to HAS-160. The step function of HAS-160 is very similar to SHA-b, so these techniques can also be applied to SHA-b.

3.3 Application to SHA-b

Based on the discussion above, we compute how many steps to satisfy Condition (5), with partial-matching and -fixing step $c \le 21$ and initial structure step $d \le 7$. The results are shown in Tables 2, 3, and 4. We do not know why, but we notice that the numbers of steps are the same when the values of $c + d$ are the same.

[2] Of course, there is a chunk partition such that $\mathrm{rank}[K_1\ K_2] < \kappa_1 + \kappa_2$; however, we are not so interested in this case. Actually, we do not have an experience with $\mathrm{rank}[K_1\ K_2] < \kappa_1 + \kappa_2$ with long steps.

Table 2. Number of Steps Such That $\mathrm{rank}\, W_1,\ \mathrm{rank}\, W_2 < 16$ for SHA-0

$c+d$	0	1–2	3	4–6	7	8–11	12–13	14–15	16–21	22	23	24–25	26–27	28
# of steps	32	33	35	37	39	42	45	47	52	54	55	57	60	61

c: number of partial-fixing step, d : number of initial structure step

Table 3. Number of Steps Such That $\mathrm{rank}\, W_1,\ \mathrm{rank}\, W_2 < 512$ for SHA-1

$c+d$	0	1–2	3	4–6	7	8–11	12–13	14–15	16–21	22	23	24–25	26–27	28
# of steps	32	33	35	37	39	42	45	47	52	54	55	57	60	61

c: number of partial-fixing step, d : number of initial structure step

For SHA-1, $\mathrm{rank}\, W_1, \mathrm{rank}\, W_2 < 512$ is a very hard condition to attack SHA-1, because we may be able to use only one neutral bit. In this case the partial-fixing technique cannot work. So, we also compute the case that $\mathrm{rank}\, W_1,\ \mathrm{rank}\, W_2 < 503$ to have the possibility to use the partial-fixing technique. Though we loose the upper bound of the rank to 503, the derived ranks are 480.

Consider the case for SHA-0. If the number of steps in chunks are 15, $\mathrm{rank}\, W_1, \mathrm{rank}\, W_2 < 16$ always holds. To set $d = 0$ and $c = 4$, that is, we do not use initial structure and partial-fixing, and the attack always works. Thus, we can trivially compute a preimage of the compression function reduced to 34 $(= 15 + 15 + 4)$ steps in 2^{128}. To see Table 2, we see 37 when $c + d = 4$. So, we can improve the attack to 37 steps with the same complexity. Consider to adopt the partial-fixing technique. To fix lower 16 bits in neutral words, it is easy to verify that we can increase 3 more steps. Following Table 2 with $c + d = 7$, we can compute a preimage of the compression function reduced to 39 steps in 2^{144}.

Note that Condition (5) is the only necessary condition for a successful attack. To construct a definite attack procedure, we need to see specific procedures for the initial structure, and for partial-fixing, and for padding. The following section describes this.

4 Detailed attack against SHA-b

This section describes detailed description of the attack against SHA-0 reduced to 52 steps. We try to increase the number of steps that can be attacked faster than the brute-force attack as large as possible. For smaller number of steps, see the previous section.

4.1 Chunk partition for 52-step SHA-0

The transformed message schedule, WR, described in the previous section is shown in Table 5. As shown in Table 5, the first chunk (steps 37–23, in total 15

Table 4. Number of Steps Such That $\mathrm{rank}\, W_1,\ \mathrm{rank}\, W_2 < 503$ for SHA-1

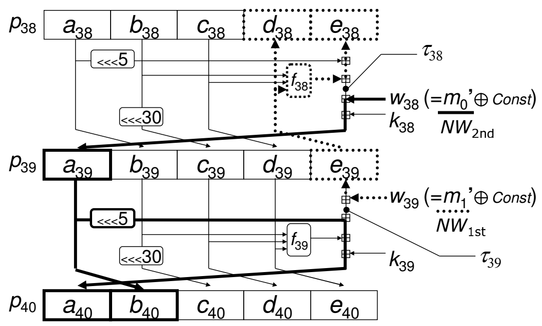
$c+d$	0–1	2	3–5	6	7	8–9	10–11	12–13	14–15	16–17	18	19	20–21	22–24	25	26–27	28
# of steps	31	33	35	36	37	39	41	43	45	47	48	49	51	54	56	57	59

