Improved (Pseudo) Preimage Attacks on Reduced-Round GOST and Grøstl-256 and Studies on Several Truncation Patterns for AES-like Compression Functions (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. In this paper, we present improved preimage attacks on the reduced-round GOST hash function family, which serves as the new Russian hash standard, with the aid of techniques such as the rebound attack, the Meet-in-the-Middle preimage attack and the multicollisions. Firstly, the preimage attack on 5-round GOST-256 is proposed which is the first preimage attack for GOST-256 at the hash function level. Then we extend the (previous) attacks on 5-round GOST-256 and 6-round GOST-512 to 6.5 and 7.5 rounds respectively by exploiting the involution property of the GOST transposition operation. Secondly, inspired by the preimage attack on GOST-256, we also study the impacts of four representative truncation patterns on the resistance of the Meet-in-the-Middle preimage attack against AES-like compression functions, and propose two stronger truncation patterns which make it more difficult to launch this type of attack. Based on our investigations, we are able to slightly improve the previous pseudo preimage attacks on reduced-round Grøstl-256.

Keywords: hash function, cryptanalysis, preimage, GOST, Grøstl-256, the Meet-in-the-Middle preimage attack, truncation patterns

1 Introduction

Cryptographic hash function is one of the fundamental building blocks in modern cryptography. A hash function takes a message of arbitrary length as input and outputs a bit string of fixed length. For a secure hash function, three classical security properties are mainly concerned, namely, the collision resistance, the second preimage resistance, and the preimage resistance. Many state-of-the-art hash functions divide the input messages into short blocks and process each block with the compression function iteratively, such as the classical Merkle-Damgård [14,38] construction. Due to the generic attacks on the Merkle-Damgård construction [25,28,27], several new domain extension schemes are proposed to fix the inherent weaknesses of the Merkle-Damgård construction. One popular instance of these new constructions is the HAIFA framework proposed by Biham and Dunkelman in [9]. It adds a salt value and a counter which denotes the number of message bits hashed so far as extra input parameters to the compression function, thus makes each compression iteration distinct and is believed to resist certain generic attacks [28,27]. In practice, it has been adopted by several SHA-3 candidates including BLAKE [5], ECHO [7], SHAvite-3 [10], and also the new Russian hash standard GOST R 34.11-2012, to name but a few.

The old GOST R 34.11-94 hash function [17] was theoretically broken in 2008 [35,36]. As a consequence, the new GOST R 34.11-2012 hash function [15,18,26] has replaced GOST R 34.11-94 as the new Russian national hash standard since January 1, 2013, and it is also included by IETF as RFC 6986 [16]. The new GOST hash function family consists of two members: GOST-256 and GOST-512, which correspond to two different output lengths. It adopts the HAIFA construction with a unique output transformation, and its compression

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

function contains two parallel AES-like permutations in the Miyaguchi-Preneel mode, which is very similar to the Whirlpool hash function [6,23]. Several cryptanalytic results have already been presented since the announcement of the new GOST hash function. Interesting results on the GOST compression function are shown in [1,43], but they seem to have limited impacts on the GOST hash function. The first cryptanalytic result at the hash function level was given by Zou *et al.* at Inscrypt 2013 [45], which presented collision attacks on 5-round GOST-256/512 and a preimage attack on 6-round GOST-512. The recent preimage attack in [2] is similar to the one in [45]. Ma *et al.* proposed several improved attacks on GOST [33], including improved (memoryless) preimage attacks on 6-round GOST-512, collision attacks on 6.5/7.5-round GOST-256/512, and the limited-birthday distinguisher [24] on 9.5-round GOST-512. More recently, Guo *et al.* presented generic second preimage attacks on the full GOST-512 hash function [22] by exploiting the misuse of the counter in the HAIFA mode of GOST. However, their attacks cannot be extended to preimage attacks which allow to invert the hash function. Due to the truncation, their attacks do not work for GOST-256 either.

To the best of our knowledge, there are no preimage attacks on the GOST-256 hash function except for the trivial brute-force attack. One of the crucial reasons that prevent the preimage attack on GOST-256 is the ChopMD-like mode adopted in the GOST-256 output transformation. The ChopMD mode [12], which truncates a fraction of the final output chaining value, is proven to be indifferentiable from a random oracle [12,11]. However, the specific truncation patterns in it have not been well studied yet. Many instances in practice just truncate the MSBs (or LSBs) of the chaining value as the digest, such as some of the most acknowledged hash functions SHA-3 [8], SHA-224/384 [39], GOST-256, Grøstl [20] and many other SHA-3 candidates. Although this type of truncation pattern is convenient to be implemented in both software and hardware, its concrete impacts on the security properties of the hash functions are less evaluated in literature. Hence, evaluating different truncation patterns and seeking for an optimal one is a very meaningful issue in cryptographic research.

Our Contributions. Firstly, we present improved preimage attacks on the reduced-round GOST hash function family. With the aid of the Meet-in-the-Middle (MitM) preimage attack [4] and the multicollisions [25] which are constructed with dedicated collisions, we overcome the obstacle of the ChopMD mode and present a preimage attack on 5-round GOST-256. Furthermore, by exploiting the weaknesses of the GOST transposition operation, if the MixRow operation of the last round is omitted, we are able to extend the 5-round preimage attack on GOST-256 and the previous preimage attack on 6-round GOST-512 [33] to 6.5 rounds and 7.5 rounds respectively. Although it is not natural to omit the MixRow operation in a single round for a hash function like GOST because it certainly disturbs the wide trail strategy [13], these attacks are theoretically meaningful since they show that any operations with the involution property (*i.e.*, the matrix transposition operation of GOST) would facilitate the attackers with more attacked rounds, thus further demonstrate that such operations with the involution property should be cautiously considered as candidates for achieving transposition in AES-like hash primitives. The results are summarized in Table 1. However, due to the limited number of rounds attacked, our results do not pose any threats to the practical use of the GOST hash function family.

Secondly, motivated by the preimage attack on GOST-256, we discuss the impacts of the truncation patterns on the resistance of the MitM preimage attack against AES-like compression functions with the ChopMD mode. We investigate four representative truncation patterns, and show that two of them certainly make it more difficult to launch this type of attack. Moreover, the last pattern studied even resists the 6-round MitM preimage attack for certain digest sizes. Based on the investigations, we are able to slightly improve the previous pseudo preimage attacks on the reduced-round SHA-3 finalist Grøstl-256.

Organization of the Paper. In Section 2, we give brief descriptions of the GOST hash function. Section 3 presents the improved preimage attacks on the reduced-round GOST hash function family. Section 4 discusses the impacts of four representative truncation patterns on the resistance of the MitM preimage attack against AES-like compression functions. Section 5 concludes and summarizes the paper. [34].

2 The GOST Hash Function Family

The GOST hash function takes any message up to 2^{512} bits as input, and outputs a 256- or 512-bit digest, *i.e.*, GOST-256 and GOST-512. These two variants are almost the same, except that they have different initial

Table 1. Summary of previous and our (second) preimage attacks on GOST

Target	Rounds Attacked	Time	Memory	Reference
GOST-256 (12 Rounds)	5	2^{192} 2^{208} §	2^{64} 2^{12}	Section 3
	6.5 †	2^{232}	2^{120}	
GOST-512 (12 Rounds)	6	2^{505}	2^{256}	[2]
		2^{505}	2^{64}	[45]
		2^{496} 2^{504} §	2^{64} 2^{11}	[33]
	7.5 †	2^{496} 2^{504} §	2^{64} 2^{11}	Section 3
	Full ‡	$2x \cdot 2^{512-x}$, for $x < 178.67$ 2^{523-x}, for $x < 259$	Not given Not given	[22]

† : Require the omission of the last MixRow operation.
‡ : Only work as second preimage attacks.
§ : The memory minimized attacks.

values, and GOST-256 only preserves the 256 MSBs of the final 512-bit chaining value as the digest. As depicted in Fig. 1, the GOST hash function family adopts the HAIFA construction with a unique output transformation. The hash computation contains three stages. Before we give specific descriptions of each stage, we define several notations.

