Exploiting Weak Diffusion of Gimli: A Full-Round Distinguisher and Reduced-Round Preimage Attacks

Fukang Liu[1,3], Takanori Isobe[2,3], Willi Meier[4]

[1] Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
liufukangs@163.com
[2] National Institute of Information and Communications Technology, Japan
[3] University of Hyogo, Hyogo, Japan
takanori.isobe@ai.u-hyogo.ac.jp
[4] FHNW, Windisch, Switzerland
willimeier48@gmail.com

Abstract. The Gimli permutation was proposed in CHES 2017, which is distinguished from other well-known permutation-based primitives for its cross-platform performance. One main strategy to achieve such a goal is to utilize a sparse linear layer (Small-Swap and Big-Swap), which occurs every two rounds. In addition, the round constant addition occurs every four rounds and only one 32-bit word is affected by it. The above two facts have been exploited by Liu-Isobe-Meier to mount a distinguishing attack on 14 rounds of Gimli permutation with time complexity 1 by utilizing the internal difference. Inspired by the 14-round distinguisher, we demonstrate that it is feasible to extend it to the full Gimli permutation with time complexity 2^{129} by further taking advantage of its weak diffusion. The corresponding technique is named as hybrid zero internal differential since the internal difference and XOR difference are simultaneously traced. Our distinguisher can be interpreted as a variant of the common differential distinguisher and zero-sum distinguisher. Apart from the permutation itself, combined with some new properties of the SP-box, the weak diffusion can also be utilized to accelerate the preimage attacks on reduced Gimli-Hash and Gimli-XOF-128 with a divide-and-conquer method. As a consequence, the preimage attack on 2-round Gimli-Hash is practical and it can reach up to 5 rounds. For Gimli-XOF-128, our preimage attack can reach up to 9 rounds. To the best of our knowledge, this is the first attack on the full Gimli permutation and our preimage attacks on reduced Gimli-Hash and Gimli-XOF-128 are the best so far. Since Gimli is included in the second round candidates in NIST's Lightweight Cryptography Standardization process, we expect that our analysis can advance the understanding of Gimli. It should be emphasized that this work can not threaten the security of the hash scheme and authenticated encryption scheme built on Gimli.

Keywords: hash function, Gimli, Gimli-Hash, Gimli-XOF, preimage attack, distinguisher

1 Introduction

The Gimli permutation was proposed by Bernstein et al. in CHES 2017 [2]. As the designers claimed, Gimli is distinguished from other well-known permutation-based primitives for its cross-platform performance. The main strategy to improve the performance of Gimli is to process the 384-bit data in four 96-bit columns independently and make only a 32-bit word swapping among the four columns every two rounds. Soon after its publication, the security of such a design strategy received a doubt from Hamburg, who posted a paper [9] to explain how dangerous such a strategy would be.

Like the AES and SHA-3 competition, NIST is currently holding a public lightweight cryptography competition, aiming at lightweight cryptography standardization [1]. Since Gimli has been included in the Round 2 candidates in NIST's Lightweight Cryptography Standardization process, it is of practical importance to further investigate its security, especially for its authenticated encryption (AE) scheme and hash scheme in the submitted Gimli document. It should be emphasized that the attack described in [9] is for an ad-hoc mode and mainly exploits the fact that there is occasional 32-bit word communication among the 4 columns. Such an attack [9] can not be directly applied to the submitted hash scheme or AE scheme.

Recently, a comprehensive study for reduced Gimli has been made by Liu-Isobe-Meier [11], covering the collision attack on Gimli-Hash, state-recovery attack on the AE scheme and distinguishing attack on the Gimli permutation. These attacks exploited the facts that there is occasional communication between the four columns of the Gimli state and that the constant addition operation occurs every four rounds. Moreover, several useful properties of the SP-box of Gimli have been revealed and become the basis of all the attacks in [11].

For the distinguishing attack on 14-round Gimli [11], only one input of a specific format is utilized. Consequently, we are curious whether it is possible to use more different inputs of the same format to construct a longer distinguisher, just as the zero-sum distinguisher which aims at finding a set of inputs such that the sum of the corresponding outputs is zero. In addition, since the preimage attack is not covered in [11], we are motivated to make a research in this direction.

Gimli-Hash is based on the well-known sponge structure [4,3], with 128-bit rate and 256-bit capacity. For such a small rate, it is challenging to devise a faster preimage attack on Gimli-Hash than the generic one, which requires 2^{128} time and 2^{128} memory. This is because the attacker has to utilize at least two message blocks to match a given 256-bit hash value. In other words, $2n$ rounds of the Gimli permutation need to be taken into account to efficiently find a preimage of n-round Gimli-Hash. Considering the progress in the cryptanalysis of Keccak [5], even with a relatively large rate, the currently best preimage attacks can only reach up to 4 rounds [8,12,10]. For Ascon [6], the preimage attack is much more

difficult due to the small rate. As a result, the designers could only mount a preimage attack [7] on up to 5 rounds of Ascon-XOF-64 with a rather high time complexity, which is almost close to exhaustive search. Especially, to demonstrate the efficiency of the new technique called linear structures [8] for the preimage attack on reduced Keccak, Guo et al. provided a practical preimage attack on 3-round SHAKE-128 as an extreme example.

Following the research on Keccak and Ascon, we believe it meaningful to apply our technique to both Gimli-XOF-128 and Gimli-Hash. On one hand, it can be used to demonstrate the limit of our technique. On the other hand, a comparison can be made between Gimli and other primitives regarding preimage resistance, especially for those selected in the second round in NIST's Lightweight Cryptography Standardization process.

Related Work. We noticed that there is an independent work on the distinguishing atttack on Gimli announced at the rump session of Eurocrypt 2020, where a full-round distinguisher is claimed to be found. According to our understanding, it shares a very similar idea with our distinguisher. However, the details on their full-round distinguisher are not clear yet.

Our Contributions. Leveraging the facts that there is little communication between the four columns of the Gimli state in the Gimli permutation and that the constant addition operation occurs only every four rounds, we extend the 14-round distinguisher as proposed in [11] to the full 24 rounds by utilizing a new technique called hybrid zero-internal-differential (ZID) distinguisher. Specifically, different from the 14-round distinguisher where only one input is used, two different inputs of a specific format will be used in our distinguisher. In addition, not only the symmetry in each internal state but also the symmetry between two different internal states generated by the two inputs will be simultaneously traced. Consequently, we could construct a distinguisher for 18 rounds of Gimli permutation with two different inputs and time complexity 2, which is further extended to the full 24 rounds with time complexity 2^{129} by using 2^{128} different input pairs.

In addition, we develop a divide-and-conquer method to accelerate the exhaustive search for the preimages of reduced Gimli-Hash and Gimli-XOF-128. It should be mentioned that similar ideas have appeared in the cryptanalysis of Gimli [9,11,14]. Therefore, this work can be viewed as an extension of previous techniques. However, as can be seen from our preimage attack on Gimli-Hash and Gimli-XOF-128, extra efforts are essential in order to attack as many rounds as possible. Especially for the preimage attack on Gimli-Hash, two message blocks of size 2^{256} have to be exhausted in less than 2^{128} time, which is quite a challenge. In addition, some new properties of the SP-box are revealed, one of which directly makes the preimage attack on 2-round Gimli-Hash practical. Our results are summarized in Table 1.

