

Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2

Jian Guo¹, San Ling¹, Christian Rechberger², and Huaxiong Wang¹

¹ Division of Mathematical Sciences,
School of Physical and Mathematical Sciences,
Nanyang Technological University, Singapore

² Dept. of Electrical Engineering ESAT/COSIC, K.U.Leuven,
and Interdisciplinary Institute for BroadBand Technology (IBBT),
Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium.

guojian@ntu.edu.sg

Abstract. We revisit narrow-pipe designs that are in practical use, and their security against preimage attacks. Our results are the best known preimage attacks on Tiger, MD4, and reduced SHA-2, with the result on Tiger being the first cryptanalytic shortcut attack on the full hash function. Previous attacks on Tiger were only on up to 17 out of 24 rounds, and had a higher complexity (2^{185} for 17 rounds). Our attacks runs in time $2^{184.3}$ and only need memory of order $2^{16.7}$, whereas all previous preimage attacks on reduced Tiger needed much more memory (e.g. 2^{160} for 17-step Tiger). The new preimage attack on MD4 is about 30 times faster than the previously best known, but requires more memory. However, using pre-computation techniques, the time complexity for finding a new preimage or second preimage for MD4 can now be as low as $2^{70.4}$ and 2^{64} MD4 computations, respectively. The second preimage attack works for all messages longer than 2 blocks.

To obtain those results, we extend the meet-in-the-middle framework recently developed by Aoki and Sasaki in a series of papers. In addition to various algorithm-specific techniques, we use a number of conceptually new ideas that are applicable to a larger class of constructions. Among them are (1) incorporating multi-target scenarios into the MITM framework, leading to faster preimages from pseudo-preimages, (2) a simple precomputation technique that allows for finding new preimages at the cost of a single pseudo-preimage, and (3) multi-word compensation instead of single-word compensation that allows to speed-up preimage search, or allows attacks on variants with more rounds. All those techniques developed await application to other hash functions. To illustrate this, we give as another example improved preimage attacks on SHA-2 members.

Keywords: Preimage, MD4, Tiger, SHA-2, Hash function, Cryptanalysis, Meet-in-the-Middle

1 Introduction

After the spectacular collision attacks on MD5 and SHA-1 by Wang et al. and follow-up work [13, 36, 40, 41], implementors reconsider their choices. While starting a very productive phase of research on design and analysis of cryptographic hash functions, the impact of these results in terms of practical and worrying attacks turned out to be less than anticipated (exceptions are e.g. [22, 35, 37]). Instead of collision resistance, another property of hash functions is more crucial for practical security: preimage resistance. Hence, research on preimage attacks and the security margin of hash functions against those attacks seems well motivated, especially if those hash functions are in practical use.

An important ongoing challenge is to find an efficient and trustworthy new hash function for long term use (e.g. in the SHA-3 competition). For new hash functions, an important first step to get confidence in them is to apply known cryptanalysis methods in order to break it. So the cryptanalysts' toolbox needs to be well equipped for this.

The new techniques we present in this paper contribute to both issues at the same time. They give new, generically applicable tools to cryptanalysts for analyzing compression functions and hash functions, and at the same time applications of them improve (sometimes by far) upon known preimage attacks on hash functions in practical use, like MD4, Tiger, and SHA-256/512. In the following we outline the new tools and new results that will be described later in the paper. We describe them to fit into the meet-in-the-middle (MITM) framework of Aoki and Sasaki as recently developed in a series of papers [6–8, 33, 34], although we note that the basic approach was pioneered by Lai and Massey [21]. Other interesting approaches to preimage attacks appeared in [11, 14, 19, 20, 24, 25, 30, 38, 42].

New methods. New methods described in this paper that are independent of a particular attack or hash functions are the following:

- **Multi-word compensation** instead of single-word compensation, leading to either faster preimage attacks, or preimage attacks on variants with a higher number of rounds. For MD4, this idea leads to speed-ups of the preimage search (see Section 3). For Tiger, this idea is used differently, namely to allow attacks on variants with many more rounds than done before, see Section 4.
- **Incorporating multi-target scenarios into the MITM framework**, leading to faster preimage attacks. The MITM framework is the basis for several theoretically interesting results on the preimage resistance of various hash functions. However, results are limited to attack complexities of 2^{n-e} for rather small e . One reason for this is that in order to exploit all the options of this framework, matching points of the meet-in-the-middle phase can be anywhere in the computation of the compression function, and not necessarily at their beginning or end. Even though this gives an attacker more freedom in the design of a compression function attack, this always led to big efficiency loss when the attack on the compression function is converted to an attack on the hash function. Hence, an attacker basically had to choose between a more restricted (and potentially much slower) strategy in the compression function attack that allows more control over the chaining values and in turn allows efficient tree- or graph based conversion methods, or to fully exploit the freedom given by the latest versions of the MITM framework in the compression function attack at the cost of inefficient conversion method. In Section 2.2 we describe a way to combine the best of both worlds. Later in the paper, this results in the best known preimage attacks for Tiger and the SHA-2 members.
- A simple **precomputation technique** that allows for finding new preimages at the cost of a single pseudo-preimage. See Section 3 for an application to MD4, where this approach is shown to outperform any point on the time/memory trade-off curve by Hellman [17] (which was proven optimal in [10] in the generic case).

New results in perspective. In addition to the conceptual ideas that contribute to the cryptanalysts' toolbox in general, we also apply those ideas and present concrete results. In fact, we manage to improve the best known preimage attacks on a number of hash functions in practical use.

- **Tiger:** One of the few unbroken, but time-tested hash functions, designed by Anderson and Biham [5] in 1996. Tiger is sometimes recommended as an alternative to MD4-like designs like SHA-1, especially because it is faster than SHA-1 on common platforms. Tiger is in practical use e.g. in decentralized file system, or in many file sharing protocols and applications, often in a Merkle-tree construction (also known as TigerTree [3]). The best collision attack on Tiger is on 19 rounds [26].¹ So far the best preimage attack on the Tiger hash function was on 17 out of 24 rounds, with time complexity 2^{185} and needs memory of order 2^{160} . Our new attack improves on all three of those aspects and seems to be the *first cryptanalytic shortcut attack on the full Tiger hash function*. Our attack on the full 24 rounds hash functions has time complexity $2^{184.3}$ (compression function attack is $2^{179.5}$) and memory requirements are only in the order of $2^{16.7}$. These results are obtained using the multi-word compensation technique and the multi-target technique mentioned above, and a dedicated technique to construct an initial structure in a precomputation.
- **MD4:** Even though very efficient collision search methods exist for MD4 [39, 31], this hash function is still in practical use. Examples include password handling in Windows NT, the S/KEY one-time-password system [16], integrity checks in popular protocols e.g. *rsync* [2] or file-sharing protocols [1] and applications like Overnet, eDonkey, etc. The time complexity for the best known compression function attack is reduced from 2^{96} (by Leurent [23]) to 2^{64} . And for the first time, we find a preimage attack on MD4 with time complexity less than 2^{100} . the new attack on the hash function has time complexity 2^{97} . Assuming precomputation, the effort for finding any new preimage (be it for the same or a different target hash value as a challenge) can now be as low as $2^{70.4}$. Both, the multi-word compensation technique, and the precomputation technique mentioned above are used to obtain those results.
- **SHA-2:** The members of the SHA-2 family of hash functions are probably one of the most interesting cryptanalytic targets. Not only because of the uptake of its adoption in all places where a hash function is needed (and they are countless), but also because they are used to compare them to candidates of the ongoing SHA-3 competition. We use SHA-2 members as an example to illustrate the effect of using the multi-target scenario. This way we also improve the best known preimage attacks on reduced SHA-256 and reduced SHA-512. They are described in Appendix A.