$A||B$ The concatenation of two bit strings A and B.
M The input message, which is divided into 512-bit blocks.
M_i The i-th 512-bit message block of M.
$|M|$ The bit length of M.
Len The bit length of the last message block of M.
Σ The 512-bit checksum of all message blocks.
CF The compression function.
h_i The i-th 512-bit chaining variable.
CT_i The i-th 512-bit counter which denotes the total message bits processed before the i-th CF call.

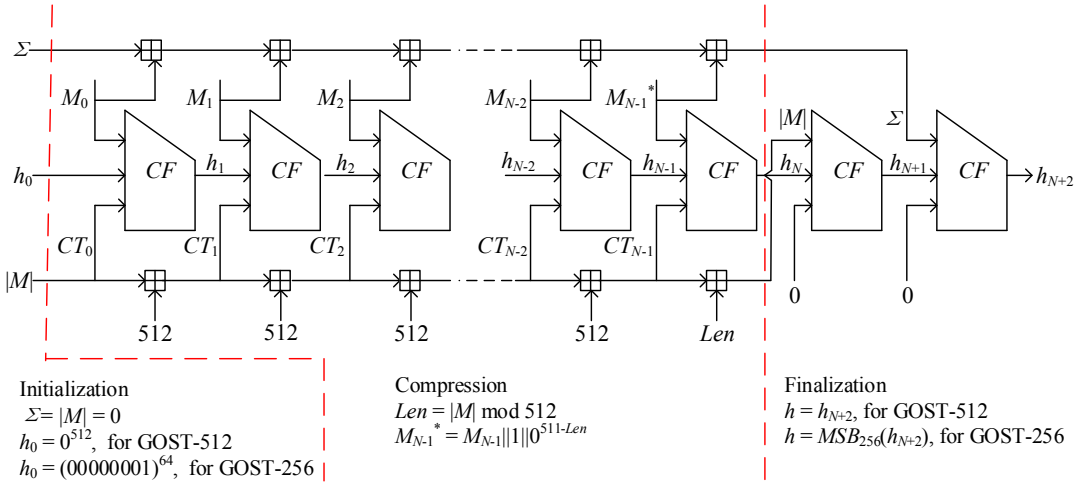

Fig. 1. Three stages of the GOST hash function

In the initialization stage, M is padded into a multiple of 512 bits, *i.e.*, $M||1||0^*$ is the padded message, which is then divided into N 512-bit blocks $M_0||M_1||...||M_{N-1}$. h_0 is assigned to the predefined IV of GOST-256 or GOST-512. $|M|$, Σ and CT_0 are assigned to 0. In the compression stage, each block M_i is processed iteratively, *i.e.*, $h_{i+1} = CF(h_i, M_i, CT_i)$ for $i = 0, 1, ..., N-1$. After each computation of the compression function, $|M|$, Σ and CT_{i+1} are updated accordingly. In the finalization stage, the output chaining value of the last message block h_N goes through the output transformation, *i.e.*, $h_{N+1} = CF(h_N, |M|, 0)$, $h_{N+2} = CF(h_{N+1}, \Sigma, 0)$. For GOST-512 (resp. GOST-256), h_{N+2} (resp. $MSB_{256}(h_{N+2})$) is the digest of M.

The compression function $CF(h_i, M_i, CT_i)$ can be seen as an AES-like block cipher E_K used in a Miyaguchi-Preneel-like mode, *i.e.*, $CF(h_i, M_i, CT_i) = E_{h_i \oplus CT_i}(M_i) \oplus M_i \oplus h_i$. As for the block cipher E_K, the 512-bit internal state is denoted as an 8×8 byte matrix. For the key schedule part, $h_i \oplus CT_i$ is assigned as the key K, then K_0 is computed from K as follows:

$$K_0 = L \circ P \circ S(K).$$

The round keys $K_1, K_2, ..., K_{12}$ are generated as follows:

$$K_{j+1} = L \circ P \circ S \circ XC(K_j) \ for \ j = 0, 1, 2, ...11,$$

where K_{12} is used as the post-whitening key:

- **AddRoundConstant(XC):** XOR a 512-bit constant predefined by the designers.
- **SubBytes(S):** process each byte of the state through the SBox layer.
- **Transposition(P):** transpose the k-th column to be the k-th row for $k = 0, 1, 2, ..., 7$, *i.e.*, transposition of the state matrix.
- **MixRows(L):** multiply each row of the state matrix by an MDS matrix.

For the data processing part, M_i is the plaintext, and is assigned to the initial state S_0. Then the state is updated 12 times with the round function as follows:

$$S_{j+1} = L \circ P \circ S \circ X(S_j), \ for \ j = 0, 1, 2, ...11,$$

where **AddRoundKey(X)** XOR the state with the round key K_j. Finally, the ciphertext $E_K(M_i)$ is computed with $S_{12} \oplus K_{12}$.

3 Improved Preimage Attacks on Reduced-Round GOST

This section illustrates improved preimage attacks on the reduced-round GOST hash function family. Firstly, we present the preimage attack on 5-round GOST-256. This attack overcomes the obstacles of the output transformation and the truncation operation of GOST-256 by combining the dedicated collision and preimage attacks on the compression function, and is the first preimage attack on GOST-256 in literature except for the trivial brute-force attack. Then by exploiting the weaknesses of the GOST transposition operation P, if the MixRow operation L in the last round is omitted, we are able to extend the 5-round preimage attack on GOST-256 and the previous 6-round preimage attack on GOST-512 to 6.5 rounds and 7.5 rounds respectively. Although such an omission seems unnatural for a construction like GOST since it certainly disturbs the wide trail strategy[1], the improved preimage attacks with more attacked rounds are of some theoretical interests since they further demonstrate that any operations with the involution property should be cautiously considered to achieve transposition in AES-like hash primitives from the aspect of preimage resistance, as similar deductions have already been presented from the aspect of collision-like attacks in [33].

3.1 Overview of the Preimage Attack on 5-Round GOST-256

Based on the preimage attacks on 6-round GOST-512 [33], a very straightforward strategy is to find a (pseudo) preimage[2] on the last compression function call of the output transformation, and convert it to a preimage

[1] Actually, similar observations have already been adopted [40], in which preimage attack on 7-round AES hashing modes was constructed by omitting the last MixColumn operation, but it is natural for AES since there is no MixColumn operation in the last round of AES. However, the omission of MixColumn/MixRow in the last round is mainly adopted by block ciphers to achieve implementation advantage in the decryption algorithm, but such omission might be improper under the hash function setting since inverse computation is commonly not required for hash functions.

[2] A pseudo preimage is a preimage whose input chaining value is not equal to the given chaining value.

attack on the GOST-256 hash function with the aid of tactfully constructed multicollisions. However, if the multicollisions are constructed with cascaded collisions which are generated with the birthday attack, the time complexity would be worse than the brute-force preimage attack on GOST-256. Moreover, the final truncation operation of GOST-256 also makes it more complex to launch the MitM preimage attack on the last compression function call of the output transformation.

To overcome the first obstacle, we could use dedicated methods to generate the collisions rather than the birthday attack, and construct the multicollisions with these dedicated cascaded collisions with unique structures. For the second problem, by a deeper look into the original MitM preimage attack framework on AES-like compression functions, we are able to find the optimal attack parameters for reduced-round GOST-256 although it performs the final truncation. Motivated by the preimage attack on reduced-round GOST-256, more studies on the MitM preimage attack for AES-like compression functions with truncation are discussed in Section 4.

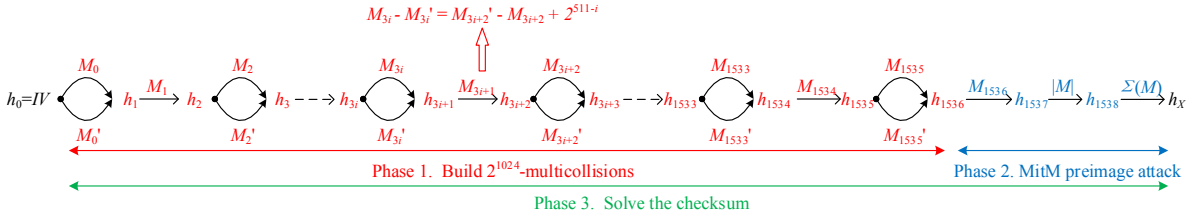

Fig. 2. Three phases of the preimage attack on reduced-round GOST-256

As depicted in Fig. 2, the preimage attack on 5-round GOST-256 consists of three phases. We briefly introduce the main thoughts of each phase which help to understand the whole attack.

