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

Jian Guo[1], San Ling[1], Christian Rechberger[2], and Huaxiong Wang[1]

[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.

ntu.guo@gmail.com

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. Our attacks runs in time $2^{188.8}$ for finding preimages, and $2^{188.2}$ for second-preimages. Both have memory requirement of order 2^8, which is much less than in any other recent preimage attacks on reduced Tiger. Using pre-computation techniques, the time complexity for finding a new preimage or second-preimage for MD4 can now be as low as $2^{78.4}$ and $2^{69.4}$ MD4 computations, respectively. The second-preimage attack works for all messages longer than 2 blocks.
To obtain these 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) probabilistic initial structures, compared with deterministic ones, to enable more neutral words, and hence to reduce the attack time complexity. All the 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 [12,38,43,44], implementors have reconsidered their choices. While starting a very productive phase of research on the 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.,* [24,37,39]). 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 them. 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 significantly 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 in a way to fit into the meet-in-the-middle (MITM) framework of Aoki and Sasaki as recently developed in a series of papers [6,7,8,35,36], although we note that the basic approach was pioneered by Lai and Massey [23]. Other interesting approaches to preimage attacks appeared in [11,13,20,21,22,27,28,33,45].

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

- **Probabilistic initial structure**, compared with (deterministic) initial structure, is found be useful for significantly reducing attack complexity for the first. To improve the time complexity of a MITM preimage attack, the attackers usually need to find more neutral words. This usually reduces the number of attackable steps, due to the fact that the more neutral words, the faster the neutrality is destroyed, and the less step can be covered for independent chunks, initial structure, and partial matching. Hence, there is a tradeoff between the size of neutral words, and attackable steps. In this paper, using MD4 in Section 3 as an example, we show one can use more neutral words, and maintain long initial structure at the same time, with cost of turning the initial structure into a probabilistic one. A similar technique has been used in [36], however it serves a purpose of better approximating the initial structure, and the attack complexity does not reduce due to limited bits for partial matching.
- **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 leads to big efficiency losses when the attack on the compression function is converted to an attack on the hash function. Hence, an attacker basically has 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 methods. 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. A table of best related works, and the comparison with our main results are shown in Table 1.

- **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 systems, or in many file sharing protocols and applications, often in a Merkle-tree construction (also

Hash	Attack	Time Complexity	Memory	Source	Remarks
MD4	pseudo-preimage	2^{96}	2^{32}	[25]	consistent padding
		2^{72}	2^{64}	Section 3	without padding
		2^{81}	2^{55}	Section 3	consistent padding
	preimage	2^{102}	2^{33}	[25]	
		$2^{99.7}$	2^{64}	Section 3	msg $\geq 2^{50}$ blocks
		$2^{78.4}$	2^{81}	Section 3	msg $\geq 2^{50}$ blocks, 2^{128} precomp.
	second-preimage	2^{56}	2^{56}	[45]	msg about 2^{56} blocks
		2^{64}	2^{64}	[19]	msg about 2^{64} blocks
		$2^{99.7}$	2^{64}	Section 3	msg ≥ 2 blocks
		$2^{69.4}$	2^{72}	Section 3	msg ≥ 2 blocks, 2^{128} precomp.
Tiger	preimage	2^{185}	2^{160}	[26]	17 steps
		2^{161}	2^{32}	[18]	16 steps, one block
		$2^{189.5}$	2^{22}	[41]	23 steps
		$2^{188.8}$	2^{8}	Section 4	full 24 steps
SHA-256	preimage	2^{240}	2^{16}	[18]	24 out of 64 steps, one block
		$2^{251.7}$	2^{12}	[6]	42 steps
		$2^{248.4}$	2^{12}	Appendix A	42 steps
		$2^{254.9}$	2^{6}	[6]	43 steps
SHA-512	preimage	2^{480}	2^{32}	[18]	24 out of 80 steps, one block
		$2^{502.3}$	2^{22}	[6]	42 steps
		$2^{494.6}$	2^{22}	Appendix A	42 steps
		$2^{511.5}$	2^{6}	[6]	46 steps

Table 1. Summary of results in this paper, and the best related works; *msg* refers to original message before padding.

known as TigerTree [3]). The best collision attack on Tiger is on 19 rounds [29].[1] So far the best preimage attack on the Tiger hash function is by Wang and Sasaki [41]. Independently of our work, they applied the MITM preimage attack to Tiger reduced to 23 steps with time complexity higher than ours (1.4×2^{189}) and requirements of 2^{22} units. Our new attack improves those in many 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 $2^{188.8}$ (compression function attack is $2^{185.4}$) and memory requirements are only in the order of 2^8. These results are obtained using 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 [42,34], 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. The time complexity for the best known compression function attack is reduced from 2^{96} (by Leurent [25]) to 2^{72}. Assuming 2^{128} precomputation using the large computation technique mentioned above, and 2^{81} storage, 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 $2^{78.4}$.
– **SHA-2**: The members of the SHA-2 family of hash functions are probably among 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, four different methods converting the pseudo-preimage to preimage (including two new ones), and also recapitulates

[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 [30]. However, this attack cannot be extended on the hash function, whereas all the attacks in this paper can.

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

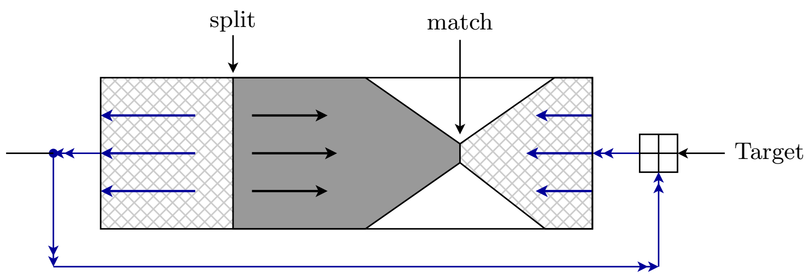


Fig. 1. Meet-in-the-Middle Pseudo-Preimage Attack against Davies-Meyer Hash Functions

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 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 [32, 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 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 $2^{n-l/2}$. So the overall complexity for finding one preimage is $2^{n-l/2+1}$, 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 the point before partial matching, and re-check the partial matching portion only). To minimize the time complexity, we require $m \geq l$, so that the partial matching costs $2^{2l-m} \leq 2^l$, which can be neglected.