Outline. This paper is organized as follows. Section 2 describes the MITM preimage attack, describes four different methods converting pseudo preimage to preimage (including two new ones), and also recapitulates techniques to extend MITM based preimage attacks. We apply these new techniques to MD4 and Tiger in Section 3 and Section 4, respectively. Section 5 concludes the paper. Results on SHA-2 appear in Appendix A.

2 The Meet-in-the-Middle Preimage Attack

The general idea of the preimage attack, illustrated in Fig 1, can be explained as follows:

1. Split the compression function into two chunks, where the values in one chunk do not depend on some message word W_p and the values in the other chunk do not depend

¹ If an attacker can choose both the difference and the actual values not only of the message, but also of the chaining input, then the full compression function can be attacked, see Mendel and Rijmen [27]. However, this attack can not be extended on the hash function, whereas all the attacks in this paper can.

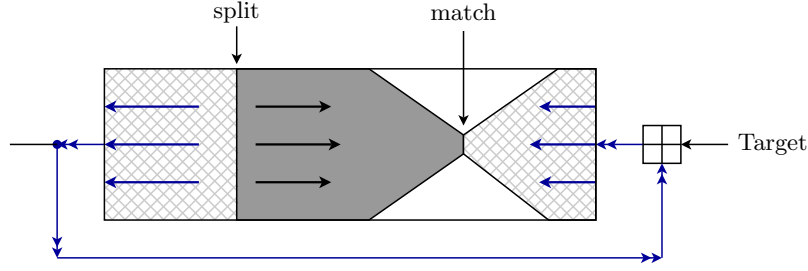


Fig. 1. Meet-in-the-Middle Pseudo-Preimage Attack against Davies-Meyer Hash Functions

on another message word W_q ($p \neq q$). 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 , containing the computed values of the chaining registers at the matching point.
4. Compare two lists to find partial matches (match for one or a few registers instead of the 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 [29, Fact 9.99]. One can compute many pseudo-preimages, and then find a message which links the IV to one of the input chaining of the pseudo-preimages, as demonstrated in Fig 2(a).

With the work effort of 2^l compression evaluations (let the space for both W_p and W_q to be 2^l), we obtain two lists, each one containing 2^l values of the register to match. When we consider all of the 2^{2l} possible pairs, we expect to get around 2^l matches (assume we match l bits at the matching point). This means that after 2^l computations we get 2^l matches on one register, effectively reducing the search space by 2^l . Leaving all the other registers to chance allows us to find a complete match and thus a pseudo-preimage in 2^{n-l} computations if the chaining is of n bits. We repeat the pseudo-preimage finding $2^{l/2}$ times, which costs $2^{n-l/2}$, and then find a message which links to one of the $2^{l/2}$ pseudo-preimages, this costs $2^{n-l/2+1}$, with memory requirement of order 2^l .

Remark on bits for partial matching. Assume we have m bits for partial matching, we expect 2^{2l-m} good candidates with the m -bit matched. However we still need to check if one of the remaining candidates gives a full match (pseudo preimage), the checking costs about 2^{2l-m} (A bit less indeed, since we can store the computed candidates up to point before partial matching, and re-check the partial matching portion only). To minimize the time complexity, we require $m \geq l$, so that $2^{2l-m} \leq 2^l$.

2.1 Multi-Target Pseudo Preimage (MTPP)

In [23], Leurent provides an unbalanced-tree multi-target pseudo preimage method (refer Fig 2(b)) to convert the pseudo preimages to preimage with complexity $(l \ln(2) + 1) \cdot 2^{n-l}$,

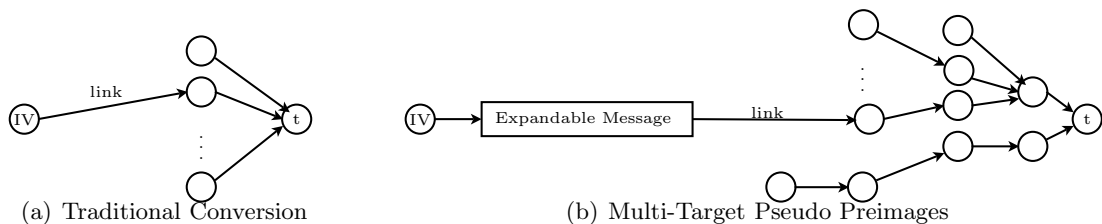


Fig. 2. Converting Pseudo Preimages to Preimage: circle denotes state, arrow denotes message block

compared with $2^{n-l/2+1}$ in [29, Fact 9.99]. Consider the matching point is at the end of compression function. The matching process is to find $L_p + L_q = t$ (t is target). When we are given k targets, the chance to find a match increases by a factor k , i.e. it takes $2^{n-l}/k$ to find a pseudo-preimage which links one of the k targets. To find 2^k many pseudo-preimages, it takes $2^{n-l}/1 + 2^{n-l}/2 + 2^{n-l}/3 + \dots + 2^{n-l}/2^k \simeq k \ln(2) \cdot 2^{n-l}$. To find a preimage, it is expected to repeat 2^{n-k} many blocks finding a message, which links to one of the 2^k targets. Taking the optimal $k = l$, the overall complexity is

$$2^{n-k} + k \ln(2) \cdot 2^{n-l} = (l \ln(2) + 1) \cdot 2^{n-l} . \quad (1)$$

Note this conversion does not necessarily increase the memory requirement, i.e. it can be the same as for finding a pseudo preimage, since we compute the 2^l pseudo preimage in sequence.

Enhanced 3-Sum Problem. The above conversion comes with an assumption that the matching can be done within 2^l . Note from each chunk, we have 2^l candidates (denote as C_p and C_q), and given 2^k targets (denote as T), we are to find *all* possible (c_p, c_q, t) , where $c_p \in C_p$, $c_q \in C_q$ and $t \in T$, such that $c_p + c_q = t$. We call this problem Enhance 3-Sum Problem, where the standard 3-sum problem *decides* whether there is a solution [4]. Current research progress [9] shows that the problem can be solved in $O(2^{2l})$ or slightly faster. So this approach expects the matching to be done in 2^{2l} (for $k = l$) instead of the assumed 2^l . However this only applies to the final feed-forward operation (“+” in most of the MD hash families), which is a small portion of the compression. Hence this approach expects 2^{2l} “+” operations to be somewhat equivalent to 2^l compression computations by counting number of “+” in the compression, when l is relatively small (e.g. ≤ 7 for MD4 and Tiger).

2.2 Generic Multi-Target Pseudo Preimage (GMTPP)

The framework of Aoki and Sasaki could not take advantage of a multi-target scenario to speed-up conversion from pseudo-preimage to preimages. The reason is a rather strong requirement on the compression function attack by the MTPP approach outlined above. By generalizing the setting, we weaken the assumption on the compression function attack, and hence allow the MITM framework to take advantage of new speed-up possibilities.

