Preimages for Step-Reduced SHA-2

Kazumaro Aoki[1], Jian Guo[2,⋆], Krystian Matusiewicz[3], Yu Sasaki[1,4], and Lei Wang[4]

[1] NTT Information Sharing Platform Laboratories, NTT Corporation
3-9-11 Midori-cho, Musashino-shi, Tokyo, 180-8585 Japan
{aoki.kazumaro,sasaki.yu}@lab.ntt.co.jp
[2] Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
guojian@ntu.edu.sg
[3] Department of Mathematics
Technical University of Denmark, Denmark
K.Matusiewicz@mat.dtu.dk
[4] University of Electro-Communications
1-5-1 Choufugaoka, Choufu-shi, Tokyo, 182-8585 Japan
wanglei@ice.uec.ac.jp

Abstract. In this paper, we present preimage attacks on up to 43-step SHA-256 (around 67% of the total 64 steps) and 46-step SHA-512 (around 57.5% of the total 80 steps), which significantly increases the number of attacked steps compared to the best previously published preimage attack working for 24 steps. The time complexities are $2^{251.9}$, 2^{509} for finding pseudo-preimages and $2^{254.9}$, $2^{511.5}$ compression function operations for full preimages. The memory requirements are modest, around 2^6 words for 43-step SHA-256 and 46-step SHA-512. The pseudo-preimage attack also applies to 43-step SHA-224 and SHA-384. Our attack is a meet-in-the-middle attack that uses a range of novel techniques to split the function into two independent parts that can be computed separately and then matched in a birthday-style phase.

Keywords: SHA-256, SHA-512, hash, preimage attack, meet-in-the-middle.

1 Introduction

Cryptographic hash functions are important building blocks of many secure systems. SHA-1 and SHA-2 (SHA-224, SHA-256, SHA-384, and SHA-512) [1] are hash functions standardized by the National Institute of Standards and Technology (NIST) and widely used all over the world. However, a collision attack on SHA-1 has been discovered recently by Wang *et al.* [2]. Since the structure of SHA-2 is similar to SHA-1 and they are both heuristic designs with no known

⋆ This work was done while visiting Technical University of Denmark and was partly supported by a DCAMM grant.

M. Matsui (Ed.): ASIACRYPT 2009, LNCS 5912, pp. 578–597, 2009.

security guarantees or reductions, an attack on SHA-2 might be discovered in the future too. To avoid a situation when all FIPS standardized functions would be broken, NIST is currently conducting a competition to determine a new hash function standard called SHA-3 [3]. From the engineering viewpoint, migration from SHA-1 to SHA-3 will take a long time. SHA-2 will take an important role during that transitional period. Hence, rigorous security evaluation of SHA-2 using the latest analytic techniques is important.

NIST requires SHA-3 candidates of n-bit hash length to satisfy a several security properties [3], first and foremost

- Preimage resistance of n bits,
- Second-preimage resistance of $n - k$ bits for any message shorter than 2^k blocks,
- Collision resistance of $n/2$ bits.

NIST claims that the security of each candidate is evaluated in the environment where they are tuned so that they run as fast as SHA-2 [4]. It seems that NIST tries to evaluate each candidate by comparing it with SHA-2. However, the security of SHA-2 is not well understood yet. Hence, the evaluation of the security of SHA-2 with respect to the security requirements for SHA-3 candidates is also important as it may influence our perspective on the SHA-3 speed requirements.

SHA-256 and SHA-512 consist of 64 steps and 80 steps, respectively. The first analysis of SHA-2 with respect to collision resistance was described by Mendel *et al.* [5], which presented the collision attack on SHA-2 reduced to 19 steps. After that, several researches have improved the result. In particular, the work by Nikolić and Biryukov improved the collision techniques [6]. The best collision attacks so far are the ones proposed by Indesteege *et al.* [7] and Sanadhya and Sarkar [8], both describing collision attacks for 24 steps. The only analysis of preimage resistance we are aware of is a recent attack on 24 steps of SHA-2 due to Isobe and Shibutani [9].

One may note the work announced at the rump session by Yu and Wang [10], which claimed to have found a non-randomness property of SHA-256 reduced to 39 steps. Since the non-randomness property is not included in the security requirements for SHA-3, we do not discuss it in this paper. In summary, the current best attacks on SHA-2 with respect to the security requirements for SHA-3 work for only 24 steps.

After Saarinen [11] and Leurent [12] showed examples of meet-in-the-middle preimage attacks, the techniques for such preimage attacks have been developed very rapidly. Attacks based on the concept of meet-in-the-middle have been reported for various hash functions, for example MD5 [13], SHA-1, HAVAL [14], and so on [15,16,17,18]. The meet-in-the-middle preimage attack is also applied to recently designed hash function ARIRANG [19], which is one of SHA-3 candidates, by Hong *et al.* [20]. However, due to the complex message schedule in SHA-2, these recently developed techniques have not been applied to SHA-2 yet.

Our contribution. We propose preimage attacks on 43-step SHA-256 and 46-step SHA-512 which drastically increase the number of attacked steps compared

to the previous preimage attack on 24 steps. We first explain various attack techniques for attacking SHA-2. We then explain how to combine these techniques to maximize the number of attacked steps. It is interesting that more steps of SHA-512 can be attacked than of SHA-256 with so-called partial-fixing technique proposed by Aoki and Sasaki [15]. This is due to the difference of the word size as functions σ and Σ mix 32-bit variables in SHA-256 more rapidly than in the case of double-size variables in SHA-512.

Our attacks are meet-in-the-middle. We first consider the application of the previous meet-in-the-middle techniques to SHA-2. We then analyse the message expansion of SHA-2 by considering all previous techniques and construct the attack by finding new independent message-word partition, which is the fundamental part of this attack.

Our attacks and a comparison with other results are summarized in Table 1.

Table 1. Comparison of preimage attacks on reduced SHA-2

Reference		Target	Steps	Complexity		Memory (approx.)
				Pseudo-preimage	Preimage	
Ours	Section 7	SHA-224	43	$2^{219.9}$	-	2^6 words
[9]		SHA-256	24	2^{240}	2^{240}	$2^{16} \cdot 64$ bits
Ours	Section 5	SHA-256	42	$2^{245.3}$	$2^{251.7}$	2^{12} words
Ours	Section 5	SHA-256	43	$2^{251.9}$	$2^{254.9}$	2^6 words
Ours	Section 7	SHA-384	43	2^{366}	-	2^{19} words
[9]		SHA-512	24	2^{480}	2^{480}	not given
Ours	Section 6	SHA-512	42	2^{488}	2^{501}	2^{27} words
Ours	Section 6	SHA-512	46	2^{509}	$2^{511.5}$	2^6 words

Outline. In Section 2, we briefly describe SHA-2. Section 3 gives an overview of the meet-in-the-middle preimage attack. In Section 4, we describe all techniques of our preimage attack. Then Sections 5 and 6 explain how these techniques can be applied together to mount an attack on SHA-256 and SHA-512, respectively. In Section 7, we put some remark on our attack. Section 8 concludes this paper.

2 SHA-2 Specification

Description of SHA-256. In this section we describe SHA-256, consult [1] for full details. SHA-256 adopts the Merkle-Damgård structure [21, Algorithm 9.25]. The message string is first padded with a single "1" bit, appropriate number of zero bits and then 64-bit length of the original message so that the length of the padded message is a multiple of 512 bits and then divided into 512-bit blocks, $(M_0, M_1, \ldots, M_{N-1})$ where $M_i \in \{0,1\}^{512}$.