2.1 Multi-Target Pseudo Preimage (MTPP)

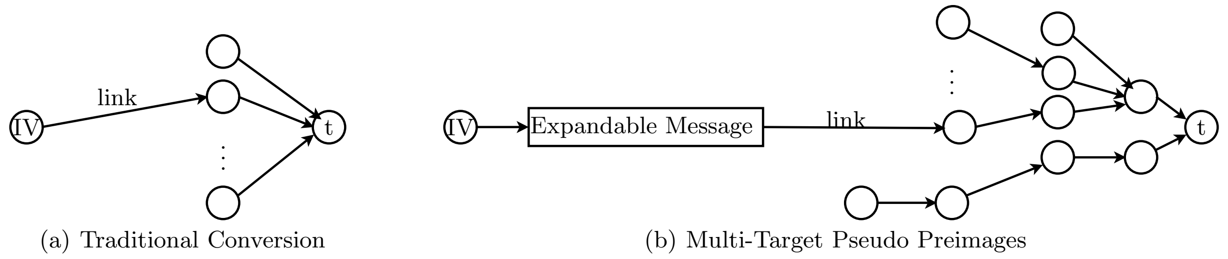


Fig. 2. Converting Pseudo-Preimages to Preimage: circle denotes state, arrow denotes message block

In [25], Leurent provides an unbalanced-tree multi-target pseudo-preimage method (refer Fig 2(b)) to convert the pseudo-preimages to preimage with complexity $(l\ln(2)+1)\cdot 2^{n-l}$, compared with $2^{n-l/2+1}$ in [32, Fact 9.99]. Suppose the matching point is at the end of compression function. The matching process is to find $l_p + l_q = t$ ($l_p \in L_p$, $l_q \in L_q$, and $t \in T$, the set of known targets). 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 to one of the k targets. To find 2^k pseudo-preimages, it takes $2^{n-l}/1 + 2^{n-l}/2 + 2^{n-l}/3 + \cdots + 2^{n-l}/2^k \simeq k\ln(2)\cdot 2^{n-l}$. To find a preimage, it is expected to repeat 2^{n-k} blocks finding a message, which links to one of the 2^k targets. Taking the optimal $k = l$, the overall complexity is

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

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-preimages 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 (denoted as L_p and L_q), and given 2^k targets (denoted as T), we are to find *all* possible (l_p, l_q, t), where $l_p \in L_p$, $l_q \in L_q$ and $t \in T$, such that $l_p + l_q = t$. We call this problem the Enhanced 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 the matching only occurs in 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 the number of "+" in the compression, when l is relatively small (*e.g.,* ≤ 7 for MD4 and Tiger, since there are about 2^7 "+" in MD4 compression; we simply count the number of operations ("+", "−", "×" and sbox lookup) in the case of 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 the 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 freely[2]. Given 2^k targets, we can distribute the $2l$ bits to $l + k/2$ and $l - k/2$, so that we can have

[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.

$2^{l+k/2}$ candidates for 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 $\Sigma_{i=1}^{2^k} 2^{n-l} \cdot i^{-1/2} \simeq 2^{n-l+1+k/2}$ (see Appendix B for the proof). So we can find the preimage in

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

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 $k = 0, \ldots, 2l/3$. 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 any 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 as 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 2. 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 [31], alternative tree and graph construction in [13]) 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,[32]	$2^{n-l/2+1}$	2^l	l	-
GMTPP	new, Section 2.2	$3 \cdot 2^{n-2l/3}$	$2^{4l/3}$	$4l/3$	redistribute neutral bits
MTPP	Section 2.1, [25]	$(l \ln(2) + 1) \cdot 2^{n-l}$	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 chaining of size 2^z

Table 2. Comparison of methods converting pseudo-preimage to preimage

2.4 Introduction to some MITM Techniques for Compression Function Attacks

There are several techniques developed recently to extend the preimage attack for more steps or to reduce the time complexity. To help understand the techniques developed later in the paper, we will introduce the concepts of 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 the two chunks can be extended. As shown in Fig 3, originally both chunks p and q contain 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.

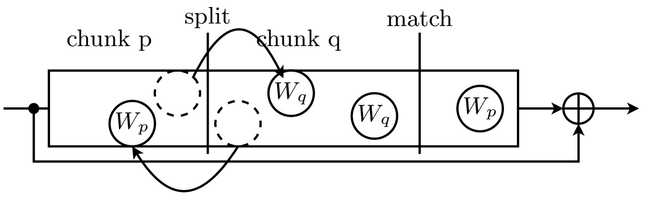


Fig. 3. Initial Structure

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 a 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 behave 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, for some functions f, σ, g, μ depending on the underlining hash function. 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.

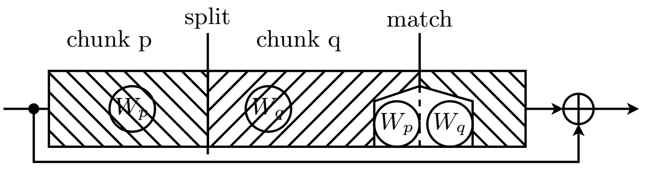


Fig. 4. Partial Matching

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. The way how initial structure and partial matching can be achieved will be demonstrated in following attack descriptions.

3 Improved Preimage Attack against MD4

3.1 Description of MD4

MD4 follows the traditional MD-strengthening, the original message is padded by 1, followed by many 0's and the 64-bit length information so that the length of padded message becomes a 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 hash. The compression function follows the Davies-Meyer construction, and comes with two major parts: message scheduling and step function. Message scheduling divides the 512-bit message block into 16 words (32 bit each) and expands them into 48 using permutations, as shown in following table.

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 the compression function. The step function takes four registers as input, and update one as $Q_i = (Q_{i-4} + \mathsf{F}_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) + M_{\pi(i)} + C_i) \lll r_i$ for $i = 0, \ldots, 47$, where C_i and r_i are predefined constants, π is a permutation defined in above table, and the functions F_i are defined as in the following table. We use typewriter font to denote the hex numbers, such as `5A827999`, $\mathbb{1}$ for `FFFFFFFF`, and $\mathbb{0}$ for `00000000`.

First pass	$0 \le i < 16$	$\mathsf{F}_i = \mathsf{IF}$	$C_i = K_0 = \mathbb{0}$
Second pass	$16 \le i < 32$	$\mathsf{F}_i = \mathsf{MAJ}$	$C_i = K_1 =$ `5A827999`
Third pass	$32 \le i < 48$	$\mathsf{F}_i = \mathsf{XOR}$	$C_i = K_2 =$ `6ED9EBA1`

3.2 Faster Pseudo Preimage Attack

In this section, we present a pseudo-preimage attack in 2^{72}. Separation of chunks is shown in Fig 5. 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 6. Note that every register and message words within the initial structure except $Q_6, M_{10}, M_{14}, M_9, Q_{26}$ are fixed to some random values. The concept of 4-cycle local-collision path has been used in [40,14,25]. However, none of those paths help in our MITM preimage attack, since we cannot find more proper choices of neutral words. In our initial structure, the relation between Q_6 and Q_{26} satisfies