When the matching point is not at the end of the compression function, we can still make use of the multi-targets. Consider the sum of the size of W_p and W_q to be $2l$, and assume we can re-distribute the $2l$ bits to W_p and W_q freely². Given 2^k targets, we can distribute the $2l$ bits to $l+k/2$ and $l-k/2$, so that we can have $2^{l+k/2}$ many candidates for

² This being a very natural assumption is illustrated by the fact that for both MD4 and SHA-2 we can give a useful application that uses this.

each direction (combining the $2^{l-k/2}$ and 2^k targets to get $2^{l+k/2}$ candidates). In this way, we can find a pseudo preimage in $2^{n-l-k/2}$ and finding 2^k targets costs $\sum_{i=1}^{2^k} 2^{n-l_i-1/2} \simeq 2^{n-l+1+k/2}$. So we can find preimage in

$$2^{n-k} + 2^{n-l+1+k/2} = 3 \cdot 2^{n-2l/3} \quad (2)$$

taking the optimal $k = 2l/3$. For this method to work, we will need more matching bits: $4l/3$ bits instead of l (we have $2^{4l/3}$ candidates for both directions). The memory requirement hence increases to $2^{4l/3}$. Here we trade memory for speed from $2^{n-l}/2^l$ (time/memory) to $2^{n-l-k/2}/2^{l+k/2}$ for $k = 0, \dots, 2l/3$. And we have full control on any other speed/memory balance in-between by making use of the proper number of given targets, i.e. less than 2^k .

2.3 Finding Preimages using Large Precomputation (FPLP)

Here, we describe a simple technique to turn a large class of pseudo preimage attacks into preimage attacks without a speed loss. The method requires an initial large precomputation of order 2^n and hence needs to be compared with the time/memory trade-off proposed by Hellman [17]. This means that the time and memory requirements of a dedicated attack need to be below the $TM^2 = N^2$ tradeoff curve in order to be considered to be an improvement over the generic attack.

The approach may be described as follows: in the precomputation phase, one tries to find messages for all possible chaining outputs (i.e. find m_i such that $hash(m_i) = h_T$ for (almost) all possible target hash values h_T , but only store those messages m_i in a table together with the output, which can actually “happen” in the pseudo-preimage attack. In the online phase, after the pseudo-preimage attack is carried out, a simple lookup into this memory is enough to find the right linking message. The memory requirement depends on the subset of all possible chaining inputs the pseudo-preimage attack can possibly result in. If this subset can be restricted enough, and the pseudo-preimage attack is fast enough, the approach may outperform the generic method. In Section 3.3, we give an actual example where this is the case for MD4, which seems to be the first result of this kind.

Four different conversion techniques are summarized in Table 1. Our point here is to illustrate and compare various approaches and the *assumptions* they make on the compression function attack. For simplicity, other conversion methods somewhat similar to MTPP (tree construction in [28], alternative tree and graph construction in [14]) are not listed. As an example, the new attack on the MD4 compression function satisfies only assumptions of the traditional and the FPLP approach, the new attack on the Tiger compression function and the SHA-2 compression function satisfy the assumption made by the GMTPP approach.

Name	Reference	Time	Memory	Bits for PM	Assumption
Traditional	Section 2, [29]	$2^{n-l/2+1}$	2^l	l	-
GMTPP	new, Section 2.2	$3 \cdot 2^{n-2l/3}$	$2^{4l/3}$	$4l/3$	Redistribution of neutral bits
MTPP	Section 2.1, [23]	$(l \ln(2) + 1) \cdot 2^{n-l}$	2^l	$2l$	Enhanced 3-SUM, PM at feedforward
FPLP	new, Section 2.3	2^{n-l}	$\max(2^z, 2^l)$	l	2^n precomputation subset of input chaining values of size 2^z

Table 1. Comparison of methods converting pseudo preimage to preimage

2.4 Intro to some MITM techniques for compression function attacks

There were several techniques developed recently to extend the preimage attack for more steps or to reduce the time complexity. To help understanding the techniques developed later in the paper, we will introduce the concepts initial structure and partial matching here.

Initial Structure. An Initial Structure can swap the order of some message words near the splitting point, so that the length of two chunks can be extended. As shown in Fig 3, originally both chunks p and q contains both neutral words W_p and W_q . After the initial structure, we essentially swap the W_p and W_q near the splitting point, so that chunk p is independent from W_q and chunk q is independent from W_p now.

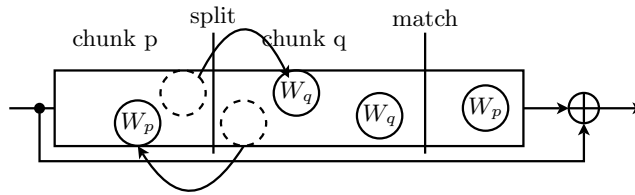


Fig. 3. Initial Structure

Partial Matching. Partial matching (PM) can extend the attack for a few additional steps. As shown in Fig 4, there are W_p and W_q near the matching point, which appear in other chunks and destroy the independence. However we can still compute few bits at the matching point, independently for both chunks, assuming no knowledge of W_p and W_q near the matching point. Partial fixing will fix part of the W_p and W_q so that we can still make use of those fixed bits, and extend the attack for a few more steps. Sometimes, W_p and W_q near the matching point behaves in such a way that we can express the matching point as $f(W_q) + \sigma(W_p)$ from chunk q, and $g(W_p) + \mu(W_q)$ from chunk p. So we can compute $f(W_q) - \mu(W_q)$ from chunk q and $g(W_p) - \sigma(W_p)$ from chunk p independently, and then find matches. This is called indirect partial matching, and was first used in [6].

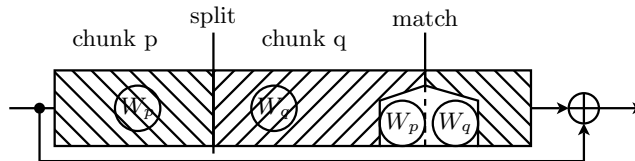


Fig. 4. Partial Matching

The success of the MITM preimage attack relies mainly on the choice of neutral words and number of steps the initial structure and partial matching can do. So we will mainly discuss those three points when the attack is applied on MD4, Tiger, and SHA-2.

3 Improved Preimage Attack against MD4

3.1 Description of MD4

MD4 follows the traditional MD-strengthening, the original message was padded by 1, followed by many 0s and 64-bit length information so that the length of padded message

becomes multiple of 512. Then divide the padded message into blocks of 512 bits and feed into the compression function iteratively. Output of the final compression is the output of the hash. The compression function follows Davies-Meyer construction, and come with two major parts: message scheduling and step function. Message scheduling divides the 512-bit message block into 16 words (32 bit each) and expand them into 48 using permutations, i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15. Starting from input chaining, the expanded words are fed into the step function iteratively. The output of the last step is added with the input chaining to give the output of compression function. The step function is shown in Fig 5, where C_i and r_i are predefined constants and W_i is the message word used at i -th step. Since only one register is modified at each step, one can use the notation $Q_i = (Q_{i-4} \boxplus F_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus M_{\pi(i)} \boxplus C_i) \lll r_i$ for $i = 0, \dots, 47$, where the constants and functions are defined as follows.

First pass	$0 \leq i \leq 16$	$F_i = \text{IF}$	$C_i = K_0 = 0$
Second pass	$16 \leq i \leq 32$	$F_i = \text{MAJ}$	$C_i = K_1 = 0x5A827999$
Third pass	$32 \leq i \leq 48$	$F_i = \text{XOR}$	$C_i = K_2 = 0x6ED9EBA1$

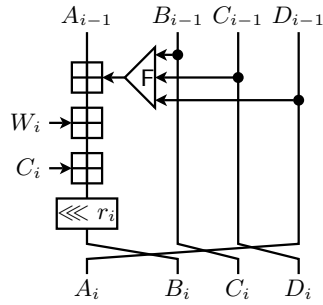


Fig. 5. MD4 Step Function

3.2 Faster Pseudo Preimage Attack

In this section, we present a pseudo preimage attack in 2^{64} . Separation of chunks is shown in Fig 6. We choose (M_9, Q_6) as W_p and (M_{14}, Q_{26}) as W_q . The initial structure covers 17 steps from Step 10 to Step 26, as shown in Fig 7. Note, once every register and message words shown in Fig 7 except $Q_6, M_{10}, M_{14}, M_9, Q_{26}$ are fixed. The relation between Q_6 and Q_{26} satisfies

$$Q_{26} - Q_6 = \varphi(M_9, M_{10}, M_{14}) \quad (3)$$

for some φ . Note φ is fixed when all other registers/message words are fixed.