Organization. This paper is organized as follows. In section 2, we introduce the notations, the Gimli permutation, some useful properties of the SP-box , the

Table 1: The analytical results of reduced Gimli-Hash and Gimli-XOF-128

Target	Attack Type	Rounds	Memory	Time	Ref.
Permutation	distinguisher	18	negligible	2	Sec. 4.2
		21	negligible	2^{65}	Sec. 4.3
		24 (full round)	negligible	2^{129}	Sec. 4.4
Gimli-Hash	preimage	2	2^{32}	$2^{42.4}$	Sec. 6
		5	$2^{65.6}$	2^{96}	Sec. 7
Gimli-XOF-128	preimage	8	2^{70}	2^{104}	App. C
		9	2^{70}	2^{104}	Sec. 8.1

hash scheme Gimli-Hash and Gimli-XOF. In section 3, two new properties of the SP-box will be presented. The distinguishing attack on full Gimli is shown in section 4. The overview of our preimage attack on reduced Gimli-Hash is described in section 5. Then, we show the preimage attacks on 2 and 5 rounds of Gimli-Hash in section 6 and section 7, respectively. How to mount the preimage attack on up to 9 rounds of Gimli-XOF-128 is explained in section 8. Finally, the paper is concluded in section 9.

2 Preliminaries

In this section, we will present some notations, the description of the Gimli permutation and its applications to hashing. Meanwhile, some useful properties of the SP-box discussed in [11] will be introduced as well.

2.1 Notation

1. $\ll$, $\gg$, $\lll$, $\ggg$, $\oplus$, $\vee$, $\wedge$ represent the logic operations *shift left, shift right, rotate left, rotate right, exclusive or, or, and*, respectively.
2. $Z[i]$ represent the $(i+1)$-th bit of the 32-bit word Z. where the least significant bit is the 1^{st} bit and the most significant bit is the 32^{nd} bit. For example, $Z[0]$ represents the least significant bit of Z.
3. $Z[i \sim j](0 \leq j < i \leq 31)$ represents the $(j+1)$-th bit to the $(i+1)$-th bit of the 32-bit word Z. For example, $Z[1 \sim 0]$ represents the two bits $Z[1]$ and $Z[0]$ of Z.
4. $A||B$ represents the concatenation of A and B. For example, if $A = 001_2$ and $B = 1001_2$, then $A||B = 0011001_2$.
5. 0^n represent an all-zero string of length n.
6. SP represents the application of the 96-bit SP-box.
7. r represents the rate part of the Gimli state.
8. c represents the capacity part of the Gimli state.
9. f represents the Gimli permutation.
10. f^{-1} represents the inverse of Gimli permutation.

2.2 Description of Gimli

Gimli was proposed in CHES 2017 [2] and is a Round 2 candidate in NIST's Lightweight Cryptography Standardization process [1]. The Gimli state can be viewed as a two-dimensional state $S = (S[i][j])$ $(0 \leq i \leq 2, 0 \leq j \leq 3)$, where $S[i][j] \in F_2^{32}$, as illustrated in Figure 1.

$S[0][0]$	$S[0][1]$	$S[0][2]$	$S[0][3]$
$S[1][0]$	$S[1][1]$	$S[1][2]$	$S[1][3]$
$S[2][0]$	$S[2][1]$	$S[2][2]$	$S[2][3]$

Fig. 1: The Gimli state

The Gimli permutation is described in Algorithm 1. As specified in [2], the permutation is composed of four operations: SP-box, Small-Swap, Big-Swap and Constant Addition. For simplicity, we denote the SP-box, Small-Swap, Big-Swap and Constant Addition by SP, S_SW, B_SW and AC, respectively. Therefore, the 24-round permutation can be viewed as 6 times of the application of the following sequence of operations:

$$(\text{SP} \rightarrow \text{S\_SW} \rightarrow \text{AC}) \rightarrow (\text{SP}) \rightarrow (\text{SP} \rightarrow \text{B\_SW}) \rightarrow (\text{SP}).$$

For convenience, denote the internal state after r-round permutation by S^r and the input state by S^0. In other words, we have

$$S^{4i} \xrightarrow{SP} S^{4i+0.5} \xrightarrow{S\_SW} \xrightarrow{AC} S^{4i+1} \xrightarrow{SP} S^{4i+2} \xrightarrow{SP} \xrightarrow{S\_BW} S^{4i+3} \xrightarrow{SP} S^{4i+4},$$

where $0 \leq i \leq 5$. Moreover, the six 32-bit round constants are denoted by c_i $(0 \leq i \leq 5)$, where $c_i = 0x9e377900 \oplus (24 - 4i)$.

2.3 SP-box

The SP-box can be viewed as a 96-bit S-Box. Denote the 96-bit input and output by $(IX, IY, IZ) \in F_2^{32\times 3}$ and $(OX, OY, OZ) \in F_2^{32\times 3}$, respectively. Formally, the following relation holds:

$$(OX, OY, OZ) = SP(IX, IY, IZ).$$

(OX, OY, OZ) is computed as follows:

$$IX \leftarrow IX \lll 24$$

$$
\begin{aligned}
IY &\leftarrow IY \lll 9 \\
OZ &\leftarrow IX \oplus IZ \ll 1 \oplus (IY \wedge IZ) \ll 2 \\
OY &\leftarrow IY \oplus IX \oplus (IX \vee IZ) \ll 1 \\
OX &\leftarrow IZ \oplus IY \oplus (IX \wedge IY) \ll 3
\end{aligned}
$$

2.4 Linear Layer

The linear layer consists of two swap operations, namely Small-Swap and Big-Swap. Small-Swap occurs every 4 rounds starting from the 1st round. Big-Swap occurs every 4 rounds starting from the 3rd round. The illustration of Small-Swap and Big-Swap can be referred to Figure 2.

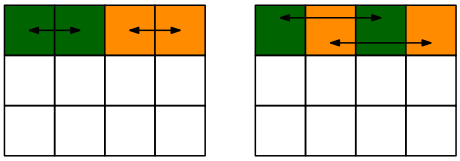

Fig. 2: The linear layer, where the left/right part represents S_SW/B_SW.

2.5 Properties of the SP-box

Suppose $(OX, OY, OZ) = SP(IX, IY, IZ)$. Several properties have been discussed in [11] and we list some useful ones for our attacks.

Property 1 [11] *If* $(IY \lll 9) \wedge \texttt{0x1fffffff} = 0$, OX *will be independent on* IX.

Property 2 [11] *A random triple* (IY, IZ, OX) *is potentially valid with probability* $2^{-15.5}$ *without knowing* IX.

Property 3 [11] *Given a random triple* (IX, OY, OZ), *it is valid with probability* 2^{-1}. *Once it is valid,* $(OX[30 \sim 0], IY, IZ[30 \sim 0])$ *can be determined.*

Property 4 [11] *Given a random triple* (IY, IZ, OZ), (IX, OX, OY) *can be uniquely determined. In addition, a random tuple* (IY, IZ, OY, OZ) *is valid with probability* 2^{-32}.

Property 5 [11] *Suppose the pair* (IY, IZ) *and* q *bits of* OY *are known. Then* t *bits of information on* IX *can be recovered by solving a linear equation system of size* q.

Property 6 [11] *Let* $(OX, OY, OZ) = SP(IX', IY', IZ')$. *If* $IY = IY'$ *and* $IZ = IZ'$, *the following relations must hold:*

$$OX[0] = OX'[0], OX[1] = OX'[1], OX[2] = OX'[2].$$
$$OY[0] \oplus OZ[0] = OY'[0] \oplus OZ'[0].$$

Property 7 [11] *Let* $(OX, OY, OZ) = SP(IX', IY', IZ')$. *If* $OY = OY'$ *and* $OZ = OZ'$, *the following relations must hold:*

$$IX[8] = IX'[8], IY[23] = IY'[23].$$

2.6 Gimli-Hash

How Gimli-Hash compresses a message is illustrated in Figure 3. Specifically, Gimli-Hash initializes a 48-byte Gimli state to all-zero. It then reads sequentially through a variable-length input as a series of 16-byte input blocks, denoted by M_0, M_1, $\cdots$.