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

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

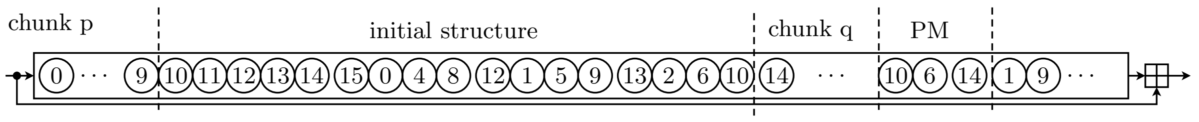


Fig. 5. Separation of chunks for Pseudo-Preimage against MD4

We fix all other registers in Fig 6 in such a way that the influence of the registers in the bold line is absorbed when passing through the F function (this is called *cross absorption property*). Note F is IF for the first pass and MAJ for the second pass. To deal with IF,

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

Similarly, we force the other two inputs equal for MAJ. All required values are shown in Fig 6. However, this setting results in no solution, since it is over-constrained on M_{12} and M_{13}. To overcome this problem, we propose a *probabilistic initial structure.*

Probabilistic Initial Structure. Consider the probability for $a = \mathsf{IF}(b, a, x)$, where a, b are fixed constants, and x is a random value in $\mathbb{F}_{2^{32}}$. The equation does not always hold for all x. However, if $|b|$ (Hamming weight) is very close to 32, then we can expect high probability for the equation to hold. Instead of setting inputs of IF to be strictly $\mathbb{1}$ or $\mathbb{0}$, we use some other values which are close to $\mathbb{1}$ or $\mathbb{0}$ (similarly, we force two inputs of MAJ to be very close), which enables us to find some solutions for the initial structure, as shown in Fig 7, where a, b are variables to be decided later. We list the equations of the constraints here:

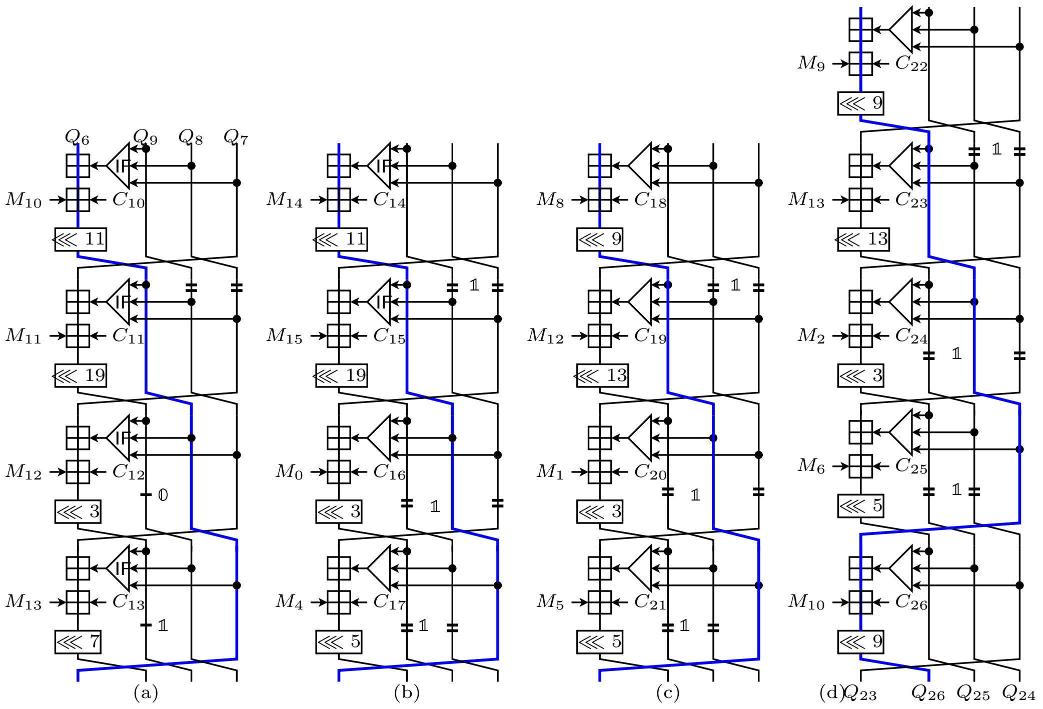


Fig. 6. 17-Step Initial Structure for MD4

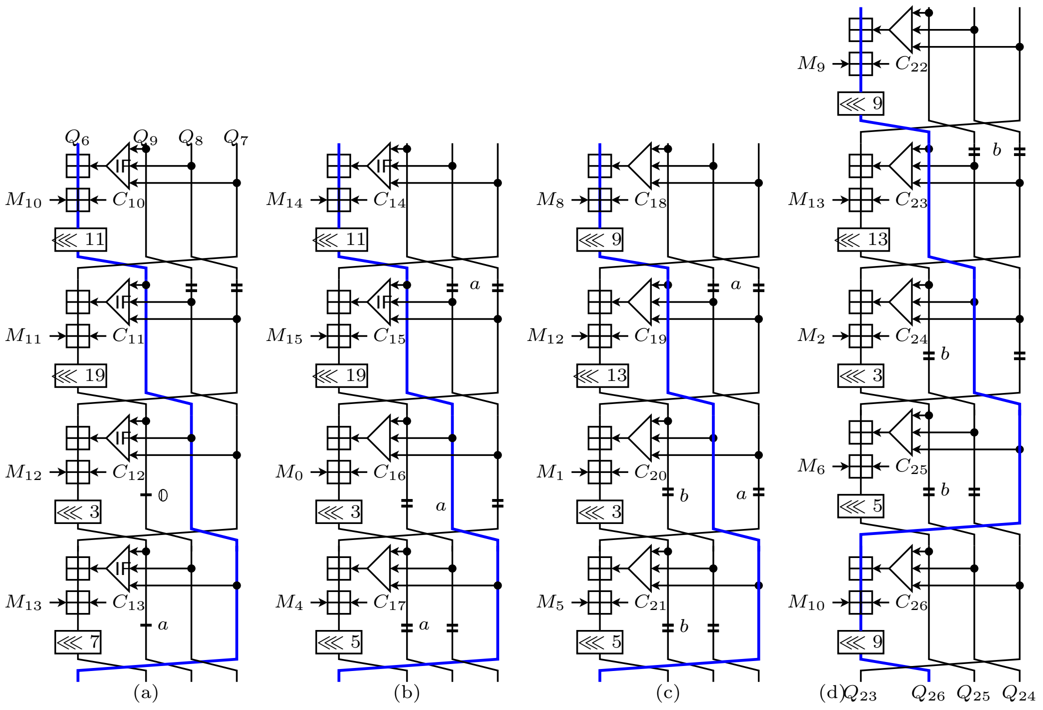


Fig. 7. 17-Step Probabilistic Initial Structure for MD4

$$\begin{array}{llcr}
\text{Step 11:} & Q_9 = Q_8 & & \\
\text{Step 12:} & Q_{11} = \mathbb{0} & \Leftrightarrow & Q_7 + Q_8 + M_{11} = \mathbb{0} \\
\text{Step 13:} & Q_{12} = a & \Leftrightarrow & (Q_8 + Q_9 + M_{12}) \lll 3 = a \\
\text{Step 15:} & Q_{13} = Q_{12} = a & \Leftrightarrow & (Q_9 + M_{13}) \lll 7 = a \\
\text{Step 16:} & Q_{15} = Q_{13} = a & \Leftrightarrow & (a + M_{15}) \lll 19 = a \\
\text{Step 17:} & Q_{16} = Q_{15} = a & \Leftrightarrow & (a + a + M_0 + K_1) \lll 3 = a \\
\text{Step 18:} & Q_{17} = Q_{16} = a & \Leftrightarrow & (a + a + M_4 + K_1) \lll 5 = a \\
\text{Step 19:} & Q_{19} = b & \Leftrightarrow & (a + a + M_{12} + K_1) \lll 13 = b \\
\text{Step 20:} & Q_{20} = Q_{19} = b & \Leftrightarrow & (a + a + M_1 + K_1) \lll 3 = b \\
\text{Step 22:} & Q_{21} = Q_{20} = b & \Leftrightarrow & (a + b + M_5 + K_1) \lll 5 = b \\
\text{Step 24:} & Q_{23} = Q_{21} = b & \Leftrightarrow & (b + b + M_{13} + K_1) \lll 13 = b \\
\text{Step 25:} & Q_{24} = Q_{23} = b & \Leftrightarrow & (b + b + M_2 + K_1) \lll 3 = b
\end{array} \tag{4}$$

The above system of equations allows us to have choices for a and b. Note that we used two probabilistic approximations in two places, *i.e.,* $\mathsf{IF}(a, \mathbb{0}, Q_{10}) = \mathbb{0}$ at Step 13, and $\mathsf{MAJ}(b, Q_{18}, a) = a$ at Step 20. Each happens with probability $2^{|a|-32}$ and $2^{-|a \oplus b|}$, respectively (assume Q_{10} and Q_{18} are uniformly distributed). To have high probability, we search the a, b which maximize $prob = 2^{|a|-|a\oplus b|-32}$. We found $a =$ `EFFFBFEF`, and $b =$ `EFCF1F6F`, which give $prob = 2^{-8}$. Solving (4) leaves $M_0 =$ `C37DFE86`, $M_1 =$ `C377EA76`, $M_2 =$ `C3D92B76`, $M_4 =$ `44FE0488`, $M_5 =$ `452D2004`, $M_{12} =$ `C0FD8501`, $M_{13} =$ `C15EC601`, $M_{15} =$ `07FE3E10`, $Q_8 = Q_9 =$ `1E81397E`, and $Q_7 + M_{11} =$ `E17EC682`. To ensure this works as expected, we verified the probability using a C program [15] , and the experiment confirms the result.

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}).

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_{26}, M_{14}) compute forward from step 27 up to step 36, and obtain the list (L_q, Q_{26}, 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_{26}, M_9, Q_6, M_{10}) fully matched).
9. Check if any pair left satisfies Eqn (3), if yes, output the pseudo-preimage; otherwise repeat the above process until a pseudo-preimage is found ($2^{32+8-32} = 2^8$ repetitions expected).

Clearly, the complexity is 2^{72} with memory requirement 2^{64}. There are some other additional properties. Note that 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 $2^{69.4}$. Furthermore, such an attack gives pseudo-preimage with chaining limited to the set L_p^i only.

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 that M_{13} is precomputed, hence the length of last block is fixed, we need to fix the least significant 9 bits of M_{14} accordingly, *i.e.,* 447 (`1BF` in hex). Note that adding more blocks will only affect the length by a multiple of 512 (2^9). We leave the number of additional blocks for chance as done in the algorithm in Section 3.2. A small modification on the algorithm (computing 2^{55} candidates for each direction during each repetition, and $2^{128-55\times 2+8} = 2^{26}$ repetitions are needed, hence the size of L_p^i increases to $2^{55+26} = 2^{81}$) will result in pseudo-preimage in $2^{69.4+9} = 2^{78.4}$ with memory requirement 2^{55}. This can be further converted to preimage in $2^{99.7}$ using the traditional conversion (link to input chaining of the last padded block), as the number of blocks can be resolved by expandable message (we compute a pseudo-preimage following the padding rule in $2^{78.4}$, then apply the traditional conversion. Now, padding is no longer a problem when inverting the second last block etc.).

The resulted message by this attack is of at least 2^{50} blocks, due to the fact that M_{15} is the most significant word of the length ($M_{15}||M_{14}$ denotes the length) and we have preset M_{15} to `07FE3E10`.

Precomputation. Similarly we can restrict the input chaining to a subset of size 2^{81}, by re-using the lists whenever looking for a new pseudo-preimage. So the pseudo-preimage can also be converted to preimage in $2^{78.4}$, when large precomputation is allowed. To achieve this, we precompute about 2^{128} different message blocks (prefixed by the expandable message) and store those with output in the restricted subset. This requires storage of order 2^{81} 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 $2^{78.4}$ only. Using a similar 2^{128} precomputation, the generic Hellman tradeoff would either require almost $2^{7.8}$ times more memory ($2^{88.8}$) to achieve the same runtime, or would lead to online computation that is almost $2^{15.6}$ times slower (2^{94}) if the same memory would be available.

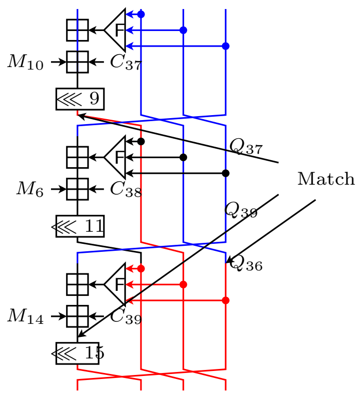


Fig. 8. 3-Step Partial Matching for MD4

3.4 Second-preimage attack on the MD4 hash function

In contrast to finding preimages, we can avoid the padding issues when finding second-preimages by finding pseudo-preimages for second last block etc., as done in [25]. Given 2^{128} precomputation, the complexity of this second-preimage attack is in $2^{69.4}$ with 2^{72} memory when $k \geq 2$, *i.e.,* it works for all messages with original length before padding at least 2 blocks (1024 bits, at least 3 blocks after padding). Similarly, it works in time $2^{99.7}$ and 2^{64} memory without precomputation. Although a faster second-preimage attack exists [45], the attack only works for very long messages, *i.e.,* at least 2^{56} blocks. For comparison, a second preimage can be found in 2^{n-k}, if the given message is of more than 2^k blocks, due to Kelsey and Schneier [19] (2^{64} for both time and memory if the optimal $k = 64$ can be achieved).