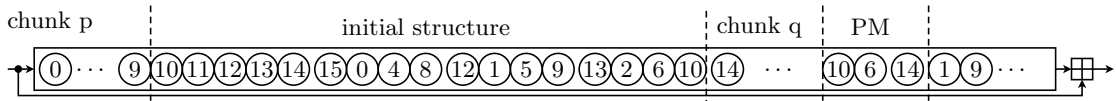


Fig. 6. Pseudo Preimage for MD4 in 2^{64}

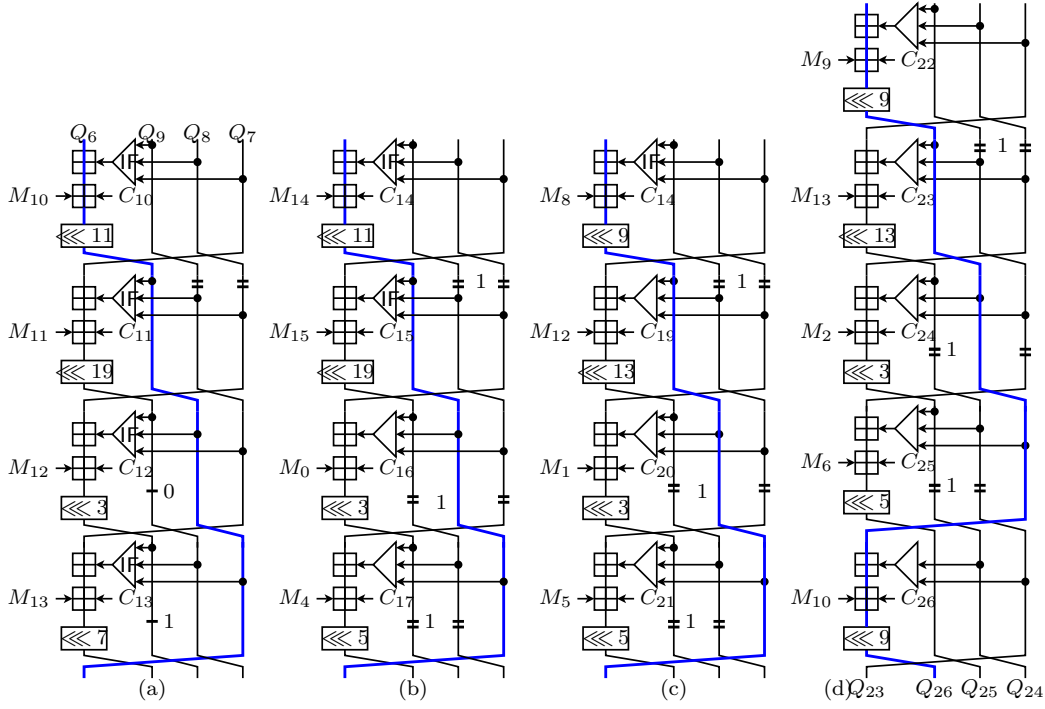


Fig. 7. 17-Step Initial Structure for MD4

We fix all other registers in the Fig 7 in such a way that the influence of the registers falling in the bold line is absorbed when passing through the F function. Note F is IF for first pass and MAJ for second pass. To deal with IF,

$$\text{IF}(x, y, z) = \begin{cases} y & \text{set } y=z \text{ when variable falls in } x \\ z & \text{set } x=0 \text{ when variable falls in } y \\ y & \text{set } x=1 \text{ when variable falls in } z \end{cases}$$

(1 denotes 0xFFFFFFFF) and we force the other two inputs equal for MAJ. All required values are shown in Fig 7. All details in each step are shown below

- step 11, $C_{10} = D_{10} \Leftrightarrow Q_8 = Q_9$
- step 12, $B_{11} = 0 \Leftrightarrow Q_{11} = 0$, use M_{11} and Q_7
- step 13, $B_{12} = 1 \Leftrightarrow Q_{12} = 1$, use M_{12} and Q_8
- step 15, $C_{14} = D_{14} = 1$, use M_{13} and Q_9
- step 16, $B_{15} = D_{15} = 1$, use M_{15}
- step 17, $B_{16} = C_{16} = 1$, use M_0
- step 19, $C_{18} = D_{18} = 1$, use M_4
- step 20, $B_{19} = D_{19} = 1$, use M_{12}
- step 21, $B_{20} = C_{20} = 1$, use M_1
- step 23, $C_{22} = D_{22} = 1$, use M_5
- step 24, $B_{23} = D_{23} = 1$, use M_{13}
- step 25, $B_{24} = C_{24} = 1$, use M_2

Solving all above leaves $M_0 = M_1 = M_2 = M_4 = M_5 = M_{12} = M_{13} = 1 - K_1$, $M_{15} = 0$, $M_{12} = 3 - 2K_1$ and $Q_8 = Q_9 = K_1 - 2$.

The pseudo preimage algorithm.

1. Fix all mentioned message words/registers as above.
2. Randomly assign all other message words, except M_9, M_{10} and M_{14}
3. Compute (Q_7, Q_8, Q_9) and (Q_{23}, Q_{24}, Q_{25})
4. For all (Q_{24}, M_{14}) compute forward from step 27 up to step 36, and obtain the list (L_q, Q_{24}, M_{14}) (expected size 2^{64})
5. For all (Q_6, M_9) compute backward from step 9 up to step 0, and obtain the list (L_p^i, Q_6, M_9) (expected size 2^{64})
6. Do feedforward and add the target, continue computing backwards up to step 40, and obtain the list (L_p, Q_6, M_9) (expected size 2^{64})
7. Do partial matching with Q_{36} and Q_{39} as shown in Fig 8 ($2^{64+64-64} = 2^{64}$ pairs left), then match with Q_{38} ($2^{64-32} = 2^{32}$ pairs left).
8. For each pair left, compute the right M_{10} , such that Q_{37} is also matched (we have 2^{32} pairs $(M_{14}, Q_{24}, M_9, Q_6, M_{10})$ fully matched)
9. Check if any pair left satisfies Eqn (3), if yes, output the pseudo preimage (Expect $2^{32-32} = 1$ match).

Clearly, the complexity is 2^{64} with memory requirement 2^{64} . There are some other additional properties. Note given a new target, we can reuse the two lists L_p^i and L_q . So that the computation starts from Step 6 in the algorithm, which results in slightly faster pseudo preimage in $2^{61.4}$. Further more, such an attack gives pseudo preimage with chaining limited to the set L_p^i only.

3-step Partial Matching. As shown in Fig 8, the partial matching works for 3 steps. Q_{36} and Q_{39} can be matched directly or using indirect partial matching. So we have 64 bits for partial matching (without using M_{10}).

3.3 Preimage attack on the MD4 hash function

To find preimage using the pseudo preimage attack above, we need to correct the padding. Note M_{13} is precomputed, hence the length of last block is fixed, we need to use least significant 9 bits of M_{14} so that the padding is consistent with last block. Note adding more blocks will only affects length by a multiple of 512 (2^9). We leave the number of additional blocks for chance as done in the Algorithm. A small modification on the algorithm (fixing the lower 9 bits of M_{14}) will result in pseudo preimage in $2^{64+9} = 2^{73}$, or $2^{61.4+9} = 2^{70.4}$ with 2^{73} precomputation, with padding correctly done. This can be further converted to preimage in 2^{97} using the traditional conversion (link to input chaining of the correctly padded pseudo preimage), as the number of block can be resolved by expandable message.