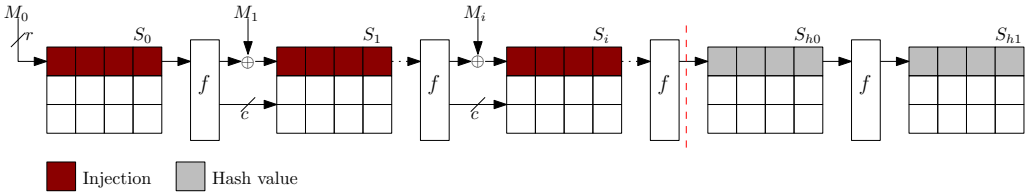

Fig. 3: The process to compress the message

Each full 16-byte input block is handled as follows:

- XOR the block into the first 16 bytes of the state (i.e. the top row of 4 words).
- Apply the Gimli permutation.

The input ends with exactly one final non-full (empty or partial) block, having b bytes where $0 \leq b \leq 15$. This final block is handled as follows:

- XOR the block into the first b bytes of the state.
- XOR 1 into the next byte of the state, position b.
- XOR 1 into the last byte of the state, position 47.
- Apply the Gimli permutation.

After the input is fully processed, a 32-byte hash output is obtained as follows:

- Output the first 16 bytes of the state (i.e. the top row of 4 words), denoted by H_0.
- Apply the Gimli permutation.

- Output the first 16 bytes of the state (i.e. the top row of 4 words), denoted by H_1.

As depicted in Figure 3, the state after M_i ($i \geq 0$) is injected is denoted by S_i and the 256-bit hash value is the concatenation of ($S_{h0}[0][0]$, $S_{h0}[0][1]$, $S_{h0}[0][2]$, $S_{h0}[0][3]$, $S_{h1}[0][0]$, $S_{h1}[0][1]$, $S_{h1}[0][2]$, $S_{h1}[0][3]$). Formally, the following relations hold:

$$\begin{aligned} S_0 &= IV \oplus (M_0||0^{256}), \\ S_{i+1} &= f(S_i) \oplus (M_i||0^{256}) \quad (i \geq 0), \end{aligned}$$

where IV is the initial state.

In our preimage attack on Gimli-Hash, two consecutive message blocks will be utilized. To distinguish the states where different message blocks are processed, we further introduce the following notations: When processing M_i, denote the internal state after r-round permutation by S_i^r and the input state by S_i^0. In other words, we have

$$S_i^{4j} \xrightarrow{SP} S_i^{4j+0.5} \xrightarrow{S\_SW} \xrightarrow{AC} S_i^{4j+1} \xrightarrow{SP} S_i^{4j+2} \xrightarrow{SP} \xrightarrow{S\_BW} S_i^{4j+3} \xrightarrow{SP} S_i^{4j+4},$$

where $0 \leq j \leq 5$ and $i \geq 0$.

Gimli-XOF In addition to Gimli-Hash, another application of the Gimli permutation called "extendable one-way function" (Gimli-XOF) is specified in the submitted Gimli document [2]. For completeness, we briefly introduce the construction of Gimli-XOF recommended by the designers for lightweight applications.

Construction. At the squeezing phase, different from Gimli-Hash which generates a fixed-length output of 32 bytes, Gimli-XOF works as follows to generate t bytes of output:

1. Concatenate $\lceil \frac{t}{16} \rceil$ blocks of 16 bytes, each of which is obtained by extracting the first 16 bytes of the state and then applying the Gimli permutation.
2. Truncate the obtained $16\lceil \frac{t}{16} \rceil$ bytes to t bytes.

At the absorbing phase, the so-called two-way fork [2] is adopted, as specified below:

1. Read the message byte by byte (imaging that there is a device). Xor the byte at the current position and then increase the current position. If the current position exceeds the end of the block (each block can absorb at most 16 bytes per time), apply the permutation and set the current position back to the first byte.
2. When reaching the "end of data", xor 1 into the state at the current position and apply the Gimli permutation.

Obviously, the difference between Gimli-Hash and Gimli-XOF at the absorbing phase exists in the padding rule.

To apply our technique, the parameter t is set as 16. In other words, the Gimli permutation is used to generate 128 bits of output. For simplicity, Gimli-XOF with a 128-bit output is denoted by Gimli-XOF-128.

3 New Properties of SP-box

In this section, two new properties of SP-box will be introduced to make our attacks efficient and reliable.

Property 8 *Suppose* $(x_1, y_1, z_1) = SP(x_0, y_0, z_0)$ *and* $(x', y', z') = SP(x_2, y_1, z_1)$. *Given a random value of* (y_0, z_0, y', z'), *all feasible solutions of* (x_0, x_2) *can be recovered with time complexity of* $2^{10.4}$.

Due to the length of the proof, the details can be referred to Appendix B.

Property 9 *Given a random constant value of* OX *and* N *random pairs of* (IY, IZ), *when* N *is sufficiently large, the expectation of the number of the solutions of* IX *is* N.

Proof. Consider the expressions to compute OX as shown in Equation 1.

$$OX[i] = \begin{cases} IZ[i] \oplus IY[i-9] \ (0 \le i \le 2) \\ IZ[i] \oplus IY[i-9] \oplus (IX[i-27] \wedge IY[i-12]) \ (3 \le i \le 31) \end{cases} \tag{1}$$

Denote the probability that there are 2^s solutions of IX for a given random triple (IY, IZ, OX) by $Pr(s)$. Therefore,

$$Pr(s+3) = 2^{-3} \times 2^{-s} \times \frac{\binom{29}{s}}{2^{29}}, \quad (0 \le s \le 29).$$

As a result, the expectation of the number of solutions of IX denoted by E can be formulated as follows:

$$E = \sum_{s=0}^{29} (2^{s+3} \times Pr(s+3)) = \sum_{s=0}^{29} (2^{s+3} \times 2^{-3} \times 2^{-s} \times \frac{\binom{29}{s}}{2^{29}}) = \sum_{s=0}^{29} \frac{\binom{29}{s}}{2^{29}} = 1.$$

In addition, according to Property 2, a random triple (IY, IZ, OX) is valid with probability $2^{-15.5}$. Thus, we can expect N solutions of OX when N is sufficiently large, e.g. $N = 2^{32}$. According to experiments, when $N = 2^{32}$, about 2^{32} (slightly greater than 2^{32}) solutions of (IX, IY, IZ) can be obtained to match a given OX.

4 Hybrid ZID Distinguisher for Full Gimli

In this section, we extend the zero-internal-differential (ZID) distinguisher for 14 rounds of the Gimli permutation as proposed in [11] to the full round. Different from [11], where only one input of a specific format is utilized and the attacker tries to trace its evolution of the symmetry of the internal states in both backward and forward directions, two different inputs of a specific format will be exploited in our new distinguisher. Specifically, both the symmetry in each internal state and the symmetry between two different internal states generated by the two inputs will be carefully investigated. Consequently, our new distinguisher is more effective and it is named as the hybrid zero-internal-differential (ZID) distinguisher. An illustration of the difference between the hybrid ZID distinguisher and the ZID distinguisher as in [11] is shown in Figure 4.

