Preimages for Step-Reduced SHA-2*

Jian Guo[1]** and Krystian Matusiewicz[2]

[1] Division of Mathematical Sciences
School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
guojian@ntu.edu.sg
[2] Department of Mathematics
Technical University of Denmark, Denmark
K.Matusiewicz@mat.dtu.dk

Abstract. In this paper, we present a preimage attack for 42 step-reduced SHA-256 with time complexity $2^{251.7}$ and memory requirements of order 2^{12}. The same attack also applies to 42 step-reduced SHA-512 with time complexity $2^{502.3}$ and memory requirements of order 2^{22}. Our attack is meet-in-the-middle preimage attack.

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

1 Introduction

A novel class of collision attacks on MD family developed by Wang et al [23, 25, 26] opened a new chapter in the history of cryptographic hash functions. Soon after MD5 was broken, a theoretical break of SHA-1 was announced [24] and refined [4].

Weaknesses of so many of these popular designs against collision attacks naturally raised the question about their level of resistance against potentially much more devastating preimage attacks. While collision attacks, especially practical, are dangerous in some scenarios, violating the preimage resistance makes the function practically useless. Unlike collision attacks, for which their impact can be sometimes mitigated by using solutions like RMX [6], the preimage attacks are more devastating. Once they are found, the only solution seems to be replacing the function with another, secure one. This motivated the community to study the popular designs also from the point of view of preimage resistance.

Preimage attacks were soon found for MD4 [28, 12, 1], reduced round MD5 and other hashes, such as HAVAL [19, 2, 21]. Finally, a preimage attack on the full MD5 was presented recently by Sasaki and Aoki [22].

The security of SHA-1 against preimage attacks was also investigated by De Canniére and Rechberger and an attack on 45 rounds out of 80 was presented in [3].

* The paper was merged and will appear in ASIACRYPT 2009.

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

SHA-256 One of the very few well-known designs from the MD family that has not been compromised yet is the current U.S. Government standard SHA-256 [16]. Due to its much more complicated structure, especially of the message expansion, attacking it seems quite a difficult task. The progress of cryptanalysis of SHA-256 has been steady but rather slow [5, 27, 13, 14, 7, 18, 17]. The best publicly known collision attack [7, 17] covers 24 steps out of the total 64. At the rump session of CRYPTO 2008 Sasaki announced a 36-step preimage attack [20], but the details have not been made public. Recently, a preimage attack on a variant reduced to 24 steps was presented [8].

SHA-256 warrants the attention for at least two reasons. The first one is its standardization and a recent move to replace the broken predecessor SHA-1 with it. The more we know about the security of reduced and simplified variants, the better we understand the complete construction and the security margin it offers.

Moreover, the NIST competition for the new hashing standard SHA-3 uses SHA-2 as some kind of a benchmark for the relative performance of the candidates. Advances in the cryptanalysis of SHA-2 may improve our understanding of how fast a secure cryptographic hash function can be.

Our Contribution We contribute to the cryptanalysis of SHA-2 by presenting a theoretical preimage attack on a variant of SHA-256 reduced to 42 steps, around 66% of the total of 64 steps for this function. We build upon the general framework developed for a number of MD hashes by Sasaki and Aoki, and add more dedicated tricks exploiting the details of the construction of SHA-2. We show how to use message expansion in both directions, achieve better partial matching and how to transfer the influence of some of the message words to be able to use them for word compensation somewhere else. Combining all those methods allows us to achieve 42 steps.

This paper is organized as follows. We start with a description of SHA-2 in Section 2 and move on to explain the general idea of the preimage attack in Section 3. Then, in Section 4, we delve into the details of the improvements, explaining how to use them to achieve more steps. Finally, we present the complete attack algorithm in Section 5 and establish its computational complexity. In the conclusions, we summarize the main points of this work and consider some possible research directions to extend the current results.

2 Description of SHA-2

SHA-256 [16] is an iterated hash function based on the Merkle-Damgård design that uses a compression function mapping 256 bits of the state and 512 bits of the message block to 256 bits of the new state. The compression function consists of 64 identical steps presented in Fig. 1 that update 256-bit internal state $\mathcal{S}_i = (A_i, \ldots, H_i)$ using one word of the expanded message W_i and a constant K_i to produce a new state $\mathcal{S}_{i+1} = (A_{i+1}, \ldots, H_{i+1})$. After the last step, a feed-forward operation is applied and the output of the compression

function is computed as a word-wise sum modulo 2^{32} of the previous state S_0 and the output of the last step S_{64}, i.e. $(A_0 + A_{64}, \ldots, H_0 + H_{64})$.