Precomputation. Similarly we can restrict the input chaining to a subset of size 2^{73} , by re-using the list whenever looking for a new pseudo preimage. So the pseudo preimage can also be converted to preimage in $2^{70.4}$, when large precomputation is allowed. To achieve this, we precompute 2^{128} different message block (prefixed by the expandable message) and store those with output falling in the restricted subset. This requires storage of order 2^{73} and precomputation effort 2^{128} . Given a target, we compute a pseudo preimage (with padding done), and it can be converted to a preimage by looking up the stored chaining values. Hence this requires online computation $2^{70.4}$ only. Using a similar 2^{128} precomputation, the generic Hellman tradeoff would either require almost 2^{20} times more memory ($2^{92.8}$) to achieve the same runtime, or would lead to online computation that is almost 2^{40} times slower (2^{110}) if the same memory would be available.

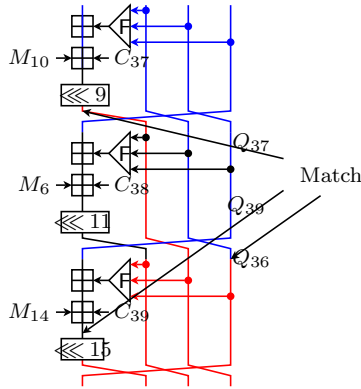


Fig. 8. 3-Step Partial Matching for MD4

3.4 Second preimage attack on the MD4 hash function

Constrast to finding preimages, we can avoid the padding issues when finding second preimages. Let $M_0 || M_1 || \dots || M_k$ be the padded message, we do the following:

1. Compute the chaining value H_1 just after processing M_1 .
2. Compute a pseudo preimage (H', M') of H_1 .
3. Lookup the table for H' , output M_{link} , which links IV to H' .
4. Output $M_{link} || M' || M_2 || \dots || M_k$ as the second preimage.

It is easy to see that the complexity of this second preimage attack is in 2^{64} when $k \geq 2$, i.e. it works for all messages with original length larger than equals to 2 blocks (1024 bits). Although faster second preimage attack exists [42], it only works for very long messages, i.e. 2^{56} blocks.

4 Preimage Attack against Tiger

Before presenting the result, we give some notations used in this Section. Let X^o and X^e denote the odd bytes and even bytes from register X , respectively. More generally, let us denote X^s so that those bits from X indexed by the set s are consistent with X and the rest are set to 0. So we can define $e = \{0, \dots, 7, 16, \dots, 23, 32, \dots, 39, 48, \dots, 55\}$ and $o = \{8, \dots, 15, 24, \dots, 31, 40, \dots, 47, 56, \dots, 63\}$.

4.1 Description of Tiger

Tiger is an iterative hash function based on the MD structure. The message is padded followed by the 64-bit length of original message so that the length of the padded message becomes multiple of 512. Then it is split into blocks of 512 bits and fed into compression function iteratively. The compression of Tiger takes 3 chaining words and 8 message words (each word is of 64 bits) as input and produces the updated 3 chaining words as output. It consists of two parts: message expansion and step function. For the details, please refer to [5]. The input chaining is fed forward, xor-ed together with output of last step function, to produce the output of the compression function, which is a variant of the Davies-Meyer construction.

Step Function. We name the three input chaining words of compression function as A, B and C . These three registers are updated as follows.

$$\begin{aligned} C &\leftarrow C + X \\ A &\leftarrow A - \text{even}(C) \\ B &\leftarrow (B + \text{odd}(C)) \times \text{mul} \end{aligned}$$

The result is then shifted around so that A, B, C become C, A, B , as shown in Fig 9. Here $+$, $-$, \times are addition, subtraction and multiplication, in $\mathbb{Z}_{2^{64}}$, respectively. The two non-linear function even and odd are defined as follows.

$$\begin{aligned} \text{even}(C) &= T_1[c_0] \oplus T_2[c_2] \oplus T_3[c_4] \oplus T_4[c_6] , \\ \text{odd}(C) &= T_4[c_1] \oplus T_3[c_3] \oplus T_2[c_5] \oplus T_1[c_7] , \end{aligned}$$

where T_1, \dots, T_4 are four S-boxes defined on $\{0, 1\}^8 \rightarrow \{0, 1\}^{64}$, the details can be found in [5]. $\text{mul} = 5, 7, 9$ for the three passes, respectively.

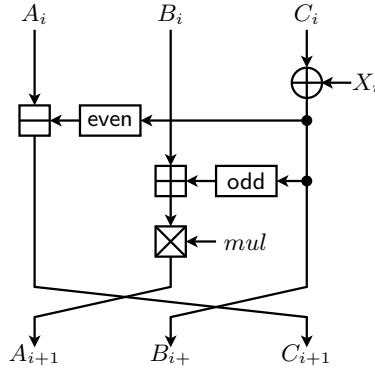


Fig. 9. Step Function of Tiger

Message Expansion. The 512-bit message block is split into 8 message words X_0, \dots, X_7 , each of 64 bits. The key scheduling function takes X_0, \dots, X_7 as input and produces message words $\{X_8, \dots, X_{15}\}$ and $\{X_9, \dots, X_{23}\}$ recursively as follows.

$$\begin{aligned} (X_8, \dots, X_{15}) &= \text{KSF}(X_0, \dots, X_7) \\ (X_9, \dots, X_{23}) &= \text{KSF}(X_8, \dots, X_{15}) \end{aligned}$$

where the key scheduling function KSF is defined as follows. We use $(X_8, \dots, X_{15}) = \text{KSF}(X_0, \dots, X_7)$ as an example here.

First Step:

$$\begin{aligned} Y_0 &= X_0 - (X_7 \oplus C_0) \\ Y_1 &= X_1 \oplus Y_0 \\ Y_2 &= X_2 + Y_1 \\ Y_3 &= X_3 - (Y_2 \oplus (\bar{Y}_1 \lll 19)) \\ Y_4 &= X_4 \oplus Y_3 \\ Y_5 &= X_5 + Y_4 \\ Y_6 &= X_6 - (Y_5 \oplus (\bar{Y}_4 \ggg 23)) \\ Y_7 &= X_7 \oplus Y_6 \end{aligned}$$

Second Step:

$$\begin{aligned} X_8 &= Y_0 + Y_7 \\ X_9 &= Y_1 - (X_8 \oplus (\bar{Y}_7 \lll 19)) \\ X_{10} &= Y_2 \oplus X_9 \\ X_{11} &= Y_3 \oplus X_{10} \\ X_{12} &= Y_4 - (X_{11} \oplus (\bar{X}_{10} \ggg 23)) \\ X_{13} &= Y_5 \oplus X_{12} \\ X_{14} &= Y_6 + X_{13} \\ X_{15} &= Y_7 - (X_{14} \oplus C_1) \end{aligned}$$

C_0 and C_1 are predefined constants with $C_0 = 0xA5A5A5A5A5A5A5A5$ and $C_1 = 0x0123456789ABCDEF$.

Attack Preview. The MITM preimage attack has been applied to Tiger, however for a variant reduced to 16 steps only [18]. The difficulty lies on finding good neutral words, longer initial structure and partial matching. In our attack, we find 4-step initial structure, extend the partial matching to 5 steps and provide choice of neutral words achieving this. However each of them comes with constraints posed on message words/registers. Throughout the description of the attack, we will explicitly give all those constraints, and explain how they can be fulfilled using the multi-word technique, i.e. utilizing the degree of freedoms of most message words and registers to fulfill these constraints, which are usually left as random in the original MITM preimage attacks.