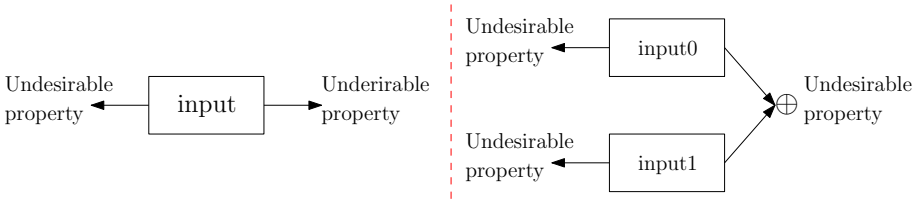

Fig. 4: Difference between the ZID distinguisher (left) [11] and our hybrid ZID distinguisher (right)

Since two different inputs are taken into account, similar to the conventional differential attack, if a specific configuration between two different outputs holds with a probability higher than that for a random permutation, a distinguisher can be constructed to distinguish the target permutation from a random one.

4.1 Revisiting the 14-Round ZID Distinguisher

First of all, we give a brief description of the 14-round ZID distinguisher [11]. For the 14-round distinguisher, the attacker starts from the internal state S^5 satisfying the following conditions and computes backwards and forwards.

$$\begin{aligned} S^5[0][0] = S^5[0][1] = S^5[0][2] = S^5[0][3], \\ S^5[1][0] = S^5[1][1] = S^5[1][2] = S^5[1][3], \\ S^5[2][0] = S^5[2][1] = S^5[2][2] = S^5[2][3]. \end{aligned}$$

Then, it can be observed that the following undesirable properties hold in $S^{0.5}$ and S^{13}:

$$S^{0.5}[1][0] = S^{0.5}[1][2],$$

$$S^{0.5}[2][0] = S^{0.5}[2][2],$$
$$S^{13}[1][1] = S^{13}[1][3],$$
$$S^{13}[2][1] = S^{13}[2][3].$$

By exploiting Property 6 and Property 7, the following undesirable properties will always exist in S^0 and S^{14}:

$$S^0[0][0][8] = S^0[0][2][8],$$
$$S^0[1][0][23] = S^0[1][2][23],$$
$$S^{14}[0][1][0] = S^{14}[0][3][0],$$
$$S^{14}[0][1][1] = S^{14}[0][3][1],$$
$$S^{14}[0][1][2] = S^{14}[0][3][2],$$
$$S^{14}[1][1][0] \oplus S^{14}[2][1][0] = S^{14}[1][3][0] \oplus S^{14}[2][3][0].$$

As explained in [11], such a 14-round ZID distinguisher can be interpreted as a zero-sum distinguisher with the data complexity reduced to the smallest one, i.e. 1. For the zero-sum distinguisher, it is common to first bound the algebraic degree after a certain number of rounds of permutation and then to find a sufficiently large number of inputs so that the sum of the corresponding outputs is zero. Naturally, to construct a longer distinguisher, the data complexity has to be increased. Thus, it is natural to ask whether it is possible to use more inputs to extend the 14-round ZID distinguisher to more rounds. Inspired by the conventional differential attack, we choose to consider two different inputs sharing a specific format and trace the symmetry between the two internal states generated by the two inputs as well as the symmetry in each internal state itself. Obviously, our distinguisher is different from the conventional internal differential where only the internal difference is traced for just one input.

4.2 Deterministic Hybrid ZID Distinguisher for 18-Round Gimli

We begin with the hybrid ZID distinguisher for 18 rounds of the Gimli permutation, which will be basis of the distinguisher for full Gimli. Such a distinguisher only requires 2 different inputs and 2 queries of the 18-round Gimli permutation.

Consider two different values (S^9, S'^9), which satisfy the following conditions:

$$S^9[0][0] \oplus S^9[0][2] = c_2,$$
$$S^9[1][0] = S^9[1][2],$$
$$S^9[2][0] = S^9[2][2],$$
$$S^9[0][1] = S^9[0][3],$$
$$S^9[1][1] = S^9[1][3],$$
$$S^9[2][1] = S^9[2][3],$$
$$S'^9[0][0] = S^9[0][2],$$

$$
\begin{aligned}
S'^9[0][2] &= S^9[0][0],\\
S'^9[1][0] &= S'^9[1][1] = S^9[1][2],\\
S'^9[2][0] &= S'^9[2][2] = S^9[2][2],\\
S'^9[0][1] &= S'^9[0][3] = S^9[0][3],\\
S'^9[1][1] &= S'^9[1][3] = S^9[1][3],\\
S'^9[2][1] &= S'^9[2][3] = S^9[2][3],
\end{aligned}
$$

where c_2 is the round constant used to compute S^9 and S'^9 in the Gimli permutation.

As illustrated in Figure 13 and Figure 14, we can trace the evolutions of the internal difference in both directions for S^9 and S'^9, respectively. The following XOR difference between S^{17} and S'^{17} can be derived:

$$
\begin{aligned}
S^{17}[1][1] \oplus S'^{17}[1][3] &= 0,\\
S^{17}[2][1] \oplus S'^{17}[2][3] &= 0,\\
S^{17}[1][3] \oplus S'^{17}[1][1] &= 0,\\
S^{17}[2][3] \oplus S'^{17}[2][1] &= 0.
\end{aligned}
$$

Consequently, according to Property 6 and Property 7, the following 8 relations always hold for (S^{18},S'^{18}).

$$
\begin{aligned}
S^{18}[0][1][0] &= S'^{18}[0][3][0],\\
S^{18}[0][1][1] &= S'^{18}[0][3][1],\\
S^{18}[0][1][2] &= S'^{18}[0][3][2],\\
S^{18}[1][1][0] \oplus S^{18}[2][1][0] &= S'^{18}[1][3][0] \oplus S'^{18}[2][3][0],\\
S'^{18}[0][1][0] &= S^{18}[0][3][0],\\
S'^{18}[0][1][1] &= S^{18}[0][3][1],\\
S'^{18}[0][1][2] &= S^{18}[0][3][2],\\
S'^{18}[1][1][0] \oplus S'^{18}[2][1][0] &= S^{18}[1][3][0] \oplus S^{18}[2][3][0].
\end{aligned}
$$

Similar to the 14-round distinguisher in [11], the following 4 relations always hold for (S^0,S'^0):

$$
\begin{cases}
S^0[0][0][8] = S^0[0][2][8],\\
S^0[1][0][23] = S^0[1][2][23],\\
S'^0[0][0][8] = S'^0[0][2][8],\\
S'^0[1][0][23] = S'^0[1][2][23].
\end{cases}
\tag{2}
$$

As a result, one could construct a distinguisher for 18 rounds of the Gimli permutation, whose data and time complexity are both 2. Such a 18-round distinguisher has been experimentally verified.