c: number of partial-fixing step, d : number of initial structure step

Table 5. Transformed Message Schedule for 52-step SHA-0

Step	m'_i: 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Second chunk
1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
2	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
5	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
8	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
9	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Skip
10	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
11	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
12	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
13	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
16	1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	
17	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	
18	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	1	
19	0	0	1	1	0	1	0	0	1	0	0	1	0	1	0	0	
20	1	0	0	1	1	0	1	0	0	1	0	0	1	0	1	0	
21	0	0	1	0	1	1	0	1	0	0	1	0	0	1	0	1	
22	1	0	1	1	0	1	1	0	0	0	0	1	0	1	1	0	
23	0	1	0	1	1	0	1	1	0	1	0	0	1	0	1	1	First chunk
24	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1	
25	0	1	1	0	0	1	1	0	0	1	0	0	0	1	0	0	
26	0	0	1	1	0	0	1	1	0	1	0	0	0	0	1	0	
27	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	1	
28	0	0	1	1	1	1	0	0	0	0	0	0	0	1	0	0	
29	0	1	0	1	1	1	1	0	0	1	1	0	0	0	1	0	
30	0	1	1	0	1	1	1	1	0	0	0	1	0	0	0	1	
31	0	0	1	1	0	1	1	1	0	0	0	0	1	1	0	0	
32	0	1	0	1	1	0	1	1	1	1	0	0	0	1	1	0	
33	0	1	1	0	1	1	0	1	1	0	0	0	0	0	1	1	
34	0	1	1	1	0	1	1	0	0	0	1	0	0	1	0	1	
35	0	1	1	1	1	0	1	1	1	1	1	1	0	1	1	0	
36	0	1	1	1	1	1	0	1	1	1	0	1	1	0	1	1	
37	0	1	0	1	1	1	1	0	0	0	1	0	1	0	0	1	↑
38	1	0	1	0	1	1	1	1	1	1	1	1	0	0	0	0	IS
39	0	1	1	1	0	1	1	1	1	1	0	1	1	0	0	0	
40	0	0	1	1	1	0	1	1	1	0	1	0	1	1	0	0	↓ Second chunk
41	1	0	0	1	1	1	0	1	1	0	1	1	0	1	1	0	
42	0	0	0	0	1	1	1	0	1	0	1	1	1	0	1	1	
43	0	0	1	0	0	1	1	1	1	0	1	1	1	0	0	1	
44	1	0	1	1	0	0	1	1	0	1	1	1	1	0	0	0	
45	0	0	1	1	1	0	0	1	1	1	1	1	1	1	0	0	
46	1	0	1	1	1	1	0	0	1	0	0	1	1	1	1	0	
47	1	0	0	1	1	1	1	0	0	0	1	0	1	1	1	1	
48	0	0	1	0	1	1	1	1	1	0	1	1	0	0	1	1	
49	0	0	1	1	0	1	1	1	0	1	0	1	1	1	0	1	
50	1	0	0	1	1	0	1	1	0	1	1	0	1	0	1	0	
51	0	0	1	0	1	1	0	1	1	1	0	1	0	1	0	1	

w_i consists of XOR of m'_j whose entry in step i is 1, e.g., $w_{10} = m'_1 \oplus m'_{10}$. IS, which appears in steps 38 and 39, stands for "Initial Structure."



The bold and dotted lines represent data lines for which values are changed depending on the value of the neutral words for the second and first chunks, respectively. Narrow lines represent data lines that are always fixed regardless of the values of the neutral words.

Fig. 2. Initial structure for 52-step SHA-0

steps) includes m'_1 but does not include m'_0, and the second chunk (steps 40–51, 0–8 in total 21 steps) includes m'_0 but does not include m'_1. Hence, by fixing m'_2 to m'_{15}, the meet-in-the-middle attack can be performed.

4.2 Initial structure for 52-step SHA-0

The construction of the initial structure is shown in Fig. 2. The goal of this construction is making p_{40} independent of the neutral words for the first chunk w_{39} $(= m'_1 \oplus Const)$, and making p_{38} independent of the neutral words for the second chunk w_{38} $(= m'_0 \oplus Const)$. This is achieved by the following procedure.

Preparation: Change the addition order in step 39, and choose an arbitrary value for τ_{38} and τ_{39}, e.g. $\tau_{38} = \tau_{39} = 0$. Moreover, fix a_{38}, b_{38}, c_{38} as arbitrary.

Table 6. Number of Known Bits in Partial-Fixing Technique for 52-Step SHA-0

j	a_j	b_j	c_j	d_j	e_j	(in w_j) m'_1	#cands of a_j	
9	All	All	All	All	All	18–0		Forward
10	18–0	All	All	All	All	18–0	1	
11	18–5	18–0	All	All	All	18–0	2^1	
12	<u>18–10</u>	<u>18–5</u>	16–0	All	All	18–0	2^2	
12	<u>20–9</u>	<u>20–11</u>	?	?	?	skipped		Backward computation
13	20–9	20–9	18–9	?	?	skipped		
14	20–9	20–9	18–7	18–9	?	skipped		
15	20–4	20–9	18–7	18–7	18–9	All	2^9	
16	20–2	20–4	18–7	18–7	18–7	18–0	2^7	
17	20–2	20–2	18–2	18–7	18–7	All	2^5	
18	20–2	20–2	18–0	18–2	18–7	18–0	2^3	
19	All	20–2	18–0	18–0	18–2	All	2^1	
20	All	All	18–0	18–0	18–0	18–0	1	
21	All	All	All	18–0	18–0	All	1	
22	All	All	All	All	18–0	18–0	1	
23	All	All	All	All	All			
j	a_j	b_j	c_j	d_j	e_j	(in w_j) m'_0	#cands of e_j	

Numbers denote the known bits of each chaining variable. Underlined variables in $j = 12$ are variables where we compare the results of two chunks.