4.2 Precomputed Initial Structure

The original initial structure does not apply to Tiger, since the message words are xor-ed into the chaining, followed by addition/subtraction operations. One can not swap the order of xor and addition/subtraction, unless the chaining values are within certain range so that we can either approximate xor by addition/subtraction, or approximate addition/subtraction by xor. In our precomputed initial structure, we force one of the inputs to be $\mathbf{0}$ so that we can replace xor by addition. Under this assumption, we are able to have 4-step initial structure as shown in Fig 10(a).

The constraints it should satisfy

Constraint 1 Variables in X_i fall on the odd bytes only, so that (X_i^e) is fixed

Constraint 2 Assume we have control over X_{i+4} on those bits so that $(\frac{X_{i+4}}{mul})^o$ is fixed, and there is no carry from even bytes to odd bytes so that we can eventually move the X'_{i+4} further up above the odd operation in step $i + 1$. The idea is to keep the input to the odd function unchanged when we move the $(\frac{X_{i+4}}{mul})^o$ as shown in Fig 10(b).

Constraint 3 C_{i+3} should be '0' for those bits, where variables of X_{i+4} falls. This can be done by precomputing all possible candidates in forward direction and filtering for the good ones as done in [6] for SHA-2.

After the precomputed initial structure (PIS) is formed, we essentially swap the order of X_i^e and $(\frac{X_{i+4}}{mul})^o$, which are 4-step far away from each other originally. We call this 4-step PIS.

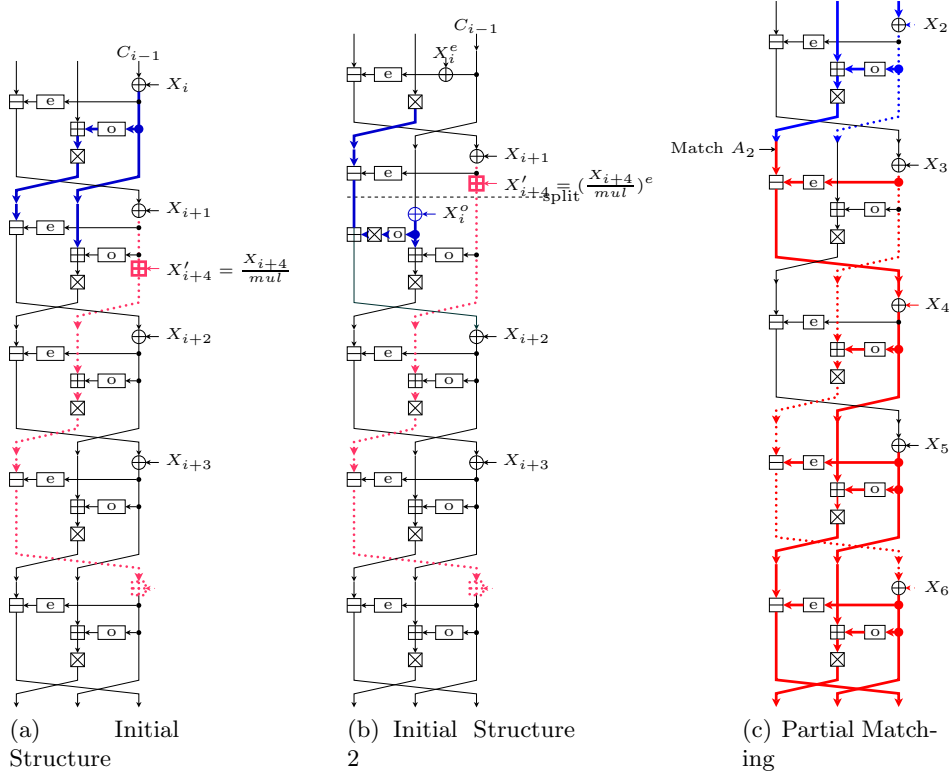


Fig. 10. 4-Step Initial Structure and 5-step Partial Matching for Tiger

4.3 Message Compensation

Message compensation method has been used in many places, however it was first explicitly named in [6]. The length of each chunk is at most 7 without splice (Splice seems to be difficult for full Tiger). Message compensation is used to achieve the maximum length (or close maximum) for each chunk. Since we are able to have 4-step PIS, we would have $7 + 4 + 1 + 7 = 19$ steps for two chunks. Details are shown in Fig 11. Where X_5, \dots, X_{11} is the first chunk (7 steps), X_{12}, \dots, X_{16} could be dealt with using precomputed initial structure as shown above, and X_{17}, \dots, X_{23} are the second chunk (7 steps). In this way, we have 19 steps extended chunks.

For the first chunk, we use a few bits of X_{18} as the neutral word, we will discuss which bits are to be used later. We force X_{18} to be the only one affected in the third pass (i.e. X_{16}, \dots, X_{23}). We come up with such a configuration following the rule that there are as little words affected in the current pass as possible. In summary, we have $\{X_2, \dots, X_6, X_{10}, X_{11}, X_{12}, X_{18}\}$ are affected as shown in Fig 11(a). Note this comes with

Constraint 4 we uses at most the least significant 23 bits of X_{18} so that these bits disappear when $(X_{18} \gg 23)$ is done (as shown in Fig. 11(a)), hence it does not affect X_{20} etc.

For the second chunk, we use few bits of X_{14} as the neutral word and avoid difference in X_7 in the first pass. In the meanwhile, we avoid differences in X_8, \dots, X_{13} and X_{15} for second pass. In the end, we have $\{X_0, \dots, X_3, X_{14}, X_{16}, \dots, X_{23}\}$ affected as shown in Fig 11(b). Note this comes with a constraint

Constraint 5 X_{15} remains constant.

The configuration in Fig 11 is consistent with what we have discussed, i.e. X_7, \dots, X_{13} are only related to X_{18} (red), $X_{19}, \dots, X_{23}, X_0, X_1$ are only related to X_{14} (blue) and the 5 steps in the middle X_{14}, \dots, X_{18} can be dealt with PIS as discussed in Section 4.2. The rest 5 steps X_2, \dots, X_6 will be dealt with in Section 4.4 later.

The two neutral words affect some common message words, i.e. X_2, X_3, X_6 and X_{16} . We will need to choose the bits from two neutral words X_{14} and X_{18} properly so that

Constraint 6 X_{14} and X_{18} will not affect any common bits of any word simultaneously, i.e. for X_2, X_3, X_6 and X_{16} .

We leave the choice to minimize the attack complexity, which will be discussed in Section 4.5.

