Cryptanalysis of Reduced-Round `Whirlwind` (Full Version)*

Bingke Ma[1,2,3], Bao Li[1,2], Ronglin Hao[1,2,4], and Xiaoqian Li[1,2,3]

[1]State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, 100093, China
[2]Data Assurance and Communication Security Research Center,
Chinese Academy of Sciences, Beijing, 100093, China
{bkma,lb,xqli}@is.ac.cn
[3]University of Chinese Academy of Sciences, Beijing, China
[4]Department of Electronic Engineering and Information Science,
University of Science and Technology of China, Hefei, 230027, China
haorl@mail.ustc.edu.cn

Abstract. The `Whirlwind` hash function, which outputs a 512-bit digest, was designed by Barreto *et al.* and published by *Design, Codes and Cryptography* in 2010. In this paper, we provide a thorough cryptanalysis on `Whirlwind`. Firstly, we focus on security properties at the hash function level by presenting (second) preimage, collision and distinguishing attacks on reduced-round `Whirlwind`. In order to launch the preimage attack, we have to slightly tweak the original Meet-in-the-Middle preimage attack framework on `AES`-like compression functions by partially fixing the values of the state. Based on this slightly tweaked framework, we are able to construct several new and interesting preimage attacks on reduced-round `Whirlpool` and `AES` hashing modes as well. Secondly, we investigate security properties of the reduced-round components of `Whirlwind`, including semi-free-start and free-start (near) collision attacks on the compression function, and a limited-birthday distinguisher on the inner permutation. As far as we know, our results are currently the best cryptanalysis on `Whirlwind`.

Keywords: cryptanalysis, hash function, `Whirlwind`, `Whirlpool`, `AES`, PGV

1 Introduction

Cryptographic hash functions are widely acknowledged as the Swiss Army Knife of modern cryptography thanks to its versatility and numerous applications. A secure hash function should at least preserve three basic security properties, namely, collision resistance, preimage resistance and second preimage resistance. Several other security properties are also concerned, such as the limited-birthday distinguisher on hash functions [14].

In order to hash an input message, many state-of-the-art hash functions divide the message into many blocks and process each block iteratively with the inner components. The Merkle-Damgård construction [27,10] is one of the most classical constructions following this principle, and the underlying component which processes the message block in the Merkle-Damgård construction is called the compression function. Security properties of the compression function often have great impacts on the hash function, thus cryptanalysis of the compression function is important and meaningful. For the compression function, there are security notions such as semi-free-start (near) collision, free-start (near) collision [1] and (pseudo) preimage. Most of the compression functions used in practice are constructed with cryptographic permutations adopting proper modes of operation such as the PGV modes [31]. Security evaluations of the underlying permutation are quite necessary and helpful in order to know the potential weaknesses, the security margins which are crucial especially in the design of new primitives and the validity of the security proofs. A noted example is the limited-birthday distinguisher on permutations [12] which has been applied to many symmetric primitives, including [12,15,16], to name but a few.

* This article is the full version of the paper published at ACISP 2015.

[1] A semi-free-start collision attack aims to find two distinct messages (M, M') and an identical chaining value h_0 satisfying $CF(h_0, M) = CF(h_0, M')$. A free-start collision attack searches for two messages (M, M') and two chaining values (h_0, h'_0) satisfying $CF(h_0, M) = CF(h'_0, M')$, and $h_0 = h'_0$ and $M_0 = M'_0$ do not hold concurrently.

The Whirlwind hash function [6] was designed by Barreto *et al.* and published by *Design, Codes and Cryptography* in 2010. It takes messages up to 2^{256} bits as inputs and outputs a 512-bit digest. At the high level construction, Whirlwind adopts a Merkle-Damgård construction with a final output transformation. At the compression function level, it employs an AES-like [28,9] design embedded in a Davies-Meyer-like mode. The state of the compression function can be seen as an 8×8 array of 16-bit elements, while the left (resp. right) half of the input state array is the 512-bit chaining value (resp. message block). Besides the high level structure of Whirlwind, it has some unique underlying components, *i.e.*, the interesting 16-bit SBox and the linear layer which are friendly to both software and hardware implementations. Actually some of its novelty design philosophies have influences on several recent works, *e.g.*, its linear layer design based on the strategy of subfield construction has been further investigated in [1] and [19]. Although Whirlwind contributes many heuristic and inspiring ideas to the design of symmetric cryptographic primitives, its concrete security properties are less evaluated in literature. Besides the analyses provided by the designers at the compression function level in [6], Riham *et al.* recently presented second preimage attacks on 5 and 6 rounds of Whirlwind [2].

Our Contributions. We present a thorough cryptanalysis on Whirlwind. Firstly, we focus on security properties at the hash function level by presenting several attacks on reduced-round versions of the 12-round Whirlwind hash function, including improved second preimage attack reduced to 6 rounds, the preimage attack reduced to 4 rounds, the collision attack reduced to 5 rounds, and the limited-birthday distinguisher reduced to 6 rounds. Secondly, security properties of the reduced-round inner components of Whirlwind are evaluated. More precisely, we launch semi-free-start collision and free-start near collision attacks on 6-round and 7-round Whirlwind compression function, and build a multiple limited-birthday distinguisher [17] on the inner permutation reduced to 9 rounds. Our attacks show that Whirlwind offers a quite comfortable security margin with respect to existing attacks. The results are summarized in Table 1.

In order to launch the preimage attacks on 4-round Whirlwind, we need to slightly tweak the original Meet-in-the-Middle preimage attack framework on AES-like compression functions by partially fixing the values of the state. Although this small tweak is rather simple, it has several further applications. For instance, we present two new preimage attacks on 5 and 6 rounds of Whirlpool [7,13], which significantly reduce the preimage lengths compared with previous attacks [35] (though our attacks require higher time complexities). We also show the first preimage attacks on several PGV hashing modes instantiated with 6-round AES at the hash function level.

Structure. The remainder of this paper is organized as follows: Section 2 provides a brief description of the Whirlwind hash function, and the techniques utilized in this paper. We present the (second) preimage attacks and the collision attacks on reduced-round Whirlwind in Section 3 and Section 4 respectively, along with the illustrations of the part-fixed Meet-in-the-Middle preimage attack and the part-fixed SuperSBox technique. The limited-birthday distinguisher on 6-round Whirlwind is elaborated in Section 5. We analyze the reduced-round components of Whirlwind in Section 6. Section 7 concludes and summarizes the paper. More details and applications of the part-fixed Meet-in-the-Middle preimage attack are provided in Appendix A.

2 Preliminaries

2.1 The Whirlwind Hash Function

The Whirlwind hash function takes any message up to 2^{256} bits as input, and outputs a 512-bit digest. It adopts the Merkle-Damgård construction with an output transformation, and the padding algorithm of Whirlwind is MD-strengthening with a 256-bit length padding L.

The compression function of Whirlwind is an AES-like design and has a 1024-bit state which can be represented by an 8×8 array of 16-bit elements. As depicted in Fig. 1, the compression function CF takes a 512-bit chaining value h^t and a 512-bit message block M^t as inputs, and performs the round function 12 times to derive the 512-bit output chaining value h^{t+1}, namely, $h^{t+1} = CF(h^t, M^t)$. Each round function consists of four maps, which are as follows:

Table 1. Summary of results on `Whirlwind` †

Target	Attack Type	Rounds	Time	Memory	Ideal	Reference
Hash Function (12 rounds)	Preimage	4	2^{496}	2^{255}	2^{512}	Section 3
			2^{497}	2^{208}		
	Second Preimage	5	2^{449}	2^{128}	2^{512}	[2]
		6	2^{505}	2^{112}	2^{512}	[2]
		6	2^{497}	2^{128}	2^{512}	Section 3
	Collision	5	2^{240}	2^{128}	2^{256}	Section 4
		4	2^{128}	2^{128}		
	LBD Distinguisher ♢	6	2^{353}	2^{160}	2^{449}	Section 5
Compression Function (12 rounds)	SFS. Near Collision ♡	5.5	2^{176}	2^{32}	2^{224}	[6]
	SFS. Collision ♡	4.5	2^{64}	2^{32}	2^{256}	[6]
		6	2^{128}	2^{128}	2^{256}	Section 6
	FS. Near Collision ♣	7	2^{224}	2^{128}	2^{448}	Section 6
Inner Permutation (12 rounds)	MLBD Distinguisher ♠	9	2^{730}	2^{128}	2^{763}	Section 6

† : We do not consider the attacks starting from a middle round.
♢: Limited-birthday distinguisher on the hash function.
♡: Semi-free-start (near) collision on the compression function.
♣: Free-start near collision (896 colliding bits out of 1024 bits) on the compression function with no truncation.
♠: Multiple limited-birthday distinguisher on the permutation.

- **SubBytes(γ):** process each cell of the state with the 16-bit SBox.
- **MixRows(θ):** multiply each row of the state array by a matrix.
- **Transposition(τ):** transpose the k-th column to be the k-th row for $k = 0, 1, 2, ..., 7$, *i.e.*, transposition of the state array.
- **AddRoundConstant(σ^r):** XOR the 1024-bit constant of the r-th round to the state array.

The designers also provide an alternative understanding for the concatenation of θ and τ, namely, $\tau \circ \theta$ as:

- θ_R: multiply each row of the state array by a matrix in even rounds.
- θ_C: multiply each column of the state array by a matrix in odd rounds.

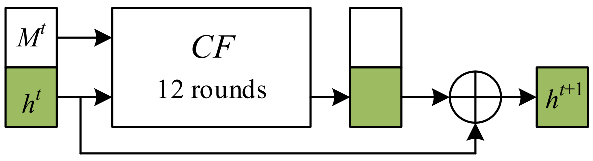


Fig. 1. The compression function of `Whirlwind`

For the remainder of this paper, we will adopt this alternative understanding of the round function. Now we give detailed algorithm of the compression function CF.

1. Initialize the state array S^0 with the input chaining value h^t and the input message block M^t:

$$\begin{cases} S^0_{i,j} = h^t_{i,j} \\ S^0_{i+4,j} = M^t_{i,j} \end{cases}, \text{ for } 0 \le i < 4,\ 0 \le j < 8.$$

2. Apply 12 iterations of the round function to the initial state S^0, and derive the final state S^{12}:

$$S^{k+1} = \begin{cases} (\sigma^k \circ \theta_R \circ \gamma)(S^k), & k \text{ is even,} \\ (\sigma^k \circ \theta_C \circ \gamma)(S^k), & k \text{ is odd,} \end{cases} \text{ for } 0 \le k < 12.$$

3. Truncate the final state S^{12} and perform the feed-forward to derive the output chaining value h^{t+1}:

$$h_{i,j}^{t+1} = h_{i,j}^{t} \oplus S_{i,j}^{12}, \text{ for } 0 \leq i < 4,\ 0 \leq j < 8.$$

After all message blocks have been processed with the compression function iteratively, the output transformation is performed to avoid trivial length extension attacks. Suppose the output chaining value of the last message block is $h^{\texttt{LAST}}$, then the final 512-bit digest h^X is computed as:

$$h^X = CF(h^{\texttt{LAST}}, 0),$$

with the message block equal to the 8×4 null matrix.

2.2 The Meet-in-the-Middle Preimage Attack

The Meet-in-the-Middle (MitM) preimage attack was first introduced by Aoki and Sasaki in their preimage attacks against `MD4` and 63-step `MD5` [5]. The basic idea of this technique, which is known as *splice-and-cut*, aims to divide the target cipher into two sub-ciphers which can be computed independently. Due to the feed-forward operations in hash functions, the MitM attack can then be applied. Several advanced techniques to further improve the basic attack are developed, such as partial matching [5], partial fixing [5], initial structure [34], indirect partial matching [3], bicliques [20] and differential MitM attack [21].