Phase 1. 2^{1024}-multicollisions are constructed with 512 cascaded 4-multicollisions pairs, namely, $(M_{3i}, M'_{3i})||M_{3i+1}||(M_{3i+2}, M'_{3i+2})$ for $i = 0, 1, ..., 511$. We choose a 3-block message pair from each of the 4-multicollisions, *i.e.*,
$(M_{3i}||M_{3i+1}||M_{3i+2}, M'_{3i}||M_{3i+1}||M'_{3i+2})$ for $i = 0, 1, ..., 511$, which satisfy that

$$M_{3i} + M_{3i+2} = M'_{3i} + M'_{3i+2} + 2^{511-i}. \tag{1}$$

Since the size of the message checksum is 512 bits, and 4-multicollisions are required for each $i = 0, 1, ..., 511$ to make Equation (1) hold, thus we need to build 2^{1024}-multicollisions. Any possible value of the message checksum can be constructed with the 512 3-block message pairs for $i = 0, 1, ..., 511$ selected from the 2^{1024}-multicollisions due to the unique structures of the collision pairs. In order to build the cascaded collisions which form the multicollisions, we utilize the dedicated collision attacks on 5-round GOST-256 compression function [45] which will be illustrated later.

Phase 2. With the output chaining value of the 2^{1024}-multicollisions h_{1536}, we randomly choose one more message block M_{1536} which satisfies padding and derive the message bit length $|M|$ as well. The value of h_{1538} can then be computed. Let h_X denote the given target, the last compression function call of the output transformation is then inverted with the MitM preimage attack, and the preimage $\Sigma(M)$ generated in this phase is the message checksum desired. Since the collision attack in phase 1 can only reach 5 rounds, we only focus on the preimage attack on 5-round GOST-256 compression function. The exact procedures of this phase will be provided later as well.

Phase 3. After phase 1 and phase 2, we get hold of the 2^{1024}-multicollisions, and also have the message checksum desired. In order to produce the desired checksum, we make use of the methods of [19,33], which first write the desired checksum in binary form and then use the unique structure of the cascaded collision pairs to produce each item of the binary expression. We refer to [33] for details of this phase.

After all three phases succeed, we manage to generate a preimage for 5-round GOST-256.

3.2 Phase 1. Construct the Multicollisions

This phase needs to generate 2^{1024}-multicollisions which can be decomposed into 512 pairs of 4-multicollisions, namely, $(M_{3i}, M'_{3i})||M_{3i+1}||(M_{3i+2}, M'_{3i+2})$ for $i = 0, 1, ..., 511$ which satisfy Equation (1). In order to do so,

we need to utilize the collision attack on 5-round GOST-256 compression function in [45]. The truncated differential trail is depicted in Fig. 3, and the rebound attack [37] is used to derive the collisions. The layers of the inbound phase, which are covered with the SuperSBox technique [21,32], are denoted in red, while the layers of the outbound phase are denoted in blue. We refer to [45] for more descriptions, and only present the results: it requires 2^{120} time and 2^{64} memory to find a collision, and the expected number of collisions with a fixed input chaining value is 2^8.

Now we show how to generate the 4-multicollisions $(M_{3i}, M'_{3i})||M_{3i+1}||(M_{3i+2}, M'_{3i+2})$ for a specific i. As depicted in Fig. 2, the exact procedures are as follows:

1. From the chaining value h_{3i}, with the aid of the rebound attack and the SuperSBox technique, we find a collision pair (M_{3i}, M'_{3i}) on 5-round GOST-256 at the cost of 2^{120} time and 2^{64} memory. Notice that the only difference between M_{3i} and M'_{3i} lies in a single cell whose exact position is determined by the value of i, and the difference can take at most 2^8 possible values.
2. From h_{3i+1}, we randomly choose the value of M_{3i+1} and compute the output chaining value h_{3i+2}.
3. From h_{3i+2}, we find a collision pair (M_{3i+2}, M'_{3i+2}) on 5-round GOST-256. Then we check whether $M_{3i} - M'_{3i} = M'_{3i+2} - M_{3i+2} + 2^{511-i}$ holds. Note that the only difference between M_{3i+2} and M'_{3i+2} lies in the same active cell as well, thus this condition holds with probability 2^{-8}.
4. From a fixed h_{3i+2}, 2^8 collision pairs (M_{3i+2}, M'_{3i+2}) can be generated, thus we expect to find one such pair which makes Equation (1) hold. However, if no desired pair is found for the fixed h_{3i+2}, we can utilize the two-block strategy, and go to step 2 with another value of M_{3i+1}, then redo step 3. As a result, step 3 will eventually succeed and the 4-multicollisions can be constructed.

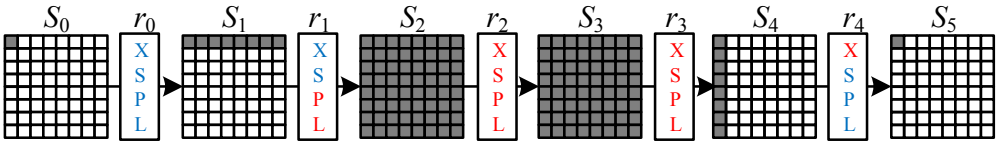

Fig. 3. Collision attack on 5-round GOST-256 compression function

Since the position of the active cell can be placed in any cell from the 64 possible positions, we can generate the 4-multicollisions for any item 2^{511-i} where $i = 0, 1, ..., 511$. For instance, the differential trail in Fig. 3 is adopted to generate the 4-multicollisions for the item 2^{511-i} where $i = 0, 1, ..., 7$. Finally, after enumerating all 64 positions of the active cell and repeating the above procedures 512 times, the 2^{1024}-multicollisions are constructed with $512 \times (2^{120} + 2^{120+8}) \approx 2^{137}$ time and 2^{64} memory.

3.3 Phase 2. Invert the Output Transformation

As depicted in Fig. 2, given the target digest h_X and the chaining value h_{1538}, this phase inverts the last compression function call of the output transformation and derives the value of the message checksum $\Sigma(M)$. In order to do so, we utilize the MitM preimage attack on AES-like compression functions. Due to the close relevances, we give brief descriptions of the MitM preimage attack on AES-like compression functions. Then we discuss how the truncation pattern of GOST-256 influences our attack. Finally, we provide the optimal attack parameters derived.

The MitM Preimage Attack Framework on AES-like Compression Functions. The MitM preimage attack was first introduced by Aoki and Sasaki in [4]. 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. Several advanced techniques to further improve the basic attack are developed, such as *partial matching* [4], *initial structure* [41], *indirect partial matching* [3], *bicliques* [30] and *differential MitM attack* [31].

In [40], 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 [44]. Thanks to the delicate descriptions of the MitM preimage attack framework on AES-like compression functions presented in [44], 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 [42], Sasaki *et al.* introduced the *guess-and-determine* approach to extend the basic attack by one more round. Based on these previous results, the basic MitM preimage attack framework on AES-like compression functions is achieved.

As shown in Fig. 4, without loss of generality, we use the chunk separation of 5-round GOST-256 in order to further illustrate the details of the attack framework. We first define several necessary notations which are used throughout this paper. These notations are also depicted in Fig. 4 for instance.

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 rows (or columns) in the initial structure.
r	Number of red rows (or columns) in the initial structure.
c	Number of constant cells in each row (or column) in the initial structure.
g	Number of guessed rows (or columns) in the backward (or forward) computation, which are denoted in purple. See Fig. 6 for instance.
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.
TIME	Time complexity of the preimage attack.
MEMORY	Memory requirement of the preimage attack.

The attack procedures can be further divided into five steps which are illustrated as follows, and the interested readers are referred to [40,44] for more details of the construction of the initial-structure and indirect-partial-matching through the MixRow layer, and [42] for the illustrations of the guess-and-determine strategy.