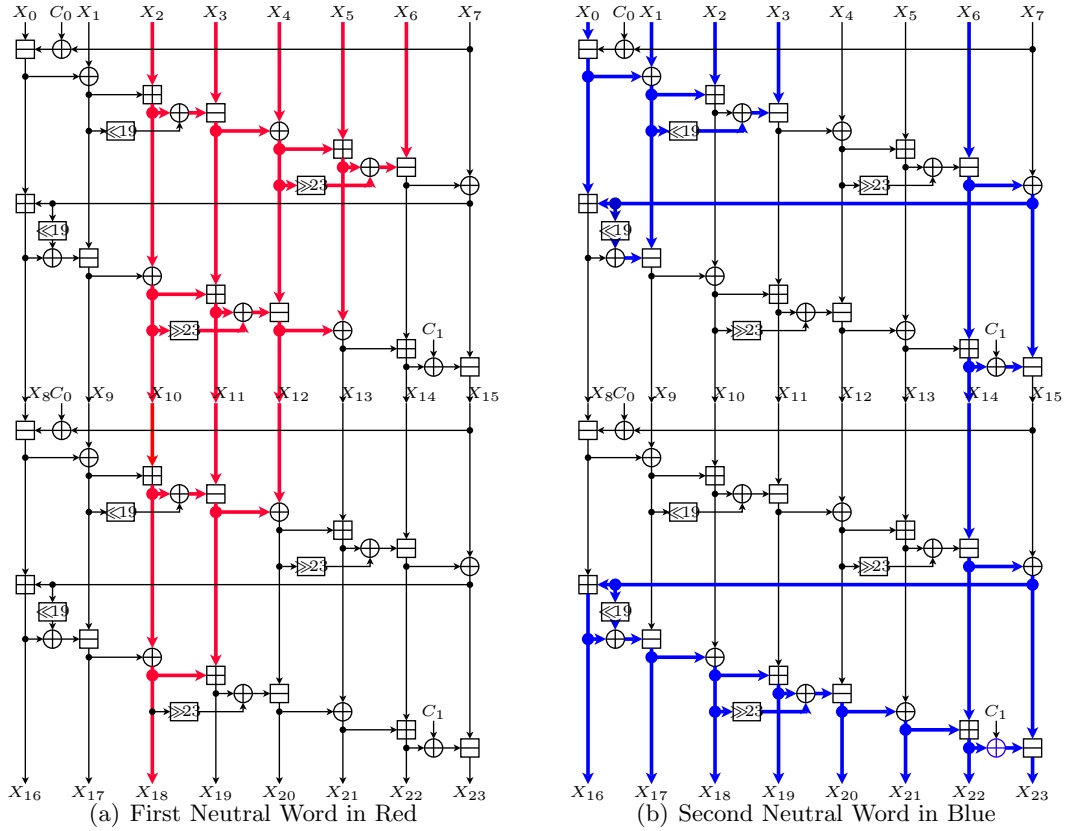


Fig. 11. The neutral words with key scheduling function for Tiger

4.4 Partial Matching and Partial Fixing

The direct partial matching works for 3 steps by computing backwards. Furthermore, by fixing the even bytes of the first message word (partial fixing technique) in forward direction, Isobe and Shibutani [18] are able to achieve 4-step partial matching. In our attack, we further fix some other message words and achieve 5-step partial matching, as shown in Fig 10(c). As described in Section 4.3, we leave $\{X_2, \dots, X_6\}$ for partial matching.

Constraint 7 *The partial information below X_3 as in Fig 10(c) computed from X_6 should cover all even bytes so that we can compute the even function in step 3*

and

Constraint 8 *X_2^o should be related to X_{14} only so that we can compute the odd function at step 2.*

4.5 Attack Description and Complexity Analysis

We define $s_b = \{0, \dots, 7, 16, \dots, 22\}$, $s_f = o$ and we force neutral bits for backward direction to $X_{18}^{s_b}$ and neutral bits for forward direction to $X_{14}^{s_f}$. Additionally we let $X_{13}^{s_f} = 0$ and $X_7^{s_f} = C_1^{s_f}$.

Setting up the message words.

1. set $X_6^{\{0\}} = 1$ so that the padding is done at the last bit of X_6 , hence the length of original message before padding is $b \cdot 2^9 + 447$ (so there are $b + 1$ message blocks).
2. set $X_7^{\{0, \dots, 8\}} = 477$, $X_7^o = C_1^o$ (this also defines 8th bit of X_7 to 1) and the rest bits of X_7 to 0. (This actually set b to $\frac{C_1^o}{2^9}$)
3. set $X_{13} = 0$.
4. set $(Y_4 \gg 23) \oplus Y_5 = 0$ so that $Y_6 = X_6$.
5. randomly choose X_0, \dots, X_3 and the rest 63 bits of X_6 , then compute all other message words.

Finding good candidates - Backward. We use few bits from X_{18} and compute the good candidates for backward direction. Constraint 4 restricts the neutral bits to the least significant 23 bits and Constraint 2 restricts the odd bytes of $\frac{X_{18}^{\{0, \dots, 22\}}}{9}$ ($mul = 9$ for the third pass) to be constant, which reduces the space of good candidates to $\frac{2^{15}}{7} \simeq 2^{12.19}$. Note the search of good candidates can be done byte-wise, i.e. search all 2^8 many $X_{18}^{\{0, \dots, 7\}}$ and 2^7 many $X_{18}^{\{16, \dots, 22\}}$ independently and filter those good ones. This gives complexity about 2^9 . Further more, we have to restrict the influence of the candidate on X_2 and X_6 (refer to 11(a)) to odd bytes (Constraint 7 and 8), which further reduces the number of good candidates to $2^{9.36}$ because of the carry effect when addition is carried out. Note we can reduce the carry effect by restricting one of the inputs of addition operation to 0 for the corresponding bits so that there is no carry between even and odd bytes (for example, we can restrict Y_{17} to 0 so that there is no carry when computing $X_{10} = Y_{18} - Y_{17}$. $Y_{17} = 0$ can be achieved during precomputation phase). Here we assume there are still $2^{12.19}$ good candidates. This computation can be done byte-wise independently too, so it gives complexity 2^9 .

Finding good candidates - Forward. We use few bits from X_{14} and compute the good candidates for backward direction. Constraint 1 restricts the neutral bits to odd bytes. We will restrict the influence of the neutral bits on X_2 and X_{18} to $o \cap \{24, \dots, 63\}$, influence on X_6 to odd bytes. We precompute all possible 2^{32} candidates, filter those good with above properties and also $C_{17}^{\{0, \dots, 7, 16, \dots, 23\}} = 0$ (Constraint 3), we obtain around 2^{13} good candidates left. Note again, the computation can be done byte-wise and this gives complexity 2^{10} .

Correctness. Note all constraints listed before are fulfilled except Constraint 5 and independence about X_3 . For Constraint 5, we set $X_{13} = 0$ and $X_7^o = C_1^o$ and $X_{15}^o = ((X_{14} - X_{13})^o \oplus X_7^o) - (X_{14}^o \oplus C_1^o) = 0$. For the independence of X_3 , neutral bits from X_{14} can only influence bits in $\{38, \dots, 63\}$ due to two $\ll 19$ operations, and the neutral bits from X_{18} have very high probability ($\simeq 1$) to influence the lower 38 bits of X_3 . So we have shown all constraints fulfilled.

Complexity of Finding a Preimage. Following the preimage attack list in Section 2, we have a pseudo-preimage attack with complexity $2^{179.5}$, which can be converted to preimage in time $2^{186.75}$ using the traditional conversion. Note we have 64 bits for partial matching, which is more than enough to fit into the GMTTPP framework. And we have the bit re-distribution control by choosing more/less bits for X_{18} during precomputation. Hence we can find preimage in $2^{184.3}$ (substitute $n = 192$, $l = 12.5$ to (2)), with memory requirement of order $2^{16.7}$ ($2^{4l/3} = 2^{16.7}$).

5 Concluding discussion

We conclude with a discussion of results, and some open problems that are independent of particular hash functions. In this paper we extend the framework around meet-in-the-middle attacks that is currently being developed by the community with a number of general approaches. We illustrated those extensions with improved preimage attacks on various time-tested hash functions, with the first cryptanalytic attack on the full Tiger hash function probably being the most interesting example. Other examples include various improved preimage attacks on MD4 and step-reduced SHA-2.

One of the generic ideas presented was the following. Under the meet-in-the-middle preimage attack framework, we presented new techniques to convert pseudo preimage into preimage faster than the traditional method, i.e the Generic Multi-Target Pseudo Preimage and a simple precomputation technique. It will be interesting to see if an algorithm solving the Enhance 3-Sum problem faster than 2^{2n} for a set size of 2^n exists, so that the MTTPP can be valid for any l . On the other hand, we found pseudo preimage for MD4 in 2^{64} , it will be interesting to see if any of the new conversion techniques/or other unknown technique works when converting pseudo preimage to preimage for MD4.

