Investigating Fundamental Security Requirements on Whirlpool: Improved Preimage and Collision Attacks

Yu Sasaki[1], Lei Wang[2,3], Shuang Wu[†4] and Wenling Wu[4]

[1] NTT Corporation
[2] The University of Electro-Communications
[3] Nanyang Technological University
[4] Institute of Software, Chinese Academy of Sciences
[†] Corresponding author. wushuang@is.iscas.ac.cn

Abstract. In this paper, improved cryptanalyses for the ISO standard hash function Whirlpool are presented with respect to the fundamental security notions. While a subspace distinguisher was presented on full version (10 rounds) of the compression function, its impact to the security of the hash function seems limited. In this paper, we discuss the (second) preimage and collision attacks for the hash function and the compression function of Whirlpool. Regarding the preimage attack, 6 rounds of the hash function are attacked with 2^{481} computations while the previous best attack is for 5 rounds with $2^{481.5}$ computations. Regarding the collision attack, 8 rounds of the compression function are attacked with 2^{120} computations, while the previous best attack is for 7 rounds with 2^{184} computations. To verify the correctness, especially for the rebound attack on the Sbox with an unbalanced Differential Distribution Table (DDT), the attack is partially implemented, and the differences from attacking the Sbox with balanced DDT are reported.

Keywords: Whirlpool, preimage, collision, meet-in-the-middle, guess-and-determine, local collision

1 Introduction

Hash functions are taking important roles in various aspects of modern cryptography. Since the collision resistance of MD5 and SHA-1 has been broken by Wang *et al.* [1, 2], cryptographers have looked for stronger hash function designs. While various new designs are discussed in the SHA-3 competition [3], some of existing hash functions seem to be much stronger than the MD4-family. Evaluating such hash functions is useful especially if they have been standardized internationally.

For hash functions, three security notions are classically considered: Collision Resistance, Second-Preimage Resistance, and Preimage Resistance. Besides, cryptographers recently have considered various non-ideal properties. Although considering such non-ideal properties is important especially for determining a

new standard, focusing on vulnerabilities that can be exploited in practice is more important especially for evaluating hash functions in practice.

Whirlpool [4] is a 512-bit hash function proposed by Rijmen and Barreto in 2000. The compression function uses a 10-round AES based cipher with $8*8$-byte internal states, and the output is computed with the Miyaguchi-Preneel mode [5, Algorithm 9.43]. Whirlpool has been adopted by ISO [6] and NESSIE [7].

Regarding the collision attack, the rebound attack proposed by Mendel *et al.* [8] is very effective with respect to the differential attack against AES based structure. Indeed, Mendel *et al.* presented a 4-round collision attack on the hash function and a 5-round collision attack on the compression function of Whirlpool. Many improved techniques of the rebound attack have been devised such as start-from-the-middle technique [9], linearized match-in-the-middle technique [9], super-(S)box analysis [10, 11], and multiple-inbound technique [11, 12]. Besides, for the AES based structure with $8 * 8$ state including Whirlpool, more techniques have been proposed such as hyper-Sbox analysis [13], non-full-active super-Sbox analysis [14], efficient list-merging technique [15], and three inbound rounds [16]. Several practical results are given for round-reduced algorithms and intermediate rounds in [9, 17, 18]. This paper exploits the differences in both of data processing part and key schedule part. Some similarities can be seen in the analysis on AES-256 [19] and two analysis on Grøstl [20, 21].

Regarding the preimage attack, meet-in-the-middle (MitM) attack with the splice-and-cut technique proposed by Aoki and Sasaki [22] has been actively discussed. Several papers proposed improved techniques [23, 24]. For the preimage attack against the AES based structure, Sasaki showed a second preimage attack on 5 rounds of Whirlpool [25]. Later, Wu *et al.* improved its complexity and extended it to the preimage attack [26]. Note that Bogdanov *et al.* showed an attack on 10-round AES in hashing modes with the biclique technique [27]. Because this attack exploits the weakness in the AES key-schedule, the attack is specific to AES and cannot be directly applied to other AES based primitives.

Our contributions. In this paper, we improve cryptanalyses on Whirlpool with respect to the fundamental security notions. The main results are a 6-round preimage attack on the hash function and an 8-round collision attack on the compression function. The results are summarized in Table 1.

Our preimage attack is based on the previous 5-round MitM attacks [25, 26]. The number of attacked rounds is extended by applying the guess-and-determine approach during the MitM attack. Moreover, we increase the number of free bits for each chunk by exploiting the freedom degrees of the key, while previous attacks fix the key as a constant. More precisely, the key schedule function shares the same round function with the data process procedure, and thus we separate the key schedule function in the same way with the data process function.

Our collision attack is based on the rebound attack. We use the key difference to cancel the difference in the data part, while previous work avoided inserting differences to the key schedule. This leads to a differential path with a high probability. In this paper, we implement our 4-round collision attack which only

Table 1. Summary of attack results

Type		Target	#Rounds	Time	Mem.	Ref.	Remarks
Fundamental Properties	Preimage	Hash Function	5	$2^{481.5}$	2^{64}	[26]	
			5	2^{448}	2^{96}	Ours	
			5	2^{465}	$O(1)$	Ours	Memoryless MitM
			6	2^{481}	2^{256}	Ours	
			6	2^{504}	$O(1)$	Ours	Memoryless MitM
	Second Preimage	Hash Function	5	2^{504}	2^{8}	[25]	
			5	2^{448}	2^{64}	[26]	
			5	2^{464}	$O(1)$	Ours	Memoryless MitM
			6	2^{481}	2^{256}	Ours	
			6	2^{504}	$O(1)$	Ours	Memoryless MitM
	Collision	Hash Function	4	2^{64}	2^{8}	[9]	
			5	2^{120}	2^{64}	[10]	
		Compress. Function	7	2^{184}	2^{8}	[28]	semi-free-start
			7	2^{120}	2^{128}	[28]	semi-free-start
			4	2^{8}	2^{8}	Ours	free-start
			7	2^{64}	2^{8}	Ours	free-start
			8	2^{120}	2^{8}	Ours	free-start
Other Properties	Near-collision	Compress. Function	9	2^{176}	2^{8}	[28]	
			9	2^{112}	2^{128}	[28]	
	Distinguisher	Compress. Function	10	2^{188}	2^{8}	[28]	
			10	2^{121}	2^{128}	[28]	

requires 2^8 computations. Because all previous collision attacks require at least 2^{64} computations even for a small number of rounds, this is the first example of the collision for a reduced compression function. We also partially implement the 7-round collision attack. We show an example of the 40-byte near-collision.