In [33], Sasaki proposed the first MitM preimage attack on `AES`-like compression functions. Two main techniques were presented, namely, initial structure in an `AES`-like compression function and indirect partial matching through an MixColumn layer. This work was later improved by Wu *et al.* in [36]. Thanks to the delicate descriptions of the MitM preimage attack framework on `AES`-like compression functions presented in [36], the chunk separations can be easily represented by introducing several essential integer parameters, and the best attack parameters can be easily derived through an exhaustive search. In [35], Sasaki *et al.* introduced the guess-and-determine approach to extend the basic attack by one more round.

2.3 The Rebound Attack and the SuperSBox Technique

The rebound attack was first introduced by Mendel *et al.* in their attacks against reduced-round `Whirlpool` and `Grøstl` [24]. It aims to find a pair of inputs satisfying some unique properties for some specific (truncated) differential characteristics faster than the ideal case. There are two main procedures of the rebound attack, namely, the inbound phase and the outbound phase. Let $\mathcal{C}$ and $\mathcal{D}$ denote the certain non-full-active differential forms (normally with very few active cells) in the inbound phase, and $\mathcal{F}$ denote the full-active differential form, then the differentials in the inbound phase of the basic rebound attack can be denoted as $\mathcal{C} \rightarrow \mathcal{F} \leftarrow \mathcal{D}$. The available freedom degrees are used to connect these middle rounds with relatively small amount of computations using the match-in-the-middle technique. In the outbound phase, solutions from the inbound phase are propagated both forwards and backwards to connect the differentials in both directions.

The SuperSBox technique for the rebound attack was independently introduced in [12,22]. This technique exploits the fact that the non-linear layers between two rounds, *i.e.*, `SB, SR, MC, AC, SB` can be computed for each column independently. Therefore, by constructing the look-up tables for each individual column, the SuperSBox technique is able to cover one more full-active state in the inbound phase, namely, $\mathcal{C} \rightarrow \mathcal{F} \leftrightarrow \mathcal{F} \leftarrow \mathcal{D}$. The look-up tables built are hence called the SuperSBoxes. In [15,16], a further extension of the rebound attack and the SuperSBox technique, which can cover three full-active states $\mathcal{C} \rightarrow \mathcal{F} \leftrightarrow \mathcal{F} \leftrightarrow \mathcal{F} \leftarrow \mathcal{D}$ in the inbound phase, was presented. The core of this technique is the advanced techniques utilized to merge solutions generated from the forward and backward SuperSBoxes in more efficient manners.

3 (Second) Preimage Attacks on Reduced-Round `Whirlwind`

Firstly, we present the second preimage attack on 6-round `Whirlwind` with the aid of the plain MitM preimage attack on `AES`-like compression functions. Then the preimage attack on 4-round `Whirlwind` is illustrated. The whole preimage attack consists of four steps. In some steps of the attack partial values of the initial state are fixed, and the plain MitM preimage attack framework on `AES`-like compression functions cannot be directly applied under this scenario. Fortunately, we slightly tweak the original attack framework by partially fixing the values of the state, and make the preimage attack feasible. Despite the simplicity of this small tweak, its effectiveness are further stated with several new and interesting preimage attacks on reduced-round `Whirlpool` and `AES` hashing modes.

3.1 Second Preimage Attack on 6-Round `Whirlwind`

Before describing all attacks, we stress that a single compression function computation (resp. the state bit size of the compression function) is used as the basic unit of time (resp. memory) throughout this paper. The second preimage attack on 6-round `Whirlwind` is less complex and more traditional. As depicted in Fig. 2, suppose we expect to find a second preimage of $M_1||M_2||...||M_s$, the attack procedures can be divided into two phases. In the first phase, we find sufficient pseudo preimages and derive a set of h'_{t+1}s for a chaining value in the middle h_{t+2}. In the second phase, we launch the traditional MitM method [26, Fact 9.99] to convert the pseudo preimages into a second preimage. More precisely, we choose random values of M_t to compute a set of h_{t+1}s from h_t, and search for a match of h_{t+1} in the set of h'_{t+1}s. A second preimage is successfully constructed if a match is derived.

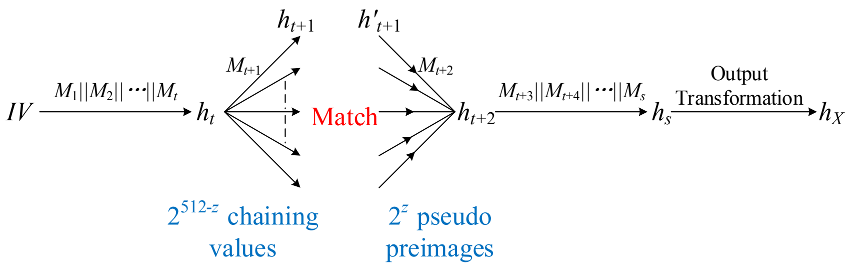


Fig. 2. Second preimage attack on 6-round `Whirlwind`

The main task lies in the first phase where sufficient pseudo preimages have to be generated. This can be done by utilizing the MitM preimage attack framework on `AES`-like compression functions which have been extensively investigated in [33,36,35]. Without loss of generality, we introduce the plain attack framework with the MitM preimage attack on 6-round `Whirlwind` compression function, and Fig. 3 depicts the basic chunk separation of this attack. Before more details are illustrated, we define several notations which are used throughout this paper.

n	Bit size of the digest.
N_c	Bit size of the cell.
N_t	Number of columns (or rows) in the state.
b	Number of blue columns (or rows) in the initial structure, and
r	Number of red columns (or rows) in the initial structure.
c	Number of constant cells in each column (or row) corresponding to the red column (or row) in the initial structure.
g	Number of guessed rows (or columns, in purple) in the backward (or forward) computation.
D_b	Freedom degrees of the blue chunk in bits.
D_r	Freedom degrees of the red chunk in bits.
D_g	Bit size of the guessed cells.
D_m	Bit size of the match point.
T	Time complexity of the attack.
M	Memory requirement of the attack.

The attack procedures of the first phase can be further divided into five steps which are elaborated as follows, and the interested readers are referred to [33,36,35] for more detailed descriptions.

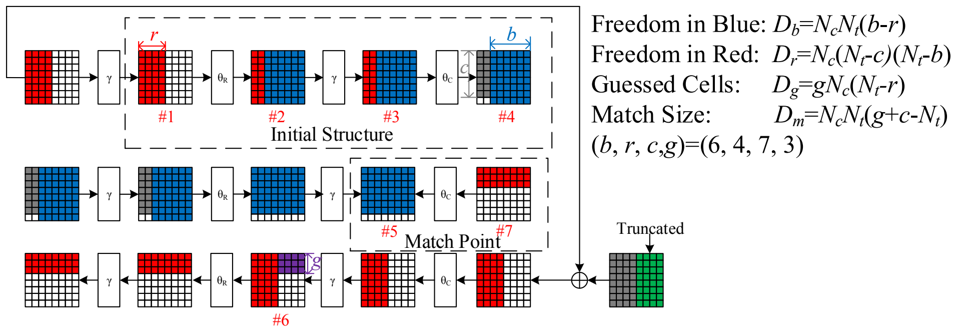
Step 1. Initial Structure. The purpose of the initial structure is to use several consecutive rounds as the starting point and divide the target cipher into two sub-ciphers which can be independently computed. For this purpose, as shown in Fig. 3, we choose random values for the constants (in grey) which are used in the linear transformations between states #1 $\leftrightarrow$ #2, and states #3 $\leftrightarrow$ #4. Following the linear relations of the θ_C and θ_R operations, compute the values in the initial structure for the forward chunk (in blue) which has D_b freedom degrees and the backward chunk (in red) which has D_r freedom degrees. After this step, the compression function is divided into two independent chunks thanks to the initial structure.

Step 2. Forward Computation. For all the blue and grey cells at state #4, the forward chunk can be computed forwards independently until state #5.

Step 3. Backward Computation. For all the red cells at #1, the backward chunk can be computed backwards independently until state #7. In order to proceed the backward computation by one more round, the guess-and-determine strategy is applied at state #6 by guessing the value of g rows.

Step 4. Indirect-Partial-Matching through the MixRow Layer. We have partial information of the red and blue cells from both directions, and linear relations of the θ_C operation can be further exploited at the match point to perform the indirect-partial-matching between states #5 $\leftrightarrow$ #7.

Step 5. Recheck. Check whether the guessed cells of the partial match derived in step 4 are guessed correctly. If so, check whether the partial match is also a full match. Repeat the above steps 1-5 until a preimage is found.



r
b
c
g
γ
θR
θC
#1
#2
#3
#4
Initial Structure
#5
#7
Match Point
#6
Truncated
Freedom in Blue: D_b=$N_cN_t(b$-$r)$
Freedom in Red: D_r=$N_c(N_t$-$c)(N_t$-$b)$
Guessed Cells: D_g=$gN_c(N_t$-$r)$
Match Size: D_m=$N_cN_t(g$+c-$N_t)$
(b, r, c, g)=(6, 4, 7, 3)

Fig. 3. Chunk separation for the 6-round pseudo preimage attack

Deriving the Attack Parameters. As studied in [36,35], the attack parameters, *e.g.*, (D_b, D_r, D_g, D_m) can be easily represented with several predefined integer parameters (b, r, c, g) as shown in Fig. 3. The attack complexities can be denoted as follows:

$$\begin{aligned} T &= 2^n(2^{-D_r} + 2^{D_g-D_b} + 2^{D_g-D_m}), \\ M &= min\{2^{D_r+D_g}, 2^{D_b}\}. \end{aligned} \tag{1}$$

We can derive the optimal attack parameters by enumerating all possible values of (b, r, c, g).

Complexity Analysis. Fig. 3 gives the optimal chunk separation for the pseudo preimage attack on the compression function, and it requires 2^{480} time and 2^{128} memory. In order to balance the overall complexities, we generate 2^{16} pseudo preimages in the first phase. In the second phase of the attack, we compute 2^{496} h_{t+1}s, and find a match with the pseudo preimages generated in the first phase with a high probability. Finally, it would require 2^{497} time and 2^{128} memory to launch a second preimage attack on 6-round `Whirlwind`.

3.2 Overview of the 4-Round Preimage Attack

Due to the strong impacts introduced by the truncation operation and the partially fixed state of `Whirlwind` (especially by the output transformation), our preimage attack can only work up to 4 rounds. The gap between the attacked rounds of preimage and second preimage attacks demonstrates that the adoptions of the truncation operation and the partially fixed state do strengthen `Whirlwind` in terms of preimage resistance.

Before describing the details, we give a brief overview of the 4-round preimage attack. As depicted in Fig. 4, the three-block preimage attack on 4-round `Whirlwind` consists of four steps. In the first step, the attack is carried out on the output transformation, and the given challenge digest h_X is inverted. In the second step, we invert the chaining value h_3, and derive the value of the last message block M_3 which contains the padding part. In the third step, we generate several pseudo preimages h_1', and sufficient values for the second last message block M_2 are obtained simultaneously. In the last step, the traditional MitM method [26, Fact 9.99] which converts pseudo preimages in to a preimage is launched, and we expect to find a match between h_1s and h_1's. If all steps succeed, the three-block message $M_1||M_2||M_3$ is a preimage for 4-round `Whirlwind`.