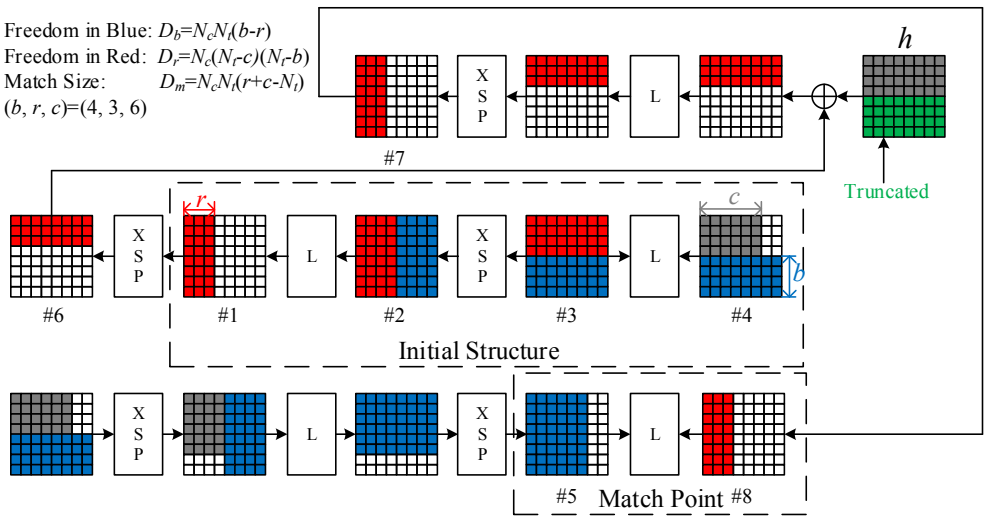

Fig. 4. Preimage attack on 5-round GOST-256 compression function

Step 1. Initial Structure. Choose random values for the constants which are used in the transformations between states #1 ↔ #2, and states #3 ↔ #4. Following the linear relations of the MixRow operation, compute the values 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 #8. Although not shown in Fig. 4, the guess-and-determine strategy can be utilized at state #7 in order to extend the attack by one more round. Note that we do not consider the truncation operation currently when describing this general attack framework. The impacts brought by

the truncation pattern through the feed-forward operation during the backward computation phase will be discussed later.

Step 4. Indirect-Partial-Matching through the MixRow Layer. The indirect-partial-matching is performed between states #5 $\leftrightarrow$ #8 with partial known information of the red and blue cells from both directions by exploiting linear relations of the MixRow operation.

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.

Deriving the Attack Parameters. As discussed in previous works, the attack parameters, *e.g.*, the freedom degrees of the forward chunk D_b and the backward chunk D_r can be easily represented and enumerated with several predefined integer parameters (b, r, c) as shown in Fig. 4. However, the size of the match point D_m is rather ad-hoc, and needs to be treated carefully mainly due to impacts introduced by the truncation operation and the feed-forward operation.

On the Match Point. For narrow pipe constructions, the state size equals with the digest size, and the indirect-partial-matching contributes to the match of the entire digest. As studied in previous works [40,44,42], the complexities can be denoted as follows (n denotes the digest size):

$$\begin{aligned} \texttt{TIME} &= 2^n(2^{-D_r} + 2^{-D_b} + 2^{-D_m}), \\ \texttt{MEMORY} &= min\{2^{D_r}, 2^{D_b}\}. \end{aligned} \tag{2}$$

However, since `GOST-256` is a wide pipe design which adopts the `ChopMD` mode, we have to ensure that the indirect-partial-matching operation between the forward blue chunk and the backward red chunk is only carried on the preserved grey cells at state #9, and is not related to the truncated green cells at state #9. As can be seen from Fig. 4, the red cells at state #7 are fully determined after the feed-forward operation of state #6 and state #9, and only the grey preserved cells at state #9 are involved in this feed-forward process, while the green truncated cells are totally irrelevant. Consequently, only the grey preserved cells of the digest at state #9 are matched through the indirect-partial-matching layer, and the attack complexities for narrow pipe constructions still hold for the `GOST-256` case. Hence, for the `GOST-256` case, the attack complexities can still be represented as Equation (2).

Impacts of the Truncation. As depicted in Fig. 4, the truncation operation has direct influences on the matching part of the attack. More precisely, in the backward computation of the red chunks, the feed-forward operation can be conducted at state #6 or #7, and different selections will result in different attacks. However, note that the attack complexities still depend on the quartets, *i.e.*, (n, D_r, D_b, D_m), and thus the expressions of the complexities are the same as Equation (2).

Case 1. Feed-forward at State #6. After the feed-forward operation with state #6, all information of the r red rows at #6 will be preserved due to the row-wise truncation as long as $r \leq 4$. Consequently, the backward computation would proceed by one more round and the original attack framework would still work as long as $r \leq 4$ is satisfied. We exhaustively search all possible attack parameters for case 1, and the optimal parameters offer the attack with time complexity 2^{192} which is optimal.

Case 2. Feed-forward at State #7. After the feed-forward operation with state #7, only the first four rows of the r red columns at #7 will be preserved due to the truncation. Although the number of attacked rounds remains unchanged, the size of the match point[3] is reduced by half which might have negative impacts on the overall complexity. Actually, we exhaustively search all possible attack parameters for case 2, and the optimal parameters offer the attack with time complexity 2^{208} which is not optimal.

[3] Similar to the pseudo preimage attack on `Grøstl` [44], although the match point is cut to half by the feed-forward operation, the indirect-partial-matching is still only related to the preserved parts of the digest, and contains no information of the truncated parts.

Optimize Phase 2. Based on the previous discussions on the generic attack framework and the impacts introduced by the truncation, we choose state #6 as the position where the feed-forward operation is performed. Because (D_b, D_r, D_m) can be represented with (b, r, c), we can easily enumerate all possible values of (b, r, c) and search for the optimal attack parameters. The optimal chunk separation for 5-round GOST-256 is denoted in Fig. 4. Since the size of the target digest h_X is 256 bits, and we have 512 bits freedom degrees in $\Sigma(M)$, phase 2 will succeed with probability 1. Finally, according to the specific attack parameters in Fig. 4, we need 2^{192} time and 2^{64} memory to generate a preimage for 5-round GOST-256 compression function.

3.4 Phase 3. Generate the Preimage

We omit the details of this phase, since it follows almost the same procedures of the preimage attack in [33]. Note that this phase only needs several simple operations, thus the complexities are negligible.

3.5 Summarize the 5-Round Attack

Minimize the Memory Requirement. We can also minimize the memory requirement after a deeper look into the above attack. Since we have to store the 2^{1024}-multicollisions, the memory requirement is at least 2^{12} 512-bit blocks. We can launch the memoryless MitM preimage attack [29,42] in phase 2, and generate a preimage with 2^{208} time and negligible memory using the parameters $(b, r, c) = (5, 4, 6)$[4]. As for phase 1, the standard time/memory tradeoff can be utilized to reduce the memory requirement of the inbound phase as stated in [32, Appendix]. More precisely, referring to Fig. 3, the exact attack steps are as follows:

1. We choose 2^s differences at S_4 and propagate the differences to $P^{-1} \circ L^{-1}(S_4)$, then save the 2^s differences in sorted lists for each SuperSBox.
2. We choose a random difference of $P^{-1} \circ L^{-1}(S_3)$, and compute the corresponding difference of $X(S_3)$.
3. For each specific SuperSBox, we enumerate all 2^{64} values according to the difference of $X(S_3)$, and compute the corresponding output differences. There are 2^s differences in the saved list, thus we expect to generate 2^s solutions (one solution average for each difference in the list) for this SuperSBox since we need to match a 64-bit difference. We can repeat step 3 for each SuperSBox independently.

Finally, we can generate 2^s solutions for the inbound phase with 2^{64} time and 2^s memory, or equivalently a single solution can be generated with 2^{64-s} time and 2^s memory. Combining the outbound phase, it takes $2^{64-s+120} = 2^{184-s}$ time and 2^s memory to derive a 5-round collision. The standard SuperSBox technique sets $s = 64$, and we could find a solution for the inbound phase with average time complexity one. But now we set $s = 12$, and we need 2^{64} time to find 2^{12} solutions for the inbound phase and 2^{172} time for a 5-round collision. Finally, in order to build and store the 2^{1024}-multicollisions, we need $512 \times (2^{172} + 2^{172+8}) \approx 2^{189}$ time and 2^{12} memory which is still not the bottleneck of the time complexity.