2 Specification and Notations

Whirlpool [4] takes any message with less than 2^{256} bits as input, and outputs a 512-bit hash value. It adopts the Merkle-Damgård structure. The input message M is padded into a multiple of 512 bits. In details, the 256-bit binary expression of the bit length ℓ is padded according to the MD-strengthening, *i.e.* $M\|1\|0^*\|\ell$. The padded message is divided into 512-bit blocks $M_0\|M_1\|\cdots\|M_{N-1}$. Let H_n be a 512-bit chaining variable. First, an initial value IV is assigned to H_0. Then, $H_{n+1} \leftarrow \mathrm{CF}(H_n, M_n)$ is computed for $n = 0, 1, \ldots, N-1$, where CF is a compression function. H_N is produced as the hash value of M. CF uses an AES based block-cipher E_k, which takes a 512-bit chaining variable H_i as a key and a 512-bit message block M_i as a plaintext. The output of CF is computed by the Miyaguchi-Preneel mode, *i.e.* $E_{H_i}(M_i) \oplus M_i \oplus H_i$.

Inside the block cipher E_k, an internal state is represented by an $8 * 8$ byte array. At first, H_i is assigned to the key value k_0. Then, ten 512-bit subkeys $k_1, k_2, \ldots, k_{10}$ are generated by the key-schedule function defined as follows:

$$k_{n+1} \leftarrow \mathrm{AC} \circ \mathrm{MR} \circ \mathrm{SC} \circ \mathrm{SB}(k_n), \text{ for } n = 0, 1, \ldots, 9.$$

- SubBytes(SB): applies the Substitution-Box to each byte.

- ShiftColumns(SC): cyclically shift the j-th column downwards by j bytes.
- MixRows(MR): multiply each row of the state matrix by an MDS matrix.
- AddRoundConstant(AC): XOR a 512-bit constant defined in the specification.

For the data processing part, M_i is assigned to the plaintext p. Then, the whitening operation is performed and the result is stored into a variable s_0, *i.e.* $s_0 \leftarrow k_0 \oplus p$. The output s_{10} of the block cipher is computed as follows, where AddRoundKey(AK) takes the XOR with k_{n+1}.

$$s_{n+1} \leftarrow \mathrm{AK} \circ \mathrm{MR} \circ \mathrm{SC} \circ \mathrm{SB}(s_n), \text{ for } n = 0, 1, \ldots, 9.$$

Notations. Byte positions in a state S are denoted by integer numbers $0, 1, \ldots, 63$, where the byte $8j + i$ corresponds to the byte in the i-th row and the j-th column of the state $\#S$, and is denoted by $\#S[8j+i]$. We denote the initial state for the data processing part in round x by $\#Dx^I$. Then, states immediately after SB, SC, MR, and AR in round x are denoted by $\#Dx^{SB}$, $\#Dx^{SC}$, $\#Dx^{MR}$, and $\#Dx^{AK}$, respectively. Obviously, $\#Dx^{AK}$ is identical with $\#D(x+1)^I$. Similarly, we use the notations $\#Kx^I, \#Kx^{SB}$, $\#Kx^{SC}$, $\#Kx^{MR}$, and $\#Kx^{AC}$ for the key schedule part. We often denote several bytes of state $\#S$ by $\#S[a, b, \ldots]$, e.g. 8 bytes in the right most column are denoted by $\#S[56, 57, \ldots, 63]$. We also use the following notations to denote specific byte positions.

- $\#S[row(i)]$: 8 byte-positions in the i-th row of state $\#S$
- $\#S[\mathrm{SC}(row(i))]$: 8 byte-positions which SC is applied to $\#S[row(i)]$
- $\#S[\mathrm{SC}^{-1}(row(i))]$: 8 byte-positions which SC^{-1} is applied to $\#S[row(i)]$

3 Related Work

3.1 Meet-in-the-Middle (Second) Preimage Attack on Whirlpool

In FSE 2011, Sasaki proposed the first MitM preimage attack on AES-like primitives [25]. Two main techniques were introduced: initial structure in an AES-like permutation and partial-matching across an MixColumn operation. As a direct application, a second preimage attack is found on 5-round Whirlpool hash function in [25]. In FSE 2012, Wu *et al.* improved the complexity of 5-round second preimage attack on Whirlpool [26] by exploiting more freedom degrees in the data state. They successfully represent the chunk separations by several essential integer parameters, and launched an automatic exhaustive search. Moreover, they also proposed a method to deal with the message padding and extended the attack into a first preimage attack.

3.2 Rebound Attack and Start-from-the-Middle Technique

The rebound attack was introduced by Mendel *et al.* [8]. If it is applied to Whirlpool, the 2-round path $8 \rightarrow 64 \rightarrow 8$ can be satisfied only with 2^8 computations. The path for rounds S and $S + 1$ is described in Fig. 1. First, an 8-byte

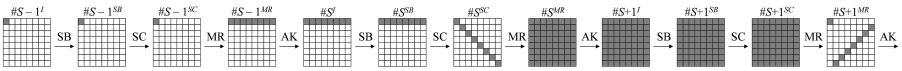

Fig. 1. Rebound and start-from-the-middle techniques

difference at $\#S+1^{MR}$ is randomly chosen, and it is propagated to $\#S+1^{SB}$. Then, a single-byte difference at one of the active bytes at $\#S^{SB}$ is randomly chosen, and it is propagated to 8 bytes of $\#S+1^{I}$. For each S-box in round $S+1$, randomly given input and output differences have solutions (paired values conforming the path) with probability about 2^{-1}, and the average number of solutions is 2. Hence, if we choose 2^8 differences for the single byte at $\#S^{SB}$, we obtain 2^8 solutions for the corresponding 8 S-boxes. By iterating it for 8 active bytes at $\#S^{SB}$, we obtain 2^8 solutions for each i of $\#S^{SB}[SC^{-1}(row(i))]$.

The start-from-the-middle technique is an improved procedure for the rebound attack, which was proposed by Mendel *et al.* [9]. It satisfies a 3-round differential path with the same complexity as the rebound attack. After obtaining 2^8 solutions for each i of $\#S^{SB}[SC^{-1}(row(i))]$ with the rebound attack, each solution is computed until $\#S-1^{MR}[SC^{-1}(row(i))]$. For each i, 127 kinds of differences are obtained at $\#S-1^{MR}$. Then, a single-byte difference at $\#S-1^{SB}$ is chosen. The attacker propagates it to $\#S-1^{MR}$, and checks whether the 8-byte difference can be produced from the solutions of the rebound attack. Because there are 127 kinds of the differences for each i, the 8-byte differences can be produced with probability about 2^{-8}. Therefore, by choosing 2^8 differences at $\#S-1^{SB}$, we expect to find the desired difference. In summary, the 3-round differential path $1 \rightarrow 8 \rightarrow 64 \rightarrow 8$ can be satisfied with a complexity of 2^8.