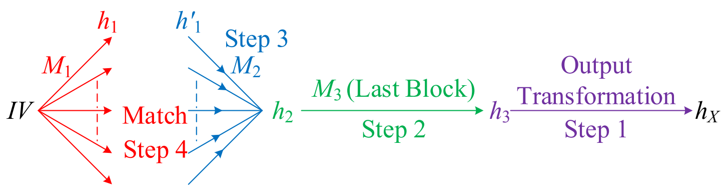


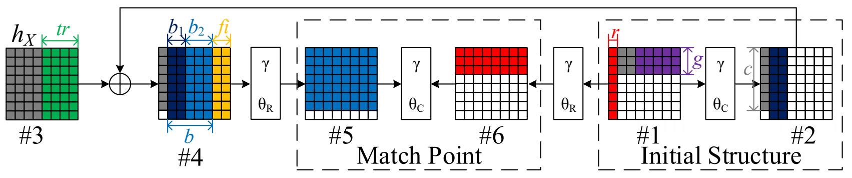
Fig. 4. Overview of the preimage attack on 4-round `Whirlwind`

3.3 The Slightly Tweaked MitM Preimage Attack on `AES`-like Compression Functions

The main bottlenecks of the preimage attack lie in step 1 and step 2 where partial values of the initial states are already fixed before launching the attack. More precisely, in step 1 the rightmost 4 columns of the state are fixed to 0 in the output transformation. As in step 2, since the preimage we try to build consists of 3 blocks, and the last block contains 255 message bits and 257 padding bits according to the MD-strengthening padding algorithm, the rightmost 2 columns of the state are fixed to the value $512 + 512 + 255 = 1279$. The plain MitM preimage attack framework on `AES`-like compression functions described in Section 3.1 seems inapplicable in this scenario. Luckily, we can apply a small and simple tweak which partially fixes the values of the state, and thus make the original attack framework feasible.

The Tweaked Attack Framework with Truncation. There are two main differences in the tweaked attack framework compared to the plain framework, namely, the partially fixed input state before the first round and the truncation operation carried after the last round. Also notice that the first and the last rounds are connected by the feed-forward operation, thus the initial structure in the tweaked framework needs to be located carefully between the first and the last rounds. Without loss of generality, we use the attack on 4-round `Whirlwind` compression function as an instance to elaborate the tweaked framework, and Fig. 5 depicts the basic chunk separation of this attack. Following the definitions in Section 3.1, several additional notations need to be specified before providing more details. Since the half truncation is performed in `Whirlwind`, we will describe the slightly tweaked attack framework under the setting that the output chaining value is truncated in the following part.

tr	Number of columns (or rows) truncated in the output chaining value.
fi	Number of columns (or rows) fixed in the initial state before the attack.
b_1	Number of blue columns (or rows) with partial freedom degrees in the initial structure.
b_2	Number of blue columns (or rows) with full freedom degrees in the initial structure, we have$b = b_1 + b_2$.
D_{b_1}	Freedom degrees of the blue chunk with partial freedom degrees in bits.
D_{b_2}	Freedom degrees of the blue chunk with full freedom degrees in bits, we have $D_b = D_{b_1} + D_{b_2}$.



Freedom in Blue 1: $D_b = N_c b_1 (N_t - g)$ Freedom in Blue 2: $D_b = N_c b_2 N_t$
Freedom in Blue: $D_b = N_c[b_1(N_t - g) + b_2 N_t]$ Freedom in Red: $D_r = r N_c (N_t - c)$
Guessed Cells: $D_g = g N_c (b_2 + fi)$ Match Size: $D_m = N_c N_t (g + c - N_t)$
Constraints: $b_1 + b_2 + r + fi = N_t$ $b_1 + b_2 = b$ $b_1 + r \leq N_t - tr$
$(tr, fi) = (4, 2)$, $(b_1, b_2, b, r, c, g) = (2, 3, 5, 1, 7, 3)$

Fig. 5. The tweaked MitM preimage attack framework with truncation

For the sake of simplicity, we only consider the cases where full columns (or rows) are fixed (resp. truncated) in the initial state (resp. the output chaining value). Given a specific attack target, the values of (fi, tr) are prefixed before the attack, and the values of (b_1, b_2, r, c, g) can be chosen by the attacker in order to derive optimal attacks. The attack procedures are as follows:

Step 1. Initialization. Set the fi yellow columns in the initial state to the prefixed values, and denote the target digest as h_X.

Step 2. Initial Structure. Randomly choose values for the constants which are denoted in grey in the initial structure. According to the prefixed values in step 1 and the chosen constants in step 2, compute all values for the blue and red cells in the initial structure. After this step, the compression function is divided into two independent chunks.

Step 3. Backward Computation. In order to perform the backward computation, we have to apply the guess-and-determine strategy by guessing g rows which are denoted in purple at state #1. For all values of the red and guessed purple cells, compute backwards to the matching point at state #6 and store all partial matching values in a sorted table TL (*e.g.* hash table).

Step 4. Forward Computation. The blue cells in the forward computation can be further classified into two categories. For the b_1 columns, their values are constrained by the constant values in the initial structure, and there are overall $2^{D_{b_1}}$ values for these blue cells. Due to the truncation, we have to ensure $r + b_1 \leq N_t - tr$. For the b_2 columns, they can take all $2^{D_{b_2}}$ possible values of these blue cells. Combining these two categories, the freedom degrees in the blue cells are $2^{D_b} = 2^{D_{b_1}+D_{b_2}}$. For all values of the blue cells, compute forwards to the matching point at state #5.

Step 5. Indirect-Partial-Matching. Check whether the partial matching values derived in step 4 is also in the table TL built in step 3.

Step 6. Recheck. Check whether the guessed cells of the partial match derived in step 5 are guessed correctly. If so, check whether the partial match is also a full match. Repeat the above steps 2-6 until a preimage is found.

Complexity Analysis. Note that we can also swap the orders of step 3 and step 4, and the selection depends on which step requires less memory. Similar to the plain framework described in Section 3.1, the complexities of the tweaked attack are as follows:

$$\begin{aligned} T &= 2^n(2^{-D_r} + 2^{D_g - D_b} + 2^{D_g - D_m}), \\ M &= min\{2^{D_r + D_g}, 2^{D_b}\}. \end{aligned}$$

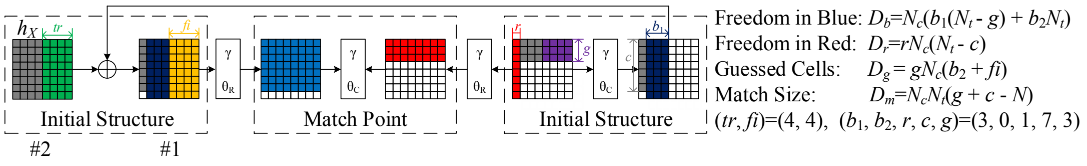
We do not provide detailed deductions, but it is convenient to check that the quartet (D_b, D_r, D_g, D_m) can be represented with the integer variables defined as given in Fig. 5. The optimal attack parameters can be easily achieved by an exhaustive search for all valid values of (b_1, b_2, r, c, g) with a prefixed (fi, tr).

The Tweaked Attack Framework without Truncation and Its Applications. Another interesting problem is to study the tweaked attack framework without truncation. We present detailed descriptions of this framework in Appendix A. As shown in the analyses, the attacked rounds can be further extended to 5 and even 6 rounds.

A natural scenario to adapt this tweaked framework without truncation is to generate short preimages for certain hash primitives by fixing partial values of the padding. The idea of generating short preimages by fixing the padding values is simple and widely applied to hash functions with the generalized Feistel structure and permutation-based message schedules such as `MD5` [34] and `SHA-0/1` [4], because the message blocks are commonly added into the round functions leaving enough freedom degrees to fix the padding message blocks. However, for most of the `AES`-like hash primitives, the input message block is immediately mixed into the input state, and thus restrain the ability to choose and fix proper values of the padding message block (normally the last one). In practice, many previous preimage attacks on `AES`-like hash functions do not impose any restraints on the padding values, and they largely rely on the expandable messages [18] which often result in preimages with very large and impractical lengths. After the investigations of this tweaked framework, we find out that for some practical `AES`-like hash constructions, there are freedom degrees to prefix partial values of the padding message block, and thus it is possilbe to derive preimages with relatively short (even practical) lengths for these constructions (though normally with higher time complexities). As direct applications, we launch two new preimage attacks against 5 and 6 rounds of `Whirlpool`, which reduce the preimage length from 2^{256} bits [35] to two blocks and 2^{128} bits respectively. Furthermore, under a very reasonable adoption of the padding algorithm, namely, MD-strengthening with the 64-bit length padding which is also used by `MD5` [32], `SHA-1` and `SHA-224/256` [29], we apply this technique to generate preimages with practical lengths for some PGV modes (including Matyas-Meyer-Oseas and Miyaguchi-Preneel) instantiated with 6-round `AES`, while no preimage attacks exist in literature for these constructions instantiated with (reduced-round) `AES` at the hash function level. The details of these proposed attacks are also provided in Appendix A, and the results are summarized in Table 4.

3.4 Details of the 4-Round Preimage Attack

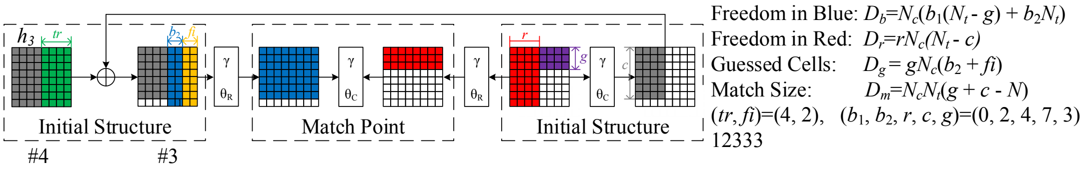
Step 1. In this step, we need to invert the output transformation and derive the chaining value h_3. Since the right half of the input state #1 is fixed to 0, and the right half of the output state #2 is truncated, we have to utilize the tweaked MitM preimage attack framework with truncation under the parameters $(fi, tr) = (4, 4)$. As shown in Fig. 6, we derive the optimal chunk separation with parameters $(b_1, b_2, r, c, g) = (3, 0, 1, 7, 3)$ after an exhaustive search. Finally, it requires 2^{496} time and 2^{208} memory to invert the output transformation of 4-round `Whirlwind`. Note that since we need to match a 512-bit target digest, but there are only 2^{512} freedom degrees in the input chaining value h_3, step 1 will succeed with probability $1 - e^{-1}$.



h_X
tr
fi
γ
θ_R
θ_C
r
g
c
b_1
Initial Structure
Match Point
Initial Structure
#2
#1
Freedom in Blue: $D_b = N_c(b_1(N_t - g) + b_2N_t)$
Freedom in Red: $D_r = rN_c(N_t - c)$
Guessed Cells: $D_g = gN_c(b_2 + fi)$
Match Size: $D_m = N_cN_t(g + c - N)$
$(tr, fi) = (4, 4)$, $(b_1, b_2, r, c, g) = (3, 0, 1, 7, 3)$

Fig. 6. Step 1. Invert the output transformation

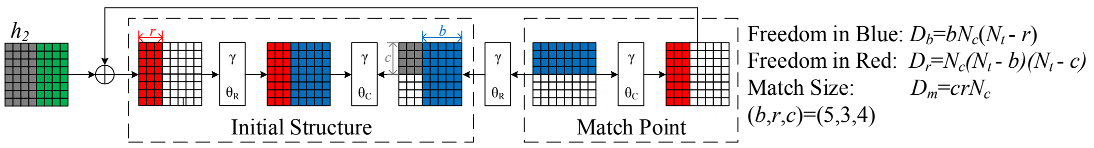
Step 2. In this step, given the value of h_3, we have to invert the compression function in order to derive the input chaining value h_2 and the last message block M_3 which contains the padding part. Since we aim to build a three-block preimage, according to the MD-strengthening padding algorithm, the last 257 bits of M_3 are fixed to the padding value. Consequently, the 2 rightmost columns of the initial state are prefixed as denoted with yellow in Fig. 7. Note that since the 256-th bit of M_3 has to be '`1`' due to the padding, we lose one bit freedom degree of the blue chunks. We utilize the tweaked MitM preimage attack framework with truncation under the parameters $(fi, tr) = (2, 4)$, and exhaustively search for all valid values of (b_1, b_2, r, c, g). The chunk separation with $(b_1, b_2, r, c, g) = (0, 2, 4, 7, 3)$ as shown in Fig. 7 is the best result achieved. As a result, step 2 requires 2^{449} time and 2^{255} memory. Since step 2 requires much less computations than step 1, one can also carry out step 2 with parameters $(b_1, b_2, r, c, g) = (3, 2, 1, 7, 3)$ to minimize the memory requirement. The corresponding complexities are $(T, M) = (2^{496}, 2^{208})$.