Table 7. Number of Known Bits in Partial-Fixing Technique for 48-Step SHA-1

j	a_j	b_j	c_j	d_j	e_j	(in w_j) m'_1	#cands of a_j	
9	All	All	All	All	All	17–0		Forward
10	17–0	All	All	All	All	All	1	
11	17–5	17–0	All	All	All	All	2^1	
12	<u>17–10</u>	<u>17–5</u>	15–0	All	All	16–0	2^2	
12	<u>27–9</u>	<u>27–11</u>	?	?	?	skipped		Backward computation
13	27–7	27–9	25–9	?	?	skipped		
14	27–9	27–7	25–7	25–9	?	skipped		
15	27–4	27–9	25–7	25–7	25–9	All	2^9	
16	27–2	27–4	25–7	25–7	25–7	25–0	2^7	
17	27–2	27–2	25–2	25–7	25–7	All	2^5	
18	27–2	27–2	25–0	25–2	25–7	25–0	2^3	
19	All	27–2	25–0	25–0	25–2	All	2^1	
20	All	All	25–0	25–0	25–0	25–0	1	
21	All	All	All	25–0	25–0	All	1	
22	All	All	All	All	25–0	25–0	1	
23	All	All	All	All	All			
j	a_j	b_j	c_j	d_j	e_j	(in w_j) m'_0	#cands of e_j	

We compare results of two chunks on a_{12} and b_{12}, in total 15 bits.

Make p_{40} independent of w_{39}: c_{40} $(= a_{38}{}^{\lll 30})$, d_{40} $(= b_{38}{}^{\lll 30})$, and e_{40} $(= c_{38})$ are already fixed. Compute b_{40} $(= a_{39} = \tau_{38} + w_{38} + k_{38})$ and a_{40} $(= \tau_{39} + a_{39}{}^{\lll 5} + f_{39}(b_{39}, c_{39}, d_{39}) + k_{39})$. Note that $b_{39} = a_{38}$, $c_{39} = b_{38}{}^{\lll 30}$, and $d_{39} = c_{38}$.

Make p_{38} independent of w_{38}: Compute d_{38} $(= \tau_{39} - w_{39})$ and e_{38} $(= \tau_{38} - f_{38}(b_{38}, c_{38}, d_{38}) - a_{38}{}^{\lll 5})$.

As described above, we can compute the first and second chunks independently of the neutral words for the second and first chunks, respectively. Hence, the meet-in-the-middle attack can be performed.

Remarks. Construction of the initial structure is dependent on the selected chunks. Since the chunk partition is different for SHA-1, we construct the initial structure of SHA-1 differently. See Section 5 for details.

4.3 Partial-fixing technique for 52-step SHA-0

In the meet-in-the-middle attack, results of two chunks must be compared efficiently. Although many steps (14 steps) between two chunks are skipped in the employed attack as shown in Table 5, a part of the results of two chunks can be compared by using the partial-fixing and partial-matching techniques. How the results of two chunks are compared is explained in Table 6. Note we first assumed that the fixed bit-positions for backward computation is represented by lower x bits and the forward computation is represented by intermediate y bits. Then, we identified the best x, y, and fixed positions. Consequently, we chose $x = 19$ and $y = 19$ from the least significant bit.

We explain how the partial computation shown in Table 6 is processed.

Forward computation for a_{10}: As a result of computing the second chunk in forward direction m'_0, we obtain the value p_9. Therefore, when we apply partial-fixing to the forward computation, we know all bits of a_9, b_9, c_9, d_9 and e_9. p_{10} is computed with $R_9(p_9, w_9)$, where w_9 can be written as $m'_1 \oplus Const$. Since the lower 19 bits of m'_1, which is the neutral word for the other chunk, are fixed, the lower 19 bits of a_{10} can be computed uniquely.

Forward computation for a_{11}: p_{11} is computed with $R_{10}(p_{10}, w_{10})$, where w_{10} can be written as $m'_1 \oplus Const$. In particular, the equation for a_{11} is as follows:

$$a_{11} = a_{10}{}^{\lll 5} + f_{10}(b_{10}, c_{10}, d_{10}) + e_{10} + w_{10} + k_{10}.$$

Since the lower 19 bits of w_{10} and all bits of f_{10}, e_{10}, and k_{10} are known, the lower 19 bits of $f_{10} + e_{10} + w_{10} + k_{10}$ can be computed uniquely. We know the lower 19 bits (bits 0 to 18) of a_{10}, hence we know bits 5 to 23 of $a_{10}{}^{\lll 5}$. When we compute $a_{11} = a_{10}{}^{\lll 5} + (f_{10} + e_{10} + w_{10} + k_{10})$, we do not know if there is a carry from bit-position 4 to 5. Therefore, we consider both possible carry bits, and obtain two candidates for bits 5 to 18 of a_{11}. Hence, for each $(a_9, b_9, c_9, d_9, e_9)$, we obtain 2^1 candidates for bits 5 to 18 of a_{11}.