Note that the behavior of the S-box is explained based on the S-box of AES. Because the S-box of Whirlpool has a different property, the evaluation for AES cannot be applied to Whirlpool directly. We later discuss this issue in Sect. 5.4.

3.3 Distinguisher for the Full Whirlpool Compression Function

Lamberger *et al.* proposed a distinguisher for the full Whirlpool compression function [11, 28]. The distinguished property is called subspace distinguisher. The dimension of the input and output differences are defined before the analysis starts. The attacker aims to find paired values whose dimension of differences at input and output are lower than the defined ones. The core technique is running the rebound attack ($8 \rightarrow 64 \rightarrow 8$) at two parts independently without determining the key value. Then, two results are connected and a long differential path ($8 \rightarrow 64 \rightarrow 8 \rightarrow 8 \rightarrow 64 \rightarrow 8$) is satisfied by searching for an appropriate key value. Although the distinguisher beautifully breaks the full-round compression function, the impact is very limited. Nevertheless, collisions on compression function are generated with this technique for 7 rounds with (Time, Memory)= $(2^{184}, 2^8)$ or $(2^{120}, 2^{128})$.

3.4 Local Collision on AES-like Primitives

For a distinguisher for AES-256, Biryukov *et al.* introduced differences to both the key and the data, and used the difference of round keys to cancel the difference of the internal states of the data process by the AddRoundKey operation, i.e. the local collision occurs [19]. The local collisions may help the attacker to build a high probability differential path on AES-like primitives.

4 Preimage Attack on 6-Round Whirlpool

4.1 Overview

Our first and main result is introducing the guess-and-determine approach to MitM preimage attack on Whirlpool hash function, and successfully increase one more attacked round. More specifically, during the independent chunk computation, even one unknown input byte of $MixRow$ makes all the 8 output bytes unknown, which is heavily unbalanced. So a chunk can guess a small number of unknown bytes in order to significantly increase the number of known bytes in the following rounds. Thus guess-and-determine approach is very effective for preimage attack on Whirlpool.

Our second result is exploiting the freedom degree in the key to increase the number of free bits in each chunk, and thus successfully reduce the complexity. Since the key schedule of Whirlpool is the same with the data process, we can separate the key schedule and the data process into two chunks in the same way, which doubles the number of free bits in both chunks.

Our third result is that we propose not only a first preimage attack on hash function with the lowest complexity, but also another memoryless preimage attack. Compared to the brute force attack, the second attack requires the same memory and a lower complexity. This is achieved by finding a last block attack first and then linking the chaining values with a fixed-key attack on the compression function. Since both the last block attack and the fixed-key attack can be implemented in a memoryless way [30], we obtain a memoryless first preimage attack.

4.2 Preimage Attack on 6-round Compression Function

The chunk separation used in the 6-round attack is illustrated in Fig. 2. Five different colors are used to indicate the categories of the bytes. The gray bytes are constants which come from the hash/output value or the initial structure. The red/blue bytes belong to the backward/forward chunk, which can be determined by the red/blue byte in the initial structure. The white bytes are affected by both red and blue bytes and we can only determine their values after a partial match is found. The purple bytes are the guessed bytes.

Since that each row of the state $\#D1^{MR}$ has unknown bytes (in white color), if we went further back through MR^{-1}, all bytes would become unknown. The values of 24 white bytes in row 0 to row 5 are guessed. Thus we can maintain 6

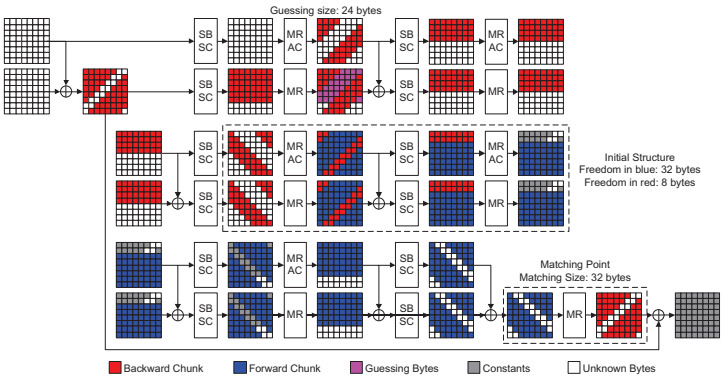

Fig. 2. Chunk separation for the preimage attack on 6-round compression function

red bytes in each of top 6 rows of the state $\#D1^{SC}$. In the key state, we do not guess the values, since the white key state $\#K1^{SC}$ does not affect the matching state through the feedforward operation.

All the possible values of the guessed bytes are used as extra freedom degrees to build the lookup table for the MitM. But after a partial match is found, we need to further check the correctness of the guessed values. More details about the guessing technique can be found in the following section.

The attack algorithm. In order to evaluate the attack complexity, we need to know the parameters: freedom degrees in red and blue bytes (D_r, D_b), size of the partial matching m and the number of guessed bits D_g. The explanation on calculating freedom degrees/size of matching point and how the partial matching works can be found in previous papers [25, 26]. Here we omit these details due to the limited space.

To summarize, the parameters for MitM attack in Fig. 2 are as follows. Freedom degrees in red bytes: $D_r = 8$ bytes $= 64$ bits (4 bytes in the key and 4 bytes in the data). Freedom degrees in blue bytes: $D_b = 32$ bytes $= 256$ bits (16 bytes in the key and 16 bytes in the data). Size of the guessed value (purple bytes):$D_g = 24$ bytes $= 192$ bits. Size of the partial match: $m = 32$ bytes $= 256$ bits (only in the data). Size of the full match: $n = 512$ bits.

The attack algorithm is as follows:

Step 1. Randomly choose the values of the constants in the initial structure.
Step 2. For all the 2^{D_r} values $\{r_i\}$ of the red bytes in the initial structure and 2^{D_g} guessed values $\{g_j\}$, go backward to the matching point and store all $2^{D_r+D_g}$ partial matching values $F(r_i, g_j)$ in a look-up table L.
Step 3. For all the 2^{D_b} values $\{b_k\}$ of the blue bytes in the initial structure, go forward to obtain the partial matching value $G(b_k)$ and check if it is in L.
Step 4. Once a partial match (r_i, g_j, b_k) such that $F(r_i, g_j) = G(b_k)$ is found, use (r_i, b_k) to compute and check if the guessed value g_j is correct. If the guess is correct, check if it is a preimage.

Step 5. Repeat the above steps 1-4 to find a preimage.