h_3
tr
b_2
fi
γ
θ_R
θ_C
r
g
c
Initial Structure
Match Point
Initial Structure
#4
#3
Freedom in Blue: $D_b = N_c(b_1(N_t - g) + b_2N_t)$
Freedom in Red: $D_r = rN_c(N_t - c)$
Guessed Cells: $D_g = gN_c(b_2 + fi)$
Match Size: $D_m = N_cN_t(g + c - N)$
$(tr, fi) = (4, 2)$, $(b_1, b_2, r, c, g) = (0, 2, 4, 7, 3)$
12333

Fig. 7. Step 2. Derive the last message block

Step 3. In this step, given the value of h_2, we have to invert the compression function in order to derive the value of the second message block M_2 and the input chaining value h_1. The only constraint of this step is the final truncation, thus the traditional MitM preimage attack can be launched. We omit these details. As depicted in Fig. 8, with parameters $(b, r, c) = (5, 3, 4)$, the attack is optimized with complexities $(T, M) = (2^{320}, 2^{192})$. Since we will perform the traditional MitM procedure [26, Fact 9.99] to convert the pseudo preimages into a preimage in step 4, we have to generate and store multiple say 2^x pairs of (h_1, M_2), which require 2^{320+x} time and 2^x memory.



h_2
r
c
b
γ
θ_R
θ_C
Initial Structure
Match Point
Freedom in Blue: $D_b = bN_c(N_t - r)$
Freedom in Red: $D_r = N_c(N_t - b)(N_t - c)$
Match Size: $D_m = crN_c$
$(b, r, c) = (5, 3, 4)$

Fig. 8. Step 3. Generate sufficient values for the second last message block

Step 4. In this step, we launch the traditional MitM method to connect the values of h_1' generated in step 3. More precisely, we choose random values of M_1 and compute the corresponding output chaining values h_1, then match it with the values of h_1' generated in step 3. After choosing 2^{512-x} distinct values of M_1, we expect to find a match and finally derive a preimage of 4-round `Whirlwind`. Since each h_1 can be checked on the fly, the memory requirement for step 4 is negligible. We choose $x = 96$ to minimize the overall time complexity for step 3 and step 4 which is $2^{512-96} + 2^{320+96} = 2^{417}$.
Complexity Analysis. Now to sum up, the time complexity is dominated by step 1, and the memory requirement is dominated by step 2. Finally, the complexities for the 4-round preimage attack are $(T, M) = (2^{496}, 2^{255})$. One can also minimize the memory requirement by adopting the alternative attack parameters provided in step 2, which results in complexities of $(T, M) = (2^{497}, 2^{208})$.

4 Collision Attacks on Reduced-Round `Whirlwind`

This section presents the collision attack on 4- and 5-round `Whirlwind` hash function. Due to the Davies-Meyer-like mode of `Whirlwind`, it is feasible to launch semi-free-start or free-start collision attacks on the reduced-round compression function with the rebound attack, but deriving a collision attack at the hash function level seems difficult. Fortunately, with the aid of the multi-block strategy, we are able to launch collision attacks on 4 and 5 rounds of the `Whirlwind` hash function.

4.1 Overview of the 5-Round Collision Attack

The previous collision attacks on `Grindahl` [30] and `Grøstl` [25] generate colliding messages which contain several consecutive message blocks. The general strategy in these attacks is after introducing the differences into the chaining values, the differences are gradually cancelled with the posterior message blocks, and eventually lead to collisions by eliminating the differences in the output chaining values. Our attack is based on a similar multi-step process which gradually introduces, restricts and finally cancels the differences. As shown in Fig. 9, the overall attack consists of three steps and outputs a 3-block colliding pair, and each step targets on a certain message block.

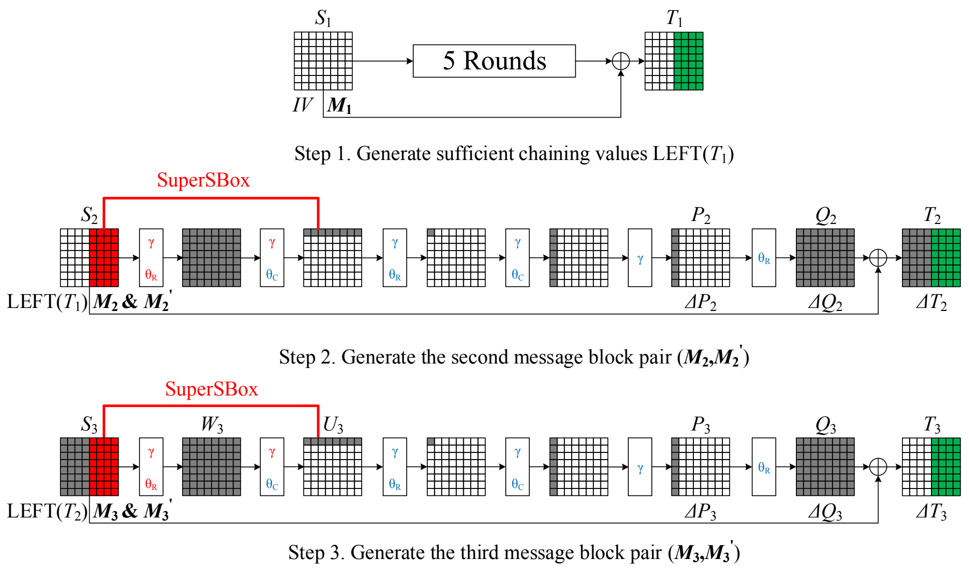


Fig. 9. Collision attack on 5-round `Whirlwind`

Before providing more detailed descriptions of the collision attack, it is necessary to specify several notations used. As depicted in Fig. 9, we use $\Delta St = St \oplus St'$ to denote the difference between two states St and St'. `LEFT`(St) and `RIGHT`(St) denote the left half and the right half of the state St respectively. Also note that in the figures of the collision attacks, active (resp. inactive) cells are denoted in grey (resp. white). The red cells correspond to the input message blocks which can be freely controlled by the attacker, and the green cells in the output chaining value of the compression function are truncated. The overall collision attack consists of three steps, and the collision pair has three blocks, *i.e.*, $(M_1||M_2||M_3, M_1||M_2'||M_3')$.

Step 1. Provide the freedom degrees. The purpose of the first step is to provide sufficient freedom degrees for the subsequent two steps. We randomly choose the values of the first message block M_1 and compute the output chaining values $\texttt{LEFT}(T_1)$.

Step 2. Introduce the input differences and restrict the output differences. We introduce the input differences by choosing different values for the second message block, *i.e.*, (M_2, M_2'), such that the difference of the output chaining values for the second compression function call, namely, $\texttt{LEFT}(\Delta T_2)$, satisfies that $\texttt{LEFT}(\Delta T_2) = \texttt{LEFT}(\Delta Q_2)$, where $\Delta Q_2 = \theta_R(\Delta P_2)$ and the difference of P_2 only lies in the first column.

Step 3. Cancel the output differences. We need to generate the message block pair (M_3, M_3'), such that the output difference of the permutation for the third compression function call, namely, $\texttt{LEFT}(\Delta Q_3)$, satisfies that $\Delta Q_3 = \theta_R(\Delta P_3)$, where the difference of P_3 only lies in the first column. It is clear that the values of ΔQ_2 and ΔQ_3 are in the same subspace thanks to their unique differential forms, and moreover, if $\Delta P_2 = \Delta P_3$ is satisfied, we can directly deduct that $\Delta Q_2 = \Delta Q_3$ due to the linear properties of θ_R. Consequently, the output difference of the third compression function call, namely, $\texttt{LEFT}(\Delta T_3)$ will be 0, since we have $\texttt{LEFT}(\Delta T_3) = \texttt{LEFT}(\Delta T_2) \oplus \texttt{LEFT}(\Delta Q_3) = \texttt{LEFT}(\Delta Q_2) \oplus \texttt{LEFT}(\Delta Q_3)$ due to the feed-forward operation. Finally, appending any identical message blocks which satisfy padding to $(M_1||M_2||M_3, M_1||M_2'||M_3')$ will lead to a collision on the hash function.

4.2 Impacts of the Partially Fixed States

As shown in Fig. 9, partial values of the input states are fixed in the last two steps of the attack. More precisely, the left half of S_2 is fixed to $\texttt{LEFT}(T_1)$ in the second step of the attack, and the left half of the paired values S_3 are fixed to $\texttt{LEFT}(T_2)$ and $\texttt{LEFT}(T_2')$ in the last step of the attack. Since the SuperSBox technique is utilized at the input states of the last two steps, it is essential to discuss the impacts brought by the partially fixed states.

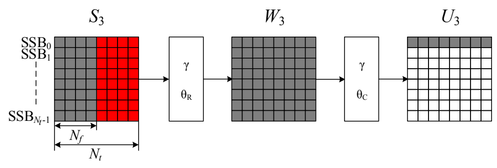


Fig. 10. Illustrations of the tweaked SuperSBox technique with partially fixed states

Without loss of generality, we use the inbound phase of step 3 which connects S_3 and U_3 as an instance to discuss a similar and generalized problem as depicted in Fig. 10. Suppose that both the differences and the values of the N_f grey columns in S_3 are fixed, and the differences and the values of the $N_t - N_f$ remaining red columns in S_3 can be freely chosen. According to the SuperSBox technique, the three layers $\gamma \to \theta_R \to \gamma$ can be seen as N_t parallel row-wise SuperSBoxes, namely, $SSB_0, SSB_1, ..., SSB_{N_t-1}$. Regarding some of the definitions in Section 3.1, the exact attack steps of the tweaked SuperSBox technique with partially fixed states are as follows:

1. From the forward direction, for every difference of the red cells in SSB_0, we enumerate all $2^{N_c(N_t-N_f)}$ possible values conforming to this difference, and compute the corresponding values and differences in $\gamma(W_3)$. We save them in a sorted table. Since we can enumerate all $2^{N_c(N_t-N_f)}$ differences in SSB_0, so the lookup table of SSB_0 has at most $2^{2N_c(N_t-N_f)}$ entries. For the remaining $N_t - 1$ SuperSBoxes, the procedures are similar.
2. From the backward direction, we select a random difference of U_3, and compute it backwards to get the value of $\theta_C^{-1}(\Delta U_3)$. We match this difference with the N_t SuperSBox tables built in step 1. Since a N_cN_t-bit difference needs to be matched for each SuperSBox, two different cases need to be considered:
   Case 1. The number of entries in each table is insufficient to derive a N_cN_t-bit match, namely, $2N_c(N_t - N_f) < N_cN_t$ (equivalent to $2N_f > N_t$). In this case, we have to compute $2^{N_cN_t-2N_c(N_t-N_f)} = 2^{N_c(2N_f-N_t)}$ differences of $\theta_C^{-1}(\Delta U_3)$ to derive a match for a single SuperSBox. Consequently, in order

to derive a match for all N_t SuperSBoxes simultaneously, we have to choose $2^{N_cN_t(2N_f-N_t)}$ different values of $\theta_C^{-1}(\Delta U_3)$. Since there are overall $2^{N_cN_t}$ possible differences of U_3, if $N_cN_t < N_cN_t(2N_f-N_t)$ which is equivalent to $2N_f > N_t+1$, we are not able to derive a match for all the SuperSBoxes at the same time.