The step transformation employs bitwise Boolean functions

$$\begin{aligned} \mathrm{MAJ}(A, B, C) &= (A \wedge B) \vee (A \wedge C) \vee (B \wedge C) \ , \\ \mathrm{IF}(E, F, G) &= (E \wedge F) \vee (\neg E \wedge G) \end{aligned}$$

and two diffusion functions

$$\begin{aligned} \Sigma_0(x) &= (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22) \ , \\ \Sigma_1(x) &= (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25) \end{aligned}$$

built from 32-bit word rotations towards least significant bit ($\ggg$) and bitwise XORs denoted by $\oplus$.

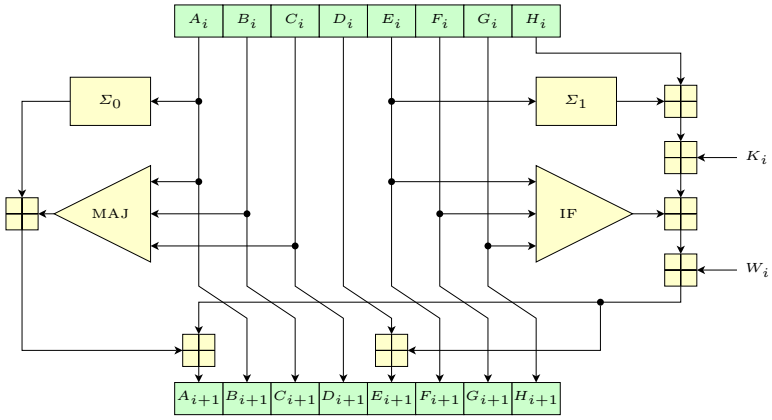

Fig. 1. One step of the SHA-256 compression function updates the state of eight chaining variables $A, \ldots, H$ using one word W_i of the expanded message.

The message expansion function splits 512-bit message block into 16 words M_i, $i = 0, \ldots, 15$, and expands them into a vector of 64 32-bit words W_i according to the following formula:

$$W_i = \begin{cases} M_i & \text{for } 0 \le i < 16 \ , \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & \text{for } 16 \le i < 64 \ . \end{cases} \tag{1}$$

The functions σ_0 and σ_1 are defined as $\sigma_0(x) = (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3)$ and $\sigma_1(x) = (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10)$, where $\gg$ denotes a shift towards least significant bit and $+$ is addition modulo 2^{32}.

3 The Preimage Attack

The general idea of the preimage attack, illustrated in Fig. 2, can be explained as follows:

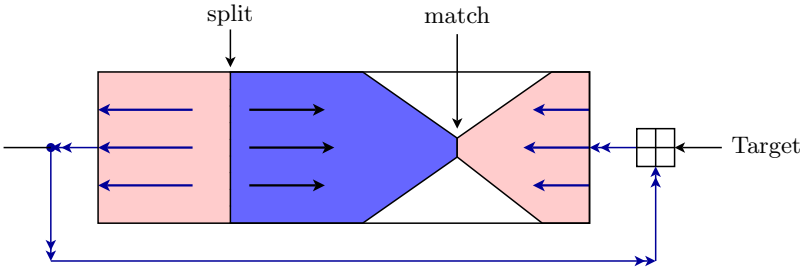

Fig. 2. Pseudo-Preimage Attack for Davies-Meyer Hash Functions

1. Split the compression function into two chunks, where the values in one chunk do not depend on some message word W_q and the values in the other chunk do not depend on another message word W_p ($p \neq q$). We follow the convention and call such words neutral with respect to the first and second chunk, respectively.
2. Fix all other values except for W_p, W_q to random values and assign random values to the chaining registers at the splitting point.
3. Start the computation both backward and forward from the splitting point to form two lists L_p, L_q indexed by all possible values of W_p and W_q and containing the computed values of the chaining registers at the matching point.
4. Compare two lists to find partial matches (match for one or few registers instead of full chaining) at the matching point.
5. Repeat the above three steps with different initial configurations (values for splitting point and other message words) until a full match is found.
6. Note that the match gives a pseudo-preimage as the initial value is determined during the attack. However, it is possible to convert pseudo-preimages to a preimage using a generic technique described in [15, Fact 9.99].