4.3 Probabilistic Hybrid ZID Distinguisher for 21-Round Gimli

Observing the above distinguisher on 18-round Gimli, it can be found that the unknown relation between $(S^{17}[0][1], S^{17}[0][3])$ and $(S'^{17}[0][1], S'^{17}[0][3])$ prevents a longer distinguisher. Thus, the attacker can impose conditions on their relation to obtain a longer distinguisher. Consider the case when the following conditions on $(S^{17}[0][1], S^{17}[0][3])$ and $(S'^{17}[0][1], S'^{17}[0][3])$ hold.

$$\begin{cases} S^{17}[0][1] = & S'^{17}[0][3], \\ S^{17}[0][3] = & S'^{17}[0][1]. \end{cases} \tag{3}$$

In this way, under the framework of the 18-round distinguisher, the following relation between S^{17} and S'^{17} holds:

$$\begin{aligned} S^{17}[0][1] &= S'^{17}[0][3], \\ S^{17}[1][1] &= S'^{17}[1][3], \\ S^{17}[2][1] &= S'^{17}[2][3], \\ S^{17}[0][3] &= S'^{17}[0][1], \\ S^{17}[1][3] &= S'^{17}[1][1], \\ S^{17}[2][3] &= S'^{17}[2][1]. \end{aligned}$$

As shown in Figure 5, by tracing the evolution of the internal difference and XOR difference for such a pair of (S^{17}, S'^{17}), one could obtain the following relations between S^{21} and S'^{21}.

$$\begin{cases} S^{21}[0][0] = & S'^{21}[0][3] \oplus c_4, \\ S^{21}[1][1] = & S'^{21}[1][3], \\ S^{21}[2][1] = & S'^{21}[2][3], \\ S^{21}[0][3] = & S'^{21}[0][0] \oplus c_4, \\ S^{21}[1][3] = & S'^{21}[1][1], \\ S^{21}[2][3] = & S'^{21}[2][1]. \end{cases} \tag{4}$$

For a random permutation, the above $32 \times 6 = 192$ bit relations hold with probability 2^{-192}. However, by choosing a pair of (S^9,S'^9) as in the 18-round distinguisher, these 192 bit relations hold with probability 2^{-64}. Specifically, for the 64 bit relations in Equation 3, since ($S^{17}[0][1]$, $S^{[}17][0][3]$) and ($S'^{17}[0][1]$, $S'^{17}[0][3]$) only depend on ($S^{13}[1][0]$, $S^{13}[2][0]$, $S^{13}[3][0]$, $S^{13}[0][2]$, $S^{13}[1][2]$, $S^{13}[3][2]$) and ($S'^{13}[1][0]$, $S'^{13}[2][0]$, $S'^{13}[3][0]$, $S'^{13}[0][2]$, $S'^{13}[1][2]$, $S'^{13}[3][2]$) and there are four times of the application of the SP-box from S^{13} to S^{17}, it can be assumed that each of 64 bit relations is independent, i.e. they hold with probability 2^{-64}. Consequently, for a random pair (S^9,S'^9) chosen in the same way as in the 18-round distinguisher, if the computed values of (S^{17}, S'^{17}) satisfy the 64 bit relations displayed in Equation 3, the 192 bit relations in Equation 4 must hold, thus distinguishing the 21-round Gimli permutation from a random permutation.

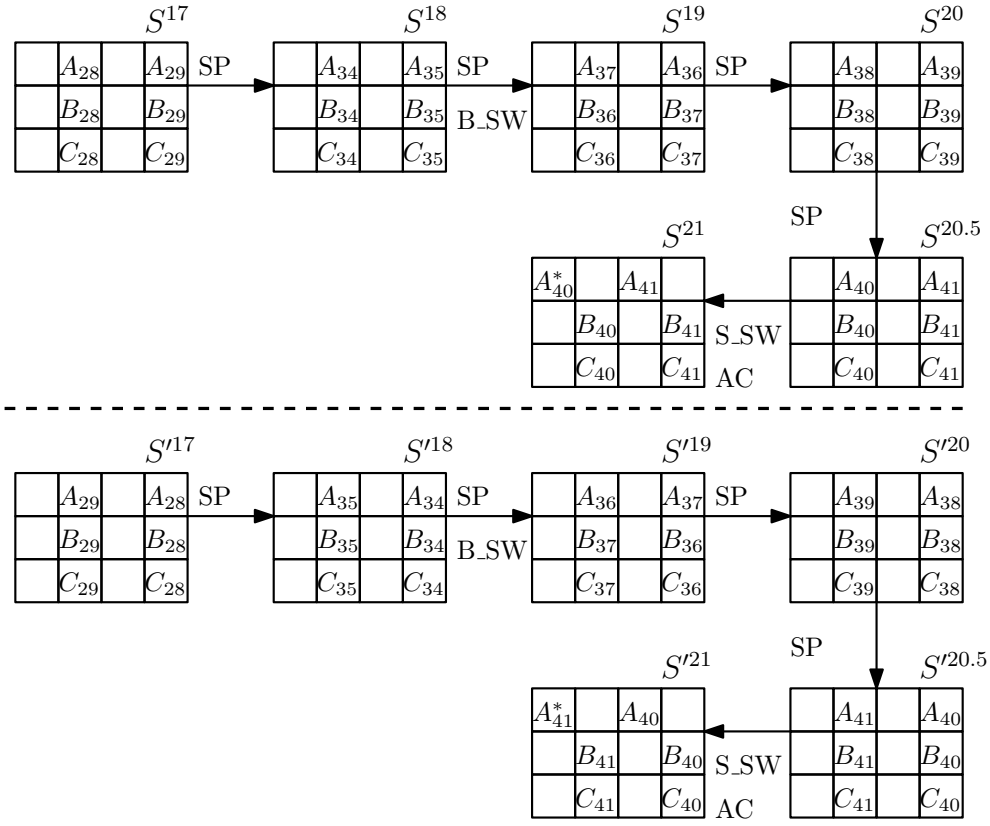

Fig. 5: The ZID distinguisher for 21-Round Gimli

Complexity Evaluation. According to the construction of the pair of (S^9, S'^9), there are in total $2^{95+96} = 2^{191}$ possible pairs. Therefore, by testing 2^{64} random pairs, one could expect to obtain an output pair of (S^{21}, S'^{21}) satisfying Equation 4 in the forward direction. For the backward direction, any pair of (S^0, S'^0) will satisfy Equation 2. Therefore, the data complexity and time complexity of our distinguisher for 21 rounds of the Gimli permutation are both 2^{65}. It should be emphasized that the 192 bit relations in Equation 4 hold with probability 2^{-192} for a random permutation.

4.4 Probabilistic Hybrid ZID Distinguisher for Full Gimli

In the same way as the extension from the 18-round distinguisher to the 21-round one, we could extend the 21-round distinguisher to the full round. Similarly, one could observe that the main obstacle to prevent a longer distinguisher exists in the relations between ($S^{21}[0][1]$, $S^{21}[0][3]$) and ($S'^{21}[0][1]$, $S'^{21}[0][3]$). If the 64 conditions on ($S^{21}[0][1]$, $S^{21}[0][3]$) and ($S'^{21}[0][1]$, $S'^{21}[0][3]$) as shown in

Equation 5 hold, the 21-round distinguisher can then be extended to the full Gimli permutation:

$$\begin{cases} S^{21}[0][1] = & S'^{21}[0][3], \\ S^{21}[0][3] = & S'^{21}[0][1]. \end{cases} \tag{5}$$

Specifically, by choosing a pair of (S^9, S'^9) in a same way as in the 18-round distinguisher, once the relations as shown in Equation 3 and Equation 5 hold, the following relations on (S^{21}, S'^{21}) will hold:

$$\begin{aligned} S^{21}[0][1] &= S'^{21}[0][3], \\ S^{21}[1][1] &= S'^{21}[1][3], \\ S^{21}[2][1] &= S'^{21}[2][3], \\ S^{21}[0][3] &= S'^{21}[0][1], \\ S^{21}[1][3] &= S'^{21}[1][1], \\ S^{21}[2][3] &= S'^{21}[2][1]. \end{aligned}$$