Complexity Analysis. Combining the complexities of the three phases, the preimage attack on 5-round GOST-256 requires 2^{192} time and 2^{64} memory. If we aim to minimize the memory requirement, the attack requires 2^{208} time and 2^{12} memory.

3.6 Extend the Preimage Attack to More Rounds

The transposition operation P of GOST is an involution, namely, $P(St) = P^{-1}(St)$ holds for any 512-bit state St. The collision and distinguishing attacks on GOST in [33] benefits from this fact with more attacked rounds. Now we show that the preimage attacks can also be extended by 1.5 more rounds by exploiting this property. More precisely, if we omit the MixRow operation L in the last round, due to the fact that the other round operations, namely, X, S, P are cell-independent operations, the backward computation can be further extended by 1.5 more rounds, and we are able to launch preimage attacks on 6.5-round GOST-256 and 7.5-round GOST-512. Although these improved attacks require the omission of the MixRow operation L in the last round which seems inappropriate for a primitive like GOST, they are worth mentioning since they further clarify that any operations with the involution property are not optimal candidates as the transposition operation of AES-like hash primitives. As a counter-example, we cannot extend the previous preimage attacks on 6-round Whirlpool [42] to 7.5 rounds by omitting the last MixRow operation, because the ShiftColumn operation of Whirlpool is not an involution.

[4] Note that $r = 4$ satisfy the above requirement $r \leq 4$ of Case 1.

Preimage Attack on 6.5-Round GOST-256. There are three phases in the preimage attack on 6.5-round GOST-256. The main ideas of each phase are identical to the 5-round attack, thus we only provide brief descriptions of the attack, and omit more specified details.

In phase 1, we generate 2^{1024}-multicollisions which can be decomposed into 512 pairs of 4-multicollisions similar to the 5-round preimage attack. In order to do so, we need to utilize the collision attack on 6.5-round GOST-256 compression function in [33]. The truncated differential trail is depicted in Fig. 5. By choosing different locations of the active column, we can build the 2^{1024}-multicollisions with $512 \times 2^{184} = 2^{193}$ time and 2^{64} memory by repeating the collision attack on the compression function 512 times.

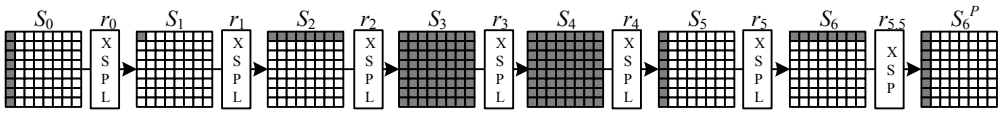

Fig. 5. Collision attack on 6.5-round GOST-256 compression function

In phase 2, we invert the last compression function call in the output transformation. We remove the MixRow operation L in the last round, and the optimal chunk separation for the 6.5-round MitM preimage attack is depicted in Fig. 6, and we can find a preimage $\Sigma(M)$ with complexities (TIME,MEMORY) $= (2^{232}, 2^{120})$.

Phase 3 is the same as the third phase of the 5-round preimage attack. Combining all 3 phases, it would require 2^{232} time and 2^{120} memory to generate a preimage for 6.5-round GOST-256. It is notable that although there exists a collision attack on 7.5-round GOST compression function [33] which can be utilized to build the 2^{1024}-multicollisions adopted in phase 1, we are not able to launch a preimage attack on 7.5-round GOST-256 due to its truncation operation.

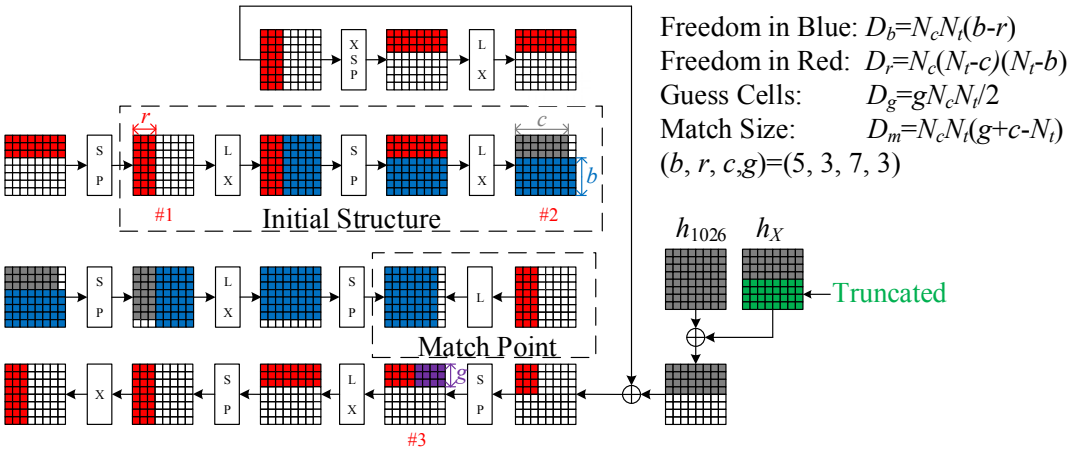

Fig. 6. Preimage attack on 6.5-round GOST-256 compression function

Preimage Attack on 7.5-Round GOST-512. The preimage attack on 7.5-round GOST-512 consists of three phases as well. The first and the last phases are the same as [33], while the improvement is carried out on the second phase. We remove the MixRow operation L in the last round, and the chunk separation for the second phase is depicted in Fig. 7. Finally, It requires 2^{496} time and 2^{64} memory to generate a preimage for 7.5-round GOST-512. We can also launch a memoryless variant of this attack following the methods in [33], and it requires 2^{504} time and 2^{11} memory to generate a preimage for 7.5-round GOST-512.

4 Discussions: Impacts of the Truncation

As indicated in Section 3.3, the truncation pattern has impacts on the optimal position of the feed-forward operation, thus influences the size of the match point and results in different attack complexities. In this section, we further look into this problem, and evaluate the impacts of four representative truncation patterns on the resistance of the MitM preimage attack against AES-like compression functions (with 8×8-byte state

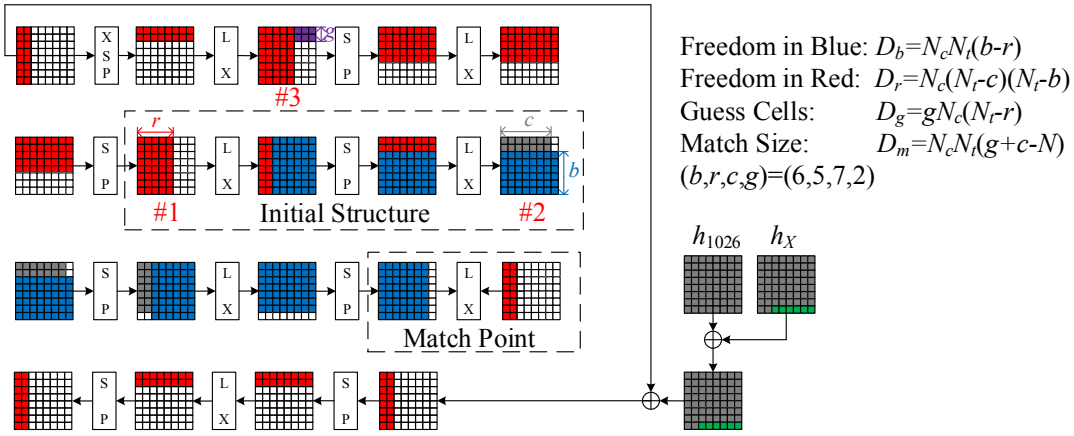

Fig. 7. Preimage attack on 7.5-round GOST-512 compression function

and the ChopMD mode). We need to stress that only the truncation patterns which truncate 512-bit chaining values to 224, 256 or 384 bits digests are considered, since they are recommended by NIST in the SHA-3 competition, and have many instances in practice. For each investigated truncation pattern, we choose carefully the position of the feed-forward operation in order to maximize the match point and derive the optimal attack. As applications of our investigations, we slightly improve the previous pseudo preimage attacks on reduced-round Grøstl-256.