With the work effort of 2^{32} compression evaluations (assuming 32-bit registers), we obtain two lists, each one containing close to 2^{32} values of the register to match. When we consider all of the 2^{64} possible pairs, we expect to get around 2^{32} matches. This means that after 2^{32} computations we get 2^{32} matches on one register, effectively reducing the search space by 2^{32}. Leaving all the other registers to chance allows us to find a complete match and thus a pseudo-preimage in $2^{256-32} = 2^{224}$ computations if the internal state is 256 bits.

4 Finding independent chunks for SHA-2

The main challenge in this type of preimage attack is to find two sufficiently long sequences of steps (*chunks*) that can be made independent from the two different neutral words.

To this end, techniques such as local collisions and initial structures [22] have been used previously to increase the number of steps that can be dealt

with during the attack. Such methods are unique for each particular function and strongly depend on the details of the design.

In this section we present three tricks developed for SHA-2 that allow us to get more steps.

4.1 Using Message Expansion

Two-way expansion The two neutral message words do not need to be taken from the set $\{W_0, \ldots, W_{15}\}$. The message expansion (1) of SHA-2 works in such a way that all the expanded message words can be determined from any consecutive 16 words $W_z, \ldots, W_{z+15}$, $0 \leq z \leq 48$. Once we choose any 16 consecutive words, we can determine the other ones recursively in both directions. Assume we start with $\{W_z, \ldots, W_{z+15}\}$. To determine the optimal choice of the splitting point and the neutral words, let us first expand the message words in both directions. For backward direction:

$$W_{z-1} = W_{z+15} - \sigma_1(W_{z+13}) - W_{z+8} - \sigma_0(W_z) \ , \tag{2}$$
$$W_{z-2} = W_{z+14} - \sigma_1(W_{z+12}) - W_{z+7} - \sigma_0(W_{z-1}) \ , \tag{3}$$
$$W_{z-3} = W_{z+13} - \sigma_1(W_{z+11}) - W_{z+6} - \sigma_0(W_{z-2}) \ , \tag{4}$$
$$W_{z-4} = W_{z+12} - \sigma_1(W_{z+10}) - W_{z+5} - \sigma_0(W_{z-3}) \ , \tag{5}$$
$$W_{z-5} = W_{z+11} - \sigma_1(W_{z+9}) - W_{z+4} - \sigma_0(W_{z-4}) \ , \tag{6}$$
$$W_{z-6} = W_{z+10} - \sigma_1(W_{z+8}) - W_{z+3} - \sigma_0(W_{z-5}) \ , \tag{7}$$
$$W_{z-7} = W_{z+9} - \sigma_1(W_{z+7}) - W_{z+2} - \sigma_0(W_{z-6}) \ , \tag{8}$$
$$W_{z-8} = W_{z+8} - \sigma_1(W_{z+6}) - W_{z+1} - \sigma_0(W_{z-7}) \ , \tag{9}$$
$$W_{z-9} = W_{z+7} - \sigma_1(W_{z+5}) - W_z - \sigma_0(W_{z-8}) \ , \tag{10}$$
$$W_{z-10} = W_{z+6} - \sigma_1(W_{z+4}) - W_{z-1} - \sigma_0(W_{z-9}) \ , \tag{11}$$
$$W_{z-11} = W_{z+5} - \sigma_1(W_{z+3}) - W_{z-2} - \sigma_0(W_{z-10}) \ . \tag{12}$$

and for forward direction:

$$W_{z+16} = \sigma_1(W_{z+14}) + W_{z+9} + \sigma_0(W_{z+1}) + W_z \ , \tag{13}$$
$$W_{z+17} = \sigma_1(W_{z+15}) + W_{z+10} + \sigma_0(W_{z+2}) + W_{z+1} \ , \tag{14}$$
$$W_{z+18} = \sigma_1(W_{z+16}) + W_{z+11} + \sigma_0(W_{z+3}) + W_{z+2} \ , \tag{15}$$
$$W_{z+19} = \sigma_1(W_{z+17}) + W_{z+12} + \sigma_0(W_{z+4}) + W_{z+3} \ , \tag{16}$$
$$W_{z+20} = \sigma_1(W_{z+18}) + W_{z+13} + \sigma_0(W_{z+5}) + W_{z+4} \ , \tag{17}$$
$$W_{z+21} = \sigma_1(W_{z+19}) + W_{z+14} + \sigma_0(W_{z+6}) + W_{z+5} \ , \tag{18}$$
$$W_{z+22} = \sigma_1(W_{z+20}) + W_{z+15} + \sigma_0(W_{z+7}) + W_{z+6} \ , \tag{19}$$
$$W_{z+23} = \sigma_1(W_{z+21}) + W_{z+16} + \sigma_0(W_{z+8}) + W_{z+7} \ , \tag{20}$$

From above, we see that choosing W_{z+8} and W_{z+9} as neutral words is reasonable as those two message words appear late in forward and backward directions, respectively. We can use $\{W_{z-4}, \ldots, W_{z+8}\}$ as the first chunk which is independent from W_{z+9}, and $\{W_{z+9}, \ldots, W_{z+22}\}$ as the second chunk which is independent from W_{z+8}.

Message compensation Using the message stealing technique explained below in Section 4.2, we can move the splitting point four steps earlier, between W_{z+4} and W_{z+5}, and "steal" W_{z+8} back to the first chunk to preserve neutrality. Now, the first chunks becomes $\{W_{z-4}, \ldots, W_{z+4}, W_{z+8}\}$ and the second one $\{W_{z+5}, W_{z+6}, W_{z+7}, W_{z+9}, \ldots, W_{z+22}\}$.

With the help of W_{z+5} and W_{z+7}, we can extend the first chunk for several more steps. We note that W_{z+5} is used in (10) and (5), 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 (8), 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}.

$$W_{z+7} = \sigma_1(W_{z+5}) \ , \tag{21}$$
$$W_{z+9} = \sigma_1^2(W_{z+5}) \ , \tag{22}$$
$$W_{z+11} = \sigma_1^3(W_{z+5}) \ , \tag{23}$$
$$W_{z+13} = \sigma_1^4(W_{z+5}) \ , \tag{24}$$
$$W_{z+15} = \sigma_1^5(W_{z+5}) \ , \tag{25}$$
$$W_{z+12} = W_{z+5} \ , \tag{26}$$
$$W_{z+14} = 2\,\sigma_1(W_{z+5}) \ . \tag{27}$$

Now, $\{W_{z-10}, \ldots, W_{z+4}\}$ are all independent from W_{z+5} (or W_{z+9} as there is a bijection between them). The first chunk becomes $\{W_{z-10}, \ldots, W_{z+4}, W_{z+8}\}$ and the second one $\{W_{z+5}, W_{z+6}, W_{z+7}, W_{z+9}, \ldots, W_{z+22}\}$.

A similar compensation is difficult in the forward direction as message words $W_z, \ldots, W_{z+7}$ are used in consecutive steps. If we use any of them, all of them have to be used. However, we already used W_{z+5} and W_{z+7} above.

In total, there are 33 steps in both chunks, regardless of the choice of z. We will pick a particular value of z later in Section 4.3.

4.2 Message Stealing

What we call a message stealing technique allows us to have more steps around the splitting point by shifting some messages related to those two neutral message words so that the neutrality property is preserved. Such shifting can only be done when it does not alter the behavior of the step function.

Message stealing makes use of the absorption property of the function $\mathrm{IF}(x, y, z) = xy \oplus \overline{x}z$. If x is $\mathbf{1}$ (all bits are 1), then $\mathrm{IF}(\mathbf{1}, y, z) = y$ which means z does not affect the result of IF 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}