The hash value h_N is computed by iteratively using the compression function CF, which takes a 512-bit message block and a 256-bit chaining variable as the input and yields an updated 256-bit chaining variable as the output,

$$\begin{cases} h_0 \leftarrow \mathrm{IV}, \\ h_{i+1} \leftarrow \mathrm{CF}(h_i, M_i)\ (i = 0, 1, \ldots, N-1), \end{cases} \tag{1}$$

where IV is a constant value defined in the specification.

The compression function is based on the Davies-Meyer mode [21, Algorithm 9.42]. It consists of a message expansion and a data processing. Let $\gg x$ and $\ggg x$ denote the x-bit right shift and rotation, respectively. First, the message block is expanded by the message expansion function,

$$W_i \leftarrow \begin{cases} m_i & \text{for } 0 \le i < 16\ , \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & \text{for } 16 \le i < 64\ . \end{cases} \tag{2}$$

where $(m_0, m_1, \ldots, m_{15}) \leftarrow M_i$ ($m_j \in \{0,1\}^{32}$) and "+" denotes addition modulo $2^{\mathrm{word\_size}}$. In SHA-256 the word size is 32 bits. Functions $\sigma_0(X)$ and $\sigma_1(X)$ are defined as

$$\begin{aligned} \sigma_0(X) &\leftarrow (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3), \\ \sigma_1(X) &\leftarrow (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10). \end{aligned} \tag{3}$$

where "$\oplus$" stands for bitwise XOR operation.

Let us use p_j to denote a 256-bit value consisting of the concatenation of eight words $A_j, B_j, C_j, D_j, E_j, F_j, G_j$ and H_j. The data processing computes h_{i+1} as follows.

$$\begin{cases} p_0 \leftarrow h_i, \\ p_{j+1} \leftarrow R_j(p_j, W_j), \quad (j = 0, 1, \ldots, 63) \\ h_{i+1} \leftarrow h_i + p_{64}, \end{cases} \tag{4}$$

Step function R_j is defined as follows

$$\begin{cases} T_1^{(j)} \leftarrow H_j + \Sigma_1(E_j) + \mathsf{Ch}(E_j, F_j, G_j) + K_j + W_j, \\ T_2^{(j)} \leftarrow \Sigma_0(A_j) + \mathsf{Maj}(A_j, B_j, C_j), \\ A_{j+1} \leftarrow T_1^{(j)} + T_2^{(j)}, \quad B_{j+1} \leftarrow A_j, \quad C_{j+1} \leftarrow B_j, \quad D_{j+1} \leftarrow C_j, \\ E_{j+1} \leftarrow D_j + T_1^{(j)}, \quad F_{j+1} \leftarrow E_j, \quad G_{j+1} \leftarrow F_j, \quad H_{j+1} \leftarrow G_j. \end{cases} \tag{5}$$

Above, K_j is a constant, different for each step, and the following functions are used

$$\begin{aligned} \mathsf{Ch}(X, Y, Z) &\leftarrow (X \vee Y) \oplus ((\neg X) \vee Z), \\ \mathsf{Maj}(X, Y, Z) &\leftarrow (X \vee Y) \oplus (X \vee Z) \oplus (Y \vee Z), \\ \Sigma_0(X) &\leftarrow (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22), \\ \Sigma_1(X) &\leftarrow (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25). \end{aligned} \tag{6}$$

where $\neg$ means bitwise negation of the word.

Description of SHA-512. The structure of SHA-512 is basically the same as SHA-256. In SHA-512, the word size is 64 bits, double of SHA-256, hence, the message-block size is 1024 bits and the size of chaining variable p_j is 512 bits.

The compression function has 80 steps. Rotation numbers in $\sigma_0, \sigma_1, \Sigma_0$, and Σ_1 are different from those used in SHA-256, which are shown below.

$$\begin{aligned}
\sigma_0(X) &\leftarrow (X^{\gg 1}) \oplus (X^{\gg 8}) \oplus (X^{\gg 7}),\\
\sigma_1(X) &\leftarrow (X^{\gg 19}) \oplus (X^{\gg 61}) \oplus (X^{\gg 6}),\\
\Sigma_0(X) &\leftarrow (X^{\gg 28}) \oplus (X^{\gg 34}) \oplus (X^{\gg 39}),\\
\Sigma_1(X) &\leftarrow (X^{\gg 14}) \oplus (X^{\gg 18}) \oplus (X^{\gg 41}).
\end{aligned} \tag{7}$$

3 Overview of the Meet-in-the-Middle Preimage Attack

A preimage attack on a narrow-pipe Merkle-Damgård hash function is usually based on a pseudo-preimage attack on its underlying compression function, where a pseudo-preimage is a preimage of the compression function with an appropriate padding. Many compression functions adopt Davies-Meyer mode, which computes $E_u(v) \oplus v$, where u is the message, v is the intermediate hash value and E is a block cipher.

First we recall the attack strategy on a compression function, which has been illustrated in Fig. 1. Denote by h the given target hash value. The high-level description of the attack for the simplest case is as follows.

1. Divide the key u of the block cipher E into two *independent* parts: u_1 and u_2. Hereafter, independent parts are called "chunks" and independent inputs u_1 and u_2 are called "neutral words".
2. Randomly determine the other input value v of the block cipher E.
3. Carry out the forward calculation utilizing v and all possible values of u_1, and store all the obtained intermediate values in a table T_F.
4. Carry out the backward calculation utilizing $h \oplus v$ and all possible values of u_2, and store all the intermediate values in a table T_B.
5. Check whether there exists a collision between T_F and T_B. If a collision exists, a pseudo-preimage of h has been generated. Otherwise, go to Step 2.

The main novelty of the meet-in-the-middle preimage attacks is, by utilizing independence of u_1 and u_2 of the key input, transforming the problem of finding a preimage of h to the problem of finding a collision on the intermediate values, which has a much lower complexity than the former one. Suppose there

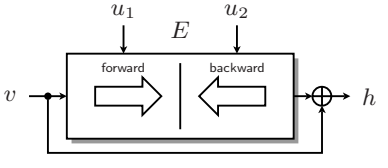

Fig. 1. Meet-in-the-middle attack strategy on a Davies-Meyer compression function $E_u(v) \oplus v$

are 2^t possible values for each of u_1 and u_2. Using 2^t compression function computations, the attacker obtains 2^t elements in *each* of T_F and T_B. The collision probability is roughly 2^{2t-n}, where n is the bit length of h, much better than the probability 2^{t-n} of finding a preimage by a brute force search with complexity 2^t.

4 The List of Attack Techniques

This section describes the list of techniques used in the attack. Some of them were used before in previous meet-in-the-middle attacks [15,18,16]. We explain them here first and then in Sections 5 and 6, we show how to combine them in an attack on SHA-2.

4.1 Splice-and-Cut

The meet-in-the-middle attack starts with dividing the key input into two independent parts. The idea of *splice-and-cut* is based on the observation made in [15] that the last and first steps of the block cipher E in Davies-Meyer mode can be regarded as consecutive by considering the feed-forward operation.

This allows the attacker to choose any step as the starting step of the meet-in-the-middle, which helps with finding more suitable independent chunks.

This technique can find only pseudo-preimages of the given hash value instead of preimages. However, pseudo-preimages can be converted to preimages with a conversion algorithm explained below.