Case 2. The number of entries in each table is sufficient to derive a match, namely, $2N_c(N_t-N_f) \geq N_cN_t$ (equivalent to $2N_f \leq N_t$). In this case, instead enumerating all differences and values of the red cells in each row of S_3, we only need to select sufficient say 2^X differences of the red cells, and exhaust all $2^{X+N_c(N_t-N_f)}$ corresponding values for each SuperSBox. We choose $X + N_c(N_t - N_f) = N_cN_t$ (equivalent to $X = N_cN_f$), because we only need to match a N_cN_t-bit difference for each SuperSBox. It requires $2^{N_cN_t}$ time and memory to generate and store the N_t SuperSBoxes tables. Given a specific difference $\theta_C^{-1}(\Delta U_3)$, we expect to get a match for each of the N_t SuperSBoxes simultaneously, thus derive a solution for the inbound phase. After exhausting all $2^{N_cN_t}$ differences of U_3, we expect to get $2^{N_cN_t}$ solutions for the inbound phase with $2^{N_cN_t}$ time and memory. Since there are overall $2^{2N_cN_t(N_t-N_f)}$ differences and corresponding values of the red cells S_3, and N_t active cells in U_3, we can generate at most $2^{2N_cN_t(N_t-N_f)-N_cN_t(N_t-1)} = 2^{N_cN_t(N_t-2N_f+1)}$ solutions for the inbound phase.

To summarize the above analysis, if $2N_f \leq N_t$, we can generate $2^{N_cN_t}$ solutions for the inbound phase with $2^{N_cN_t}$ time and memory, and at most $2^{N_cN_t(N_t-2N_f+1)}$ solutions can be generated for the inbound phase. Otherwise if $2N_f > N_t$, the attack would be infeasible due to the lack of freedom degrees.

4.3 Details of the Collision Attack

For `Whirlwind`, $N_c = 16, N_t = 8, N_f = 4$ which satisfies the condition discussed in Section 4.2, and the SuperSBox technique can be adapted. We illustrate the detailed procedures of the 5-round collision attack as shown in Fig. 9.

Step 1. From the specified IV, we choose a random value for the first message block M_1, and compute its output chaining value which is $\texttt{LEFT}(T_1)$ due to the truncation.

Step 2. From $\texttt{LEFT}(T_1)$, we utilize the SuperSBox technique by exploiting the freedom degrees in both the values and the differences of M_2 in the inbound phase (in red). We get 2^{128} solutions for the inbound phase with 2^{128} time and 2^{128} memory. Based on the analysis in Section 4.2, the overall number of solutions for the inbound phase is 2^{128}. The probability of the outbound phase (in blue) is $2^{-16\times 7} = 2^{-112}$, thus we expect to get 2^{16} pairs of (M_2, M_2').

Step 3. From a specific pair of $\texttt{LEFT}(T_2)$ and $\texttt{LEFT}(T_2')$ which is generated in step 2, we utilize the SuperSBox technique by exploiting the freedom degrees in both the values and the differences of M_3 in the inbound phase (in red). We get 2^{128} solutions for the inbound phase with 2^{128} time and 2^{128} memory. The probability of the outbound phase (in blue) is $2^{-16\times 7} = 2^{-112}$, and we need to guarantee $\Delta P_3 = \Delta P_2$ which happens with probability $2^{-16\times 8} = 2^{-128}$ in order to eliminate the difference. So we need 2^{240} solutions from the inbound phase of step 3, and the freedom degrees seem insufficient. By choosing 2^{96} different values of M_1 in step 1, and combining the 2^{16} solutions $(\texttt{LEFT}(T_2), \texttt{LEFT}(T_2'))$ generated in step 2 for each value of M_1, the overall number of solutions reaches $2^{96+16+128} = 2^{240}$, and we expect to derive a collision.

Complexity Analysis. As illustrated above, it requires 2^{240} compression function computations to generate a collision for 5-round `Whirlwind`. The memory requirement is 2^{128} due to the SuperSBox technique.

4.4 The Four Round Collision Attack

As depicted in Fig. 11, we can also launch a collision attack on 4-round `Whirlwind`. Note that the freedom degrees are sufficient in the 4-round attack, thus we only need to launch a two-block attack. The attack strategy is similar to the 5-round attack through a multi-step process by introducing, restricting and cancelling the differences. We omit more descriptions and it requires 2^{128} time and 2^{128} memory to find a collision for 4-round `Whirlwind`. Since the designers recommend the adoption of `Whirlwind` with truncated digest, it is notable that the 4-round collision attack can also be applied to `Whirlwind-384`, *i.e.*, the `Whirlwind` variant with the 384 LSBs as the digest.

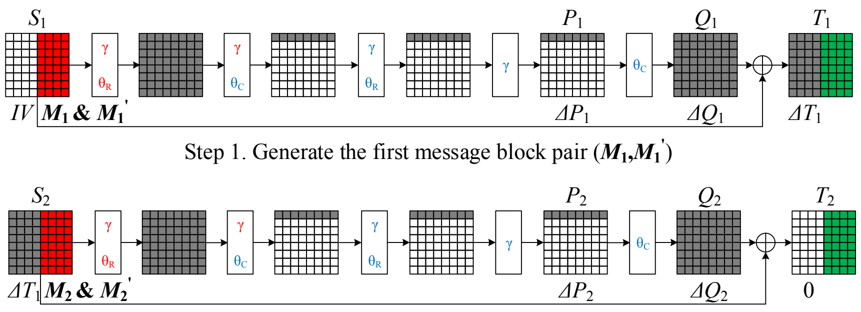
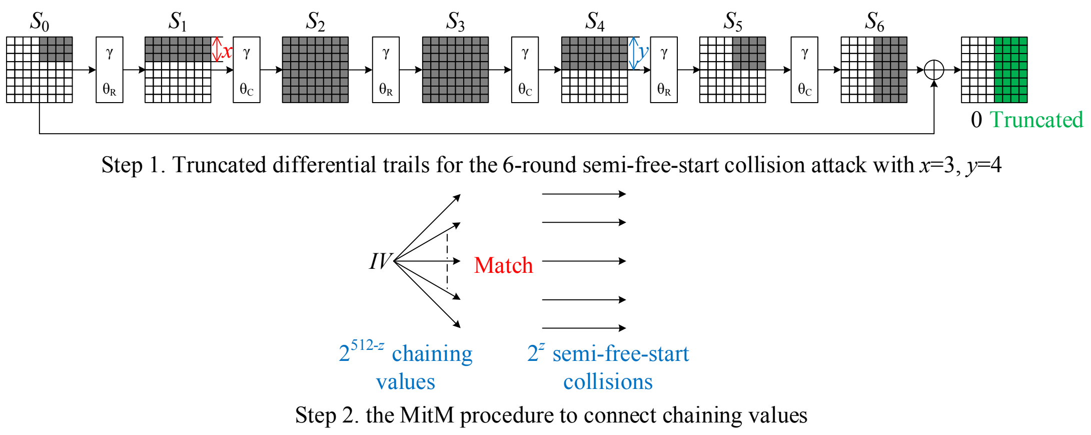


Fig. 11. Collision attack on 4-round `Whirlwind`

5 Limited-Birthday Distinguisher on Reduced-Round `Whirlwind`

The limited-birthday distinguisher on hash functions [14], which was proposed at ASIACRYPT 2013, converts semi-free-start collisions on the compression function into a distinguisher on the hash function. This section presents the limited-birthday distinguisher on 6-round `Whirlwind`. There are two main procedures to build a limited-birthday distinguisher, *i.e.*, generating sufficient semi-free-start collisions on the compression function and connecting the chaining values with the traditional MitM method. In order to achieve better balance between these two procedures, we follow the ideas presented in [23].



S_0
S_1
S_2
S_3
S_4
S_5
S_6
x
y
0 Truncated
Step 1. Truncated differential trails for the 6-round semi-free-start collision attack with x=3, y=4
IV
Match
2^{512-z} chaining values
2^z semi-free-start collisions
Step 2. the MitM procedure to connect chaining values

Fig. 12. Limited-birthday distinguisher on 6-round `Whirlwind`

Step 1. Build the Semi-free-start Collisions. The truncated differential trail for the 6-round semi-free-start collision attack is shown in Fig. 12, and it follows the following pattern:

$$4x \xrightarrow{r_0} 8x \xrightarrow{r_1} 64 \xrightarrow{r_2} 64 \xrightarrow{r_3} 8y \xrightarrow{r_4} 4y \xrightarrow{r_5} 32y,$$

where $x, y = 1, 2, ..., 8$ denote the number of active rows in S_1 and S_4 respectively. The rebound attack and the SuperSBox technique are utilized to find the collisions, and the specific details are omitted. In the inbound phase, given a random difference of $\gamma(S_1)$, we calculate the corresponding SuperSBoxes with 2^{128} time and 2^{128} memory. Then we choose 2^{128} differences from all possible differences of S_4, and match with the SuperSBoxes. According to the SuperSBox technique, we expect to generate 2^{128} solutions for the inbound phase at the cost of 2^{128} time and 2^{128} memory. In other words, the average complexity to generate a solution for the inbound phase is only 1. If the solutions are insufficient, we can choose other difference values of $\gamma(S_1)$ and S_4 and repeat the above procedures.

The outbound phase has a probability of $2^{-64(x+y)}$, thus the average complexities to generate one semi-free-start collision is $2^{64(x+y)}$ time and 2^{128} memory. Note that there are 2^{128x} differences in $\gamma(S_1)$ and 2^{128y} differences in S_4, so we can generate at most $2^{128(x+y)}$ solutions for the inbound phase, and consequently $2^{128(x+y)-64(x+y)} = 2^{64(x+y)}$ semi-free-start collisions.

Step 2. Build the Limited-Birthday Distinguisher. As depicted in Fig. 12, suppose 2^z semi-free-start collisions are generated in step 1, we compute 2^{512-z} output chaining values and hope to find a match using the traditional MitM method with a high probability.

Complexity Analysis. Combining both steps, the time complexity to launch the attack is

$$T = 2^{z+64(x+y)} + 2^{512-z}.$$

Note that the difference of the input messages lies in a subspace of size $2^{16\times 4x} = 2^{64x}$, and the difference of the output lies in a subspace of size 1 due to the collision property. Hence, for an ideal one-way function, it would require $max\{2^{256}, 2^{513-64x}\}$ computations to find a message pair corresponding to the limited-birthday problem.

Now we show how to minimize T and also make the distinguisher valid. Two different occasions need to be considered. In the first occasion, the amount of the semi-free-start collisions is sufficient to achieve balance for step 2, thus we let $z + 64(x + y) = 512 - z$ in order to balance the complexities of the two steps, and we need to guarantee $z \leq 64(x + y)$. The above conditions are equivalent to $x + y \geq 3$ and $z = 256 - 32(x + y)$, and the time complexity can be rewritten as

$$T = 2^{257+32(x+y)}.$$

In the second occasion, the amount of the semi-free-start collisions is insufficient to achieve balance for step 2, thus we need to maximize z to minimize the time complexity. These restraints are equivalent to $x + y \leq 2$ and $z = 64(x + y)$, and the time complexity can be rewritten as

$$T = 2^{512-64(x+y)}.$$

Finally, we consider both occasions and find out that when $x = 1, y = 2$ the overall time complexity is minimized to 2^{353} computations, where the generic attack would require 2^{449} computations. As a result, a valid limited-birthday distinguisher is constructed for 6-round `Whirlwind` with 2^{353} time and 2^{160} memory (the semi-free-start collisions need to be stored).