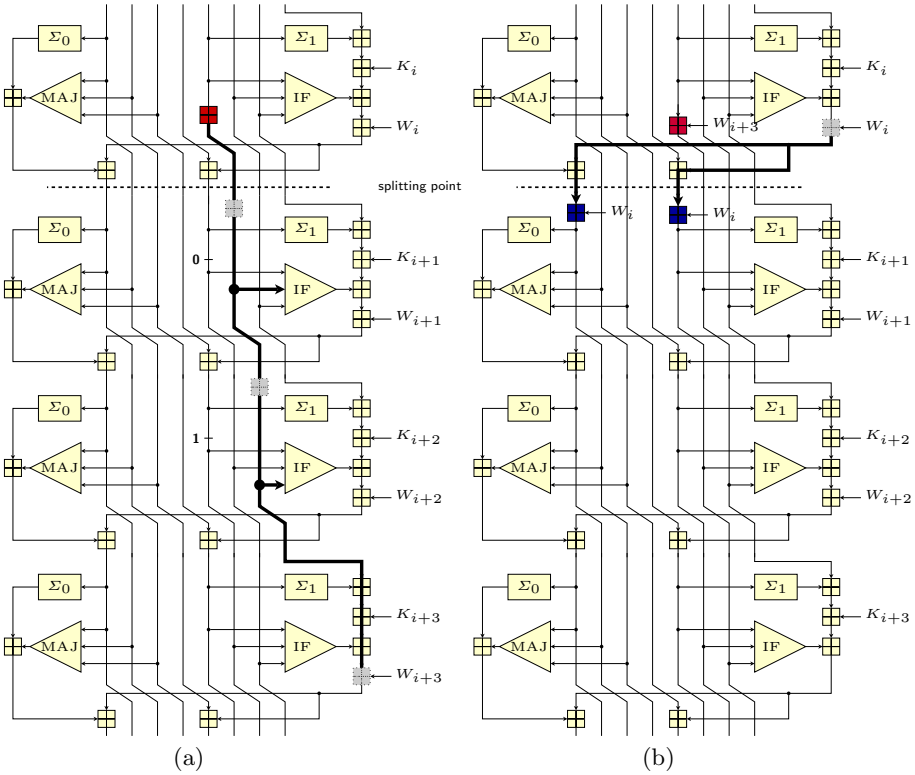

Fig. 3. Message stealing allows to (a) move the addition of W_{i+3} upwards provided that the IF functions absorb the appropriate inputs; (b) move W_i one step downwards.

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 IF, 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. 3(a). 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 further reduces the space of W_i to 2^{32-2l}. We choose $l = 10$ so that we have more or less same space for both W_i and W_{i+3}, the reason will be explained in Section 5.

The important thing to note here is that if we fix the values of F_{i+1}, G_{i+1} and of the sum $D_{i+1} + H_{i+1}$ (this is possible since $\mathcal{S}_{i+1}$ is a splitting point we have complete control over), we can precompute the set of good values for W_i and store them in a table. Then, we can later recall them at no cost.

On the other hand, message word W_i can be moved to step $i + 1$ with no constraint, as shown in Fig. 3(b).

This essentially swaps the order of the two words W_i and W_{i+3} which was used in message compensation.

4.3 Indirect Partial Matching

In this section we explain how to use a modified partial matching method to extend the attack by 9 more steps. The basic partial matching technique would give us 7 more steps [8], since we need at least one register to perform the matching and the computation backward "looses" one register per step. However, using some tricks we explain below we can get two more steps.

The partial matching is shown in Fig 4. Let us fix $z = 11$ (this choice will become clear in a moment). The two neutral words are W_{16} and W_{19} and the corresponding chunks are $\{W_1, \ldots, W_{15}, W_{19}\}$ and $\{W_{16}, W_{17}, W_{18}, W_{20}, \ldots, W_{33}\}$. We want to gain two more steps, one step at each end of both chunks, where message words W_0 and W_{34} are used.

We compute the value of A_{35} from both directions and try to find matches, other seven registers are computed and checked if A_{35} matches. In the forward direction, A_{35} can be calculated from $\mathcal{S}_{34}$ and W_{34}. $\mathcal{S}_{34}$ is independent from W_{19}, however W_{34} is not. Substituting $z = 11$ into (20) we get $W_{34} = \sigma_1(W_{32}) + W_{27} + \sigma_0(W_{19}) + W_{18}$ and see that although W_{34} is not neutral about W_{19}, it can be expressed as a sum of two independent functions of W_{19} and W_{16}. Moreover, the value of W_{34} is added to get the value of A_{35}, so we can express A_{35} as $\psi(W_{16}) + \sigma_0(W_{19})$.