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 indexed by the set s are the same as in X and the rest are set to 0. To be consistent, we can define $e = \{0, \ldots, 7, 16, \ldots, 23, 32, \ldots, 39, 48, \ldots, 55\}$ and $o = \{8, \ldots, 15, 24, \ldots, 31, 40, \ldots, 47, 56, \ldots, 63\}$.

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 the original message so that the length of the padded message becomes a multiple of 512. Then it is split into blocks of 512 bits and fed into the 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. The input chaining is fed forward, together with output of last step function, to produce the output of the compression function, which is a variant of the Davies-Meyer construction. We introduce the step function and message expansion in details as follows.

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

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

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.

$$\begin{aligned} \mathsf{even}(x) &= T_1[x_B^0] \oplus T_2[x_B^2] \oplus T_3[x_B^4] \oplus T_4[x_B^6] \ , \\ \mathsf{odd}(x) &= T_4[x_B^1] \oplus T_3[x_B^3] \oplus T_2[x_B^5] \oplus T_1[x_B^7] \ , \end{aligned}$$

where $T_1, \ldots, T_4$ are four S-boxes defined on $\{0,1\}^8 \rightarrow \{0,1\}^{64}$, and x_B^i denotes the i-th least significant *Byte* of x, the details can be found in [5]. *mul* is 5, 7, 9 for the three passes, respectively.

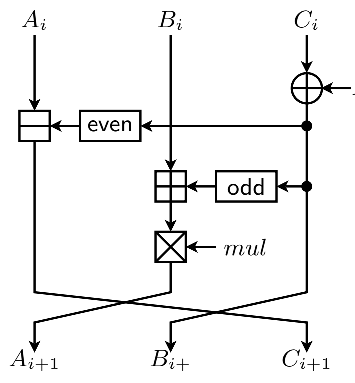


Fig. 9. Step Function of Tiger

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

$\{X_{16}, \ldots, X_{23}\}$ recursively as follows.

$$(X_8, \ldots, X_{15}) = \mathsf{KSF}(X_0, \ldots, X_7)$$
$$(X_{16}, \ldots, X_{23}) = \mathsf{KSF}(X_8, \ldots, X_{15}) \ ,$$

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

First Step:

$$\begin{aligned}
Y_0 &= X_0 - (X_7 \oplus K_3)\\
Y_1 &= X_1 \oplus Y_0\\
Y_2 &= X_2 + Y_1\\
Y_3 &= X_3 - (Y_2 \oplus (\overline{Y}_1 \ll 19)\\
Y_4 &= X_4 \oplus Y_3\\
Y_5 &= X_5 + Y_4\\
Y_6 &= X_6 - (Y_5 \oplus (\overline{Y}_4 \gg 23))\\
Y_7 &= X_7 \oplus Y_6
\end{aligned}$$

Second Step:

$$\begin{aligned}
X_8 &= Y_0 + Y_7\\
X_9 &= Y_1 - (X_8 \oplus (\overline{Y}_7 \ll 19))\\
X_{10} &= Y_2 \oplus X_9\\
X_{11} &= Y_3 + X_{10}\\
X_{12} &= Y_4 - (X_{11} \oplus (\overline{X}_{10} \gg 23))\\
X_{13} &= Y_5 \oplus X_{12}\\
X_{14} &= Y_6 + X_{13}\\
X_{15} &= Y_7 - (X_{14} \oplus K_4)
\end{aligned}$$

with $K_3 = \texttt{A5A5A5A5A5A5A5A5}$, $K_4 = \texttt{0123456789ABCDEF}$, and $\overline{Y}$ denotes bitwise complement of Y.

Attack Preview. The MITM preimage attack has been applied to Tiger, however for variants reduced to 16 and 23 steps [18,41], out of 24 in full Tiger. The difficulty lies on finding good neutral words, longer initial structure and partial matching. In our attack, we find a 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, due to the very complicated message scheduling used in Tiger. 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 degrees of freedom 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

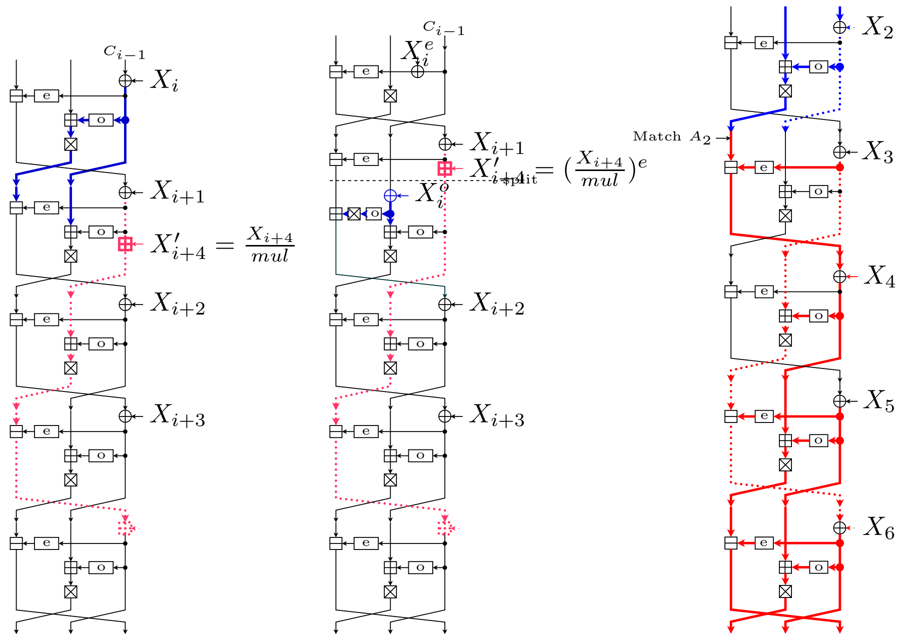
The original initial structure does not apply to Tiger, since the message words are xor-ed into the chaining, followed by addition/subtraction operations. One cannot swap the order of xor and addition/subtraction, unless the chaining values are within certain range so that we can either approximate xor by addition, or approximate addition by xor. We can either restrict one of the inputs to $\mathbb{0}$, or force the output to be $\mathbb{1}$, *e.g.,* $X \oplus \mathbb{0} = X + \mathbb{0}$, and $X \oplus Y = \mathbb{1}$ if and only if $X + Y = \mathbb{1}$. Under this restriction, we are able to have a 4-step initial structure as shown in Fig 10(a), which comes with the following three constraints.

Constraint 1 *Variables from X_i lie 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 *function in step $i+1$. The idea is to keep the input to the* odd *function unchanged when we move the $(\frac{X_{i+4}}{mul})^e$ as shown in Fig 10(b).*

Constraint 3 *$C_{i+3} \oplus X_{i+4}$ should be 1 for those bits, where variables from X_{i+4} lie.*

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 steps away from each other originally.