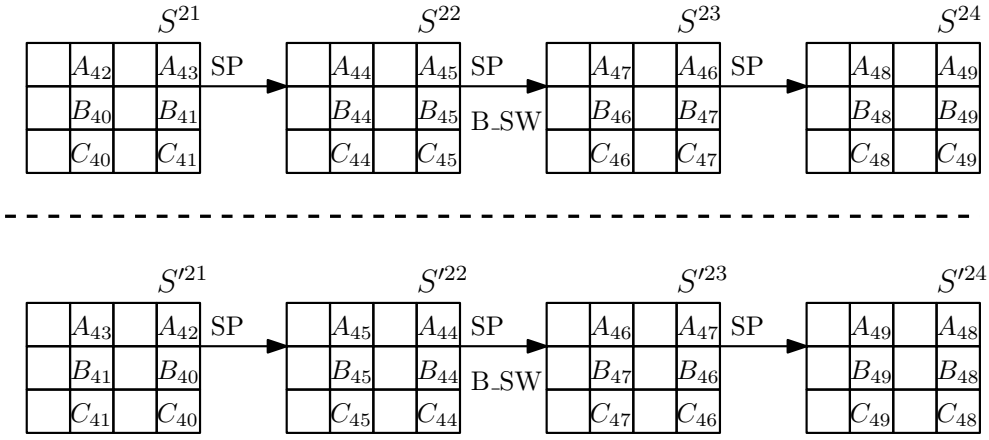

Fig. 6: The ZID distinguisher for full Gimli

As shown in Figure 6, by tracing the internal difference and XOR difference for such a pair of (S^{21}, S'^{21}), the following 192 bit relations will exist in (S^{24}, S'^{24}):

$$\begin{cases} S^{24}[0][1] = & S'^{24}[0][3], \\ S^{24}[1][1] = & S'^{24}[1][3], \\ S^{24}[2][1] = & S'^{24}[2][3], \\ S^{24}[0][3] = & S'^{24}[0][1], \\ S^{24}[1][3] = & S'^{24}[1][1], \\ S^{24}[2][3] = & S'^{24}[2][1]. \end{cases} \tag{6}$$

Similar to the 21-round distinguisher, for a random permutation, the 192 bit relations shown in Equation 6 will hold with probability 2^{-192}. However, by choosing a pair of (S^9, S'^9) as in the 18-round distinguisher, based on a similar assumption in the 21-round distinguisher for bit conditions, these relations will hold with probability $2^{-64-64} = 2^{-128}$, i.e. the 128 bit relations in Equation 3 and Equation 5 hold, thus distinguishing the full Gimli permutation from a random one.

Complexity Evaluation. Since there are in total 2^{191} pairs of (S^9, S'^9) in our construction, one could expect an output pair of (S^{24}, S'^{24}) satisfying Equation 6 in the forward direction after testing 2^{128} random pairs of (S^9, S'^9), while Equation 6 holds with probability 2^{-192} for a random permutation. In the backward direction, any pair of (S^0, S'^0) will satisfy Equation 2. Therefore, the data complexity and time complexity of our distinguisher for the full Gimli permutation are both 2^{129}.

Remark. Although such a distinguisher cannot be a threat for the AE or hash scheme of Gimli, it is the first attack on full Gimli. In addition, the ratio of the time complexity of our distinguisher to the whole value space of the Gimli state is $2^{129-384} = 2^{-255}$, which is much smaller than that of the zero-sum distinguisher for 24-round (full) Keccak [13] with 1600-bit state, i.e. $2^{1575-1600} = 2^{-25}$. Moreover, it should be mentioned that our distinguisher mainly exploits the features of the linear layer and the constant addition operation, where the linear layer of Gimli is a main strategy to help it outperform other primitives. However, as shown by our distinguisher, there are unexpected properties underlying such a design, even if these properties do not lead to an attack on the encryption or hash mode.

5 Overview of Preimage Attacks on Gimli-Hash

As can be observed from the hybrid ZID distinguisher for full Gimli, we take many advantages of the weak diffusion to construct the full-round distinguisher. Different from Keccak [5], in which the diffusion is strong, the diffusion of Gimli is rather weak. As pointed out by the designers, the avalanche effect requires 10 rounds of Gimli permutation. Exploiting such a feature of the Gimli permutation, the divide-and-conquer method may work well to accelerate the preimage finding procedure. In this section, we describe how the generic preimage attack on Gimli-Hash works and give an overview of our preimage attack on reduced Gimli-Hash.

5.1 The Generic Preimage Attack on Gimli-Hash

The generic preimage attack on Gimli-Hash is based on a meet-in-the-middle method. Specifically, consider five message blocks (M_0, M_1, M_2, M_3, M_4) and

utilize them to find a preimage for a given hash value. In other words, consider the following sequence of state transitions:

$$S_0 \xrightarrow{f} S_1 \xrightarrow{f} S_2 \xrightarrow{f} S_3 \xrightarrow{f} S_4 \xrightarrow{f} S_{h0} \xrightarrow{f} S_{h1}. \quad (7)$$

Given a hash value, ($S_{h0}[0][0]$, $S_{h0}[0][1]$, $S_{h0}[0][2]$, $S_{h0}[0][3]$, $S_{h1}[0][0]$, $S_{h1}[0][1]$, $S_{h1}[0][2]$, $S_{h1}[0][3]$) become known. As a result, the generic preimage attack can be described as follows:

Phase 1: Randomly choose a value for the 256-bit capacity part of S_{h0} and compute the corresponding S_{h1}. Repeat it until the computed 128-bit rate part of S_{h1} is consistent with that in the given hash value.

Phase 2: At this phase, the full state of S_{h0} becomes known. Thus, randomly choosing 2^{128} values for (M_3, M_4) by taking the padding in S_4 into account, compute backward the corresponding 2^{128} values of the capacity part of S_2 and store them in a table denoted by T_0.

Phase 3: Randomly choose a value for (M_0, M_1) and compute forward the corresponding value of the capacity part of S_2. Repeat it until the computed value is in T_0 and record the corresponding (S_{h0}, M_0, M_1, M_2, M_3).

Phase 4: Compute $S_2' = f(S_1)$ and $S_2 = f^{-1}(S_3)$. Then, $M_2||0^{256} = S_2 \oplus S_2'$.

Complexity Evaluation. Obviously, the time complexity at Phase 1 is 2^{128} since a 128-bit value needs to be matched. For Phase 2, the time and memory complexity are both 2^{128}. At Phase 3, the time complexity is 2^{128} since 2^{256} pairs need to be generated in order to match the 256-bit capacity part of S_2. Consequently, the time and memory complexity of the generic attack on Gimli-Hash are both 2^{128}.

5.2 The Preimage Attack with Divide-and-Conquer Methods

Our attack procedure is slightly different from the generic one. To gain advantages, Phase 1 has to be finished in less than 2^{128} time. In addition, at Phase 2, we only choose 1 random value for (M_3, M_4) by considering the padding in S^4. In this way, the capacity part of S_2 is fixed and only takes one value. Then, at Phase 3, instead of only choosing 2^{128} values for (M_0, M_1), our aim is to exhaust all the 2^{256} possible values of (M_0, M_1) in less than 2^{128} time to match the 256-bit capacity part of S_2 obtained at Phase 2. Finally, compute M_2 in the same way as in the generic attack.

Since (M_0, M_1) can take 2^{256} possible values, it is expected that Phase 2 is only performed once or twice. Obviously, the main obstacle in our method is how to achieve Phase 1 and Phase 3 efficiently, i.e. in less than 2^{128} time. In the following description of our preimage attacks on 2 and 5 rounds of Gimli-Hash, Phase 1 is called **Finding a Valid Capacity Part** and Phase 3 is called **Matching the Capacity Part**. If the two phases can be finished in less than 2^{128} time, advantages over the generic attack are obtained.

Specifically, when the Gimli permutation is reduced to n rounds, **Finding a Valid Capacity Part** is equivalent to the following problem:

Given the rate part of S^0 and S^n ($n \leq 24$), how to find a solution of the capacity part of S^0 to match the given rate part of S^n?