Forward computation for a_{12}: By almost the same procedure as above, we can obtain two candidates for bits 10 to 18 a_{12} for each candidate of p_{11}. Hence, for each $(a_9, b_9, c_9, d_9, e_9)$, we obtain 2^2 candidates for bits 10 to 18 of a_{12}.

Backward computation for e_{22}: As a result of computing the first chunk in backward direction m'_1, we obtain the value of p_{23}. p_{22} is computed with $R_{22}^{-1}(p_{23}, w_{22})$, where w_{22} can be written as $m'_0 \oplus Const$. Since the lower 19 bits of m'_0 are fixed, the lower 19 bits of e_{22} can be computed uniquely.

Backward computation for e_{17}: With similar techniques to the forward computation, we can compute 2^3 candidates for p_{18} as shown in Table 6 for each p_{23}. We next explain how to compute p_{17} with $R_{17}^{-1}(p_{18}, w_{17})$, in particular,

$$\begin{aligned} e_{17} &= a_{18} - k_{17} - w_{17} - f_{17}(c_{18}{}^{\ggg 30}, d_{18}, e_{18}) - b_{18}{}^{\lll 5}, \\ w_{17} &= m'_1 \oplus Const = m'_1 \oplus m'_3 \oplus m'_9 \oplus m'_{14}. \end{aligned}$$

In order to reduce the number of unknown carries, the number of additions (or subtractions) should be reduced as much as possible. For this purpose, we fix the lower 19 bits of w_{17} to $-k_{17}$. This can be achieved by first fixing the lower 19 bits of $m'_1 \oplus m'_3 \oplus m'_9$, and then compute $m'_{14} = m'_1 \oplus m'_3 \oplus m'_9 \oplus (-k_{17})$ with respect to the lower 19 bits.

Remarks for the rest: In a similar manner, we obtain Table 6. Note, we need to fix $w_{16} = m'_0 \oplus m'_1 \oplus m'_2 \oplus m'_8 \oplus m'_{13}$ to $-k_{16}$ and $w_{15} = m'_{15}$ to $-k_{15}$ with respect to the lower 19 bits. With adequate message space, this can be easily achieved. Backward computation is done until p_{15}. Steps 14-11 are skipped in the partial-matching technique. Finally, we compare the results from both chunks at bits 10–18 of a_{12} and bits 11–18 of b_{12}, in total 17 bits.

4.4 Attack procedure for 52-step SHA-0

For a given hash value, H_m, the attack procedure is as follows.

1. Fix $m'_i, (i \notin \{0,1\})$ and the lower 19 bits of m'_0 and m'_1 to randomly chosen values.
2. Fix chaining variables in the initial structures (steps 38-39) as described in Section 4.2.
3. For all 13 free bits of the neutral words for the second chunk, namely the higher 13 bits of m'_0,
   (a) Compute a_{40} and b_{40} from w_{38} as explained in Section 4.2.
   (b) Compute: $\begin{cases} p_{j+1} \leftarrow R_j(p_j, w_j) & \text{for } j = 40, 41, \ldots, 51 \\ p_0 \quad \leftarrow H_m - p_{52}, & \\ p_{j+1} \leftarrow R_j(p_j, w_j) & \text{for } j = 0, 1, \ldots, 8 \end{cases}$
   (c) Compute bits 0–18 of a_{10}, 2^1 candidates for bits 5–18 of a_{11} and 2^2 candidates for bits 10–18 of a_{12} as explained in Section 4.3.
   (d) Make a table of $(m'_0, p_9, a_{10}, a_{11}, a_{12})$s. Since we have 13 free bits in neutral words, and 2^2 candidates of partial a_{12} for each choice of free bits, we have 2^{15} items in the table.
4. For all 13 free bits of the neutral words for the first chunk, namely the higher 13 bits of m'_1,
   (a) Compute e_{38} and d_{38} as described in Section 4.2.
   (b) Compute: $p_j \leftarrow R_j^{-1}(p_{j+1}, w_j) \qquad$ for $j = 37, 36, \ldots, 23$,
   (c) Compute the lower 19 bits of e_{22}, e_{21}, and e_{20}, bits 2–18 of e_{19}, bits 7–18 of e_{18}, e_{17}, and e_{16}, and bits 9–18 of e_{15} as explained in Section 4.3.
   (d) For each item in the table, check whether or not bits 10–18 of a_{12}, and bits 11–18 of b_{12} computed from both chunks match.
   (e) If a match is found, compute p_{10} to p_{13} by the corresponding message word, and check the match of the additionally computed bits, and check the correctness of the guess for the carry for a_{11} and a_{12} step by step.
   (f) If a match is found, compute p_{22} to p_{11} by the corresponding message word, and check whether all values from both chunks match and check the correctness of the guess for the carry for e_{19} to e_{15}.
   (g) If all bits match, the pair of the corresponding message and p_0 is a pseudo-preimage.