4.1 The Generic MitM Preimage Attack Framework on AES-like Compression Functions

AES-like Compression Functions. We first define what we call an AES-like block cipher (or permutation). An AES-like block cipher applies N_r rounds of a round function to update a plaintext represented as a matrix of N_t columns and N_t rows, where each of the N_t^2 cells has a size of N_c bits. Since the key of the block cipher is always fixed in the presented analysis model, we omit specification of the key schedule and denote the round keys as constants. The round function consists 4 sequent operations, the AddRoundConstant operation (AC), the SubBytes operation (SB) which applies a $N_c \times N_c$ nonlinear bijective SBox to each cell, the ShiftRow operation (SR)[5] which rotationally moves each cell in the x-th row by x positions to the left in its own row, and the MixColumn operation (MC) which mixes all the columns with an MDS matrix. Note that for the AES-like block ciphers considered in this paper, $N_t = 8, N_c = 8$.

When an AES-like block cipher is adopted as the underlying block cipher to build a compression function with the PGV modes, we call such a compression function an AES-like compression function.

The Attack Model. We have checked that for all the 12 PGV modes, either a preimage or a pseudo preimage attack on the specific AES-like compression function can be derived with the MitM preimage attack. Hence, without loss of generality, we only study the pseudo preimage resistance of AES-like compression functions with the Davies-Meyer mode. Similar analyses can be carried on the remaining PGV modes.

As briefly summarized in Section 3.3, the MitM preimage attack on AES-like compression functions can reach 5 and 6 rounds combining several techniques like initial structure, the guess-and-determine strategy and indirect-partial matching. We first focus on the 6-round MitM preimage attack, and the typical chunk separation for the 6-round attack is depicted in Fig. 8. If the size of the digest is n bits, then the time complexity for the 6-round attack is computed as follows:

$$\mathtt{T} = 2^n(2^{-D_r} + 2^{D_g-D_b} + 2^{D_g-D_m}).$$

[5] There are many alternatives to achieve transposition, such as the matrix transposing operation of GOST which is found to be non-optimal in [33] and this paper. For the sake of simplicity, we only consider the AES-like ShiftRow operation (SR).

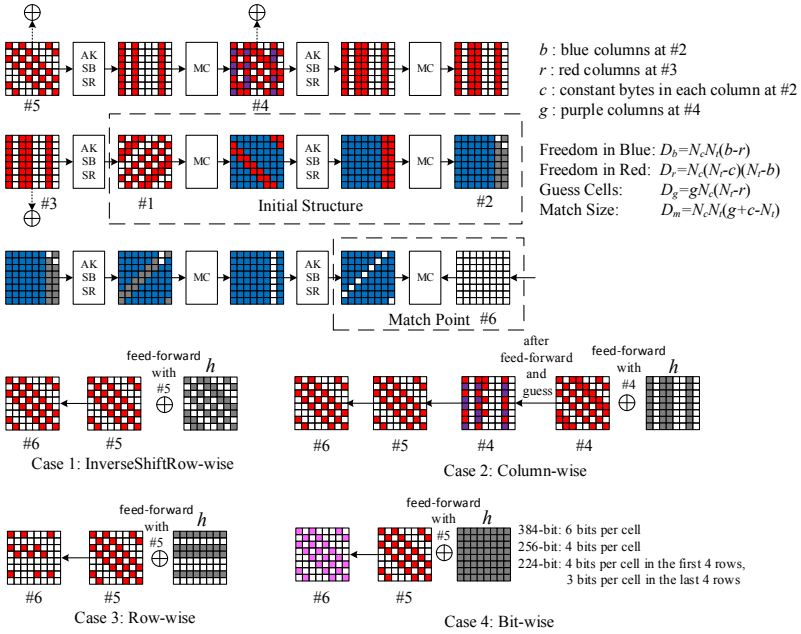

Fig. 8. MitM preimage attack on 6-round AES-like compression function

Assessment Criterion. In order to measure the impacts of different truncation patterns on the resistance of the MitM preimage attack, we define **the advantage Adv of the time complexity for the dedicated MitM preimage attack over the brute-force attack** as follows:

$$\mathtt{Adv} = log(\mathtt{T}_{\mathtt{MitM}}) - log(\mathtt{T}_{\mathtt{Brute}}).$$

Thus, the advantage for the 6-round attack can be denoted as follows:

$$\mathtt{Adv} = min\{D_r, D_b - D_g, D_m - D_g\}.$$

If no truncation is performed, the corresponding advantage is denoted with $\mathtt{Adv}_{\mathtt{6R}}^{\mathtt{Full}}$. We search for all possible chunk separations, and derive that $\mathtt{Adv}_{\mathtt{6R}}^{\mathtt{Full}} = 16$.

For an AES-like compression function truncating from 512 bits to n bits, we denote the corresponding advantage for the 6-round attack with $\mathtt{Adv}_{\mathtt{6R}}^{\mathtt{n}}$. For a specific truncation pattern, if $\mathtt{Adv}_{\mathtt{6R}}^{\mathtt{n}} < \mathtt{Adv}_{\mathtt{6R}}^{\mathtt{Full}}$ holds, it certainly narrows (even eliminates) the gap between the brute-force attack and the dedicated MitM preimage attack, thus strengthens the compression function in terms of the resistance of this type of attack. Based on this observation, we expect to find the optimal truncation patterns among all the possible patterns. However, the search space is to large[6], and we do not come up with advanced techniques to significantly reduce the search space at the moment. Instead of enumerating all possible truncation patterns, we study 4 representative patterns which are convenient in both software and hardware implementations, and show that 2 of the 4 patterns provide extra strength against the MitM preimage attack.

4.2 Case Study: Preimage Resistance of Four Truncation Patterns

Firstly, we would like to point out two observations which are crucial in our analyses as shown in Fig. 8. The first observation comes from the simple fact that the actual positions of the red cells can be freely chosen to a certain extent in the backward computation. For instance, the attacker can freely choose the positions of the r red columns from the eight columns at state #3. The second observation comes from the fact that the feed-forward operation can be performed at the start of any round in the backward computation. In fact, these observations have already been adopted in our preimage attack on 5-round GOST-256.

[6] For example, if we truncate from 512 bits to 256 bits, there are a total $C_{512}^{256} \approx 2^{507}$ patterns. If we consider byte-wise truncation, there are still $C_{64}^{32} \approx 2^{60.67}$ patterns. Although we find several trivial methods to further classify all the truncation patterns, the search space is still too large for a practical implementation.

Observation 1. *As depicted in Fig. 8, if there are r (resp. g) red (resp. purple) columns at state #3 (resp. #4), then the r (resp. g) red (resp. purple) columns can be randomly chosen from the N_t columns of state #3 (resp. #4), and there are overall $C^r_{N_t}$ (resp. $C^g_{N_t}$) possible patterns for the positions of the red (resp. purple) columns.*

Observation 2. *In the chunk separation denoted in Fig. 8, the feed-forward operation can be carried out at the begining of any round in the backward computation, namely, state #3, #4 or #5.*

Security Analyses of the Four Representative Truncation Patterns. The 4 truncation patterns considered are denoted in Fig. 8. For the first three cases, the grey cells of h are preserved while the white cells of h are truncated. For the last case, only partial bits of each cell of h are preserved while the remaining bits are truncated. Due to the space limit, we only provide detailed analyses of the situation where the 512-bit state is truncated to the 256-bit digest. The 384-bit and the 224-bit cases can be analyzed analogously, and the details are hence omitted. The overall results are summarized in Table 2.