4.2 Converting Pseudo-preimages to Preimages

In x-bit iterated hash functions, a pseudo-preimage attack with complexity $2^y, y < x - 2$ can be converted to a preimage attack with complexity of $2^{\frac{x+y}{2}+1}$ [21, Fact9.99]. The idea is applying the unbalanced meet-in-the-middle attack with generating $2^{(x-y)/2}$ pseudo-preimages and generating $2^{(x+y)/2}$ 1-block chaining variables starting from IV.

4.3 Partial-Matching

The example in Fig. 1 is the simplest and optimistic case. In fact, in the previous attacks, the key input cannot be divided into just two independent chunks. Usually besides the two independent chunks u_1 and u_2, there is another part, which depends on both u_1 and u_2. Hence, the stored intermediate values in T_F and T_B are ones at different steps. This raises a problem: how the values in T_F and T_B can be compared. However, many hash functions, including SHA-2, have Unbalanced Feistel Network structure, where the intermediate values will only be updated partially at one step. This means that a part of the intermediate values does not change during several steps and the attacker can check the match of two values partially.

Consider SHA-2, assume one chunk produces the value of p_j and the other chunk produces the value of p_{j+s}. The attacker wants to efficiently check whether or not p_j and p_{j+s} match without the knowledge of $W_j, W_{j+1}, \ldots, W_{j+s-1}$. In SHA-2, the maximum number of s is 7.

Assume the value of $p_{j+7} = A_{j+7} \| B_{j+7} \| \cdots \| H_{j+7}$ is known and W_{j+6} is unknown. By backward computation, we can obtain the values of $A_{j+6}, B_{j+6}, \ldots, G_{j+6}$. This is because $A_{j+6}, B_{j+6}, C_{j+6}, E_{j+6}, F_{j+6}$, and G_{j+6} are just copies of corresponding values in p_{j+7} and D_{j+6} is computed as follows.

$$D_{j+6} \leftarrow E_{j+7} - (A_{j+7} - (\Sigma_0(B_{j+7}) + \mathsf{Maj}(B_{j+7}, C_{j+7}, D_{j+7}))). \tag{8}$$

By repeating the similar computation, in the end, A_j is computed from p_{j+7} without the knowledge of $W_j, W_{j+1}, \ldots, W_{j+6}$. Note that this technique was already used (but not explicitly named) in [9].

4.4 Partial-Fixing

This is an extension of the partial-matching technique that considers *parts* of registers of the internal state. It increases the number of steps that can exist between two independent chunks. Assume that the attacker is carrying out the computation using u_1 and he is facing a step whose key input depends on both u_1 and u_2. Because the computation cannot go ahead without the knowledge of u_2, the chunk for u_1 must stop at this step. The *partial-fixing* technique is partially fixing the values of u_1 and u_2 so that we can obtain partial knowledge even if the full computation depends on both u_1 and u_2.

The partial-fixing technique for SHA-2 has not been considered previously. Assume we can fix the lower x bits of the message word in each step. Under this assumption, 1 step can be partially computed easily. Let us consider the step function of SHA-2 in the forward direction. Equations using W_j is as follows.

$$\begin{cases} T_1^{(j)} \leftarrow H_j + \Sigma_1(E_j) + \mathsf{Ch}(E_j, F_j, G_j) + K_j + W_j, \\ A_{j+1} \leftarrow T_1^{(j)} + T_2^{(j)}, \quad E_{j+1} \leftarrow D_j + T_1^{(j)}. \end{cases} \tag{9}$$

If the lower x bits of W_j are fixed, the lower x bits of A_{j+1} (and E_{j+1}) can be computed independently of the upper $32 - x$ bits of W_j. Let us consider to skip another step in forward direction. The equation for A_{j+2} is as follows:

$$A_{j+2} \leftarrow T_1^{(j+1)} + \Sigma_0(A_{j+1}) + \mathsf{Maj}(A_{j+1}, B_{j+1}, C_{j+1}). \tag{10}$$

We know only the lower x bits on A_{j+1}. Hence, we can compute Maj function for only the lower x bits. How about the Σ_0 function? We analysed the relationship of the number of consecutive fixed bits from LSB in the input and output of $\sigma_0, \sigma_1, \Sigma_0$, and Σ_1. The results are summarized in Table 2.

From Table 2, if x is large enough, we can compute the lower $x - 22$ bits of A_{j+2} in SHA-256 and the lower $x - 39$ bits in SHA-512, though the number of known bits is greatly reduced after the Σ_0 function. This fact also implies

Table 2. Relationship of number of consecutive fixed bits from LSB in input and output of σ and Σ

	SHA-256				SHA-512			
	Σ_0	Σ_1	σ_0	σ_1	Σ_0	Σ_1	σ_0	σ_1
Input	x	x	x	x	x	x	x	x
output	$x-22$	$x-25$	$x-18$	$x-19$	$x-39$	$x-41$	$x-8$	$x-61$

When x agrees with the word size, the output is x. When the number described in the output is negative, the output is 0.

that we cannot obtain the value of A_{j+3} since the number of fixed bits will be always 0. In the end, the partial-fixing technique can be applied for up to 2 steps in forward direction. Similarly, we considered the partial-fixing technique in backward, and found that it can be applied up to 6 steps.

However we have another problem in the first assumption; the lower x bits of each message word can be fixed. This is difficult to achieve because the fixed bits in message words are mixed by the σ function in the message expansion. In fact, we could apply the partial-fixing technique for computing only 1 step in forward, and only 2 steps in backward for SHA-256. However, in SHA-512, the bit-mixing speed of σ is relatively slow due to the double word size. In fact, we could compute 2 steps in forward, and 6 steps in backward. Finally, 10 steps in total can be skipped by the partial-matching and partial-fixing techniques for SHA-256, and 15 steps for SHA-512. (These numbers of steps are explained in Sections 5 and 6.)

4.5 Indirect-Partial-Matching

This is another extension of partial-matching. Consider the intermediate values in T_F and T_B. We can express them as functions of u_1 and u_2, respectively. If the next message word used in forward direction can be expressed as $\psi_1(u_1)+\psi_2(u_2)$ and computation of chaining register at the matching point does not destroy this relation (because the message word is also added), the matching point can still be expressed as a sum of two independent functions of u_1, u_2, e.g. $\psi_F(u_1)+\xi_F(u_2)$. Similarly, we can express the matching point from backward as $\psi_B(u_1)+\xi_B(u_2)$, and we are to find match. Now, instead of finding a match directly, we can compute $\psi_F(u_1)-\psi_B(u_1)$ in forward direction and $\xi_B(u_2)-\xi_F(u_2)$ in backward direction independently and find a match.

In case of SHA-2, it is possible to extend the 7-step partial-matching to 9-step indirect-partial-matching by inserting one step just before and after the partial matching.

Note this technique can be combined with partial-fixing technique by applying them in order: partial-fixing, partial-matching and indirect-partial-matching. However, there are some constraints that need to be satisfied, such as the independence of message word used in indirect-partial-matching, while we need to be able to compute enough bits at the matching point in order to carry out the partial-matching efficiently.