Similarly, we compute backward and express A_{35} as $\mu(W_{19}) - W_{16}$. Now we need to find matches such that $\psi(W_{16}) + \sigma_0(W_{19}) = \mu(W_{19}) - W_{16}$, which is equivalent to $\psi(W_{16}) + W_{16} = \mu(W_{19}) - \sigma_0(W_{19})$. Let us define $\psi'(W_{16}) := \psi(W_{16}) + W_{16}$ and $\mu'(W_{19}) := \mu(W_{19}) - \sigma_0(W_{19})$. Instead of finding matches for A_{35} directly, we can compute ψ' and μ' independently and match.

This is possible only when both message words used at the beginning and the end of the partial match can be expressed as a sum of two independent functions. This is true when $z = 11$ and it explains our choice.

4.4 Overview of the Attack

The overview of the separation of chunks is shown in Fig. 5. The pink/lighter filled boxes (▫) denote registers from the first chunk that depend only on W_{19}. The blue/darker boxes (■) denote variables from the second chunk that depend only on W_{16}. Mixed color boxes (both ◪ and ◪) denote registers that can be expressed as a sum modulo 2^{32} of two independent functions of neutral variables

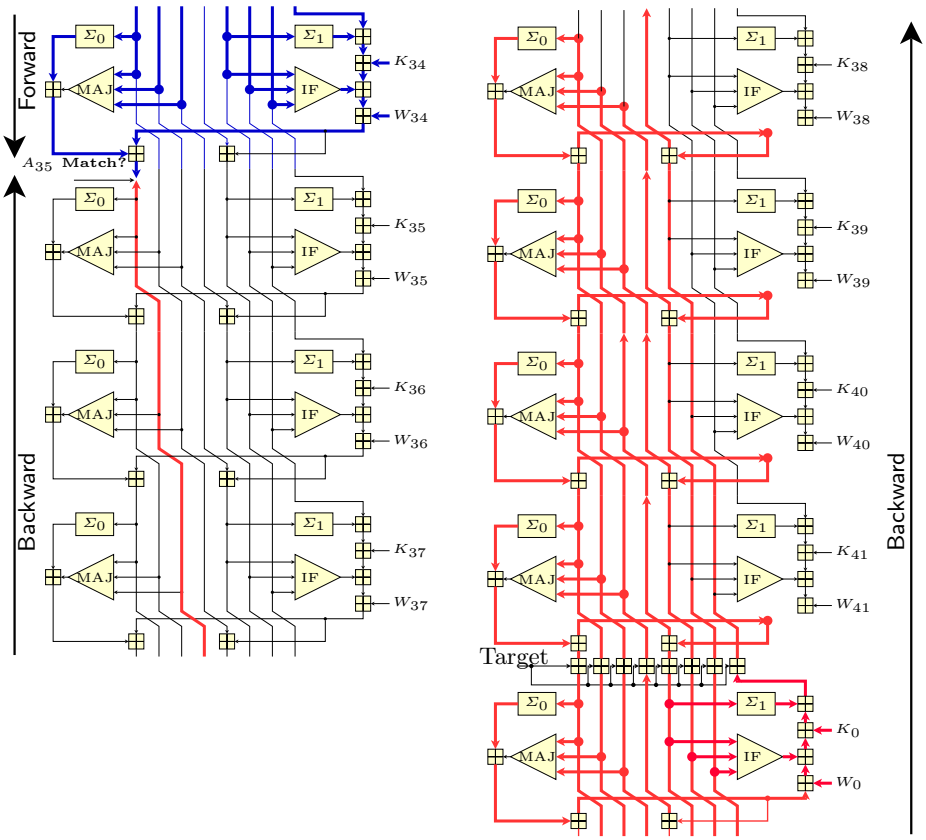

Fig. 4. Partial Matching

W_{19} and W_{16}. Register A_{35} is the matching point and is outlined with a bold line (⊠). Finally, registers that depend on both neutral variables in a complicated way are drawn as crossed-out boxes (⊠) and empty boxes represent registers independent from both neutral words.

5 Algorithm and Complexity

Once we have explained all the elements of our attack, we can put them together in one algorithm as described below.