4.5 Complexity estimation for 52-step SHA-0

Assume the complexity for computing 1 step is $\frac{1}{52}$ 52-step SHA-0 compression function, and the memory access cost is negligible compared with the cost of the computation of the step function.

- The complexity of Steps 1 and 2 are negligible.
- The complexity of Step 3a is approximately $2^{13} \cdot \frac{2}{52}$.
- The complexity of Step 3b is approximately $2^{13} \cdot \frac{21}{52} (= 2^{13} \cdot \frac{12}{52} + 2^{13} \cdot \frac{9}{52})$.
- The complexity of Step 3c is approximately $2^{13} \cdot \frac{7}{52}$ $(=2^{13}(\frac{1}{52} + 2^1 \cdot \frac{1}{52} + 2^2 \cdot \frac{1}{52}))$.
- The complexity of Step 4a is approximately $2^{13} \cdot \frac{2}{52}$.
- The complexity of Step 4b is $2^{13} \cdot \frac{15}{52}$.
- The complexity of Step 4c is approximately $2^{13} \cdot \frac{685}{52}$ $(=2^{13}(\frac{1}{52} + \frac{1}{52} + \frac{1}{52} + 2^1 \cdot \frac{1}{52} + 2^3 \cdot \frac{1}{52} + 2^5 \cdot \frac{1}{52} + 2^7 \cdot \frac{1}{52} + 2^9 \cdot \frac{1}{52}))$.
- The first chunk produces $2^{22} (= 2^{13} \cdot 2^9)$ items. Therefore, at Step 4d, $2^{37} (= 2^{22} \cdot 2^{15})$ pairs are compared and 2^{20} $(=2^{37} \cdot 2^{-17})$ pairs will remain.

- At Step 4e, the complexity of computing p_{10} and p_{11} is approximately $2^{20} \cdot \frac{2}{52}$. Then, by comparing two additional bits of a_{11} (bit-positions 19 and 20) and checking the correctness of the 1 guess for the carry for a_{11}, the number of remaining pairs becomes 2^{17} ($=2^{30} \cdot 2^{-3}$). The complexity of computing p_{12} is approximately $2^{17} \cdot \frac{1}{52}$ and by comparing three additional bits of a_{12} (bit-positions 9, 19, and 20) and checking the correctness of the 1 guess of carry for a_{12}, the number of remaining pairs becomes 2^{13} ($=2^{17} \cdot 2^{-4}$). The complexity of computing p_{13} is approximately $2^{13} \cdot \frac{1}{52}$ and by comparing twelve additional bits of a_{13} (bit-positions 9-20), the number of remaining pairs becomes 2^{1} ($=2^{13} \cdot 2^{-12}$).
- Complexity for Step 4f is negligible since the number of remaining pairs is sufficiently reduced compared to the previous part. By checking the correctness of the guesses of carry for e_{22} to e_{15}, the number of remaining pair becomes 2^{-8} ($=2^{1} \cdot 2^{-9}$). By checking the entire p_{13} for matches, the number of remaining pair becomes 2^{-134} ($=2^{-8} \cdot 2^{-126}$). Therefore, by repeating this attack 2^{134} times, we can expect to find a pseudo-preimage.
  The complexity for Step 3 is $2^{13}(\frac{2}{52} + \frac{21}{52} + \frac{7}{52}) = 2^{13} \cdot \frac{30}{52}$. The complexity for Step 4 is $2^{13}(\frac{2}{52} + \frac{15}{52} + \frac{685}{52}) + 2^{20} \cdot \frac{2}{52} + 2^{17} \cdot \frac{1}{52} + 2^{13} \cdot \frac{1}{52} = 2^{13}(\frac{702}{52} + \frac{256}{52} + \frac{16}{52} + \frac{1}{52}) = 2^{13} \cdot \frac{975}{52}$. Hence, we can find a pseudo-preimage with a complexity of $2^{13}(\frac{30}{52} + \frac{975}{52}) \cdot 2^{134} = 2^{17.20} \cdot 2^{134} \approx 2^{151.2}$. This can be converted to the preimage attack with a complexity of $2^{156.6}$ by using the algorithm described in Section 2.2.

In this attack, we use a memory to store 2^{15} $(m'_0, p_9, a_{10}, a_{11}, a_{12})$s in Step 3d. Therefore, the memory complexity of this attack is approximately $2^{15} \times 9$ words. To apply the technique [6, Remark 9.93], we can convert our attack into memoryless version. The converted attack requires negligible memory and finds a pseudo-preimage in $2^{152.2}$. The attack is converted to the preimage attack by using the algorithm described in Section 2.2 with complexity of $2^{157.1}$ with $5 \times 2^{3.9}$ words of memory, which is negligible.