4.6 Initial Structure

In some cases, the two independent chunks u_1 and u_2 will overlap with each other. The typical example is that the order of the input key of E is $u_1u_2u_1u_2$. This creates a problem: how should the attacker carry out the forward and backward computations independently. The *Initial Structure* technique was proposed by [16] to solve such a problem. Previous attacks usually set a certain step as the starting step, then randomly determine the intermediate value at that step, and carry out the independent computations. However, the initial structure technique sets all the steps of u_2u_1 in the middle of $u_1u_2u_1u_2$ together as the starting point. Denote the intermediate values at the beginning and last step of u_2u_1 as I_1 and I_2 respectively. For each possible value of u_1, the attacker can derive a corresponding value I_1. Similarly, for each possible value of u_2, the attacker can derive a corresponding value I_2. Moreover, any pair (I_1, u_1) and (I_2, u_2) can be matched at the steps of u_2u_1 of $u_1u_2u_1u_2$. Thus, the attacker can carry out independent computations utilizing (I_1, u_1) and (I_2, u_2).

Initial structure for SHA-2 makes use of the absorption property of the function $\mathsf{Ch}(x, y, z) = xy \oplus (\neg x)z$. If x is $\mathbf{1}$ (all bits are 1), then $\mathsf{Ch}(\mathbf{1}, y, z) = y$ which means z does not affect the result of Ch function in this case; similarly when x is $\mathbf{0}$ (all bits are 0), y does not affect the result. When we want to control partial output (few bits), we need to fix the corresponding bits of x instead of all bits of x.

We consider 4 consecutive step functions, i.e. from step i to step $i + 3$. We show that, under certain conditions, we can move the last message word W_{i+3} to step i and move W_i to step $i+1$ while keeping the final output after step $i+3$ unchanged.

Assume we want to transfer upwards a message word W_{i+3}. Due to the absorption property of Ch, we can move W_{i+3} to step $i + 2$ (adding it to register G_{i+2}) if all the bits of E_{i+2} are fixed to 1. This is illustrated in Fig. 2 (left). Similarly, we can further move W_{i+3} to step $i + 1$ (adding it to register F_{i+1}) if all the bits of E_{i+1} are 0. Then, we still can move it upwards by transferring it to register E_i after step transformation in step i.

The same principle applies if we want to transfer only part of the register W_{i+3}. If l most significant bits (MSB) of W_{i+3} are arbitrary and the rest is set to zero (to avoid interference with addition on least significant bits), we need to fix l MSB of E_{i+2} to one and l MSB of E_{i+1} to zero.

As l MSB of E_{i+1} need to be 0, we need to use l MSB of W_i to satisfy this requirement. This reduces the space of W_i to 2^{32-l}. Similarly, we need to choose those W_i that fix l MSB of E_{i+2} to one. This is possible because changing the value of W_i influences the state of register E_{i+2} through Σ_1 at step $i + 1$. We experimentally checked that changing W_i generates changes in E_{i+2} that are sufficiently close to uniformly distributed. Satisfying additional constraints on l bits further reduces the space of W_i to 2^{32-2l}.

The important thing to note here is that if we fix the values of F_{i+1}, G_{i+1} and of the sum $D_{i+1} + H_{i+1}$ we can precompute the set of good values for W_i and store them in a table. Then, we can later recall them at negligible cost.

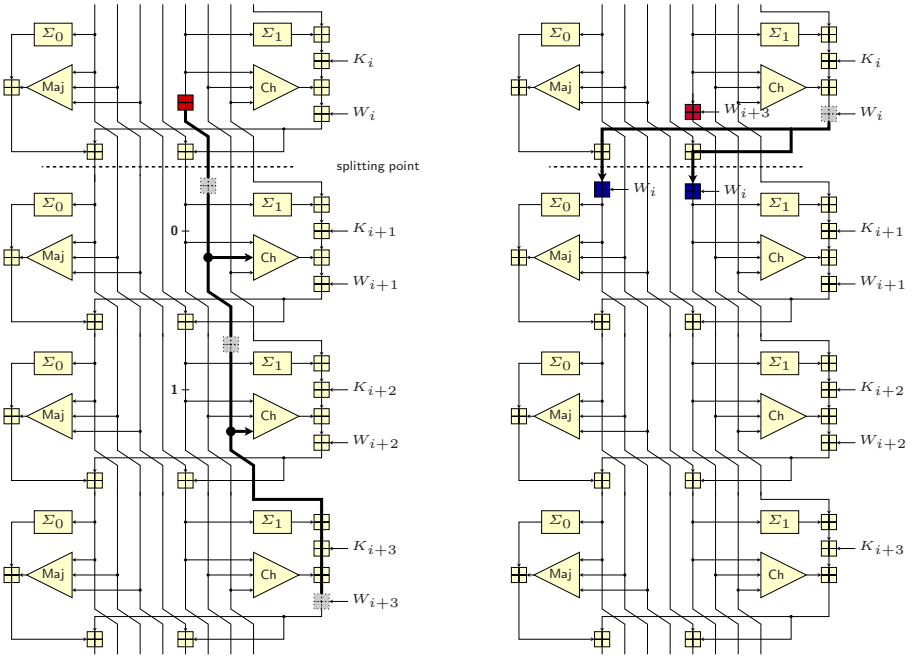

Fig. 2. Initial structure for SHA-2 allows to move the addition of W_{i+3} upwards provided that the Ch functions absorb the appropriate inputs (left); move W_i one step downwards (right)

On the other hand, message word W_i can be moved to step $i+1$ with no constraint, as shown in Fig. 2 (right).

This procedure essentially swaps the order of words W_i and W_{i+3}.

4.7 Two-Way Expansion

Message expansion usually works in such a way that some consecutive several messages can determine the rest. For SHA-2, any consecutive 16 message words can determine the rest since the message expansion is a bijective mapping. This enables us to control any intermediate 16 message words and then expand the rest in both ways. This technique gives us more freedom of choices of neutral words, and extends the number of steps for the two chunks a lot. Note that the maximum number of consecutive steps for the two chunks is 30 for SHA-2. Since the message expansion is a bijective mapping, no matter which neutral word is chosen, it must be used to compute at least one of the any consecutive 16 message words. So each chunk of consecutive steps is of length at most 15.

4.8 Message Compensation

For some choice of neutral words, two chunks are not able to achieve the optimal length. By forcing some of the other message words to cancel the change introduced by neutral words, the optimal or near-optimal length could be achieved.

Combining the initial structure, two-way expansion and message compensation techniques, we are able to find two chunks of length 33. We choose to control on $\{W_z, \ldots, W_{z+15}\}$, for some z which we will determine later. We choose W_{z+5} and W_{z+8} as neutral words. We show the first chunk $\{W_{z-10}, \ldots, W_{z+4}, W_{z+8}\}$ to be independent from W_{z+5} and second chunk $\{W_{z+5}, W_{z+6}, W_{z+7}, W_{z+9}, \ldots, W_{z+22}\}$ to be independent from W_{z+8}. Note that W_{z+8} is "moved" to first chunk by method explained in initial structure. For forward direction, we need to show $\{W_{z-10}, \ldots, W_{z-1}\}$ are independent from W_{z+5} when they are expanded from $\{W_z, \ldots, W_{z+15}\}$.