1. Randomly choose the values for internal chaining $\mathcal{S}_{17}$ (after the movement of message words by message stealing) and message words not related to neutral words, i.e. W_{11}, W_{12}, W_{13}, W_{14}, W_{15}, W_{17}, W_{21}. Let us call this an initial configuration.
2. For all possible W_{16} (there are 2^{32-2l} different values as described in Section 4.2), compute the corresponding W_{18}, W_{20}, W_{22}, W_{23}, W_{24}, W_{25}, W_{26}

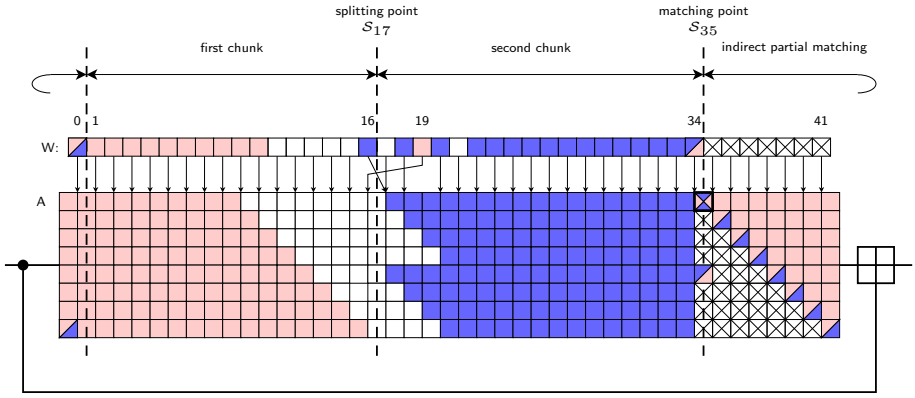

Fig. 5. Separation of chunks and dependencies of state words in the attack.

as in (21)-(27). Compute backward and find $\psi'(W_{16})$ as described in Section 4.3. Store the result $(W_{16}, \psi'(W_{16}))$ in a list L_p.

3. For all possible W_{19} (there are 2^l different values), compute forward and find μ' as described in Section 4.3. Store the result $(W_{19}, \mu'(W_{19}))$ in a list L_q.
4. Compare the values of ψ' and μ' and find matches. If a match is found, compute other registers $B_{35}, \ldots, H_{35}$ and see whether they match from both directions. If they do, compute the pseudo-preimage as $\mathcal{S}_0$ and $W_0, \ldots, W_{15}$.
5. Repeat steps 1 - 4 with different initial configurations until a full match is found.
6. Repeat step 5 to find sufficiently many pseudo-preimages, then find a preimage according to Fact 9.99 [15].

The computational complexity for step 2 and 3 is 2^{l-1} and 2^{31-2l} (We estimate this by approximating one chunk as half of the compression function, it is less in fact), and it generates 2^{32-l} pairs. The chance for one pair to be a good match is 2^{-256}, so we need to repeat step 1-3 for $2^{256-32+l}$ times. The overall complexity for finding one pseudo-preimage becomes $2^{224+l} \cdot (2^{l-1} + 2^{31-2l})$. Choosing the optimal value $l = 10$, the complexity is $2^{245.32}$. According to Fact 9.99 [15], sufficiently many pseudo-preimages in step 6 in our case is $2^{(256-245.32)/2} = 2^{5.34}$, then the overall complexity for finding preimage is $2^{(256+245.32)/2+1} = 2^{251.66}$.

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 W_{13}, W_{14} and W_{15} 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 W_{14} and W_{15} as random, later we can still resolve the length using expandable messages [10].

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 [9], then follow the expandable message 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 less than $2^{(n-l)/2}$. We need additional $128k$ bytes memory to store the k block multi-collisions.

We note most of the message words are randomly chosen, it naturally gives second preimages with high probability. Above multi-preimages are most probably multi-second preimages.

6 Conclusions

In this paper we presented a preimage attack on a version of SHA-256 reduced to 42 steps out of 64. This number of steps is possible thanks to four new techniques we presented: message stealing, two-way message expansion, message compensation and indirect partial matching. Each one of them allows to improve over the standard preimage attack and allows to add more steps.

The same attack also works for SHA-512 as the message expansion is the same and the attack does not depend on the details of functions $\sigma_{0/1}$ and $\Sigma_{0/1}$. We also expect that a variant of this attack would work for the function DHA-256 [11].

The preimage attack we presented creates a very interesting situation for SHA-256 when a preimage attack, covering 42 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.