Case 1 performs the *InverseShiftRow*-wise truncation, and there are overall $C^{N_t/2}_{N_t}$ specific *InverseShiftRow*-wise truncation patterns. As depicted in Fig. 8, according to Observation 2, if the feed-forward operation is carried out at state #5, no information of the red cells will be lost after the feed-forward, and the backward computation can proceed to the match point at state #6. One may say that there are $C^{N_t/2}_{N_t}$ truncation patterns for the *InverseShiftRow* truncations, and a different selection of these patterns may not preserve this property. However, as identified in Observation 1, if $N_t/2 \geq g$, we can choose the positions of the purple columns at state #4 according to the specific truncation pattern adopted, so that we do not miss any information of the red cells after the feed-forward. In other words, since the positions of the purple columns can be adaptively chosen, all the *InverseShiftRow*-wise truncation patterns are equivalent in this sense. Consequently, the values of (D_b, D_r, D_g, D_m) for the *InverseShiftRow* truncations remain unchanged as the case when no truncation is performed, and we can directly conclude that

$$\mathtt{Adv}^{256}_{\mathtt{Case1,6R}} = \mathtt{Adv}^{\mathtt{Full}}_{\mathtt{6R}} = 16.$$

The truncation patterns of case 1 do not narrow the gap between the 6-round MitM preimage attack and the brute-force attack.

Case 2 performs the *Column*-wise truncation. Similar to case 1, if the feed-forward operation is carried out at state #4, we can adaptively choose the corresponding positions of the purple columns at state #4 from all $C^g_{N_t}$ patterns according to the specific truncation pattern, and make sure that no information of the red cells will be lost after the feed-forward if $N_t/2 \geq g$. Then we are able to proceed the backward computation by one more round to the match point at state #6 (which is also state #5 in this case). We conclude that

$$\mathtt{Adv}^{256}_{\mathtt{Case2,6R}} = \mathtt{Adv}^{\mathtt{Full}}_{\mathtt{6R}} = 16,$$

and the truncation patterns of case 2 do not narrow the gap between the 6-round MitM preimage attack and the brute-force attack either.

Case 3 performs the *Row*-wise truncation. The feed-forward should be carried out at state #5 instead of state #4 and #3, since it requires full columns to compute a full round backwards, and the feed-forward operation certainly disturbs the full columns. By easily and exhaustively enumerating the values of (b, r, c, g), we can directly compute the values of (D_b, D_r, D_g). However, the matching part is affected by the truncation, and therefore the size of the match point D_m is rather ad-hoc. More precisely, as can be seen from Case 3 of Fig. 8, we lose partial information of the red cells at state #6 compared to state #5 due to the truncation after the feed-forward operation at state #5. Luckily, the overall number of the positions for the purple columns of state #4 is only $C^g_{N_t}$ which can be exhaustively searched. Moreover, since the purple columns of state #4 can be adaptively chosen, all the $C^{N_t/2}_{N_t}$ Row-wise truncation patterns are equivalent. Consequently, for each valid quartet (b, r, c, g), we can enumerate all possible positions of the purple columns, derive the

maximum size of the match point D_m, and then compute the corresponding time complexity for the specific quartet (b, r, c, g). After testing all valid values of (b, r, c, g), we obtain the optimal chunk separations and the minimized time complexity which are summarized in Table 2, and derive that

$$\texttt{Adv}^{256}_{\texttt{Case3,6R}} = 8 < \texttt{Adv}^{\texttt{Full}}_{\texttt{6R}} = 16.$$

Based on this result, it is obvious that all the truncation patterns of case 3 do narrow the gap between the 6-round MitM preimage attack and the brute-force attack, thus provide extra strength against this type of attack.

Case 4 performs the *Bit*-wise truncation, *i.e.*, 4 bits are selected from each cell. Due to the same reason in case 3, the feed-forward should be carried out at state #5, and the backward computation proceeds until reaching the match point at state #6. We denote the known cells at #6 in pink since only partial bits are preserved due to the bit-wise truncation. The size of the match point[7] can be denoted as $N_t(gN_c/2 + cN_c - N_tN_c)$ if $gN_c/2 + cN_c - N_tN_c > 0$, otherwise no match point will be derived. We search for all possible values of (b, r, c, g), and compute the corresponding time complexities. However, no chunk separation is obtained which would make the MitM preimage attack faster than exhaustive search. Thus we conclude that

$$\texttt{Adv}^{256}_{\texttt{Case3,6R}} = 0,$$

and the truncation pattern of case 4 effectively prevents the 6-round MitM preimage attack for the 256-bit digest.

Influences on Other Types of Attacks. A wide range of collision-like attacks on `AES`-like hash primitives are based on multi-block procedures, and mainly focus on the prior message blocks. The collisions of these attacks are derived by appending identical message blocks which satisfy padding to the previous colliding message blocks. However, the truncation operation is normally performed only once at the last message block, and therefore have no impacts on these collision-like attacks which mainly target prior message blocks. Thus, the stronger truncation patterns discussed above do not weaken the resistance against these typical collision-like attacks. Regarding other types of attacks, the discussions seem to be rather ad-hoc, and depend on the specific structures of the underlying primitives. However, we find that the stronger truncation patterns do not weaken the security properties of `GOST-256`, `Grøstl-256` and the truncated variants of `Whirlpool` with respect to existing attacks on these primitives, and we believe this fact should hold for a wide range of other primitives with similar structures.

Overview of the Four Cases. Due to the fact that the truncation operation is normally much simpler than the compression functions, and is most commonly processed only once at the end of the hash computation, the extra efforts to implement these stronger truncation patterns are negligible (especially when processing long input messages). Taking both security properties and implementation issues into account, the truncation patterns of case 3 and case 4 are promising alternatives in future designs of `AES`-like hash primitives where truncation needs to be performed.

Applications to `Grøstl-256`. For `Grøstl-256`, the column-wise truncation operation in the output transformation corresponds to case 2. As depicted in Fig. 9, we can carry out the feed-forward operation earlier in the backward computation without losing any information of the red cells. As a result, the time complexity of the previous 6-round (resp. 5-round) attack can be decreased from 2^{248} [46] (resp. 2^{208} [44]) to 2^{240} (resp. 2^{192}) output transformation computations. The complexities of the pseudo preimage attacks on 5- and 6-round `Grøstl-256` in [44,46] can also be slightly improved, we omit these details. Based on these slightly improved attacks, we recommend `Grøstl-256` to perform the stronger row-wise or bit-wise truncations.

[7] We find that the relations between the red and blue cells which are utilized in the indirect-partial-matching can be written in bit-wise forms.

Table 2. Summary of the results for the four truncation cases, the truncation patterns which resist the MitM preimage attack are denoted in red.

Case	Rounds Attacked	Digest Size	Advantage	Parameters I $\#(b, r, c, g)$	Parameters II $\#(D_b, D_r, D_g, D_m)$	Time	Memory
No Truncation	6	512	16	(6,5,7,2)	(64,16,48,64)	2^{496}	2^{64}
	5	512	64	(4,3,6,-)	(64,64,-,64)	2^{448}	2^{64}
Case 1 and Case 2	6	384	16	(6,5,7,2)	(64,16,48,64)	2^{368}	2^{64}
		256	16	(6,5,7,2)	(64,16,48,64)	2^{240}	2^{64}
		224	16	(6,5,7,2)	(64,16,48,64)	2^{208}	2^{64}
	5	384	64	(4,3,6,-)	(64,64,-,64)	2^{320}	2^{64}
		256	64	(4,3,6,-)	(64,64,-,64)	2^{192}	2^{64}
		224	64	(4,3,6,-)	(64,64,-,64)	2^{160}	2^{64}
Case 3	6	384	8	(7,6,7,2)	(64,8,32,48)	2^{376}	2^{40}
		256	8	(7,6,7,3)	(64,8,48,64)	2^{248}	2^{56}
		224	8	(7,6,7,3)	(64,8,48,64)	2^{216}	2^{56}
	5	384	48	(5,4,6,-)	(64,48,-,128)	2^{336}	2^{48}
		256	48	(5,4,6,-)	(64,48,-,64)	2^{208}	2^{48}
		224	48	(5,4,6,-)	(64,48,-,64)	2^{176}	2^{48}
Case 4	6	384	8	(7,6,7,3)	(64,8,48,80)	2^{376}	2^{56}
		256	-	-	-	2^{256}	$\mathcal{O}(1)$
		224	-	-	-	2^{224}	$\mathcal{O}(1)$
	5	384	48	(5,4,6,-)	(64,48,-,64)	2^{336}	2^{48}
		256	32	(4,3,7,-)	(64,32,-,32)	2^{224}	2^{32}
		224	24	(5,4,7,-)	(64,24,-,48)	2^{200}	2^{24}