The complexity is explained as follows:

Step 2. It takes $2^{D_r+D_g}$ computations and memory to build the look-up table.
Step 3. It takes 2^{D_b} computations to find all the $2^{D_b+D_r+D_g-m}$ partial matches.
Step 4. $2^{D_b+D_r+D_g-m}$ computations are needed to verify the correctness for all the partial matches. There would be $2^{D_b+D_r-m}$ valid partial matches that pass the correctness test, since the probability that g_j is correct is 2^{-D_g}.
Step 5. The probability that steps 1-4 succeed is $2^{D_b+D_r-m} \cdot 2^{-(n-m)} = 2^{D_b+D_r-n}$. The above steps are repeated for $2^{n-D_b-D_r}$ times to find a preimge.

Therefore, the complexity of the above algorithm is

$$2^{n-D_b-D_r} \cdot (2^{D_r+D_g} + 2^{D_b} + 2^{D_b+D_r+D_g-m}) = 2^n \cdot (2^{-D_r} + 2^{D_g-D_b} + 2^{D_g-m}) \quad (1)$$

With the given parameters, the complexity is about $2^{512} \cdot (2^{-64} + 2^{192-256} + 2^{192-256}) \approx 2^{448}$ compression function calls. Only step 2 requires $2^{64+192} = 2^{256}$ memory.

It is observed that the pattern for the chunk separation can be represented as several numbers: b= the number of blue rows in $\#D2^{MR}$, r= the number of red rows in $\#D2^{I}$, w=the number of white rows in $\#D5^{SC}$, g=the number of guessed rows in $\#D1^{MR}$. Then the parameters for the MitM attack can be calculated as: $D_b = 16(b-r)$ bytes, $D_r = 2w(8-b)$ bytes, $D_g = g(8-r)$ bytes and $m = 8(g+(8-w)-8) = 8(g-w)$ bytes. In the following sections, we will continue using the parameters of b, r, w and g to identify the pattern for chunk separations. We searched for all the possible patterns of the chunk separation by exhaustively enumerating the parameters b, r, w and g. Fig. 2 shows the optimal complexity case $(b, r, w, g = 6, 4, 2, 6)$. Note that the 6-round attack is also applicable without using freedom degrees of the key.

Memoryless MitM attacks. In [30], Morita *et al.* proposed the memoryless MitM technique, which can be applied in our attack by designing the following three functions:

1) a mapping from the partial matching value to the blue value,
2) a mapping from the partial matching value to the red (and purple) value,
3) a pseudo-random boolean switching function taking the partial matching value as the input.

However, we found that the memoryless MitM has some limitations. The memoryless MitM is very efficient to find one match, its complexity is limited by half of the matching size m and increases linearly with the number of matches. Namely, at most $2^{max\{0,min\{D_b-D_g,D_r,m/2-D_g\}\}}$ computations can be saved using memoryless MitM. Using look-up tables, we can save at most $2^{max\{0,min\{D_b-D_g,D_r,m-D_g\}\}}$ computations. This difference results in different optimal chunk separations, which is considered in the following sections.

4.3 The First Preimage Attacks

A first preimage attack is the combination of a second preimage attack and an attack on the last compression function which produces message block with correct padding. In order to find optimal first preimage attacks, we need to consider a lot of different attacks.

Two types of last block attacks. The first preimage attack must fulfill the message length padding. In a fixed-key attack on the compression function, 10 padding bits can be chosen if the initial structure is placed at the beginning of the encryption. This technique was used in [26]. The probability that a random message block satisfies a constraint of the padding string is 2^{-9}. Details are explained in Appendix B.

In the chosen-key preimage attacks, the initial structure cannot be placed at the beginning of the compression function. So the chosen padding technique is not applicable. However, we can repeat the attack 2^9 times to obtain a valid last message block.

Since Whirlpool uses 256-bit length padding and we just satisfied a small part of it, the rest part of the length cannot be known before the attack. Therefore, we need the expandable messages [31] to fulfill it.

Two types of second preimages. In previous attacks, the key (chaining value) is known before the attack. The preimage attack on the compression function is to find a message block that connect two chaining values. The fixed-key attack is equivalent to a second preimage attack if the input and output chaining values are chosen consecutively from the known ones.

If the key is chosen, the value of the key (chaining value) can only be determined after the attack. Then we need to connect it to one of the known chaining value. This is done using a MitM step on the chaining values.

Different combinations for the first preimage attack. First, we analyzed all the 5/6-round fixed-/chosen-key attacks on compression functions and turn them into second preimage and last-block attacks. Second, we considered the fixed-key attacks with chosen padding and found more attacks on the last message block. At last, we combine the second preimage attacks and the last-block attacks to found the first preimage attacks with the lowest computations and the lowest memory respectively.

The detailed results of all preimage attacks are summarized in Table 2. Note that we can adjust the time-memory tradeoff by choosing different combinations of second preimages and the last-block attacks or changing the tradeoff of MitM on the chaining value for chosen key attacks.

Table 2. Detailed Results on all preimage attacks on CF and hash function

5-Round Attacks		b r w	D_b D_r m	Compression Function	Second Preimage	Last Block	Preimage
chosen-key	-	5 4 2	128 96 128	$2^{416}, 2^{96}$	$2^{465}, 2^{96}$	$2^{425}, 2^{96}$†	$2^{448}, 2^{96}$†
	ml	4 3 1	128 64 128	$2^{448}, O(1)$	$2^{481}, 2^{32}$	$2^{457}, 2^{32}$	
fixed-key	-	4 3 2	64 64 64	$2^{448}, 2^{64}$	$2^{448}, 2^{64}$†	$2^{457}, 2^{64}$	
	ml	5 4 2	64 48 128	$2^{464}, O(1)$	$2^{464}, O(1)$‡	$2^{473}, O(1)$	
fixed-key	-	4 3 2	55 63 64	-	-	$2^{457}, 2^{55}$	$2^{465}, O(1)$‡
chosen padding	ml	5 4 2	54 48 128	-	-	$2^{464}, O(1)$‡	

6-Round Attacks		b r w g	D_b D_r D_g m	Compression Function	Second Preimage	Last Block	Preimage
chosen-key	-	6 4 2 6	256 64 192 256	$2^{448}, 2^{256}$	$2^{481}, 2^{256}$†	$2^{457}, 2^{256}$†	$2^{481}, 2^{256}$†
	ml	7 6 2 6	128 32 96 256	$2^{480}, O(1)$	$2^{497}, 2^{16}$	$2^{489}, O(1)$‡	
fixed-key	-	6 5 1 2	64 16 48 64	$2^{496}, 2^{64}$	$2^{496}, 2^{64}$	$2^{505}, 2^{64}$	
	ml	7 5 1 5	128 8 120 256	$2^{504}, O(1)$	$2^{504}, O(1)$‡	$2^{513}, O(1)$	
fixed-key	-	6 4 1 3	118 16 96 128	-	-	$2^{496}, 2^{112}$	$2^{504}, O(1)$‡
chosen padding	ml	7 6 1 3	54 8 48 128	-	-	$2^{506}, O(1)$	

† : The attacks with the lowest computations.
‡ : The attacks with the lowest memory.
ml : The memoryless MitM attacks.