$$
\begin{aligned}
W_{z-1} &= W_{z+15} - \sigma_1(W_{z+13}) - W_{z+8} - \sigma_0(W_z) \ , && (11)\\
W_{z-2} &= W_{z+14} - \sigma_1(W_{z+12}) - W_{z+7} - \sigma_0(W_{z-1}) \ , && (12)\\
W_{z-3} &= W_{z+13} - \sigma_1(W_{z+11}) - W_{z+6} - \sigma_0(W_{z-2}) \ , && (13)\\
W_{z-4} &= W_{z+12} - \sigma_1(W_{z+10}) - \mathbf{W}_{z+5} - \sigma_0(W_{z-3}) \ , && (14)\\
W_{z-5} &= W_{z+11} - \sigma_1(W_{z+9}) - W_{z+4} - \sigma_0(W_{z-4}) \ , && (15)\\
W_{z-6} &= W_{z+10} - \sigma_1(W_{z+8}) - W_{z+3} - \sigma_0(W_{z-5}) \ , && (16)\\
W_{z-7} &= W_{z+9} - \sigma_1(W_{z+7}) - W_{z+2} - \sigma_0(W_{z-6}) \ , && (17)\\
W_{z-8} &= W_{z+8} - \sigma_1(W_{z+6}) - W_{z+1} - \sigma_0(W_{z-7}) \ , && (18)\\
W_{z-9} &= W_{z+7} - \sigma_1(\mathbf{W}_{z+5}) - W_z - \sigma_0(W_{z-8}) \ , && (19)\\
W_{z-10} &= W_{z+6} - \sigma_1(W_{z+4}) - W_{z-1} - \sigma_0(W_{z-9}) \ . && (20)
\end{aligned}
$$

We note that W_{z+5} is used in (19) and (14), we compensate them by using W_{z+7} and W_{z+12}. By "compensating" we mean making the equation value independent from W_{z+5} by forcing $W_{z+7} - \sigma_1(W_{z+5}) = C$ (C is some constant, we use 0 for simplicity) and $W_{z+12} - W_{z+5} = C$. W_{z+7} is also used in (17), however we can use W_{z+9} to compensate for it, i.e. set $W_{z+9} = \sigma_1(W_{z+7}) = \sigma_1^2(W_{z+5})$. Then W_{z+9} and W_{z+12} are used in steps above, so we continue this recursively and finally have the following constraints that ensure the proper compensation of values of W_{z+5}.

$$
\begin{aligned}
W_{z+7} &= \sigma_1(W_{z+5}) \ ,\\
W_{z+9} &= \sigma_1^2(W_{z+5}) \ ,\\
W_{z+11} &= \sigma_1^3(W_{z+5}) \ ,\\
W_{z+13} &= \sigma_1^4(W_{z+5}) \ ,\\
W_{z+15} &= \sigma_1^5(W_{z+5}) \ ,\\
W_{z+12} &= W_{z+5} \ ,\\
W_{z+14} &= 2\,\sigma_1(W_{z+5}) \ .
\end{aligned} \qquad (21)
$$

The second chunk is independent from W_{z+8} automatically without any compensation. The 33-step two-chunk is valid regardless of the choice of z as long as $z > 10$. To simplify the notation, we use $W_j, \ldots, W_{j+32}$ to denote the two chunks, then W_{j+15} and W_{j+18} are the two neutral words. We reserve the final choice of j for later to pick the one that allows to attack the most steps, as described later.

5 Preimage Attack against 43 Steps SHA-256

5.1 Number of Attacked Steps

The attack on SHA-256 uses 33-step two-chunk $W_j, \ldots, W_{j+32}$ explained in Section 4. Hence, in forward direction, p_{j+33} can be computed independently of the other chunk and in backward direction, p_j can be computed independently of the other chunk. We extend the number of attacked steps as much as possible with partial-fixing (PF) and indirect-partial-matching (IPM) techniques.

Forward computation of A_{j+34}: The equation for A_{j+34} is as follows.

$$\begin{cases} A_{j+34} = \Sigma_0(A_{j+33}) + \mathsf{Maj}(A_{j+33}, B_{j+33}, C_{j+33}) + H_{j+33} \\ \qquad\qquad + \Sigma_1(E_{j+33}) + \mathsf{Ch}(E_{j+33}, F_{j+33}, G_{j+33}) + K_{j+33} + W_{j+33}, \\ W_{j+33} = \sigma_1(W_{j+31}) + W_{j+26} + \sigma_0(W_{j+18}) + W_{j+17} \end{cases}$$

We can use either PF or IPM to compute A_{j+34}. If we use PF, we fix the lower l bits of W_{j+18}, which is a neutral word for the other chunk. According to Table 2, this fixes the lower $l-18$ bits of $\sigma_0(W_{j+18})$. Finally, the lower $l-18$ bits of A_{j+34} can be computed. If we use IPM, we describe A_{j+34} as a sum of functions of each neutral words i.e. $A_{j+34} = \psi_F(W_{j+15}) + \xi_F(W_{j+18})$. From the above equations, they can be easily done. Note that IPM is more efficient than PF with respect to only computing A_{j+34} because IPM does not need to fix a part of neutral word.

Forward computation of A_{j+35}: The equation for A_{j+35} is as follows.

$$\begin{cases} A_{j+35} = \Sigma_0(A_{j+34}) + \mathsf{Maj}(A_{j+34}, B_{j+34}, C_{j+34}) + \cdots + W_{j+34}, \\ W_{j+34} = \sigma_1(W_{j+32}) + W_{j+27} + \sigma_0(W_{j+19}) + W_{j+18} \end{cases}$$

Neither PF nor IPM can compute A_{j+35}. If we used PF for A_{j+34}, only the lower $l-18$ bits are known. This makes all bits of A_{j+35} unknown after the computation of $\Sigma_0(A_{j+34})$. If we used IPM, A_{j+34} is described as a sum of two independent functions. However, because Σ_0 consists of XOR of three self-rotations, it seems difficult to describe $\Sigma_0(A_{j+34})$ as a sum of two independent functions.

In summary, we can skip only 1 step in forward. In this case, using IPM is more efficient than using PF.

Backward computation of H_{j-1}: The equation for H_{j-1} is as follows.

$$\begin{cases} H_{j-1} = A_j - (\Sigma_0(B_j) + \mathsf{Maj}(B_j, C_j, D_j)) \\ \qquad\qquad - \Sigma_1(F_j) - \mathsf{Ch}(F_j, G_j, H_j) - K_{j-1} - W_{j-1}, \\ W_{j-1} = W_{j+15} - \sigma_1(W_{j+13}) - W_{j+8} + \sigma_0(W_j) \end{cases}$$

We can use either PF or IPM to compute H_{j-1}. If we use PF, we fix the lower l bits of W_{j+15}, and then, the lower l bits of H_{j-1} can be computed. If we use IPM, we describe H_{j-1} as a sum of functions of each neutral word.