4.6 Padding issue

In the attack described above, we use R in Figure 4. Focusing on the padding part with $M^T = RM'^T$, we should satisfy the padding rule with

$$m_{13} = m'_1 \oplus m'_{13}, \qquad m_{14} = m'_{14}, \qquad m_{15} = m'_{15}. \tag{8}$$

- $m'_{14} \leftarrow 0$. When m'_{14} is used in the partial-fixing technique, m'_{14} is XORed with other m'_js. So, there is room to fix m'_{14}. This means the number of message strings is less than 2^{32} bits.
- Set the least significant bit of m_{13} ($= m'_1 \oplus m'_{13}$) to '1'. Although m'_1 is a neutral word, the partial-fixing technique fixes the least significant 20 bits. By appropriately setting the least significant bit of m'_{13}, this condition is satisfied.
- $m'_{15} \bmod 2^9 \leftarrow 447$. This agrees with the padding rule for m_{13}. However, we specified $m'_{15} = -k_{15}$ in the attack procedure when we perform the partial-fixing technique for step 15. To observe $m'_{15} + k_{15}$, the least significant three

$$W = \begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0\\
 & & & & & & & &\vdots& & & & & & &
\end{bmatrix}$$

Fig. 3. SHA-0 Message Schedule in Matrix Form

$$R = \begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}$$

Fig. 4. Linear Transformation Used in the Attack of Reduced SHA-0

bits are zero. Additionally, since we know 21-2 bits of a_{16}, we can determine the carry from the 8th bit to the 9th bit. This is the same effect as setting $m'_{15} = -k_{15}$.

In conclusion, we can compute a pseudo-preimage following the padding rule at the same complexity, $2^{151.2}$ as describe above, and we can compute a 2-block preimage with the regular padding in $2^{156.6}$.

Note, even if the padding rule cannot be satisfied, the attack is valid as a second-preimage attack.

5 Attack sketch for 48-step SHA-1

This section describes the sketch of the attack against SHA-1 reduced to 48 steps.

5.1 Chunk partition

Let E be E_{32}. The transformed message schedule, $W' = WR$, is shown in Table 8. In SHA-1, the size of W' is 512. We searched for chunk partition of 48-step SHA-1, and found the pattern where κ_1 and κ_2 in Eq.(6) are 32. When we attack SHA-1, we use the first 64 bits of M' as neutral words, and fix the other 448 bits. Hence, we show only the first 64 bits of W'.

5.2 Initial structure and partial-fixing technique

The construction of the initial structure is shown in Fig. 5. To fix the output of f_2, we use the cross absorption property presented by [11]. We manually optimized the number of n in the initial structure shown in Fig. 5 by considering the efficiency of the partial-fixing technique together. As a result, we select $n = 24$.

The partial-fixing technique for 48-step SHA-1 skips 14 steps as shown in Table 8. How the results of two chunks are compared is explained in Table 7. In

Table 8. Transformed Message Schedule for 48-step SHA-1

Step	1st 32 cols of W'	2nd 32 cols of W'	
0	$E \oplus E^{\lll 2}$	0	IS
1	0	E	
2	E	0	
3	0	E	
4	$E^{\lll 1}$	0	$\downarrow$ Second chunk
5	0	0	
6	E	0	
7	E	0	
8	$E^{\lll 1}$	0	
9	0	0	
10	E	0	
11	E	0	
12	$E^{\lll 1}$	0	
13	0	0	
14	0	0	
15	E	0	
16	$E \oplus E^{\lll 1}$	0	
17	0	0	
18	E	0	

Step	1st 32 cols of W'	2nd 32 cols of W'	
19	E	$E^{\ggg 1}$	Skip
20	0	0	
21	0	0	
22	E	$E^{\ggg 2}$	
23	0	0	
24	0	0	
25	0	$E^{\ggg 3}$	
26	E	0	
27	0	$E^{\ggg 2}$	
28	E	$E^{\ggg 4}$	
29	0	0	
30	E	0	
31	0	$E^{\ggg 5}$	
32	E	0	
33	0	$E^{\ggg 2} \oplus E^{\ggg 4}$	First chunk
34	0	$E^{\ggg 6}$	
35	0	$E^{\ggg 2} \oplus E^{\ggg 3}$	
36	0	0	
37	0	$E^{\ggg 7}$	
38	0	$E^{\ggg 4}$	
39	0	$E^{\ggg 4} \oplus E^{\ggg 6}$	
40	0	$E^{\ggg 8}$	
41	0	$E^{\ggg 4}$	
42	0	0	
43	0	$E^{\ggg 4} \oplus E^{\ggg 9}$	
44	0	0	
45	0	$E^{\ggg 6} \oplus E^{\ggg 8}$	
46	0	$E^{\ggg 10}$	
47	0	$E^{\ggg 3} \oplus E^{\ggg 6} \oplus E^{\ggg 11}$	$\uparrow$

The second and the third columns of the table show the first and second 32 columns of W'. In each step, 32 rows of W' are related. Hence, each entry of the table denotes corresponding 32×32 submatrix of W'.
Since all 'j's of E_j used in this table are 32, we simply write E to denote E_{32}.

forward computation, we fix the lower 18 bits of m'_1, and in backward computation, we fix the lower 26 bits of m'_0. Finally, bit positions 10 to 17 of a_{12} and bit positions 11 to 17 of b_{12}, in total 15 bits, are compared.