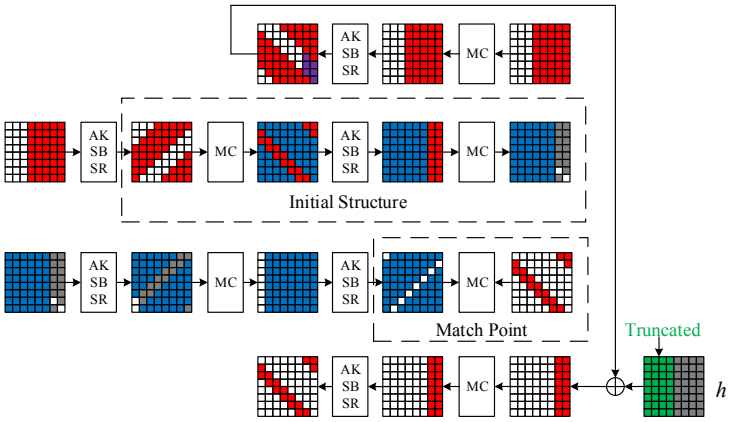

Fig. 9. Pseudo preimage attack on 6-round Grøstl-256 output tansformation

5 Conclusion and Open Problems

In this paper, we present improved preimage attacks on the reduced-round GOST hash function family by combining the dedicated collision attack and the MitM preimage attack on the GOST compression function. As far as we know, our result is the first preimage attack on GOST-256 at the hash function level. We also investigate the impacts of four representative truncation patterns on the resistance of the MitM preimage attack against AES-like compression functions, and propose two strengthened truncation patterns, which make it more difficult to launch the MitM preimage attack.

However, due to the large search space, we are not able to study the impacts of all possible truncation patterns at the moment. An interesting and open problem is to seek for advanced approaches to efficiently enumerate all possible truncation patterns, and investigate their impacts on various security properties of many AES-based hash primitives.

References

1. AlTawy, R., Kircanski, A., Youssef, A.: Rebound Attacks on Stribog. In: Lee, H.S., Han, D.G. (eds.) ICISC 2013. LNCS, vol. 8565, pp. 175–188. Springer International Publishing (2014), also available at http://eprint.iacr.org/2013/539

2. AlTawy, R., Youssef, A.: Preimage Attacks on Reduced-Round Stribog. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT 2014. LNCS, vol. 8469, pp. 109–125. Springer International Publishing (2014)
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.: 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)
5. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 Proposal BLAKE. Submission to NIST (Round 3) (2010), http://131002.net/blake/
6. Barreto, P., Rijmen, V.: The Whirlpool Hashing Function. Submitted to NESSIE (September 2000) (2000), http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html
7. Benadjila, R., Billet, O., Gilbert, H., Macario-Rat, G., Peyrin, T., Robshaw, M., Seurin, Y.: SHA-3 Proposal: ECHO. Submission to NIST (updated) (2009), http://crypto.rd.francetelecom.com/ECHO/
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak Reference. Submission to NIST (Round 3) (2011), http://keccak.noekeon.org/Keccak-reference-3.0.pdf
9. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278 (2007), http://eprint.iacr.org/2007/278
10. Biham, E., Dunkelman, O.: The SHAvite-3 Hash Function. Submission to NIST (Round 2) (2009), http://www.cs.technion.ac.il/~orrd/SHAvite-3/
11. Chang, D., Nandi, M.: Improved Indifferentiability Security Analysis of chopMD Hash Function. In: Nyberg, K. (ed.) FSE 2008, LNCS, vol. 5086, pp. 429–443. Springer, Heidelberg (2008)
12. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård Revisited: How to Construct a Hash Function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005)
13. Daemen, J., Rijmen, V.: The Wide Trail Design Strategy. In: Honary, B. (ed.) IMACC 2001, LNCS, vol. 2260, pp. 222–238. Springer, Heidelberg (2001)
14. 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)
15. Dolmatov, V., Degtyarev, A.: GOST R 34.11-2012: Hash Function (2013)
16. Dolmatov, V., Degtyarev, A.: Request for Comments 6986: GOST R 34.11-2012: Hash Function. Internet Engineering Task Force (IETF) (2013), http://www.ietf.org/rfc/rfc6986.txt
17. Information Protection and Special Communications of the Federal Security Service of the Russian Federation: GOST R 34.11-94, Information Technology Cryptographic Data Security Hashing Function (1994), in Russian
18. Information Protection and Special Communications of the Federal Security Service of the Russian Federation: GOST R 34.11-2012, Information Technology Cryptographic Data Security Hashing Function (2012), http://www.tc26.ru/en/GOSTR3411-2012/GOST_R_34_11-2012_eng.pdf
19. Gauravaram, P., Kelsey, J.: Linear-XOR and Additive Checksums Don't Protect Damgård-Merkle Hashes from Generic Attacks. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 36–51. Springer, Heidelberg (2008)
20. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl–a SHA-3 Candidate. Submission to NIST (Round 3) (2011), http://www.groestl.info/Groestl.pdf
21. 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)
22. Guo, J., Jean, J., Leurent, G., Peyrin, T., Wang, L.: The Usage of Counter Revisited: Second-Preimage Attack on New Russian Standardized Hash Function. In: Joux, A., Youssef, A. (eds.) SAC 2014, LNCS, vol. 8781, pp. 195–211. Springer International Publishing (2014), also available at http://eprint.iacr.org/2014/675
23. International Organization for Standardization: ISO/IEC 10118-3:2004: Information technology - Security techniques - Hash-functions - Part 3: Dedicated hash-functions (2004)
24. 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)
25. Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 306–316. Springer, Heidelberg (2004)
26. Kazymyrov, O., Kazymyrova, V.: Algebraic Aspects of the Russian Hash Standard GOST R 34.11-2012. Cryptology ePrint Archive, Report 2013/556 (2013), http://eprint.iacr.org/2013/556
27. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 183–200. Springer, Heidelberg (2006)
28. 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)
29. Khovratovich, D., Nikolić, I., Weinmann, R.P.: Meet-in-the-Middle Attacks on SHA-3 Candidates. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 228–245. Springer, Heidelberg (2009)
30. 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)

31. 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)
32. 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)
33. 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
34. Ma, B., Li, B., Hao, R., Li, X.: Improved (Pseudo) Preimage Attacks on Reduced-Round GOST and Grøstl-256 and Studies on Several Truncation Patterns for AES-like Compression Functions (Full Version). Cryptology ePrint Archive (2015)
35. Mendel, F., Pramstaller, N., Rechberger, C.: A (Second) Preimage Attack on the GOST Hash Function. In: Nyberg, K. (ed.) FSE 2008, LNCS, vol. 5086, pp. 224–234. Springer, Heidelberg (2008)
36. Mendel, F., Pramstaller, N., Rechberger, C., Kontak, M., Szmidt, J.: Cryptanalysis of the GOST Hash Function. In: Wagner, D. (ed.) CRYPTO 2008, LNCS, vol. 5157, pp. 162–178. Springer, Heidelberg (2008)
37. 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)
38. 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)
39. 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
40. 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)
41. 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)
42. 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)
43. Wang, Z., Yu, H., Wang, X.: Cryptanalysis of GOST R Hash Function. Cryptology ePrint Archive, Report 2013/584 (2013), http://eprint.iacr.org/2013/584
44. 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)
45. Zou, J., Wu, W., Wu, S.: Cryptanalysis of the Round-Reduced GOST Hash Function. In: Lin, D., Xu, S., Yung, M. (eds.) Inscrypt 2013. LNCS, vol. 8567, pp. 309–322. Springer International Publishing (2014)
46. Zou, J., Wu, W., Wu, S., Dong, L.: Improved (Pseudo) Preimage Attack and Second Preimage Attack on Round-Reduced Grøstl. Cryptology ePrint Archive, Report 2012/686 (2012), http://eprint.iacr.org/2012/686