6 Analyses of the Reduced-Round Components of `Whirlwind`

This section presents analyses of the reduced-round compression function and inner permutation of `Whirlwind`.

6.1 Semi-free-start Collision Attack on 6-Round `Whirlwind` Compression Function

The truncated differential trail is depicted in Fig. 13. The three middle rounds are covered with the inbound phase by utilizing the SuperSBox technique, and in the outbound phase two rounds and one round are propagated in the forward and the backward directions respectively. The inbound phase generates 2^{128} solutions with 2^{128} time and 2^{128} memory. There are two $8 \rightarrow 4$ transitions in the outbound phase, thus the probability of the outbound phase is $2^{-16\times 8} = 2^{-128}$. Consequently, it requires 2^{128} time and 2^{128} memory to find a semi-free-start collision.

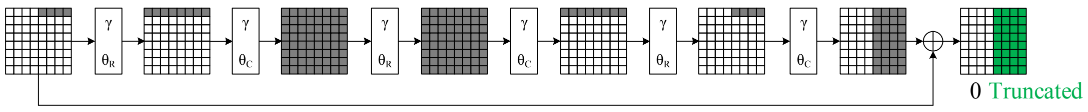


Fig. 13. Semi-free-start collision attack on 6-round `Whirlwind` compression function

6.2 Free-start Near Collision Attack on 7-Round `Whirlwind` Compression Function

We consider the `Whirlwind` Compression Function where the whole state is preserved without the half truncation. The truncated differential trail is depicted in Fig. 14. The three middle rounds are covered with the inbound phase by utilizing the SuperSBox technique, and in the outbound phase two rounds are propagated in both the forward and the backward directions. The inbound phase generates 2^{128} solutions with 2^{128} time and 2^{128} memory, and can generate at most 2^{256} solutions. There are two $8 \to 1$ transitions in the outbound phase, thus the probability of the outbound phase is $2^{-16\times 14} = 2^{-224}$. Consequently, it requires 2^{224} time and 2^{128} memory to find a free-start near collision on 896 bits out of 1024 bits, while for an ideal compression function it would require 2^{448} computations. Note that in order to find a free-start collision, we have to cancel the four leftmost cells of the active row, thus it would require $2^{224+64} = 2^{288}$ solutions from the inbound phase. Consequently, it is almost impossible to derive a free-start collision with this truncated differential trail due to the lack of freedom degrees.

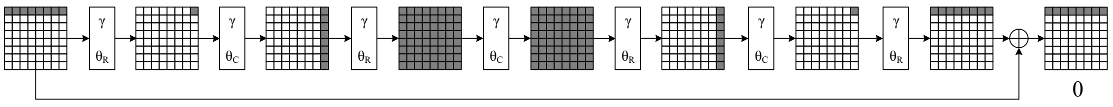


Fig. 14. Free-start near collision attack on 7-round `Whirlwind` compression function

6.3 Limited-Birthday Distinguisher on 9-Round `Whirlwind` Inner Permutation

The truncated differential trail to build the limited-birthday distinguisher on the permutation [12] is depicted in Fig. 15. The distinguisher is actually a straightforward application of the generic distinguisher on `AES`-like permutations which has been comprehensively studied [15,16], thus we omit more descriptions. In a word, we can find two messages satisfying the limited-birthday problem with 2^{736} time and 2^{128} memory, while for an ideal permutation it would require 2^{769} computations. Finally, combining the multiple-birthday distinguisher [17] which takes all possible outbound patterns into account, the time complexity can be further reduced to 2^{730}, while the best known generic algorithm would require no less than 2^{763} computations.

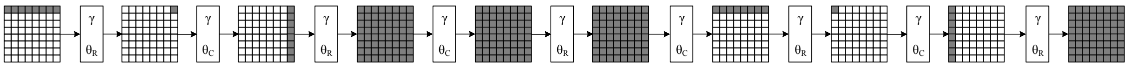


Fig. 15. Limited-birthday distinguisher on 9-round `Whirlwind` inner permutation

7 Conclusion

We provide a thorough security analysis of `Whirlwind`. Firstly, we focus on security properties at the hash function level, and present 6-round second preimage, 4-round preimage, 5-round collision and 6-round distinguishing attacks out of the 12-round hash function. Then we investigate security properties of the `Whirlwind` components with several reduced-round attacks on the compression function and the underlying permutation.

Moreover, we show how to generate preimages with significantly reduced lengths for reduced-round versions of `Whirlpool`, and also present the first preimage attacks at the hash function level for several PGV modes instantiated with 6-round `AES`.

Acknowledgements. We would like to thank the anonymous reviewers of ACISP 2015 for their valuable comments and suggestions.

References

1. Albrecht, M., Driessen, B., Kavun, E., Leander, G., Paar, C., Yalçın, T.: Block Ciphers - Focus on the Linear Layer (feat. PRIDE). In: Garay, J., Gennaro, R. (eds.) CRYPTO 2014, LNCS, vol. 8616, pp. 57–76. Springer, Heidelberg (2014)
2. AlTawy, R., Youssef, A.: Second Preimage Analysis of Whirlwind. In: Lin, D., Yung, M., Zhou, J. (eds.) Inscrypt 2014. LNCS, vol. 8957, pp. 311–328. Springer International Publishing (2015)
3. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., Wang, L.: Preimages for Step-Reduced SHA-2. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 578–597. Springer, Heidelberg (2009)
4. Aoki, K., Sasaki, Y.: Meet-in-the-Middle preimage attacks against reduced SHA-0 and SHA-1. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 70–89. Springer, Heidelberg (2009)

5. Aoki, K., Sasaki, Y.: Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In: Avanzi, R., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009)
6. Barreto, P., Nikov, V., Nikova, S., Rijmen, V., Tischhauser, E.: Whirlwind: A New Cryptographic Hash Function. In: Designs, Codes and Cryptography. vol. 56, pp. 141–162. Springer, US (2010)
7. Barreto, P., Rijmen, V.: The Whirlpool Hashing Function. Submitted to NESSIE (2000), `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`
8. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
9. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - the Advanced Encryption Standard. Springer, Heidelberg (2002)
10. Damgård, I.B.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, New York (1990)
11. Floyd, R.W.: Nondeterministic Algorithms. J. ACM 14(4), 636–644 (1967)
12. 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)
13. International Organization for Standardization: ISO/IEC 10118-1:2004: Information technology - Security techniques - Hash-functions - Part 3: Dedicated hash-functions (2004)
14. Iwamoto, M., Peyrin, T., Sasaki, Y.: Limited-Birthday Distinguishers for Hash Functions. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013. LNCS, vol. 8270, pp. 504–523. Springer, Heidelberg (2013)
15. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved Rebound Attack on the Finalist Grøstl. In: Canteaut, A. (ed.) FSE 2012, LNCS, vol. 7549, pp. 110–126. Springer, Heidelberg (2012)
16. Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved Cryptanalysis of AES-like Permutations. In: J. Cryptology. pp. 1–27. Springer, US (2013), published online
17. Jean, J., Naya-Plasencia, M., Peyrin, T.: Multiple Limited-Birthday Distinguishers and Applications. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013, pp. 533–550. LNCS, Springer, Heidelberg (2014)
18. 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)
19. Khoo, K., Peyrin, T., Poschmann, A., Yap, H.: FOAM: Searching for Hardware-Optimal SPN Structures and Components with a Fair Comparison. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 433–450. Springer, Heidelberg (2014), also available at `http://eprint.iacr.org/2014/530`
20. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 244–263. Springer, Heidelberg (2012)
21. Knellwolf, S., Khovratovich, D.: New Preimage Attacks against Reduced SHA-1. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012, LNCS, vol. 7417, pp. 367–383. Springer, Heidelberg (2012)
22. 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)
23. Ma, B., Li, B., Hao, R., Li, X.: Improved Cryptanalysis on Reduced-Round GOST and Whirlpool Hash Function. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 289–307. Springer International Publishing (2014), full version available at `http://eprint.iacr.org/2014/375`
24. 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)
25. Mendel, F., Rijmen, V., Schläffer, M.: Collision Attack on 5 Rounds of Grøstl. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 509–521. Springer, Heidelberg (2015)
26. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC press (2010)
27. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 428–446. Springer, New York (1990)
28. National Institute of Standards and Technology (NIST): FIPS-197: Advanced Encryption Standard. Federal Information Processing Standards Publication 197, U.S. Department of Commerce (November 2001), `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`
29. National Institute of Standards and Technology (NIST): FIPS PUB 180-3: Secure Hash Standard. Federal Information Processing Standards Publication 180-3, U.S. Department of Commerce (October 2008), `http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf`
30. Peyrin, T.: Cryptanalysis of Grindahl. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 551–567. Springer, Heidelberg (2007)
31. Preneel, B., Govaerts, R., Vandewalle, J.: Hash Functions Based on Block Ciphers: A Synthetic Approach. In: Rueppel, R. (ed.) CRYPTO 1993. LNCS, vol. 658, pp. 368–378. Springer, Heidelberg (1993)
32. Rivest, R.L.: Request for Comments 1321: The MD5 Message Digest Algorithm. The Internet Engineering Task Force (IETF) (1992), `http://www.ietf.org/rfc/rfc1321.txt`

33. Sasaki, Y.: Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 378–396. Springer, Heidelberg (2011)
34. 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)
35. Sasaki, Y., Wang, L., Wu, S., Wu, W.: Investigating Fundamental Security Requirements on Whirlpool: Improved Preimage and Collision Attacks. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 562–579. Springer, Heidelberg (2012)
36. Wu, S., Feng, D., Wu, W., Guo, J., Dong, L., Zou, J.: (Pseudo) Preimage Attack on Round-reduced Grøstl Hash Function and Others. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 127–145. Springer, Heidelberg (2012)

A The Tweaked MitM Preimage Attack Framework without Truncation and Its Applications

A.1 The Tweaked MitM Preimage Attack Framework without Truncation

Fig. 16 shows the basic chunk separation for the tweaked MitM preimage attack framework on 6-round AES-like compression function without truncation. The symbols follow the same definitions in Section 3, and we fix $n = 512, N_c = N_t = 8$ for this case. The initial structure has to be built in the last round (or the first round) due to the influences from both the partially fixed values in the input state before the first round and the feed-forward computation performed after the last round. After the construction of the initial structure, the blue and the red chunks can be computed independently. By utilizing the guess-and-determine technique, the backward computation of the red chunk can be extended by one more round. We omit detailed descriptions of the attack procedures. Also note that we have to remove the guessed round under some circumstances because of the lack of freedom degrees in the state, and the attack can only reach 5 rounds. The memoryless attacks can also be achieved with proper attack parameters. The complexities for the 5- and 6-round attacks can be denoted as follows:

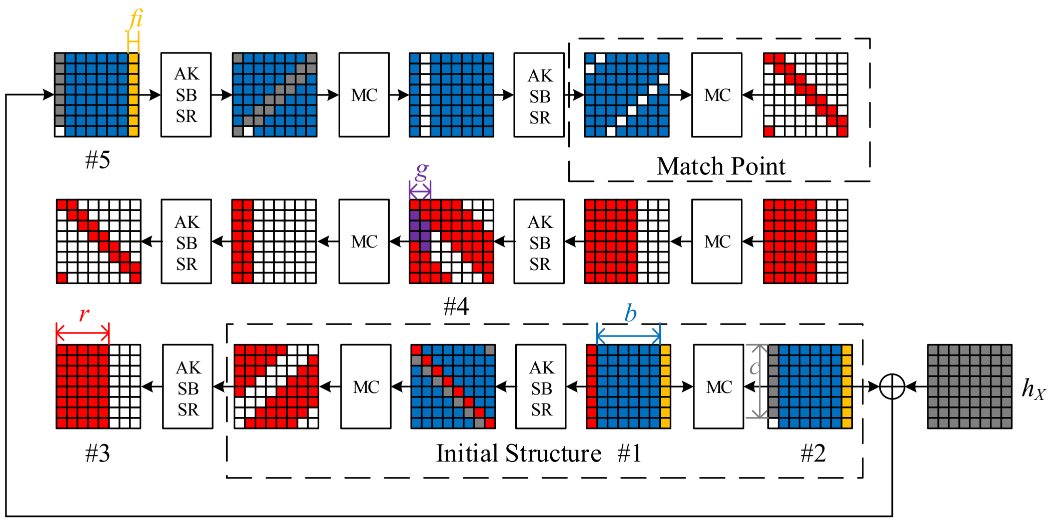
$$\begin{aligned}
&T_{6R} = 2^n(2^{-D_r} + 2^{D_g-D_b} + 2^{D_g-D_m}), && M_{6R} = min\{2^{D_r+D_g}, 2^{D_b}\},\\
&T_{6R,ML} = 2^n(2^{-D_r} + 2^{D_g-D_b} + 2^{D_g-D_m/2}), &&\\
&T_{5R} = 2^n(2^{-D_r} + 2^{-D_b} + 2^{-D_m}), && M_{5R} = min\{2^{D_r}, 2^{D_b}\},\\
&T_{5R,ML} = 2^n(2^{-D_r} + 2^{-D_b} + 2^{-D_m/2}). &&
\end{aligned}$$

where ML refers to the memoryless attacks. The attack parameters can be easily enumerated, and we list the results in Table 2.

Table 2. Summary of the results for the tweaked MitM preimage attacks without truncation

Rounds	fi	Parameters I $\#(b, r, c, g)$	Parameters II $\#(D_b, D_r, D_g, D_m)$	Time	Memory
6	1	(6,5,7,2)	(64,8,48,64)	2^{504}	2^{56}
	2	(5,3,7,3)	(128,8,120,128)	2^{504}	2^{128}
5	1	(5,4,5,-)	(64,48,-,64)	2^{464}	2^{48}
		(5,3,6,-)†	(128,32,-,64)	2^{480}	$\mathcal{O}(1)$
	2	(4,3,6,-)	(64,32,-,64)	2^{480}	2^{32}
		(4,3,6,-)†	(64,32,-,64)	2^{480}	$\mathcal{O}(1)$
	3	(4,3,6,-)	(64,16,-,64)	2^{496}	2^{16}
		(4,3,6,-)†	(64,16,-,64)	2^{496}	$\mathcal{O}(1)$
	4	(3,2,7,-)	(64,8,-,64)	2^{504}	2^{8}
		(3,2,7,-)†	(64,8,-,64)	2^{504}	$\mathcal{O}(1)$

† : The memoryless MitM preimage attack.



b : blue rows at #1	r : red rows at #3
c : constant cells in each row at #2	g : purple rows at #4
fi : fixed padding rows at #5	$(b, r, c, g, fi) = (6, 5, 7, 2, 1)$

6-Round Attack:

Freedom in Blue: $D_b = N_c N_t (b-r)$	Freedom in Red: $D_r = N_c (N_t - c)(N_t - b - fi)$
Guess Cells: $D_g = g N_c (N_t - r)$	Match Size: $D_m = N_c N_t (g + c - N_t)$

5-Round Attack:

Freedom in Blue: $D_b = N_c N_t (b-r)$	Freedom in Red: $D_r = N_c (N_t - c)(N_t - b - fi)$
Match Size: $D_m = N_c N_t (r + c - N_t)$	

Fig. 16. The tweaked MitM preimage attack framework on AES-like compression function without truncation

A.2 Two-Block Preimage Attack on 5-Round Whirlpool and More

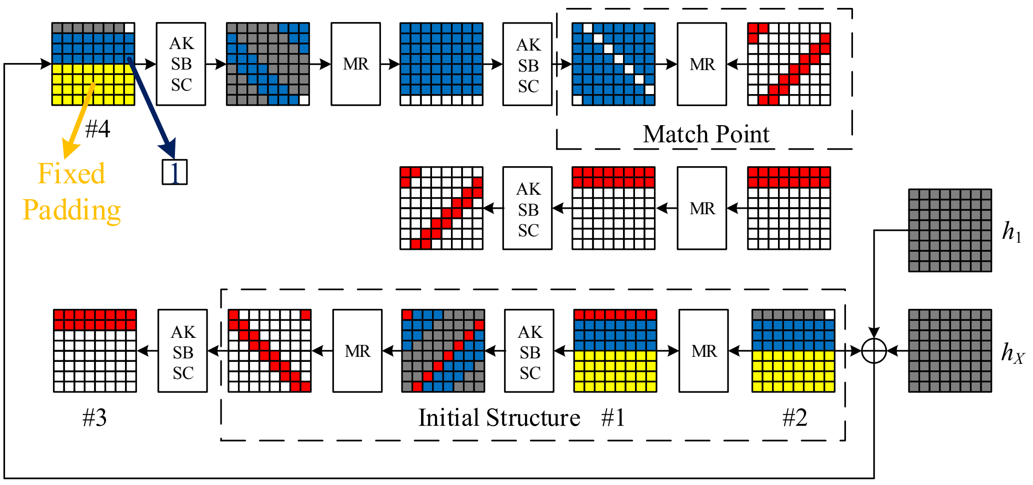
Whirlpool is an ISO standardized hash function [13] which operates on 512-bit input message blocks and produces a 512-bit digest. It adopts the classical MD-strengthening padding, which adds a single bit '1' followed with multiple '0's, and appends a 256-bit length padding L. Its compression function adopts the Miyaguchi-Preneel mode. We refer to [7] for the detailed specifications of Whirlpool. There are several preimage attacks on reduced-round Whirlpool [36,35], but these attacks rely on the expandable messages and construct preimages with very long and impractical lengths. We present two preimage attacks on reduced-round Whirlpool, namely, the two-block preimage attack on 5-round Whirlpool and the preimage attack on 6-round Whirlpool which significantly reduce the length of the preimage from a maximum size of 2^{256} bits to a maximum size of 2^{128} bits.

Two-Block Preimage Attack on 5-Round Whirlpool. As shown in Fig. 17 (a), the two-block preimage attack consists of two steps. In the first step, we choose random values of the first message block M_1, and compute its output chaining value h_1 from the initial value of Whirlpool. In the second step, given h_1 and the target digest h_X, we launch the tweaked MitM preimage attack on the 5-round compression function to derive the second message block M_2. Since we have to match a 512-bit digest, and partial values of M_2 are prefixed due to padding, it seems the freedom degrees are insufficient. However, since we can freely choose the values of M_1 and repeat the second step, thus we have 512 bits freedom degrees in h_1 which guarantee the feasibility of the attack.

Before we show how to deal with the padding part of M_2, we clarify that the technique utilized in [35] which exploits the freedom degrees in the chaining value is not applicable in our attack because the initial-structure now has to be placed in the last attacked round. As depicted in Fig. 17 (b), the yellow cells of the initial state #4 relate to the 256-bit length padding L, and are fixed to the hex value 0x2ff which is the bit length of $M_1||M_2$ before padding. Since we have to append a single bit '1' to M_2 according to the MD-strengthening padding, we also lose 1-bit freedom degree in the blue chunk. The expressions of the attack parameters are provided in Fig. 17 (b). After an exhaustive search for the best attack parameters, the two-block preimage attack on 5-round `Whirlpool` can be launched with 2^{504} time and 2^{8} memory. The attack can also be memoryless with the same time complexity.

$$\begin{tikzcd} IV \arrow[r, "M_1", "\text{Step 1}"'] & h_1 \arrow[r, "M_2", "\text{Step 2}"'] & h_X \end{tikzcd}$$

(a) sketch of the 5-round preimage attack



b : blue rows at #1 $\quad$ r : red rows at #3

c : constant cells in each row at #2 $\quad$ fi : fixed padding rows at #4

$(b, r, c, fi) = (3, 2, 7, 4)$

Freedom in Blue: $D_b = N_c N_t (b-r) - 1$ $\quad$ Freedom in Red: $D_r = N_c (N_t - c)(N_t - b - fi)$

Match Size: $D_m = N_c N_t (r + c - N_t)$

(b) chunk separation for step 2

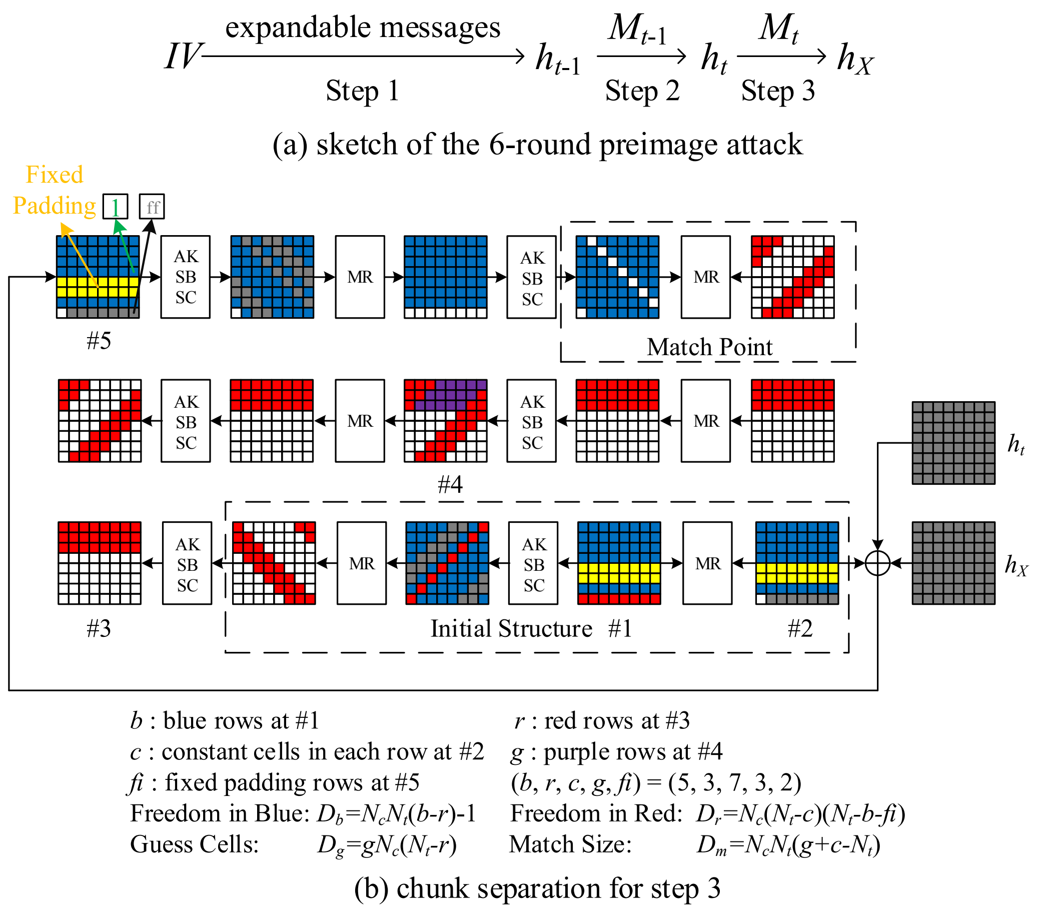
Fig. 17. Two-block preimage attack on 5-round `Whirlpool`

Extend the Attack to 6-Round `Whirlpool`. The current best preimage attack on `Whirlpool` [35] reaches 6 rounds with 2^{481} time and 2^{256} memory. However, this attack adds no restraints on the padding part, and the length of the preimage generated can reach a maximum size of 2^{256} bits. Now we show how to launch the preimage attack on 6-round `Whirlpool` which significantly reduces the maximum size of the preimage from 2^{256} bits to 2^{128} bits.