5.3 Summary of attack

In this attack, a_4 and m'_0 are the neutral words for the second chunk where, m'_0 is the first 32 bits of M'. Similarly, b_0 and m'_1 are the neutral words for the first chunk, where, m'_1 is the second 32 bits of M'.

To construct the initial structure appropriately and apply the partial-fixing technique efficiently, we need to fix a part of neutral words. In the first chunk, we fix bit positions 26, 27, 28, 29, 30, 31, 0, and 1, in total 8 bits, of b_0s. This results in fixing the upper 8 bits of c_1, which is necessary for the initial structure. We also fix bit positions 1 to 18 of, in total 18 bits, of m'_1. This results in fixing the lower 18 bits of w_{19}, which is a message word used in the first step in the partial-fixing technique in forward direction. Note, the number of unfixed bits in neutral words for the first chunk is 38. In the second chunk, we fix the lower 26 bits of m'_0. This result in fixing the lower 24 bits of w_0 $(= (E \oplus E^{\lll 2}) \times m'_0)$, which is required for the initial structure, and fixing the lower 24 bits of w_j $(= E), j \in \{32, 30, 28, 26\}$, which is required for the partial-fixing technique.

We roughly estimate the complexity of the attack. Considering the unknown carries in the partial-fixing, the meet-in-the-middle attack examines the match

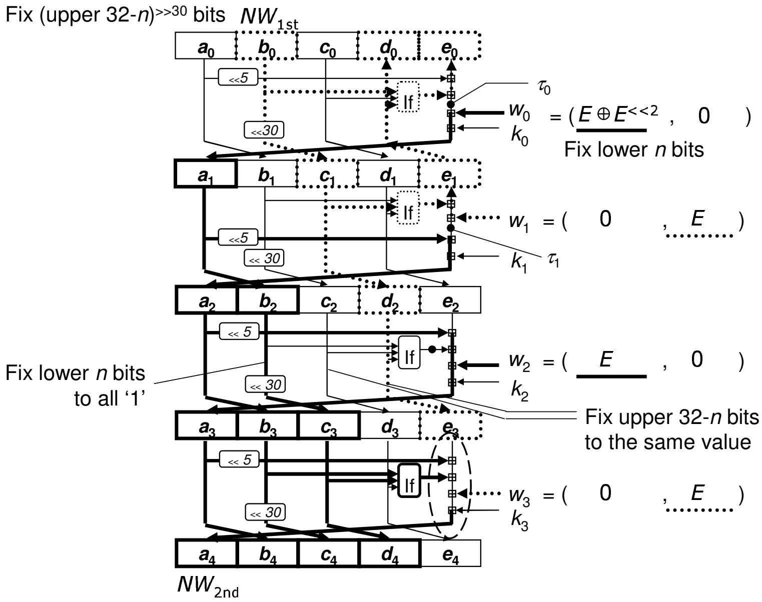


Fig. 5. Initial structure for 48-step SHA-1

Note that we perform the efficient consistency check described in the initial structure part of Section 2.4 in dashed circle in the figure.

of 2^{87} $(= 2^{38+2} \times 2^{38+9})$ pairs. Unfortunately, since we can only match 47 bits, the number of resulting pairs are $2^{40} \gg 2^{38}$. So, the attack requires more time than the brute-force attack. To reduce the time complexity, we analyze the probabilistic behavior of carry propagation in the partial-fixing. Observing Table 7, we notice that we can estimate the existence of carry with a probability higher than $1/2$ for several additions with unknown carry. In the backward computation, we can estimate the existence of carry with a probability higher than $3/4$ for two cases. Thus, the meet-in-the-middle attack examines the match of 2^{85} $(= 2^{38+2} \times 2^{38+7})$ pairs. Since the matching bit is 47 bits, the number of resulting pairs are $2^{38} \approx 2^{38}$. So, the time complexity for the dominant part is computing the chunks, and is approximately 2^{39} and the success probability is approximately $2^{-117.7}$ $(= 2^{32+6+6-160} \times (3/4)^2)$. To iterate the above procedure $2^{117.7}$ times, we find a pseudo-preimage with high probability, and the total time complexity is approximately $2^{156.7}$ $(= 2^{39} \times 2^{117.7})$. Consider a second preimage attack whose block is longer than 3. applying the above attack to the second block, and using the conversion described in Section 2.2, a preimage will be found in $2^{159.3}$.