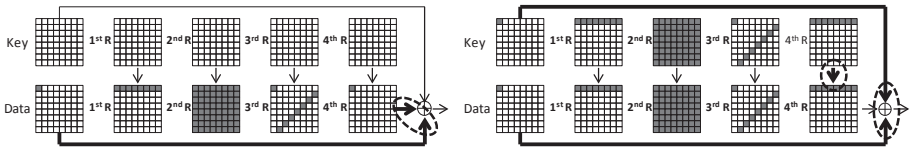

Fig. 3. Left: previous approach Right: our approach

5 Collision Attacks on the Compression Function

5.1 Overview

In order to generate collisions with previous rebound approaches, the state at the beginning and the end must have the same differential form so that they can cancel each other with the feed-forward operation. This is a strong constraint. We overcome this constraint by generating local collisions several times,*i.e.*, canceling differences of the data by using differences of the key. The idea is illustrated in Fig. 3. Because the diffusions for the data and key are identical, we can keep the same differential form. This makes possible to use the differential path with different differential forms between the beginning and the end.

The idea of using the key difference is advantageous not only for canceling the output difference but also constructing a high probability differential path by using the local collision. For example, we use the following differential path for an 8-round collision attack. Here, "WH" represents the whitening operation.

$$\begin{aligned}\text{Key: } 64 &\xrightarrow{\text{WH}} 64 \xrightarrow{1^{\text{st}}R} 8 \xrightarrow{2^{\text{nd}}R} 1 \xrightarrow{3^{\text{rd}}R} 8 \xrightarrow{4^{\text{th}}R} 64 \xrightarrow{5^{\text{th}}R} 8 \xrightarrow{6^{\text{th}}R} 1 \xrightarrow{7^{\text{th}}R} 8 \xrightarrow{8^{\text{th}}R} 64,\\ \text{Data: } 64 &\xrightarrow{\text{WH}} 0 \xrightarrow{1^{\text{st}}R} 8 \xrightarrow{2^{\text{nd}}R} 1 \xrightarrow{3^{\text{rd}}R} 8 \xrightarrow{4^{\text{th}}R} 0 \xrightarrow{5^{\text{th}}R} 8 \xrightarrow{6^{\text{th}}R} 1 \xrightarrow{7^{\text{th}}R} 8 \xrightarrow{8^{\text{th}}R} 0\,, \quad (2)\end{aligned}$$

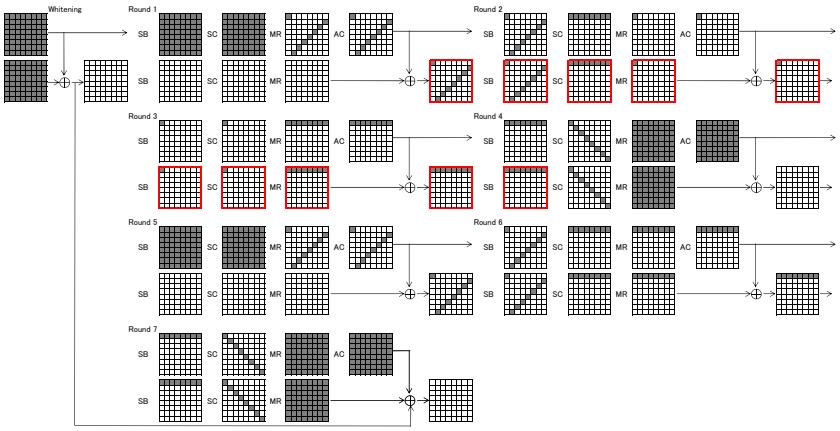

Fig. 4. Differential path for 7R attack. Grey bytes are active bytes. The inbound phase for the data processing part is stressed by red squares.

where, the most expensive part (full active state) is avoided for the data processing part to reduce the attack complexity and to keep enough freedom degrees.

We use a rebound-attack approach to search for the values. First, the values for the key are searched. Then, the values for the data are searched for the fixed key pairs. The complexity is a sum of two searching phases, not a product.

5.2 7-Round Collision Attack

We explain our 7-round collision attack, with 2^{64} computations and memory to store 2^8 state. The differential path is as follows. See its illustration in Fig. 4.

$$\text{Key: } 64 \xrightarrow{\text{WH}} 64 \xrightarrow{1^{\text{st}}R} 8 \xrightarrow{2^{\text{nd}}R} 1 \xrightarrow{3^{\text{rd}}R} 8 \xrightarrow{4^{\text{th}}R} 64 \xrightarrow{5^{\text{th}}R} 8 \xrightarrow{6^{\text{th}}R} 8 \xrightarrow{7^{\text{th}}R} 64,$$

$$\text{Data: } 64 \xrightarrow{\text{WH}} 0 \xrightarrow{1^{\text{st}}R} 8 \xrightarrow{2^{\text{nd}}R} 1 \xrightarrow{3^{\text{rd}}R} 8 \xrightarrow{4^{\text{th}}R} 0 \xrightarrow{5^{\text{th}}R} 8 \xrightarrow{6^{\text{th}}R} 8 \xrightarrow{7^{\text{th}}R} 0\,.$$

The key and the plaintext should have the same difference so that the plaintext difference can be canceled by the whitening operation. Then, we make a local collision after the 4th round, and another local collision after the 7th round.

Searching procedure for key schedule part. The goal is finding a single pair of key values satisfying the differential path for the key. The essential part of this procedure is finding two values satisfying the middle three rounds, $1 \rightarrow 8 \rightarrow 64 \rightarrow 8$. This can be done with the Start-from-the-Middle attack [9]. The complexity is only 2^8 computations and the amount of memory is 2^8 state. If the middle three rounds are satisfied, the entire path are also satisfied by simply

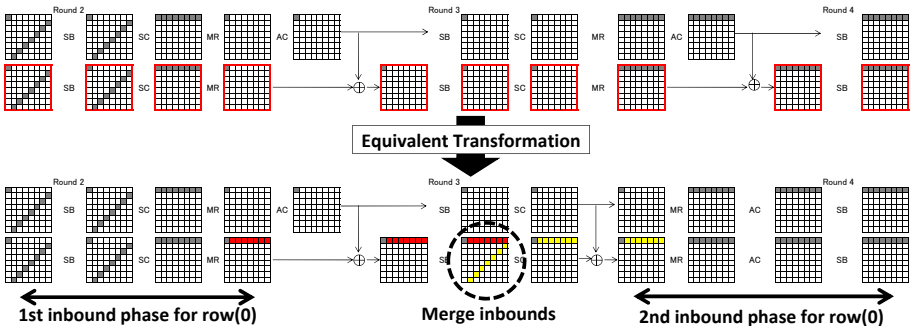

Fig. 5. Details of the inbound phase.

extending the path by 2 rounds in backward and 2 rounds in forward. Because this transformation is deterministic, the complexity for 7 rounds is unchanged.