We expect the techniques outlined in this paper to also improve existing preimage attacks on well studied hash functions like MD5, SHA-1, Haval, and others. Also, several SHA-3 candidates seem to be a natural target.

Acknowledgements

The third author would like to thank Yu Sasaki for interesting discussions during an ECRYPT2 meeting. Part of this work was done while the first author was visiting Graz University of Technology, supported by ECRYPT2. The work of Jian Guo, San Ling and Huaxiong Wang is supported in part by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03 and the Singapore Ministry of Education under Research Grant T206B2204.

References

1. Multisource File Transfer Protocol. http://en.wikipedia.org/wiki/Multisource_File_Transfer_Protocol.

2. Rsync. <http://rsync.samba.org/>.
3. TigerTree Hash Code. <http://tigertree.sourceforge.net/>.
4. 3-Sum Problem. <http://en.wikipedia.org/wiki/3SUM>.
5. R. J. Anderson and E. Biham. TIGER: A Fast New Hash Function. In D. Gollmann, editor, *FSE 1996*, volume 1039 of *Lecture Notes in Computer Science*, pages 89–97. Springer, 1996.
6. K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for Step-Reduced SHA-2. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 578–597. Springer, 2009.
7. K. Aoki and Y. Sasaki. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In S. Halevi, editor, *Advances in Cryptology — CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89, Berlin, Heidelberg, New York, 2009. Springer-Verlag.
8. K. Aoki and Y. Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In R. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography - SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119, Sackville, Canada, 2009. Springer-Verlag.
9. I. Baran, E. D. Demaine, and M. P. trascu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.
10. E. Barkan, E. Biham, and A. Shamir. Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs. In C. Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2006.
11. E. Biham. New Techniques for Cryptanalysis of Hash Functions and Improved Attacks on Snefru. In Nyberg [32], pages 444–461.
12. R. Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
13. C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
14. C. De Cannière and C. Rechberger. Preimages for Reduced SHA-0 and SHA-1. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 179–202. Springer, 2008.
15. O. Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009.
16. N. Haller. RFC1760 - The S/KEY One-Time Password System, 1995.
17. M. E. Hellman. A Cryptanalytic Time - Memory Trade-Off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
18. T. Isobe and K. Shibutani. Preimage Attacks on Reduced Tiger and SHA-2. In Dunkelman [15], pages 139–155.
19. D. Khovratovich, I. Nikolic, and R.-P. Weinmann. Meet-in-the-Middle Attacks on SHA-3 Candidates. In Dunkelman [15], pages 228–245.
20. L. R. Knudsen and J. E. Mathiassen. Preimage and Collision Attacks on MD2. In H. Gilbert and H. Handschuh, editors, *FSE*, volume 3557 of *LNCS*, pages 255–267. Springer, 2005.
21. X. Lai and J. L. Massey. Hash Function Based on Block Ciphers. In *EUROCRYPT*, pages 55–70, 1992.
22. G. Leurent. Message Freedom in MD4 and MD5 Collisions: Application to APOP. In A. Biryukov, editor, *FSE*, volume 4593 of *LNCS*, pages 309–328. Springer, 2007.
23. G. Leurent. MD4 is Not One-Way. In Nyberg [32], pages 412–428.
24. F. Mendel, N. Pramstaller, and C. Rechberger. A (Second) Preimage Attack on the GOST Hash Function. In Nyberg [32], pages 224–234.
25. F. Mendel, N. Pramstaller, C. Rechberger, M. Kontak, and J. Szmids. Cryptanalysis of the GOST Hash Function. In D. Wagner, editor, *CRYPTO*, volume 5157 of *LNCS*, pages 162–178. Springer, 2008.
26. F. Mendel, B. Preneel, V. Rijmen, H. Yoshida, and D. Watanabe. Update on Tiger. In R. Barua and T. Lange, editors, *INDOCRYPT*, volume 4329 of *LNCS*, pages 63–79. Springer, 2006.
27. F. Mendel and V. Rijmen. Cryptanalysis of the Tiger Hash Function. In K. Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 536–550. Springer, 2007.
28. F. Mendel and V. Rijmen. Weaknesses in the HAS-V Compression Function. In K.-H. Nam and G. Rhee, editors, *ICISC*, volume 4817 of *LNCS*, pages 335–345. Springer, 2007.
29. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
30. F. Muller. The MD2 Hash Function Is Not One-Way. In P. J. Lee, editor, *ASIACRYPT*, volume 3329 of *LNCS*, pages 214–229. Springer, 2004.

31. Y. Naito, Y. Sasaki, N. Kunihiro, and K. Ohta. Improved Collision Attack on MD4 with Probability Almost 1. In D. Won and S. Kim, editors, *ICISC*, volume 3935 of *LNCS*, pages 129–145. Springer, 2005.
32. K. Nyberg, editor. *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*. Springer, 2008.
33. Y. Sasaki and K. Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In J. P. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271, Berlin, Heidelberg, New York, 2008. Springer-Verlag.
34. Y. Sasaki and K. Aoki. Finding preimages in full MD5 faster than exhaustive search. In A. Joux, editor, *Advances in Cryptology — EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152, Berlin, Heidelberg, New York, 2009. Springer-Verlag.
35. Y. Sasaki, L. Wang, K. Ohta, and N. Kunihiro. Security of md5 challenge and response: Extension of apop password recovery attack. In T. Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008.
36. M. Stevens, A. K. Lenstra, and B. de Weger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *LNCS*, pages 1–22. Springer, 2007.
37. M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In S. Halevi, editor, *29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, Proceedings*, volume 5677 of *LNCS*, pages 55–69. Springer, 2009.
38. S. S. Thomsen. An improved preimage attack on MD2. Cryptology ePrint Archive, Report 2008/089, 2008. <http://eprint.iacr.org/>.
39. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [12], pages 1–18.
40. X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
41. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In Cramer [12], pages 19–35.
42. H. Yu, G. Wang, G. Zhang, and X. Wang. The Second-Preimage Attack on MD4. In Y. Desmedt, H. Wang, Y. Mu, and Y. Li, editors, *CANS*, volume 3810 of *LNCS*, pages 1–12. Springer, 2005.

A Improved Preimage Attack against SHA-2

In [6], Aoki *et al.* give preimage on 42-step reduced SHA-2. We note that matching point (together with the choice of neutral words) can be moved to the end of the compression function, as done for attacking SHA-224/384 in [6]. The number of neutral bits in two direction is around $32/3$ ($64/3$ for SHA-512) and the number of bits for partial matching is 32 (64 for SHA-512), which is more than enough. Applying the MTPP framework, we find preimages in $2^{248.4}$ (substitute $n = 256$ and $l = 32/3$ to (1)), compared with $2^{251.7}$ for 42-step SHA-256 and $2^{494.6}$ (substitute $n = 512$ and $l = 64/3$ to (1)), compared with $2^{502.3}$ for 42-step SHA-512. The memory requirements remain unchanged.

Note partial matching works in such a way that, more bits are fixed, less bits for neutral words and more steps/more bits can be used for partial matching. So there is a balance between bits for neutral words and bits for partial matching. When multi-targets are available, we are to use less bits for neutral bits, and more for partial matching, in order to reduce the complexity for finding pseudo preimages. This trick can be applied to the attack on 43-step SHA-256 and 46-step SHA-512 in [6], hence the complexity can be reduced. As mentioned in our conclusions, we expect this method to be directly applicable to more existing results.