As depicted in Fig. 18 (a), this attack consists of three steps. In the first step, we build a $(119, 119 + 2^{119} - 1)$-expandable messages with $2^{119} + 119 \times 2^{256+1} \approx 2^{264}$ time and negligible memory[2]. We denote the output chaining value of the expandable messages as h_{t-1}. In the second step, from h_{t-1}, we choose random values for the message block M_{t-1} and compute the output chaining value h_t. In the third step, given h_t and the target digest h_X, we launch the tweaked MitM preimage attack on the 6-round compression function to derive the last message block M_t. We need to repeat the last two steps to exploit the freedom degrees in h_t which guarantee the validity of the attack. After we derive the value of M_t, we choose message blocks from the expandable messages corresponding to the padding part of M_t. There is a negligible probability that the message length does not correspond to the expandable messages. However, we can repeat the last two steps to derive another value of M_t and make adaptive selection from the expandable messages. If all three steps succeed, we manage to construct a preimage for 6-round `Whirlpool`.

[2] The collision search in order to build the expandable messages can be memoryless by adopting the cycle detection techniques such as Floyd's cycle-finding algorithm [11].

Based on the results in Appendix A.1, the 6-round attack can work with at most two prefixed columns (or rows, depending on whether the MixColumn or the MixRow is performed). Consequently, two yellow rows are fixed to 0 in the initial state #5 as depicted in Fig. 18 (b), and the maximum size of the message reduces to 2^{128} bits (equivalent to 2^{119} 512-bit message blocks, thus explain the structure of the expandable messages built in the first step). We also have to fix the 256-th bit to '`1`' due to the MD-strengthening padding, and thus lose 1 bit freedom degree of the blue chunk. The last 9 bits have to be fixed to `0xff` since the last block M_t contains 255 message bits, but these bits lie in the constants in the initial structure which can be freely chosen by the attacker, thus we do not lose any freedom degrees of the red chunk. Finally, we derive the optimal attack parameters through an exhaustive search, and this step requires 2^{505} time and 2^{127} memory. Combining all three steps of the 6-round attack, we need 2^{505} time and 2^{127} memory to find a preimage for 6-round `Whirlpool` whose length is at most 2^{128} bits.



IV
expandable messages
Step 1
h_{t-1}
M_{t-1}
Step 2
h_t
M_t
Step 3
h_X
(a) sketch of the 6-round preimage attack
Fixed Padding
1
ff
AK SB SC
MR
Match Point
#5
#4
#3
#1
#2
Initial Structure
h_t
h_X
b : blue rows at #1
r : red rows at #3
c : constant cells in each row at #2
g : purple rows at #4
fi : fixed padding rows at #5
$(b, r, c, g, fi) = (5, 3, 7, 3, 2)$
Freedom in Blue: $D_b=N_cN_t(b-r)-1$
Freedom in Red: $D_r=N_c(N_r-c)(N_t-b-fi)$
Guess Cells: $D_g=gN_c(N_t-r)$
Match Size: $D_m=N_cN_t(g+c-N_t)$
(b) chunk separation for step 3

Fig. 18. Preimage attack on 6-round `Whirlpool` with relatively short length

A.3 Preimage Attacks on 6-Round `AES` Hashing Modes with Practical Lengths

It is also interesting to apply the tweaked MitM preimage attack to the PGV modes instantiated with (reduced-round) `AES`. Table 3 lists the 12 secure PGV modes. In [33], Sasaki evaluated the (second) preimage resistance of the PGV modes instantiated with 7-round `AES`. More precisely, preimage attacks were launched for No.1 to No.4. However, only second preimage attacks were launched for No.5 to No.12 (including Matyas-Meyer-Oseas and Miyaguchi-Preneel) because the padding cannot be satisfied. The preimage resistance of `AES` hashing modes has also been studied in later works including [36] (slight improvements of [33]) and [8] (accelerated full-round exhaustive search at the compression function level), but it remains an open problem to launch preimage attacks on the PGV modes from No.5 to No.12 instantiated with (reduced-round) `AES`.

Table 3. Twelve secure PGV modes, X_i refers to $H_i \oplus M_i$.

No.	Computation	No.	Computation	No.	Computation	No.	Computation
1	$E_{M_i}(H_i) \oplus H_i$ †	2	$E_{M_i}(X_i) \oplus X_i$	3	$E_{M_i}(H_i) \oplus X_i$	4	$E_{M_i}(X_i) \oplus H_i$
5	$E_{H_i}(M_i) \oplus M_i$ ‡	6	$E_{H_i}(X_i) \oplus X_i$	7	$E_{H_i}(M_i) \oplus X_i$ §	8	$E_{H_i}(X_i) \oplus M_i$
9	$E_{X_i}(M_i) \oplus M_i$	10	$E_{X_i}(H_i) \oplus H_i$	11	$E_{X_i}(M_i) \oplus H_i$	12	$E_{X_i}(H_i) \oplus M_i$

†: Davies-Meyer ‡: Matyas-Meyer-Oseas §: Miyaguchi-Preneel

We present the first preimage attacks on the PGV modes from No.5 to No.12 instantiated with 6-round `AES` under a very reasonable adoption of the padding algorithm. Since the digest size is 128 bits, we assume the padding algorithm is MD-strengthening with the 64-bit length padding. Before describing the attacks, we stress that only `AES-128` is applicable to instantiate the PGV modes from No.5 to No.12 since the message block and the chaining value have identical length. Also note that the MixColumn operation in the last `AES` round is omitted in our attacks as [33] already did, because the full `AES` does not perform this operation in the final round either.

As depicted in Fig. 19 (a), the attack consists of three steps, and the first two steps are almost identical to the attack on 6-round `Whirlpool` in Appendix A.2, hence we omit more descriptions. The main difference is that we now build $(33, 33 + 2^{33} - 1)$-expandable messages with $2^{33} + 33 \times 2^{64+1} \approx 2^{70}$ time and negligible memory.

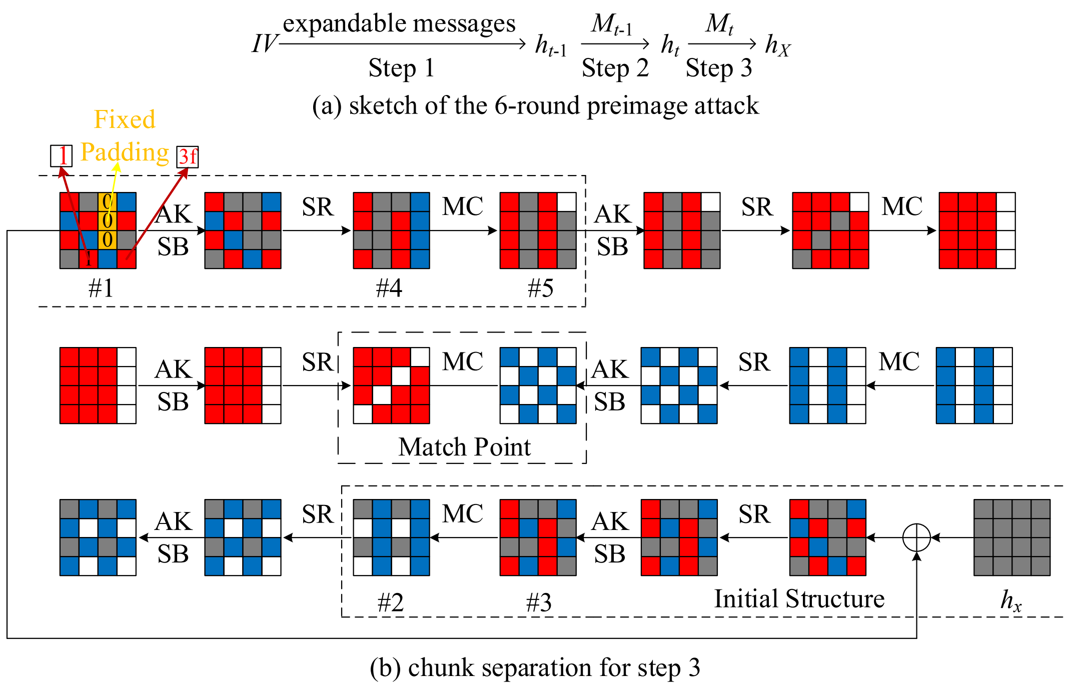


Fig. 19. Preimage attack on 6-round `AES-128` in the Matyas-Meyer-Oseas mode

The chunk separation for the MitM preimage attack in the last step on 6-round `AES` in the Matyas-Meyer-Oseas mode is depicted in Fig. 19 (b), while the remaining PGV modes from No.5 to No.12 can be analyzed similarly. Since the initial structure has to be placed in the first round and the last round, the attack cannot be extended to 7 rounds as the attacks in [33]. The 3 yellow cells are fixed to 0, thus the maximum size of the message is reduced from 2^{64} bits to 2^{40} bits (equivalent to 2^{33} 128-bit message blocks, and also equivalent to a practical 128 GB storage). We have to fix the 64-th bit to '`1`' and the last 7 bits to `0x3f` due to the padding algorithm, thus lose 8 bits freedom degrees in the red chunk. As a result, we can see from #3 to #2 that the freedom degree in the red chunk is $16 - 8 = 8$ bits. From #4 to #5, we can see freedom degree in the blue chunk is 8 bits. The size of the match point is 32 bits. Finally, the last step requires $2^{128} \times (2^{-8} + 2^{-8} + 2^{-32}) \approx 2^{120}$ time and 2^8 memory. This step can also be memoryless thanks to the specific attack parameters with a relatively small size of the match point.

For the PGV modes from No.5 to No.8, the input chaining value of the last block can be prefixed, thus the preimage attack on these modes instantiated with 6-round AES requires 2^{120} time and negligible memory. However, for the PGV modes from No.9 to No.12, the input chaining value is generated after the last step, thus cannot be predefined. We need to invoke the generic MitM method [26, Fact 9.99] to convert the pseudo preimages into a preimage, and the complexities of the preimage attack become 2^{125} time and 2^4 memory. Table 4 summarizes our new preimage attacks on Whirlpool and AES hashing modes.

Table 4. Summary of preimage attacks on Whirlpool and AES hashing modes, only preimage attacks on the hash functions are listed.

Target	Rounds	Time	Memory	Reference	Remarks
Whirlpool	5	2^{448} 2^{465}	2^{96} $\mathcal{O}(1)$	[35]	Long preimage up to 2^{256} bits.
		2^{504}	$\mathcal{O}(1)$	Appendix A.2	Two-block preimage.
	6	2^{481} 2^{504}	2^{256} $\mathcal{O}(1)$	[35]	Long preimage up to 2^{256} bits.
		2^{505}	2^{127}	Appendix A.2	Long preimage up to 2^{128} bits.
AES hashing modes	7	2^{125} $2^{122.7}$	2^8 2^{16}	[33] [36]	PGV modes No.1 to No. 4, preimage with short length.
	6 †	2^{120}	$\mathcal{O}(1)$	Appendix A.3	PGV modes No.5 to No. 8, long preimage up to 128 GB.
	6 †	2^{125}	2^4	Appendix A.3	PGV modes No.9 to No. 12, long preimage up to 128 GB.

† : The padding algorithm is MD-strengthening with 64-bit length padding.