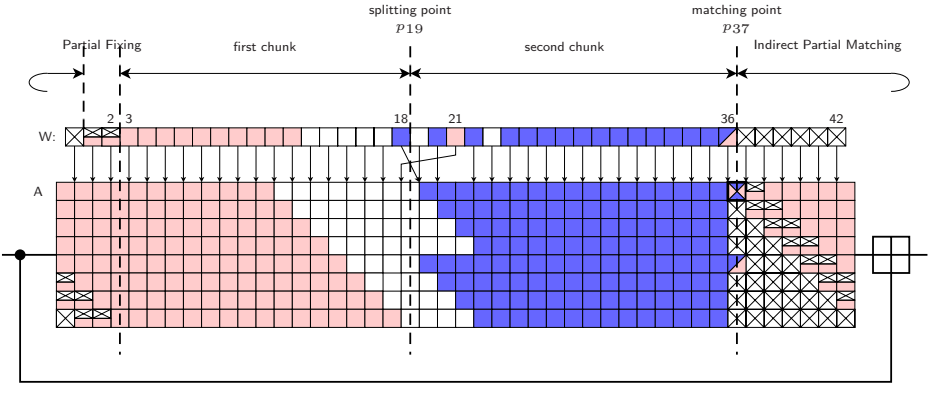

Fig. 3. Separation of chunks and dependencies of state words for SHA-256

Backward computation of H_{j-2}: The equation for H_{j-2} is as follows.

$$\begin{cases} H_{j-2} = A_{j-1} - (\Sigma_0(B_{j-1}) + \mathsf{Maj}(B_{j-1}, C_{j-1}, D_{j-1})) \\ \qquad\qquad - \Sigma_1(F_{j-1}) - \mathsf{Ch}(F_{j-1}, G_{j-1}, H_{j-1}) - K_{j-2} - W_{j-2}, \\ W_{j-2} = W_{j+14} - \sigma_1(W_{j+12}) - W_{j+7} + \sigma_0(W_{j-1}) \end{cases}$$

We can use PF to compute H_{j-2} but cannot use IPM. To describe $\mathsf{Ch}(F_{j-1}, G_{j-1}, H_{j-1})$ and $\sigma_0(W_{j-1})$ as a sum of two independent functions seems difficult. If we used PF for H_{j-1}, we can obtain the lower l bits of $\mathsf{Ch}(F_{j-1}, G_{j-1}, H_{j-1})$ and lower $l-18$ bits of $\sigma_0(W_{j-1})$. Finally, we can compute the lower $l-18$ bits of H_{j-2}.

By the similar analysis, we confirmed that we cannot compute H_{j-3}. In summary, we can skip 2 steps in backward with PF which fixes the lower $l, l > 18$ bits of W_{j+15}.

The attack uses 33-step two-chunk $W_j, \ldots, W_{j+32}$ including 4-step initial structure. Apply PF for W_{j-1} and W_{j-2}, and apply IPM for W_{j+34}. Finally, 43 steps are attacked by skipping additional 7 steps using partial-matching technique.

36 steps (W_{j-2} to W_{j+34}) must be located sequentially. We have several options for j. We choose $j = 3$ for the following two purposes; (1) W_{13}, W_{14}, and W_{15} can be freely chosen to satisfy message padding rules, (2) pseudo-preimage attack on SHA-224 is possible (explained in Section 7).

We need to fix the lower $l+18$ bits of W_{18} to fix the lower l bits of W_2 by PF. Besides, we lose half of remaining freedom to construct 4-step initial structure. Hence, we choose l to balance $l-18$ and $\frac{32-l}{2}$, i.e. we choose $l = 23$.

The overview of the separation of chunks is shown in Fig. 3. ▭ denotes variables depending only on W_{21}; ■ denotes variables depending only on W_{18}; ◪ and ◪ denote registers that can be expressed as a sum modulo 2^{32} of two independent functions of neutral variables W_{18} and W_{21}; ▩ denotes registers with few bits depending only on W_{21}; ⊠ denotes registers depending on both W_{18} and W_{21} in a complicated way.

5.2 Attack Procedure

1. Randomly choose the values for internal chaining variable p_{19} (after the movement of message words by initial structure) and message word W_{19}. Randomly fix the lower 23 bits of W_{18}. By using the remaining 9 free bits of W_{18}, find 2^5 values on average that correctly construct the 4-step initial structure, and store them in the table T_W. Let us call this an initial table-preparation.
2. Randomly choose message words not related to initial structure and neutral words, i.e. $W_{13}, W_{14}, W_{15}, W_{16}, W_{17}, W_{23}$. Let us call this an initial configuration.
3. For all 2^5 possible W_{18} in T_W, compute the corresponding W_{20}, W_{22}, W_{24}, $W_{25}, W_{26}, W_{27}, W_{28}$ as shown in equations (21). Compute forward and find $\psi_F(W_{18})$. Store the pairs $(W_{18}, \psi_F(W_{18}))$ in a list L_F.
4. For all 2^4 possible values (the lower 4 bits) of W_{21}, compute backward and find $\xi_F(W_{21})$, which is $\sigma_0(W_{21})$ in this attack, and the lower 4 bits of A_{37}.
5. Compare the lower 4 bits of $A_{37} - \sigma_0(W_{21})$ and the lower 4 bits of $\psi_F(W_{18})$ stored in L_F.
6. If a match is found, compute $A_{37}, B_{37}, \ldots, H_{37}$ with the corresponding W_{18} and W_{21} and check whether results from both directions match each other. If they do, output p_0 and $W_0, \ldots, W_{15}$ as a pseudo-preimage.
7. Repeat steps 2 – 6 for all possible choices of $W_{13}, W_{16}, W_{17}, W_{21}$. Note, the MSB of W_{13} is fixed to 1 to satisfy message padding. Hence, we have 2^{127} freedom for this step.
8. If no freedom remains in step 7, repeat steps 1 – 7.
9. Repeat steps 1 – 8 2^4 times to obtain 2^4 pseudo-preimages. Then, convert them to a preimage according to [21, Fact9.99].

5.3 Complexity Estimation

We assume the complexity for 1 step function and 1-step message expansion is $\frac{1}{43}$ compression function operation of 43-step SHA-256. We also assume that the speed of memory access is negligible compared to computation time for step function and message expansion. Complexity for step 1 is 2^9 and use a memory of 2^5 words. Complexity for step 2 is negligible. In step 3, we compute $p_{j+1} \leftarrow R_j(p_j, W_j)$ for $j = 18, 19, \ldots, 36$ and corresponding message expansion. Hence, the complexity is $2^5 \frac{19}{43}$. We use a memory of $2^5 \times 2$ words. Similarly, in step 4, we compute $p_j \leftarrow R_{j+1}^{-1}(p_{j+1}, W_j)$ for $j = 20, 19, \ldots, 2$ and 6 more steps for partial-fixing and partial-matching. Hence, the complexity is $2^4 \frac{25}{43}$. In step 5, we compare the match of lower 4 bits of $2^9 (= 2^4 \cdot 2^5)$ items. Hence, 2^5 results will remain. Complexity for step 6 is $2^5 \frac{8}{43}$ and the probability that all other bits match is 2^{-252}. Hence, the number of remaining pair becomes $2^{-247} (= 2^5 \cdot 2^{-252})$. So far, the complexity from step 2 to 6 is $2^5 \frac{19}{43} + 2^4 \frac{25}{43} + 2^5 \frac{8}{43} = 2^5 \frac{39.5}{43} \approx 2^{4.878}$. In step 7, this is repeated 2^{127} times and its complexity is $2^{131.878}$. Step 8 is computed 2^{120} times. This takes $2^{120} \cdot (2^9 + 2^{131.878}) \approx 2^{251.9}$. This is the complexity of the pseudo-preimage attack on SHA-256 43-steps. Finally, at Step 9, preimages are

found with a complexity of $2^{1+(251.878+256)/2} = 2^{254.939} \approx 2^{254.9}$. The required memory for finding a pseudo-preimage is 2^5 words and $2^5 \times 2$ words in Steps 1 and 3, which is $2^5 \times 3$ words. For finding a preimage, we need to store $2^{1.9}$ pseudo-preimages for unbalanced meet-in-the-middle. This requires a memory of $2^{1.9} \times 24$ words.

5.4 Attack on 42 Steps SHA-256

When we attack 42 steps, We use 1-step IPM instead of 2-step PF in backward. This allows the attacker to use more message freedom. We choose $l = 10$ so that l and $\frac{32-l}{2}$ are balanced. Because each chunk has at least 10 free bits, the complexity for finding pseudo-preimages is approximately $2^{246} (= 2^{256} \cdot 2^{-10})$. The precise evaluation is listed in Table 1.