For **Matching the Capacity Part**, since two message blocks need to be considered, we distinguish the states by S_0 and S_1 as depicted in Figure 3 for convenience. Then, it is equivalent to the following problem:

Given the capacity part of S_0^0 and S_1^n, how to find a solution of the rate part of S_0^0 and S_1^0 to match the given capacity part of S_1^n?

6 The Preimage Attack on 2-Round Gimli-Hash

In this section, how to mount a preimage attack on 2-round Gimli-Hash with a practical time complexity is given. As described above, we only focus on Phase 1 and Phase 3. It should be emphasized that like the generic preimage attack, our preimage attack is over 5 message blocks.

6.1 Finding a Valid Capacity Part

For a better understanding of our attack, it is better to refer to Figure 7. The corresponding attack procedure is described as follows.

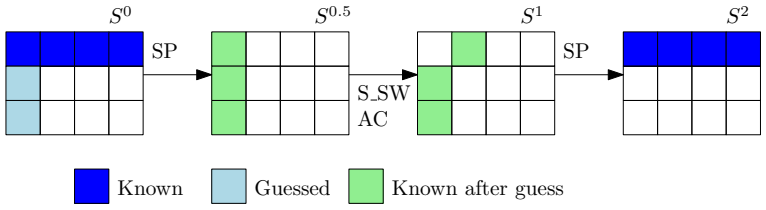

Fig. 7: Generate a valid capacity part for the preimage attack on 2-round Gimli-Hash

Step 1: Choose a random value for $(S^0[1][0], S^0[2][0])$ and compute $(S^1[0][1]$, $S^1[1][0]$, $S^1[2][0])$. Check whether $(S^1[1][0]$, $S^1[2][0]$, $S^2[0][0])$ is valid based on Property 2. If it is, store $(S^0[1][0]$, $S^0[2][0])$ in a table denoted by T_1. Otherwise, choose another value for $(S^0[1][0], S^0[2][0])$ and repeat this step until about 2^{32} random values are tried.

Step 2: Similarly, choose a random value for $(S^0[1][1], S^0[2][1])$ and compute $(S^1[0][0]$, $S^1[1][1]$, $S^1[2][1])$. Check whether $(S^1[1][1]$, $S^1[2][1]$, $S^2[0][1])$ is valid based on Property 2. If it is, store $(S^0[1][1]$, $S^0[2][1])$ in a table denoted by T_2. Otherwise, choose another value for $(S^0[1][1]$, $S^0[2][1])$ and repeat this step until 2^{32} random values are tried.

Step 3: Consider all possible combinations between T_1 and T_2. For each combination, ($S^0[1][0]$, $S^0[2][0]$, $S^0[1][1]$, $S^0[2][1]$) are fully known. Therefore, it is possible to compute ($S^2[0][0]$, $S^2[0][1]$) and check whether it is consistent with the given value. Once a solution of ($S^0[1][0]$, $S^0[2][0]$, $S^0[1][1]$, $S^0[2][1]$) is found to match ($S^2[0][0]$, $S^2[0][1]$), output the solution and move to Step 4.

Step 4: Similarly, we can first try 2^{32} possible values for ($S^0[1][2]$, $S^0[2][2]$) and store the valid ones which can possibly match $S^2[0][2]$ in a table denoted by T_3. Then, try 2^{32} possible values for ($S^0[1][3]$, $S^0[2][3]$) and store the valid ones which can possibly match $S^2[0][3]$ in a table denoted by T_4. Finally, exhaust all possible combinations between T_3 and T_4 and compute the corresponding ($S^2[0][2]$, $S^2[0][3]$). Check whether the computed one is consistent with the given value. If it is, output the solution of ($S^0[1][2]$, $S^0[2][2]$, $S^0[1][3]$, $S^0[2][3]$) and exit.

Complexity Evaluation. Obviously, the time complexity to compute the table T_i ($i \in \{1, 2, 3, 4\}$) is 2^{32} and the memory complexity is $2^{32-15.5} \approx 2^{17}$ due to the effect of Property 2. Since 2^{64} random values of ($S^0[1][0]$, $S^0[2][0]$, $S^0[1][1]$, $S^0[2][1]$) are used to match the 64-bit ($S^2[0][0]$, $S^2[0][1]$), it is expected there will be one combination between T_1 and T_2 to match ($S^2[0][0]$, $S^2[0][1]$). Similarly, it is expected that there will be one combination between T_3 and T_4 to match ($S^2[0][2]$, $S^2[0][3]$). Based on Property 2, there will be $2^{(32-15.5)\times 2} = 2^{33}$ possible combinations between T_1 and T_2. Similarly, there are 2^{33} combinations between T_3 and T_4. Consequently, the time complexity and memory complexity to find a valid capacity part are 2^{33} and $2^{17+1} = 2^{18}$, respectively.

6.2 Matching the Capacity Part

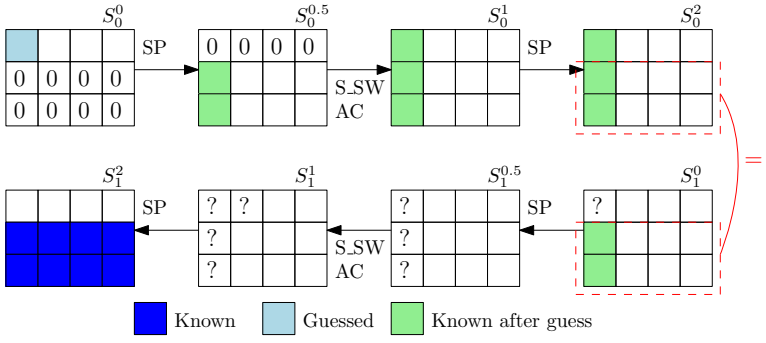

Fig. 8: Illustration of the preimage attack on 2-round Gimli-Hash

As illustrated in Figure 8, the corresponding procedure to match a given capacity part by utilizing two message blocks can be described as follows.

Step 1: Guess the value of $S_0^0[0][0]$. Based on Property 1, ($S_1^0[1][0]$, $S_1^0[2][0]$) can be uniquely determined. According to Property 8, we can find all the solutions of ($S_1^0[0][0]$, $S_1^1[0][0]$) with the knowledge of ($S_1^0[1][0]$, $S_1^0[2][0]$, $S_1^2[1][0]$, $S_1^2[2][0]$). Once the solution is obtained, compute the corresponding $S_1^1[0][1]$ by using ($S_1^0[0][0]$, $S_1^0[1][0]$, $S_1^0[2][0]$) and store the values of ($S_0^0[0][0]$, $S_1^0[0][0]$, $S_1^1[0][0]$, $S_1^1[0][1]$) in a table denoted by T_5. Repeat this step until all 2^{32} values of $S_0^0[0][0]$ are traversed.

Step 2: Similarly, guess the value of $S_0^0[0][1]$ and compute the corresponding ($S_1^0[1][1]$, $S_1^0[2][1]$). Based on Property 8, compute all the solutions of ($S_1^0[0][1]$, $S_1^1[0][1]$) which can match ($S_1^2[1][1]$, $S_1^2[2][1]$). Then, compute the corresponding $S_1^1[0][0]$ by using ($S_1^0[0][1]$, $S_1^0[1][1]$, $S_1^0[2][1]$). Check whether the computed ($S_1^1[0][0], S_1^1[0][1]$) exists in T_5. If it does, record the corresponding tuple ($S_0^0[0][0], S_0^0[0][1], S_1^0[0][0], S_1^0[0][1]$) and move to Step 3. Otherwise, repeat guessing $S_0^0[0][1]$ until all 2^{32} values of $S_0^0[0][1]$ are traversed.