(a) Initial Structure (b) Initial Structure 2 (c) Partial Matching

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

4.3 Message Compensation

The length of each independent chunk is at most 7 steps, due to the fact that any consecutive 8 message words can generate all other words (*i.e.*, related to all other words). Message compensation is used to achieve the maximum length (or close to 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 $X_7, \ldots, X_{13}$ form the first chunk (7 steps), $X_{14}, \ldots, X_{18}$ may be dealt with using precomputed initial structure as shown above, and $X_{19}, \ldots, X_{23}, X_0, X_1$ are the second chunk (7 steps). In this way, we have 19-step 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.*, $X_{16}, \ldots, X_{23}$). We come up with such a configuration following the rule that there are as few words affected in the current pass as possible. In summary, we have $\{X_2, \ldots, X_6, X_{10}, X_{11}, X_{12}, X_{18}\}$ affected as shown in Fig 11(a). Note this comes with

Constraint 4 *We use 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.*

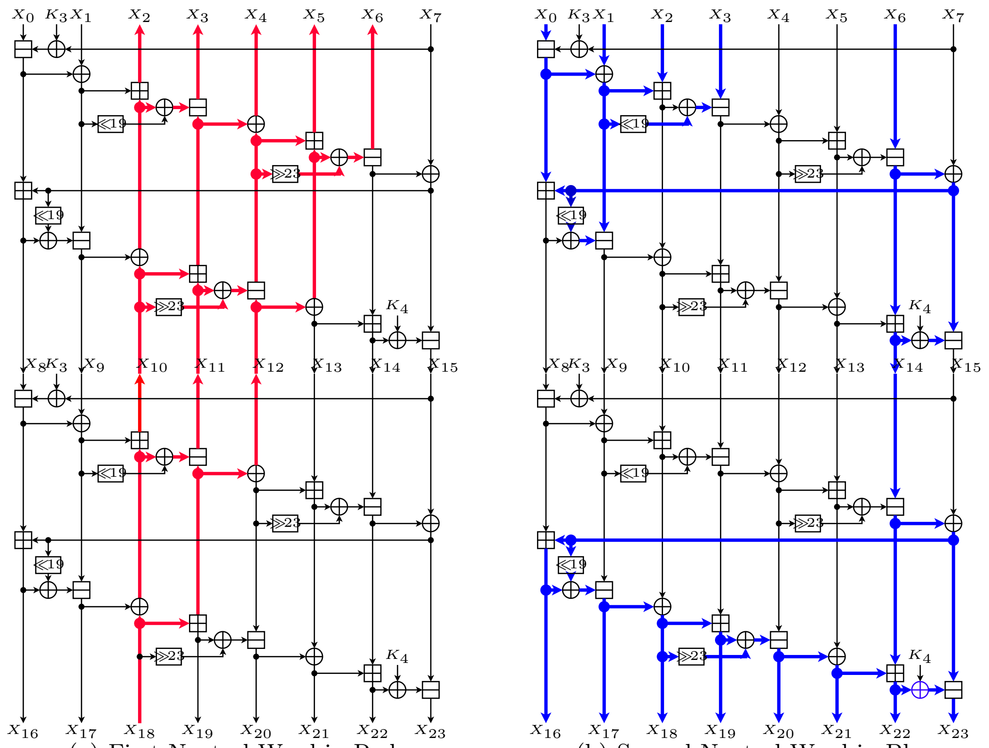
For the second chunk, we use a few bits of X_{14} as the neutral word and avoid difference in X_7 in the first pass. Meanwhile, we avoid differences in $X_8, \ldots, X_{13}$ and X_{15} for the second pass. In the end, we have $\{X_0, \ldots, X_3, X_{14}, X_{16}, \ldots, X_{23}\}$ affected as shown in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

Constraint 5 X_{15} *remains constant.*

The two neutral words affect some common message words, *i.e.*, X_2, X_3, X_6 and X_{18}. 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_{18}.

We are left with the choices of the neutral bits for minimizing the attack complexity, which will be discussed later in Section 4.5.



(a) First Neutral Word in Red (b) Second Neutral Word in Blue

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, and 5-step by Wang and Sasaki [41]. In addition to the 4-step initial structure, we further post more conditions on message words in order to achieve 5-step partial matching (different from [41]), as shown in Fig 10(c), it covers step 2 to step 6.

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;*

Constraint 8 *X_2^o should be related to X_{14} only, so that we can compute the* odd *function at step 2 independently of X_{18}.*

To summarize, we are to use $\{X_7, \ldots, X_{13}\}$ as one chunk, $\{X_{19}, \ldots, X_{23}, X_1, X_2\}$ as the other chunk; precomputed initial structure covers steps using $\{X_{14}, \ldots, X_{18}\}$ ($i = 14$ for Section 4.2); and partial matching works for $\{X_2, \ldots, X_6\}$. Hence, the full Tiger of all 24 steps is covered.

4.5 Attack Description and Complexity Analysis

In this section, we show how to set the message words and registers for the PIS in order to have all constraints fulfilled. We also give algorithms with complexity evaluations, when necessary, to demonstrate how the attack works.

Fulfilling all Constraints. To have constraints about X_{18} fulfilled (*i.e.,* Constraints 2, 4, and 8), we choose neutral bits from $X_{18}^{s_b}$, where $s_b = \{0, \ldots, 7, 16, \ldots, 22\}$. Similarly, to have Constraint 1 on X_{14} fulfilled, we restrict the neutral bits from bytes 3, 5, 7 of X_{14}, *i.e.,* $X_{14}^{s_f}$ with $s_f = \{24, \ldots, 31, 40, \ldots, 47, 57, \ldots, 63\}$ (bit 56 is reserved for padding). Due to the fact that addition/subtraction will only propagate differences towards MSB, the least significant bits of $X_{14}^{s_f}$ that may affect on X_2, X_3, X_6, X_{18} are 43 (due to $\lll 19$), 62 (due to $\lll 19$ twice), 24, and 24, respectively. However, $X_{18}^{s_b}$ has very low chance ($\simeq 0$) of affecting up to bit 43 of X_2, bit 62 of X_3, bit 24 of X_{18}, and we will filter candidates so that the influence on X_6 is limited to up to bit 23. Hence, Constraint 6 can be fulfilled. To fulfill Constraint 5, we force $Y_6^{s_f} = X_{14}^{s_f}$ (through setting $X_{13}^{s_f} = \mathbb{0}$), and $X_7^{s_f} = K_4^{s_f}$. We leave Constraint 3 for PIS setup, and Constraint 7 for partial matching, to be addressed later.