In this attack, we use a memory to store 2^{40} items. Therefore, the memory complexity is approximately $2^{40} \times 11$ words.

6 Conclusion

This paper proposes a method for analyzing the linear message schedule in SHA-0 and SHA-1 for a preimage attack using the meet-in-the-middle attack. Thanks to recently developed auxiliary techniques such as splice-and-cut, partial-fixing, and initial structure, the results of the application of the proposed method can be used to compute preimages of reduced SHA-0 and SHA-1 up to 52 and 48 steps, respectively, faster than the brute-force attack. The results shows that the meet-in-the-middle attack is also effective for a linear message schedule compared to permutations of the message words. Since SHA-0 and SHA-1 have 80 steps and the attack described herein does not reach the same number of steps, the preimage resistance of SHA-0 and SHA-1 is still sufficient. We should pay attention to the progress of the techniques related to preimage resistance.

Acknowledgments

The authors would like to thank the anonymous referee for pointing out that we can construct initial structure with deterministic way against 52-step SHA-0. It also reduces the memory complexity by a factor of 2^{32}.

References

1. Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In Roberto Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography — Workshop Records of 15th Annual International Workshop, SAC 2008*, pages 82–98, Sackville, New Brunswick, Canada, 2008.
2. Jean-Philippe Aumasson, Willi Meier, and Florian Mendel. Preimage attacks on 3-pass HAVAL and step-reduced MD5. In Roberto Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography — Workshop Records of 15th Annual International Workshop, SAC 2008*, pages 99–114, Sackville, New Brunswick, Canada, 2008. (also appears in IACR Cryptology ePrint Archive: Report 2008/183 `http://eprint.iacr.org/2008/183`).
3. Christophe De Cannière and Christian Rechberger. Preimages for reduced SHA-0 and SHA-1. In David Wagner, editor, *Advances in Cryptology — CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 179–202, Berlin, Heidelberg, New York, 2008. Springer-Verlag. (slides on preliminary results presented at ESC 2008 seminar `http://wiki.uni.lu/esc/`).
4. Deukjo Hong, Donghoon Chang, Jaechul Sung, Sangjin Lee, Seokhie Hong, Jesang Lee, Dukjae Moon, and Sungtaek Chee. New FORK-256. (IACR Cryptology ePrint Archive: Report 2007/185 `http://eprint.iacr.org/2007/185`), 2007.
5. Gaëtan Leurent. MD4 is not one-way. In Kaisa Nyberg, editor, *Fast Software Encryption — 15th International Workshop, FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 412–428, Berlin, Heidelberg, New York, 2008. Springer-Verlag.
6. Alfred John Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997.
7. Ronald L. Rivest. The MD4 message digest algorithm. In Alfred John Menezes and Scott A. Vanstone, editors, *Advances in Cryptology — CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer-Verlag, Berlin, Heidelberg, New York, 1991. Also appears in RFC 1320 (`http://www.ietf.org/rfc/rfc1320.txt`).

8. Ronald L. Rivest. *Request for Comments 1321: The MD5 Message Digest Algorithm.* The Internet Engineering Task Force, 1992. (http://www.ietf.org/rfc/rfc1321.txt).
9. Markku-Juhani Olavi Saarinen. A meet-in-the-middle collision attack against the new FORK-256. In Kannan Srinathan, Chanrasekharan Pandu Rangan, and Moti Yung, editors, *Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India*, volume 4859 of *Lecture Notes in Computer Science*, pages 10–17. Springer-Verlag, Berlin, Heidelberg, New York, 2007.
10. Yu Sasaki and Kazumaro Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In Josef Pieprzyk, editor, *Advances in Cryptology — ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271. Springer-Verlag, Berlin, Heidelberg, New York, 2008.
11. Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer-Verlag, Berlin, Heidelberg, New York, 2009.
12. Yu Sasaki and Kazumaro Aoki. A preimage attack for 52-step HAS-160. In Pil Joong Lee and Jung Hee Cheon, editors, *Information Security and Cryptology - ICISC 2008, 11th International Conference*, volume 5461 of *Lecture Notes in Computer Science*, pages 302–317. Springer-Verlag, Berlin, Heidelberg, New York, 2009.
13. U.S. Department of Commerce, National Institute of Standards and Technology. *Secure Hash Standard (SHS) (Federal Information Processing Standards Publication 180-3)*, 2008. (http://csrc.nist.gov/publications/PubsFIPS.html#FIPS%20186-3).
14. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology — EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, Berlin, Heidelberg, New York, 2005.
15. Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry. HAVAL — one-way hashing algorithm with variable length of output. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology — AUSCRYPT'92*, volume 718 of *Lecture Notes in Computer Science*, pages 83–104. Springer-Verlag, Berlin, Heidelberg, New York, 1993.