Acknowledgement

We would like to thank Lars R. Knudsen, Christian Rechberger, and the anonymous reviewers of ASIACRYPT 2009 for the helpful comments. Jian Guo was supported in part 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. Kazumaro Aoki and Yu Sasaki. Preimage attack on one-block MD4, 63-step MD5 and more. In *Selected Areas in Cryptography - SAC 2008*, volume 5381 of *LNCS*, pages 103–119. Springer, 2009.

2. Jean-Philippe Aumasson, Willi Meier, and Florian Mendel. Preimage attacks on 3-pass HAVAL and step-reduced MD5. In *Selected Areas in Cryptography - SAC 2008*, volume 5381 of *LNCS*, pages 120–135. Springer, 2009.
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 *LNCS*, pages 179–202. Springer, 2008.
4. Christophe De Canniére and Christian Rechberger. Finding SHA-1 characteristics: General results and applications. In *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 1–20. Springer, Dec 2006.
5. Henri Gilbert and Helena Handschuh. Security analysis of SHA-256 and sisters. In *Selected Areas in Cryptography 2003*, volume 3006 of *LNCS*, pages 175–193. -Verlag, 2003.
6. Shai Halevi and Hugo Krawczyk. Strengthening digital signatures via randomized hashing. In *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *LNCS*, pages 41–59. Springer, 2006.
7. Sebastiaan Indesteege, Florian Mendel, Bart Preneel, and Christian Rechberger. Collisions and other non-random properties for step-reduced SHA-256. In *Workshop Records of SAC 2008*, pages 257–274, 2008.
8. Takanori Isobe and Kyoji Shibutani. Preimage attacks on reduced Tiger and SHA-2. In *Fast Software Encryption – FSE 2009*, LNCS. Springer, 2009. to appear.
9. Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *LNCS*, pages 306–316. Springer, 2004.
10. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 474–490. Springer, 2005.
11. Jesang Lee, Donghoon Chang, Hyun Kim, Eunjin Lee, and Deukjo Hong. A new 256-bit hash function dha-256 - enhancing the security of sha-256. NIST - First Cryptographic Hash Workshop, October 31-November 1, 2005.
12. Gaëtan Leurent. MD4 is not one-way. In *Fast Software Encryption – FSE 2008*, volume 5086 of *LNCS*, pages 412–428. Springer, 2008.
13. Krystian Matusiewicz, Josef Pieprzyk, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Analysis of simplified variants of SHA-256. In *Western European Workshop on Research in Cryptology – WEWoRC 2005*, volume P-74 of *Lecture Notes in Informatics*. Gesellschaft für Informatik, 2005.
14. Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen. Analysis of step-reduced SHA-256. In *Fast Software Encryption – FSE 2006*, volume 4047 of *LNCS*, pages 126–143. Springer, 2006.
15. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. available online from `http://www.cacr.math.uwaterloo.ca/hac/`.
16. National Institute of Standards and Technology. Secure hash standard (SHS). FIPS 180-2, August 2002.
17. Somitra Kumar Sanadhya and Palash Sarkar. New collision attacks against up to 24-step sha-2. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 91–103. Springer, 2008.
18. Somitra Kumar Sanadhya and Palash Sarkar. Non-linear reduced round attacks against SHA-2 hash family. In *Information Security and Privacy – ACISP 2008*, volume 5107 of *LNCS*, pages 254–266. Springer, 2008.

19. Yu Sasaki and Kazumaro Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In *Advances in Cryptology – ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 253–271. Springer, 2008.
20. Yu Sasaki and Kazumaro Aoki. Preimage attacks on MD, HAVAL, SHA, and others. CRYPTO 2008 Rump session presentation, August 2008. Slides available at `http://rump2008.cr.yp.to/`.
21. Yu Sasaki and Kazumaro Aoki. Preimage attacks on step-reduced MD5. In *Information Security and Privacy – ACISP 2008*, volume 5107 of *LNCS*, pages 282–296. Springer, 2008.
22. Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 134–152. Springer, 2009.
23. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.
24. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
25. Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.
26. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *LNCS*, pages 1–16. Springer, 2005.
27. Hirotaka Yoshida and Alex Biryukow. Analysis of a SHA-256 variant. In *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *LNCS*, pages 245–260. Springer, 2006.
28. Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The second-preimage attack on MD4. In *Cryptology and Network Security – CANS 2005*, volume 3810 of *LNCS*, pages 1–12. Springer, 2005.