Precomputed Initial Structure. For the precomputed initial structure to work, we have to preset several message words. Besides $X_{13}^{s_f} = 0$ and $X_7^{s_f} = K_4^{s_f}$, we still need to take care of the padding. We set $X_6^{56} = 1$, *i.e.,* the length of original message in last block is 447 ($7 \times 64 - 1$). Hence, we need to set $X_7^{\{0,\ldots,8\}} = 447$. Note that adding more blocks will affect the length by a multiple of 2^9, which has no effect on the 9 LSBs of X_7. To reduce the influence of $X_{14}^{s_f}$ on X_6, we further set $(\overline{Y}_4 \gg 23 \oplus Y_5)^{s_f} = \mathbb{0}$, so that only $X_6^{s_f}$ out of X_6 will be affected. Note the PIS can be done in 2^{15} evaluations of key scheduling (leaving restriction on $X_{14}^{s_f}$ for probability only). This is negligible since we can reuse the PIS for at least 2^{16} times, to be discussed later.

Finding good candidates - Backward. We use bits from $X_{18}^{s_b}$ to compute the good candidates for backward direction. Constraint 2 further restricts us to choose values such that $X_{18}^{\{0,\ldots,7\}}$ and $X_{18}^{\{16,\ldots,23\}}$ are multiple of 9 ($mul = 9$ for third pass). Hence, we can have $\lceil 2^8/9 \rceil \times \lceil 2^7/9 \rceil = 2^{8.8}$ good candidates. Finally, we filter out candidates which do not fulfill Constraint 6. Experiments show that the remaining good candidates are about 2^8. Note these good candidates need to be computed under the constrainted PIS, we use message modification techniques to fulfill the constraints for PIS, and to get the 2^8 good candidates in less than 2^{19} key scheduling evaluations. Details can be found in [15].

Finding good candidates - Forward. We use bits from $X_{14}^{s_f}$ to compute the good candidates for backward direction. To have Constraint 3 fulfilled, we need to filter the candidates, such that it gives $\mathbb{1}$ for $C_{i+3}^{s_b}$ as in Fig 10(b), this reduces the number of candidates to $2^{23-15} = 2^8$. Note that this part can be re-used for many different (at least 2^{16}) C_i, by changing the even bytes, which we can freely set at the very beginning of the MITM preimage attack. Hence, the time complexity for this part is also negligible.

Probabilistic Partial Matching. Partial matching matches A_2 from both sides, where we can compute A_2 in the forward direction without any problem. However, in the backward direction, we only know information of bytes 0, 1, 2, 4, 6 of X_6 (red), as to compute B_3^e. Note that $B_3 = (B_6 \oplus X_6 + \mathsf{even}(B_6))/5 - \mathsf{odd}(B_5)$ ($mul = 5$ for first pass), where B_5 and B_6 are known. We rewrite it to $B_3 = (B_6 \oplus X_6)/5 + K_5$, where $K_5 = \mathsf{even}(B_5)/5 - \mathsf{odd}(B_4)$. We can compute bytes 0, 1, 2 of B_3, yet we still need bytes 4, 6 from information of bytes 4, 6 of X_6 only. Note that $B_3^{\{32,\ldots,39\}} = (B_6^{\{32,\ldots,39\}} \oplus X_6^{\{32,\ldots,39\}} - \mathsf{Bo} \times 2^{32})/5 + K_5^{\{32,\ldots,39\}} + \mathsf{Ca} \times 2^{32}$, where $\mathsf{Bo} \in \{0, \ldots, 4\}$ denote borrow from bit 31 when '/5' is carried out, and $\mathsf{Ca} \in \{0, 1\}$ denote the carry for the '+' from bit 31. We deal with the Bo by computing all possible choices, and guess the $\mathsf{Ca} = K_5^{31}$ which results in a probability 3/4 for the Ca to be correct. This gives an example for byte 4, and we can deal with byte 6 similarly. The process results in 25 times more computations for partial matching, together with probability 9/16. However, we shall only need to repeat the even and the '$-$' at Step 3, so that the essential repetition is equivalent to less than 2^{-1} compression computations per candidate.

Complexity of Finding a (Second) Preimage. Following the MITM preimage attack framework, the pseudo-preimage attack works as follows.

1. Randomly choose A_{14}, B_{14}, C_{14}.
2. Compute precomputed initial structure.
3. Compute candidates in backward and forward directions.
4. Repeat for 2^{16} values of C_{14} by looping all values in byte 4 and 6 (this step is to make time complexity for first three steps negligible):
   (a) For each candidate for backward and forward directions, compute A_2 independently.

(b) Carry out probabilistic partial matching. If a full match on A_2 is found, further re-check if the "guess" is correct.

5. Repeat 1-4 until a pseudo-preimage is found.

The pseudo-preimage attack works in time $2^{185.4}$ ($2^{192-8} \times 1.5 \times (3/4)^{-2}$), which can be reduced to $2^{182.4}$ when more than 2^4 targets are available (by using targets as part of backward candidates as in GMTPP). The pseudo-preimage can be converted to preimage attack with time complexity $2^{189.7}$ using the traditional conversion, with memory requirement of order 2^8. Following the GMTPP framework, the time complexity can be further reduced to $2^{188.8}$ (by computing 24 pseudo preimages and $2^{192}/24$ linking messages), with the same memory requirement. Similarly, the second-preimage attack works in $2^{188.2}$, when the given message is of more than 2^4 blocks.

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 have extended 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 Enhanced 3-Sum Problem faster than 2^{2n} for a set size of 2^n exists, so that the MTPP can be valid for any l. On the other hand, we found pseudo-preimage for MD4 in 2^{72}, it will be interesting to see if any of the new conversion techniques or other unknown techniques 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, the narrow-pipe SHA-3 candidates seem to be natural targets.

Acknowledgements

We would like to thank Kazumaro Aoki, Gaëtan Leurent, Yu Sasaki, Lei Wang, and anonymous reviewers of EUROCRYPT 2010 and ASIACRYPT 2010 for their interesting discussions and helpful comments. Part of this work was done while the first author was visiting Graz University of Technology, supported by ECRYPT2. The work 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. Ross J. Anderson and Eli Biham. TIGER: A Fast New Hash Function. In Dieter Gollmann, editor, *FSE*, volume 1039 of *LNCS*, pages 89–97. Springer, 1996.
6. Kazumaro Aoki, Jian Guo, Krysitan Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for Step-Reduced SHA-2. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 578–597. Springer, 2009.
7. Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In Shai Halevi, editor, *CRYPTO*, LNCS, pages 70–89. Springer, 2009.

8. Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In Roberto Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC*, volume 5381 of *LNCS*, pages 103–119. Springer, 2009.
9. Ilya Baran, Erik D. Demaine, and Mihai P trascu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.
10. Elad Barkan, Eli Biham, and Adi Shamir. Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *LNCS*, pages 1–21. Springer, 2006.
11. Eli Biham. New Techniques for Cryptanalysis of Hash Functions and Improved Attacks on Snefru. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 444–461. Springer, 2008.
12. Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
13. Christophe De Cannière and Christian Rechberger. Preimages for Reduced SHA-0 and SHA-1. In David Wagner, editor, *CRYPTO*, volume 5157 of *LNCS*, pages 179–202. Springer, 2008.
14. Hans Dobbertin. The First Two Rounds of MD4 are Not One-Way. In Serge Vaudenay, editor, *FSE*, volume 1372 of *LNCS*, pages 284–292. Springer, 1998.
15. Jian Guo. The C Program Verifies the Preimage Attacks against Tiger and MD4, 2010. Available: `http://www.jguo.org/docs/Tiger-MD4-AC10.zip`.
16. N. Haller. RFC1760 - The S/KEY One-Time Password System, 1995.
17. Martin E. Hellman. A Cryptanalytic Time - Memory Trade-Off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
18. Takanori Isobe and Kyoji Shibutani. Preimage Attacks on Reduced Tiger and SHA-2. In Orr Dunkelman, editor, *FSE*, volume 5665 of *LNCS*, pages 139–155. Springer, 2009.
19. John kelsey and Bruce Schneier. Second Preimage on n-bit hash functions for much less than 2^n work. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*. Springer, 2005.
20. Dmitry Khovratovich, Ivica Nikolic, and Ralf-Philipp Weinmann. Meet-in-the-Middle Attacks on SHA-3 Candidates. In Orr Dunkelman, editor, *FSE*, volume 5665 of *LNCS*, pages 228–245. Springer, 2009.
21. Lars R. Knudsen and John Erik Mathiassen. Preimage and Collision Attacks on MD2. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *LNCS*, pages 255–267. Springer, 2005.
22. Lars R. Knudsen, John Erik Mathiassen, Frédéric Muller, and Søren S. Thomsen. Cryptanalysis of MD2. *Journal of Cryptology*, 23(1):72–90, 2010.
23. Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In *EUROCRYPT*, pages 55–70, 1992.
24. Gaëtan Leurent. Message Freedom in MD4 and MD5 Collisions: Application to APOP. In Alex Biryukov, editor, *FSE*, volume 4593 of *LNCS*, pages 309–328. Springer, 2007.
25. Gaëtan Leurent. MD4 is Not One-Way. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 412–428. Springer, 2008.
26. Florian Mendel. Two Passes of Tiger Are Not One-Way. In Bart Preneel, editor, *AFRICACRYPT*, volume 5580 of *LNCS*, pages 29–40. Springer, 2009.
27. Florian Mendel, Norbert Pramstaller, and Christian Rechberger. A (Second) Preimage Attack on the GOST Hash Function. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 224–234. Springer, 2008.
28. Florian Mendel, Norbert Pramstaller, Christian Rechberger, Marcin Kontak, and Janusz Szmidt. Cryptanalysis of the GOST Hash Function. In David Wagner, editor, *CRYPTO*, volume 5157 of *LNCS*, pages 162–178. Springer, 2008.
29. Florian Mendel, Bart Preneel, Vincent Rijmen, Hirotaka Yoshida, and Dai Watanabe. Update on Tiger. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *LNCS*, pages 63–79. Springer, 2006.
30. Florian Mendel and Vincent Rijmen. Cryptanalysis of the Tiger Hash Function. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 536–550. Springer, 2007.
31. Florian Mendel and Vincent Rijmen. Weaknesses in the HAS-V Compression Function. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC*, volume 4817 of *LNCS*, pages 335–345. Springer, 2007.
32. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
33. Frédéric Muller. The MD2 Hash Function Is Not One-Way. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *LNCS*, pages 214–229. Springer, 2004.
34. Yusuke Naito, Yu Sasaki, Noboru Kunihiro, and Kazuo Ohta. Improved Collision Attack on MD4 with Probability Almost 1. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *LNCS*, pages 129–145. Springer, 2005.
35. Yu Sasaki and Kazumaro Aoki. Preimage attacks on 3, 4, and 5-pass HAVAL. In Josef Pawel Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *LNCS*, pages 253–271. Springer, 2008.

36. Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster than Exhaustive Search. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *LNCS*, pages 134–152. Springer, 2009.
37. Yu Sasaki, Lei Wang, Kazuo Ohta, and Noboru Kunihiro. Security of MD5 Challenge and Response: Extension of APOP Password Recovery Attack. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *LNCS*, pages 1–18. Springer, 2008.
38. Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *LNCS*, pages 1–22. Springer, 2007.
39. Marc Stevens, Alex Sotirov, Jake Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *LNCS*, pages 55–69. Springer, 2009.
40. Serge Vaudenay. On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In Bart Preneel, editor, *FSE*, volume 1008 of *LNCS*, pages 286–297. Springer, 1994.
41. Lei Wang and Yu Sasaki. Finding Preimages of Tiger Up to 23 Steps. In Seokihie Hong and Tetsu Iwata, editors, *FSE*, volume 6147 of *LNCS*, pages 116–133. Springer, 2010.
42. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.
43. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
44. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.
45. Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The Second-Preimage Attack on MD4. In Yvo Desmedt, Huaxiong Wang, Yi Mu, and Yongqing 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 the 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 directions 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, the more bits are fixed, the fewer bits for neutral words and more steps/more bits can be used for partial matching. So there is a trade-off between bits for neutral words and bits for partial matching. When multi-targets are available, we are to use fewer 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.

B Proof

This section proves $\Sigma_{i=1}^{2^k} 2^{n-l} \cdot i^{-1/2} \simeq 2^{n-l+1+k/2}$.

Proof. Using Riemann integral, we can estimate $\Sigma_{i=1}^{2^k} i^{-1/2}$ as $\int_{i=0}^{2^k} i^{-1/2} = 2^{k/2+1}$, with error in the range (-2, 0]. A concrete proof is shown below.

Let $f(x) = \lceil x \rceil^{-1/2}$ for $x > 0$, and $g(x) = x^{-1/2}$, we immediately have $\Sigma_{i=1}^{2^k} i^{-1/2} = \int_{i=0}^{2^k} f(x)dx \leq \int_{i=0}^{2^k} g(x)dx = 2^{k/2+1}$ (note $g(x) \geq f(x)$ for all $x > 0$). Similarly, $f(x) \geq g(x+1)$ for all $x > 0$, hence $\Sigma_{i=1}^{2^k} i^{-1/2} \geq \int_{i=0}^{2^k} g(x+1)dx = 2((2^k+1)^{1/2} - 1) > 2^{k/2+1} - 2$. Hence the estimation $\Sigma_{i=1}^{2^k} i^{-1/2} \simeq 2^{k/2+1}$ is valid. Multiplying both side by 2^{n-l} proves above.