6 Preimage Attack against 46 Steps SHA-512

6.1 Basic Strategy for SHA-512

For SHA-512, we can attack more steps than SHA-256 by using PF. This occurs by the following two properties;

- Message-word size of SHA-512 is bigger than that of SHA-256. Hence, the bit-mixing speed of σ and Σ functions are slower than SHA-256.
- The choice of three rotation numbers for the σ_0 function is very biased.

To consider the above, we determine to use the message freedom available to the attacker for applying PF as much as possible.

Construction of the 4-step initial structure explained in Section 4 consumes a lot of message freedom. Therefore, we do not use the 4-step initial structure for SHA-512. Construction of the 3-step initial structure also needs a lot of message freedom. On the other hand, 2-step initial structure does not consume any message freedom because we do not have to control Ch functions. Finally, in our attack, we use a 31-step two-chunk including 2-step initial structure. Because construction of 2-step initial structure is much simpler than that of 4-step initial structure, we omit the detailed explanation of the construction.

6.2 Chunk Separation

The 31 message words we use are W_j to W_{j+30}. We apply the 2-step initial structure for W_{j+15} and W_{j+16}, hence the neutral words for the first chunk is W_{j+16} and for the second chunk is W_{j+15}. Whenever we change the value of W_{j+16}, we change the values of $W_{j+7}, W_{j+6}, \ldots, W_j$ by message compensation technique so that the change does not impact to the second chunk. Similarly, whenever we change W_{j+15}, we change $W_{j+17}, W_{j+19}, W_{j+21}, W_{j+22}, \ldots, W_{j+30}$. Finally, W_j to W_{j+30} can form the 31-step two-chunks.

6.3 Partial-Fixing Technique

We skip 6 steps in backward and 2 steps in forward by PF. Namely, we need to partially compute $W_{j-1}, W_{j-2}, \ldots, W_{j-6}$ independently of W_{j+15}, and partially compute W_{j+31} and W_{j+32} independently of W_{j+16}, The equations for these message words are as follows.

$$
\begin{aligned}
\underline{W_{j-1}}_{l} &= \underline{W_{j+15}}_{l} - \sigma_1(W_{j+13}) - W_{j+8} + \sigma_0(W_j),\\
\underline{W_{j-2}}_{l-8} &= W_{j+14} - \sigma_1(W_{j+12}) - W_{j+7} + \sigma_0(\underline{W_{j-1}}_{l}),\\
\underline{W_{j-3}}_{l-16} &= W_{j+13} - \sigma_1(W_{j+11}) - W_{j+6} + \sigma_0(\underline{W_{j-2}}_{l-8}),\\
\underline{W_{j-4}}_{l-24} &= W_{j+12} - \sigma_1(W_{j+10}) - W_{j+5} + \sigma_0(\underline{W_{j-3}}_{l-16}),\\
\underline{W_{j-5}}_{l-32} &= W_{j+11} - \sigma_1(W_{j+9}) - W_{j+4} + \sigma_0(\underline{W_{j-4}}_{l-24}),\\
\underline{W_{j-6}}_{l-40} &= W_{j+10} - \sigma_1(W_{j+8}) - W_{j+3} + \sigma_0(\underline{W_{j-5}}_{l-32}),
\end{aligned}
$$

$$
\begin{aligned}
\underline{W_{j+31}}_{l-8} &= \sigma_1(W_{j+29}) + W_{j+24} + \sigma_0(\underline{W_{j+16}}_{l}) + W_{j+15},\\
\underline{W_{j+32}}_{l} &= \sigma_1(W_{j+30}) + W_{j+25} + \sigma_0(W_{j+17}) + \underline{W_{j+16}}_{l}.
\end{aligned}
$$

Remember Table 2. If the lower l bits of input of σ_0 is fixed, we can compute the lower $l-8$ bits of its output. In backward, if we fix the lower l bits of W_{j+15}, the lower l bits of W_{j-1}, the lower $l-8$ bits of W_{j-2}, the lower $l-16$ bits of W_{j-3}, the lower $l-24$ bits of W_{j-4}, the lower $l-32$ bits of W_{j-5}, and the lower $l-40$ bits of W_{j-6} can become independent of the second chunk. This results in computing the lower l bits of H_{j-1}, the lower $l-8$ bits of H_{j-2}, the lower $l-16$ bits of H_{j-3}, the lower $l-41$ bits of H_{j-4}, the lower $l-49$ bits of H_{j-5}, and the lower $l-57$ bits of H_{j-6}. Note that we also need to consider Σ_1 to compute H_{j-4}, H_{j-5}, and H_{j-6}. If we fix the lower l bits of W_{j+16}, the lower $l-8$ bits of W_{j+31}, and the lower l bits of W_{j+32} can become independent of the first chunk. This results in computing the lower $l-8$ bits of A_{j+32}, and the lower $l-47$ bits of A_{j+33}.

Therefore, if we choose $l = 60$, we can match the lower 3 bits of H_{j-6} and 13 bits of A_{j+33} after we skip 7 steps by the partial-matching technique.

6.4 Attack Overview

The attack uses 31-step two-chunk $W_j, \ldots, W_{j+30}$ including 2-step initial structure. Apply PF for $W_{j-1}, W_{j-2}, \ldots, W_{j-6}$, and W_{j+31}, W_{j+32}. Finally, 46 steps are attacked by skipping additional 7 steps using partial-matching technique.

39 steps (W_{j-6} to W_{j+32}) must be located sequentially. Because W_{j+8}, W_{j+9}, $W_{j+10}, W_{j+11}, W_{j+12}, W_{j+13}, W_{j+14}, W_{j+18}, W_{j+20}$ are the message words we fix in advance, we choose $j = 6$ so that W_{14} and W_{15} can be chosen to satisfy message padding rules. The MSB of W_{13} can also be satisfied. In this chunk separation, W_{j+7} can be described as $W_{j+7} = \mathrm{Const} - W_{j+16}$, where Const is a chosen fixed value and the lower l bits of W_{j+16} are fixed. If we fix Const and the MSB of W_{j+16} to 0 and some value, respectively, and choose the lower l bits of W_{j+16} so that the MSB of $-W_{j+16}$ does not change for all active bits of W_{j+16}, we can always fix the MSB of W_{j+7}.

The number of free bits in W_{j+16} is 3. ($l = 60$ but we fix the MSB for satisfying padding for W_{13}.) The number of free bits in W_{j+15} is 4. Results from both chunks are compared with 3 bits. Therefore, the final complexity of pseudo-preimage attack is approximately 2^{509}. This is converted to a preimage attack whose complexity is approximately $2^{511.5}$. For finding pseudo-preimages, this attack needs to store 2^3 items. Hence, the required memory is $2^3 \times 9$ words. For finding preimages, we need to store $2^{1.5}$ pseudo-preimages for unbalanced meet-in-the-middle. This requires a memory of $2^{1.5} \times 24$ words.

6.5 Attack on 42 Steps SHA-512

When we attack 42 steps, we stop using 1-step PF in forward and 3-step PF in backward. We choose $l = 40$. Because each chunk has at least 24 free bits, the complexity for finding pseudo-preimages is approximately $2^{488} (= 2^{512} \cdot 2^{-24})$. The precise evaluation is listed in Table 1.