Searching procedure for data processing part. This phase is performed after the key values are fixed. The goal is finding a pair of plaintexts which follow the differential path and generate a collision in the output. The procedure is divided into the inbound phase and the outbound phase.

Inbound phase. The inbound phase is from state $\#D2^I$ to state $\#D4^{SB}$, which are stressed by red squares in Fig. 4. For the inbound phase, we search for the values with a similar approach to Mendel *et al.* [11]. The details of the inbound phase are described in Fig. 5. Note that the key values are already fixed. Hence, the differences for $\#2D^I$ and $\#D4^{SB}$ are uniquely fixed. First, we apply an equivalent transformation to the third round, *i.e.* AK is performed between SC and MR. Then, the inbound phase is further divided into three parts; first inbound phase, second inbound phase, and merge two inbounds.

First inbound phase for row 0: We aim to find 2^8 paired values that satisfy the differential path between $\#D2^I[SC^{-1}(row(0))]$ and $\#D3^{SB}[row(0)]$ which are described by red in Fig. 5. We only compute a single row. The other rows remain unfixed. The difference for 8 bytes at $\#D2^I[SC^{-1}(row(0))]$ is fixed to the same as $\#K2^I[SC^{-1}(row(0))]$ so that the difference of $\#D2^I[SC^{-1}(row(0))]$ can be canceled by AK^{-1} in the first round. Then, for all 2^8 differences in $\#D2^{MR}[0]$, we compute the corresponding 8-byte difference at $\#D2^{SB}[SC^{-1}(row(0))]$. The average probability that the fixed difference at $\#D2^I[SC^{-1}(row(0))]$ and a computed one in $\#D2^{SB}[SC^{-1}(row(0))]$ have solutions for all 8 bytes is 2^{-8}. Because 2^8 differences are examined in $\#D2^{MR}[0]$, one pair is expected to have solutions and the number of obtained solutions is 2^8 on average. Finally, for all 2^8 solutions, we compute the corresponding 8 bytes at $\#D3^{SB}[row(0)]$ and store them in a list L_1.

Second inbound phase for row 0: This part is similar to the first inbound phase. We aim to find 2^8 paired values that satisfy the differential path between $\#D3^{SB}[SC^{-1}(row(0))]$ and $\#D4^{SB}[row(0)]$ which are described by yellow in Fig. 5. Again we only compute a single row. The difference for 8 bytes at $\#D4^{SB}[row(0)]$ is fixed to the same as $\#K4^{SB}[row(0)]$ so that it can be canceled after the AK operation in the fourth round. For all 2^8 differences in $\#D3^{SC}[0]$, we compute the corresponding 8-byte difference at $\#D4^{I}[row(0)]$, and check if solutions exist between the fixed $\#D4^{SB}[row(0)]$ and computed $\#D4^{I}[row(0)]$. After 2^8 trials, we expect to obtain 2^8 solutions on average. Finally, for 2^8 solutions, we compute the corresponding 8 bytes at $\#D3^{SB}[SC^{-1}(row(0))]$ and store them in a list L_2.

Merge two inbounds: One byte (in position 0) is overlapped in 8 bytes stored in L_1 and L_2, hence we need to find the match. Both of value and difference need to match, and thus the probability of the match is 2^{-16}. Because 2^{16} combinations of the results in L_1 and L_2 are available, we expect to find a match. We use the other 49 unfixed bytes at $\#D3^{SB}$ as freedom degrees for the outbound phase. Because it can produce $2^{49*8} = 2^{392}$ values for the outbound phase, finding one match is enough for this phase.

The complexity for the inbound phase is 2^8 computations for both of the first and second inbound phases. A memory to store 2^8 state is required to generate L_1 and L_2. In summary, with 2^8 computations and a memory to store 2^8 state, up to 2^{392} solutions of the inbound phase can be produced.

Outbound phase. Due to the inbound phase, the differential path is ensured to be satisfied up to the fourth round. The outbound phase is a brute force approach to satisfy the differential path after the fourth round by using solutions of the inbound phase. The only probabilistic event for the outbound phase is the cancelation of the difference at the final output. This occurs when the differences for $\#D7^{SB}[row(0)]$ is the same as $\#K7^{SB}[row(0)]$. Therefore, by examining 2^{64} solutions of the inbound phase, we can obtain a collision at the final output.

In summary, a collision is generated with 2^{64} in time and 2^8 in memory.

5.3 Extension to 8-Round Collision Attack and Other Variants

The 7-round attack in Sect. 5.2 can be extended to 8 rounds. The differential path up to the 4th round is exactly the same as the one for the 7-round attack. Therefore, the inbound part is unchanged. In the outbound phase, $8 \rightarrow 8 \rightarrow 64$ is replaced with $8 \rightarrow 1 \rightarrow 8 \rightarrow 64$. The entire path is given in Eq.(2).

Because the attack procedure is very similar, we only mention the difference from the 7-round attack. To search for the key values, we use the Start-from-the-Middle approach. In this time, the differential propagation $8 \xrightarrow{6^{\text{th}}R} 1$ needs to be satisfied probabilistically. Therefore, the complexity for the key schedule part is 2^{56} in time and 2^8 in memory. Note that the complexity can be improved to 2^{48} with the linearized match-in-the-middle technique [9]. Because this part

is not the bottle-neck, we omit its detailed explanation. Also note that only 1 result is enough because the data processing part can produce many solutions.

For the data processing part, the inbound phase is exactly the same as the one for the 7-round attack, which requires 2^8 in time and 2^8 in memory, and can produce up to 2^{392} solutions. In the outbound phase, the probabilistic events are the differential propagation $8 \xrightarrow{6^{\text{th}}R} 1$ and the differential cancelation at the output state. Therefore, a collision for 8 rounds can be generated with a complexity of $2^{56+64} = 2^{120}$ computations and 2^8 state of memory.