7 Remarks

7.1 Length of Preimages

The preimages are of at least two blocks, last block is used to find pseudo-preimages and the second last block links to the input chaining of last block. Two block preimages is only possible if we can preset the message words used for encoding the length (m_{14} and m_{15} for SHA-2) of last block according to the padding and length encoding rules. In our case, this can be done in the first step of the algorithm. On the other hand, we can leave m_{14} and m_{15} as random, later we can still resolve the length using expandable messages [22].

7.2 SHA-224 and SHA-384

Our attack on 43 steps SHA-256 can also produce pseudo-preimages for SHA-224 by using the approach by Sasaki [23]. In our attack, we match 4-bits of A_{37} which is essentially equivalent to G_{43}. Then, we repeat the attack until other registers randomly match i.e. we wait until $A_{43}, B_{43}, \ldots, F_{43}$, and H_{43} randomly match. In SHA-224, the value of H_{43} is discarded in the output. Hence, we do not have to care the match of H_{43}, which results in decreasing the complexity by 2^{32} bits. Hence, pseudo-preimages of SHA-224 can be computed with a complexity of $2^{219.9} (= 2^{251.9} \cdot 2^{-32})$. Note, this cannot be converted to a preimage attack on SHA-224 because the size of intermediate chaining variable is 256 bits.

If we apply our attack on SHA-512 to SHA-384, W_{13}, W_{14}, and W_{15} will depend on neutral words. Hence, we cannot confirm 46 steps SHA-384 can be attacked or not because of padding problem. However, 43 steps SHA-384 can be attacked by using the same chunk as SHA-256. By considering the difference of word size and application of PF, we can optimize the complexity by choosing $l = 27$ so that $l - 8$ and $\frac{64-l}{2}$ are balanced.

7.3 Multi-preimages and Second-Preimages

We note that the method converting pseudo-preimage to preimages can be further extended to find multi-preimages. We find first k block multi-collisions [24], then follow the expandable message [22] to link to the final block. This gives 2^k multi-preimages with additional $k2^{n/2}$ computations, which is negligible when k is much smaller than $2^{(n-t)/2}$ (t denotes number of bits for each chunk, refer to Section 3). We need additional $128k$ bytes of memory to store the k block multi-collisions. Furthermore, most of the message words are randomly chosen, this attack naturally gives second preimages with high probability. Above multi-preimages are most probably multi-second preimages.

8 Conclusions

In this paper, we presented preimage attacks on 43 steps SHA-256 and 46 steps SHA-512. The time complexity of the attack for 43-step SHA-256 is $2^{254.9}$ and it requires $2^5 \cdot 3$ words of memory. The time complexity of the attack for 46-step SHA-512 is $2^{511.5}$ and it requires $2^3 \cdot 9$ words of memory. The number of attacked steps is greatly improved from the best previous attack, in other words, the security margin of SHA-256 and SHA-512 is greatly reduced. Because SHA-256 and SHA-512 have 64 and 80 steps, respectively, they are currently secure.

An open question worth investigating would be to see if the current attacks may still be improved. Perhaps finding $15+4+15$ pattern of chunks with 4-step initial structure in the middle or using better partial-fixing technique that would utilize middle bits of the message word would extend the attacks.

The preimage attack we presented creates a very interesting situation for SHA-2 when a preimage attack, covering 43 or 46 steps, is much better than the best known collision attack, with only 24 steps. Our attack does not convert to collision attack because of the complexity above the birthday bound. However, we believe that the existence of such a preimage attack suggests that a collision attack of similar length could be also possible. In that light, the problem of finding collisions for reduced variants of SHA-256 definitely deserves more attention.

Acknowledgements

We would like to thank Lars R. Knudsen, Christian Rechberger, and the anonymous reviewers for the helpful comments. Jian Guo was supported by the Singapore Ministry of Education under Research Grant T206B2204. Krystian Matusiewicz was supported by grant 274-07-0246 from the Danish Research Council for Technology and Production Sciences.

References

1. 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/fips/fips180-3/fips180-3_final.pdf`

2. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
3. U.S. Department of Commerce, National Institute of Standards and Technology: Federal Register Vol. 72(212) Friday, November 2, 2007 Notices (2007), http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf
4. U.S. Department of Commerce, National Institute of Standards and Technology: NIST's Plan for Handling Tunable Parameters. Presentation by Souradyuti Paul at The First SHA-3 Candidate Conference (February 2009), http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/Feb2009/
5. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: Analysis of step-reduced SHA-256. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 126–143. Springer, Heidelberg (2006)
6. Nikolić, I., Biryukov, A.: Collisions for step-reduced SHA-256. In: [25], pp. 1–15
7. Indesteege, S., Mendel, F., Preneel, B., Rechberger, C.: Collisions and other non-random properties for step-reduced SHA-256. In: [26], pp. 276–293
8. Sanadhya, S.K., Sarkar, P.: New collision attacks against up to 24-step SHA-2 (extended abstract). In: Rijmen, V., Das, A., Chowdhury, D.R. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 91–103. Springer, Heidelberg (2008)
9. Isobe, T., Shibutani, K.: Preimage attacks on reduced Tiger and SHA-2. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 139–155. Springer, Heidelberg (2009)
10. Yu, H., Wang, X.: Non-randomness of 39-step SHA-256 (2008), http://www.iacr.org/conferences/eurocrypt2008v/index.html
11. Saarinen, M.J.O.: A meet-in-the-middle collision attack against the new FORK-256. In: Srinathan, K., Pandu Rangan, C., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 10–17. Springer, Heidelberg (2007)
12. Leurent, G.: MD4 is not one-way. In: [25], pp. 412–428
13. Rivest, R.L.: Request for Comments 1321: The MD5 Message Digest Algorithm. The Internet Engineering Task Force (1992), http://www.ietf.org/rfc/rfc1321.txt
14. Zheng, Y., Pieprzyk, J., Seberry, J.: HAVAL — one-way hashing algorithm with variable length of output. In: Seberry, J., Zheng, Y. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 83–104. Springer, Heidelberg (1993)
15. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: [26], pp. 103–119
16. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg (2009)
17. Aoki, K., Sasaki, Y.: Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 70–89. Springer, Heidelberg (2009)
18. Sasaki, Y., Aoki, K.: Preimage attacks on 3, 4, and 5-pass HAVAL. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 253–271. Springer, Heidelberg (2008)
19. Chang, D., Hong, S., Kang, C., Kang, J., Kim, J., Lee, C., Lee, J., Lee, J., Lee, S., Lee, Y., Lim, J., Sung, J.: ARIRANG. NIST home page: http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html
20. Hong, D., Kim, W.H., Koo, B.: Preimage attack on ARIRANG. Cryptology ePrint Archive, Report 2009/147 (2009), http://eprint.iacr.org/2009/147
21. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC Press, Boca Raton (1997)

22. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 474–490. Springer, Heidelberg (2005)
23. Sasaki, Y.: Meet-in-the-middle attacks using output truncation in 3-pass HAVAL. In: Samarati, P., Yung, M., Martinelli, F. (eds.) ISC 2009. LNCS, vol. 5735, pp. 79–94. Springer, Heidelberg (2009)
24. Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
25. Nyberg, K. (ed.): FSE 2008. LNCS, vol. 5086. Springer, Heidelberg (2008)
26. Avanzi, R., Keliher, L., Sica, F. (eds.): SAC 2008. LNCS, vol. 5381. Springer, Heidelberg (2009)