It seems worth mentioning that our differential path is an iterative form;

$$\begin{aligned} \text{Key: } 64 &\xrightarrow{x} 8 \xrightarrow{x+1} 1 \xrightarrow{x+2} 8 \xrightarrow{x+3} 64, \\ \text{Data: } 0 &\xrightarrow{x} 8 \xrightarrow{x+1} 1 \xrightarrow{x+2} 8 \xrightarrow{x+3} 0\,. \end{aligned}$$

Therefore, constructing a differential path for $4n$ rounds or $4(n-1)+3$ rounds is possible. However, we cannot find the attack for three iterations (12-rounds or 11-rounds) due to a too high complexity and too small freedom degrees.

Practical near-collision attack on 7 rounds. In some case, near-collisions can be a real threat because hash values are used after the truncation. Our 7-round attack in Sect. 5.2 can generate a 40-byte near-collision with a complexity of 2^{40} computations and 2^8 state of memory. For this attack, we only cancel the difference in 5-bytes between $\#K7^{SB}[row(0)]$ and $\#D7^{SB}[row(0)]$. Note that the brute force attack for 40-byte near-collision takes 2^{160} computations, and thus our attack is much faster. We also implemented the attack on a PC, and confirmed that the attack could work correctly. An example of the generated data is provided in Table 3 in Appendix C.

Practical collisions on 4 rounds. All previous attacks require at least 2^{64} computations to generate a collision even for a small number of rounds. Therefore, we investigate the practical collision attack on a small number of rounds.

Our differential path generates a local collision after the fourth round, and up to fourth round can be covered by the inbound phase. Therefore, we can generate collisions of the 4-round Whirlpool compression function only with 2^8 computations and 2^8 state of memory. No extra practical example is given here since the 7-round near-collision in Table 3 is also a 4-round collision.

5.4 Theory vs Practice: Implementation of Rebound Attacks

The DDT of the S-box is the core of the rebound attack, which provides an efficient method for satisfying the differential paths. The S-box of Whirlpool is not as balanced as the one in AES. For a non-zero difference pair, if there is a conforming value, we call it a match. The matching probability of Whirlpool S-box is lower than the one in AES.

The property of the Whirlpool S-box results in big differences between theory and practice. Theoretically, one valid key pair is enough to find a match of the

MitM phase in the data processing part. But, practically, we tried 109 different valid key pairs to find a solution for the data part. In every matching step, we have to try more times to find a match. So the complexity to find one solution is increased. However, the expected number of solutions for a random difference pair does not depend on DDT. Hence, the total complexity is not increased if we need many solutions of the inbound phase. As a result, the complexity of our 7-round and 8-round attacks is not affected, since the complexity mainly depends on a lot of iterations in the outbound phase. The theoretical complexity of our inbound phase for both key and data (to find a 4-round collision) is 2^8. Because we only need one solution from the inbound phase, experiments show that the practical complexity for the inbound phase is increased by 2^4 to 2^7 times.

6 Concluding Remarks

In this paper, we improved the attacks on Whirlpool with respect to the fundamental security notions. For the preimage attack, the number of attacked rounds was extended by the guess-and-determine technique. Moreover, the complexity was improved by exploiting the freedom in the key value. For the collision attack, the difference was introduced in the key value, and a high probability differential path was constructed by canceling the difference in the data with the difference in the key. These results show several risks of using similar diffusions for the key and data. These also indicate that Whirlpool is still secure in practice.

References

1. Wang, X., Yu, H.: How to break MD5 and other hash functions. In Cramer, R., ed.: EUROCRYPT 2005. LNCS, Vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
2. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In Shoup, V., ed.: CRYPTO 2005. LNCS, Vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
3. U.S. Department of Commerce, National Institute of Standards and Technology: Federal Register /Vol. 72, No. 212/Friday, November 2, 2007/Notices. (2007)
4. Rijmen, V., Barreto, P.S.L.M.: The WHIRLPOOL hashing function. Submitted to NISSIE (2000)
5. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC Press (1997)
6. International Organization for Standardization: ISO/IEC 10118-3:2004, Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions. (2004)
7. New European Schemes for Signatures, Integrity, and Encryption(NESSIE): NESSIE PROJECT ANNOUNCES FINAL SELECTION OF CRYPTO ALGORITHMS. (2003)
8. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In Dunkelman, O., ed.: FSE 2009. LNCS, Vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
9. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved cryptanalysis of the reduced Grøstl compression function, ECHO permutation and AES block cipher. In Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R., eds.: Selected Areas in Cryptography 2009. LNCS, Vol. 5867, pp. 16–35. Springer, Heidelberg (2009)

10. Gilbert, H., Peyrin, T.: Super-Sbox cryptanalysis: Improved attacks for AES-like permutations. In Hong, S., Iwata, T., eds.: FSE 2010. LNCS, Vol. 6147, pp. 365–383. Springer, Heidelberg (2010)
11. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound distinguishers: Results on the full Whirlpool compression function. In Matsui, M., ed.: ASIACRYPT 2009. LNCS, Vol. 5912, pp. 126–143. Springer, Heidelberg (2009)
12. Matusiewicz, K., Naya-Plasencia, M., Nikolić, I., Sasaki, Y., Schläffer, M.: Rebound attack on the full LANE compression function. In Matsui, M., ed.: ASIACRYPT 2009. LNCS, Vol. 5912, pp. 106–125. Springer, Heidelberg (2009)
13. Wu, S., Feng, D., Wu, W., Su, B.: Hyper-Sbox view of AES-like permutations: A generalized distinguisher. In Lai, X., Yung, M., Lin, D., eds.: Inscrypt 2010. LNCS, Vol. 6584, pp. 155–168. Springer, Heidelberg (2011)
14. Sasaki, Y., Li, Y., Wang, L., Sakiyama, K., Ohta, K.: Non-full-active super-sbox analysis: Applications to ECHO and Grøstl. In Abe, M., ed.: ASIACRYPT 2010. LNCS, Vol. 6477, pp. 38–55. Springer, Heidelberg (2010)
15. Naya-Plasencia, M.: How to improve rebound attacks. In Rogaway, P., ed.: CRYPTO 2011. LNCS, Vol. 6841. pp. 188–205. Springer, Heidelberg (2011)
16. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved rebound attack on the finalist Grøstl. In Canteaut, A., ed.: FSE 2012. LNCS, Springer, Heidelberg (2012) To appear.
17. Jean, J., Fouque, P.A.: Practical near-collisions and collisions on round-reduced ECHO-256 compression function. In Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R., eds.: FSE 2011. LNCS, Vol. 5867, pp. 107–127. Springer, Heidelberg (2011)
18. Wu, S., Feng, D., Wu, W.: Practical rebound attack on 12-round Cheetah-256. In Lee, D., Hong, S., eds.: ICISC 2009. LNCS, Vol. 5984, pp. 300–314. Springer, Heidelberg (2010)
19. Biryukov, A., Khovratovich, D., Nikolić, I.: Distinguisher and related-key attack on the full AES-256. In Halevi, S., ed.: CRYPTO 2009. LNCS, Vol. 5677, pp. 231–249. Springer, Heidelberg (2009)
20. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Rebound attack on the reduced Grøstl hash function. In Pieprzyk, J., ed.: CT-RSA 2010. LNCS, Vol. 5985, pp. 350–365. Springer, Heidelberg (2010)
21. Peyrin, T.: Improved differential attacks for ECHO and Grøstl. In Rabin, T., ed.: CRYPTO 2010. LNCS, Vol. 6223, pp. 370–392. Springer, Heidelberg (2010)
22. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In Avanzi, R.M., Keliher, L., Sica, F., eds.: Selected Areas in Cryptography 2008. LNCS, Vol. 5381, pp. 103–119. Springer, Heidelberg (2009)
23. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced Meet-in-the-Middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In Abe, M., ed.: ASIACRYPT 2010. LNCS, Vol. 6477, pp. 56–75. Springer, Heidelberg (2010)
24. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In Joux, A., ed.: EUROCRYPT 2009. LNCS, Vol. 5479, pp. 134–152. Springer, Heidelberg (2009)
25. Sasaki, Y.: Meet-in-the-middle preimage attack on AES hashing modes and an application to Whirlpool. In Joux, A., ed.: FSE 2011. LNCS, Vol. 6733, pp. 378–396. Springer, Heidelberg (2011)
26. Wu, S., Feng, D., Wu, W., Guo, J., Dong, L., Zou, J.: (Pseudo) preimage attack on reduced-round Grøstl hash function and others. In Canteaut, A., ed.: FSE 2012. LNCS, Springer, Heidelberg (2012) To appear.

27. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In Lee, D.H., Wang, X., eds.: ASIACRYPT 2011. LNCS, Vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
28. Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: The rebound attack and subspace distinguishers: Application to Whirlpool. Cryptology ePrint Archive, Report 2010/198 (2010) `http://eprint.iacr.org/2010/198`.
29. Aoki, K., Sasaki, Y.: Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In Halevi, S., ed.: CRYPTO 2009. LNCS, Vol. 5677, pp. 70–89. of LNCS, Springer, Heidelberg (2009)
30. Morita, H., Ohta, K., Miyaguchi, S.: A switching closure test to analyze cryptosystems. In Feigenbaum, J., ed.: CRYPTO 1991. LNCS, Vol. 576, pp. 183–193. Springer, Heidelberg (1992)
31. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In Cramer, R., ed.: EUROCRYPT 2005. LNCS, Vol. 3494, pp. 474–490. Springer, Heidelberg (2005)

A Chunk Separation for Preimage Attack

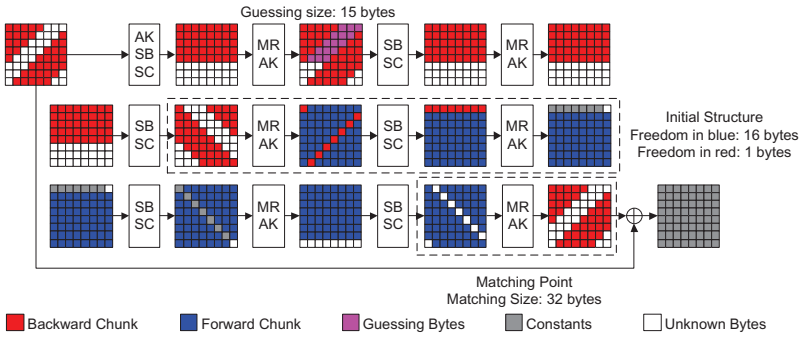

Fig. 6. Chunk separation $(b, r, w, g) = (7, 5, 1, 5)$ for the memoryless second preimage attack on 6-round hash function

B On the Message Length Padding

In order to convert the attack on the compression function into an attack on the hash function, we need to deal with the message padding first. For the last message block, the lower half are the message length in binary expression. Here, we use L to denote the message length. If the last bit of the fourth row $\#M[row(3)]$ in the message block $\#M$ is 1, we can obtain that $L \equiv 255 \ mod \ 512$. So the last 9 bits of $\#M[row(7)]$ should be 011111111. If the last two bits of $\#M[row(3)]$ are 10, we know that $L \equiv 254 \ mod \ 512$. So the last 9 bits of $\#M[row(7)]$ should be 011111110. So, we can calculate the probability that a

random message block is a valid block with correct padding by adding up all the probability for different suffix of the upper half of the message block:

$$\sum_{i=1}^{256} 2^{-(9+i)} \approx 2^{-9}.$$

C Examples of Data Generated in the Experiments

Table 3. Collision for 4- and Near-Collision for 7-round Compression Function

Chaining Value	Message Block	Difference in Round 4	Difference in Round 7
7B A9 ED 44 E2 7A FE 2B	71 68 DA 09 4F B6 D0 B2		
FD 53 A5 EE 97 A6 72 F3	97 93 7B 9A FF 6C 41 BB		
FD 4E EF 3B F1 65 E8 64	C0 BD AF 12 72 FD A4 17		
B4 D0 84 01 F9 75 18 57	30 82 86 46 FF 83 47 D0		
02 BB 5E 6F CA A3 E5 76	99 D8 0E 3C 03 C5 8E 06	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
86 F2 38 76 2B 9B 7F 58	EE 78 EF 01 74 65 7D AF	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0E 80 06 67 58 65 90 0A	84 03 52 1B C3 F7 F2 BC	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
DD A7 64 C7 3A 6F ED AC	95 C9 BD 81 20 26 12 57	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
9A CB 57 95 CE 6B F7 17	90 0A 60 D8 63 A7 D9 8E	00 00 00 00 00 00 00 00	42 6E 9E 0D 4F 4F 21 4F
D9 35 99 D8 94 7D 35 F4	B3 F5 47 AC FC B7 06 BC	00 00 00 00 00 00 00 00	7D CF 94 FA B2 7D 7D E9
5E 1D 25 7F 45 10 E2 B2	63 EE 65 56 C6 88 AE C1	00 00 00 00 00 00 00 00	FA B0 E9 4A 7D 59 B0 B0
00 0F B4 6A A0 10 89 A0	84 5D B6 2D A6 E6 D6 27	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
A6 16 D6 D4 6B 37 75 D4	3D 75 86 87 A2 51 1E A4		
B7 F8 03 25 F8 0D 9D 9D	DF 72 D4 52 A7 F3 9F 6A		
E8 1D 70 13 40 0E 47 94	62 9E 24 6F DB 9C 25 22		
52 58 53 E9 D0 C2 B5 0E	1A 36 8A AF CA 8B 4A F5		