Step 3: Similarly, exhaust all 2^{32} values of $S_0^0[0][2]$ and store the corresponding solutions of ($S_0^0[0][2]$, $S_1^0[0][2]$, $S_1^1[0][2]$, $S_1^1[0][3]$) in a table denoted by T_6. Finally, exhaust all 2^{32} values of $S_0^0[0][3]$ and compute the corresponding solutions of ($S_1^0[0][3]$, $S_1^1[0][2]$, $S_1^1[0][3]$). If the solution of ($S_1^1[0][2]$, $S_1^1[0][3]$) also exists in T_6, record the corresponding ($S_0^0[0][2]$, $S_0^0[0][3]$, $S_1^0[0][2]$,)$S_1^0[0][3]$) and exit.

Complexity Evaluation. At Step 1, all 2^{32} possible values of $S_0^0[0][0]$ need to be traversed. For each value, Property 8 is utilized to compute ($S_1^0[0][0]$, $S_1^0[0][0]$). Therefore, the time complexity at Step 1 is $2^{32+10.4} = 2^{42.4}$. Moreover, it is expected that there will be 2^{32} elements in T_5 since each guess of $S_0^0[0][0]$ can correspond to 1 solution of ($S_1^0[0][0]$, $S_1^1[0][0]$) on average based on Property 8. Similarly, the time complexity at Step 2 is also $2^{42.4}$. Since there are 2^{32} solutions of ($S_1^1[0][0]$, $S_1^1[0][1]$) in T_5 and there will be another 2^{32} solutions of ($S_1^1[0][0]$, $S_1^1[0][1]$) at Step 2, it is expected there will be a match in ($S_1^1[0][0]$, $S_1^1[0][1]$) after traversing all 2^{32} values of $S_0^0[0][1]$. Similarly, the time complexity and memory complexity at Step 3 are $2^{42.4}$ and 2^{32}, respectively. In a word, considering the complexity to find a valid capacity part, the time complexity and memory complexity of the preimage attack on 2-round Gimli-Hash are $2^{42.4}$ and 2^{32}, respectively.

Experiments. To verify the correctness of our attack, we provide a solution of $(M_0, M_1, M_2, M_3, M_4)$ which can lead to an all-zero state in Table 2. Note that with such a message, we can construct arbitrary second preimage and colliding message pairs for 2-round Gimli-Hash with time complexity 1. Specifically, given a message M_x, $(M_x, M_0||M_1||M_2||M_3||M_4||M_x)$ is a colliding message pair. Moreover, given a message M_x and its hash value H_x, $M_0||M_1||M_2||M_3||M_4||M_x$ is a second preimage of H_x.

Table 2: A message leading to an all-zero state for 2-round Gimli-Hash

M_0	0x1c5c59da	0x41b61bb7	0	0
M_1	0x9cf49a4e	0x9a80d115	0	0
M_2	0xa31c3903	0x41e6e73c	0	0
M_3	0x456723c6	0xdc515cff	0	0
M_4	0x98694873	0x944a58ec	0	0
Full-state Value	0	0	0	0
	0	0	0	0
	0	0	0	0

7 The Preimage Attack on 5-Round Gimli-Hash

In this section, how to mount the preimage attack on 5-round Gimli-Hash will be introduced. Similarly, we only focus on **Finding a Valid Capacity Part** and **Matching the Capacity Part**.

7.1 Finding a Valid Capacity Part

As illustrated in Figure 9, the corresponding procedure can be divided into 4 steps, as shown below.

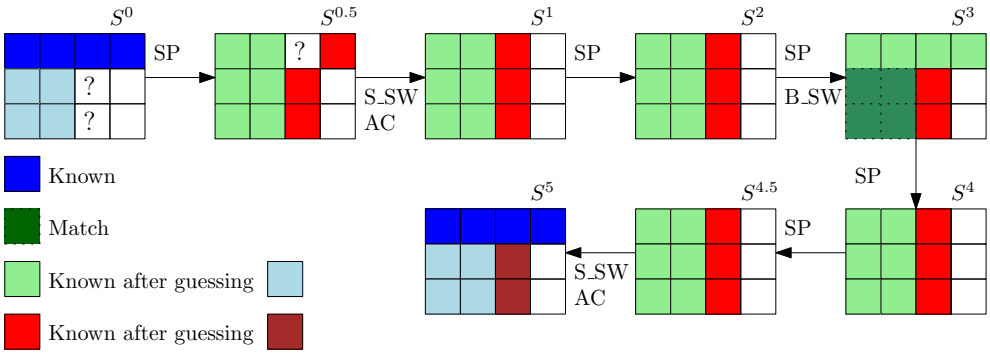

Fig. 9: Generate a valid capacity part for the preimage attack on 5-round Gimli-Hash

Step 1: Randomly choose a value for ($S^0[1][0]$, $S^0[2][0]$, $S^0[1][1]$, $S^0[2][1]$) and compute the corresponding ($S^3[0][2]$, $S^3[0][3]$, $S^3[1][0]$, $S^3[2][0]$, $S^3[1][1]$, $S^3[2][1]$). Store the values of ($S^0[1][0]$, $S^0[2][0]$, $S^0[1][1]$, $S^0[2][1]$, $S^3[0][2]$, $S^3[0][3]$, $S^3[1][0]$, $S^3[2][0]$, $S^3[1][1]$, $S^3[2][1]$) in a table denoted by T_7. Repeat this step for 2^{64} random values of ($S^0[1][0]$, $S^0[2][0]$, $S^0[1][1]$, $S^0[2][1]$).

Step 2: Randomly choose a value for ($S^5[1][0]$, $S^5[2][0]$, $S^5[1][1]$, $S^5[2][1]$) and compute the corresponding ($S^3[0][0]$, $S^3[0][1]$, $S^3[1][0]$, $S^3[2][0]$, $S^3[1][1]$, $S^3[2][1]$). Check whether the computed ($S^3[1][0]$, $S^3[2][0]$, $S^3[1][1]$, $S^3[2][1]$) is in T_7. If it is, record the corresponding value of ($S^5[1][0]$, $S^5[2][0]$, $S^5[1][1]$, $S^5[2][1]$, $S^3[0][0]$, $S^3[0][1]$, $S^3[0][2]$, $S^3[0][3]$) and move to Step 3. Otherwise, repeat trying different random values for ($S^5[1][0]$, $S^5[2][0]$, $S^5[1][1]$, $S^5[2][1]$).

Step 3: It should be emphasized that ($S^0[1][0]$, $S^0[2][0]$, $S^0[1][1]$, $S^0[2][1]$, $S^3[0][0]$, $S^3[0][1]$, $S^3[0][2]$, $S^3[0][3]$) is a fixed value at this step. Randomly choose a value for ($S^5[1][2]$, $S^5[2][2]$) and compute the corresponding ($S^3[0][2]$, $S^3[1][2]$, $S^3[2][2]$). Check whether the computed $S^3[0][2]$ is consistent with the one obtained at Step 2. If it is not, repeat choosing a random value for ($S^5[1][2]$, $S^5[2][2]$). If it is, continue computing the corresponding ($S^{0.5}[0][3]$, $S^{0.5}[1][2]$, $S^{0.5}[2][2]$) with the knowledge of ($S^3[0][0]$, $S^3[1][2]$, $S^3[2][2]$). According to Property 3, ($S^0[0][2]$, $S^{0.5}[1][2]$, $S^{0.5}[2][2]$) is valid with probability 2^{-1}. Once it is valid, compute ($S^{0.5}[0][2][30 \sim 0]$, $S^0[1][2]$, $S^0[2][2][30 \sim 0]$) and store the value of ($S^5[1][2]$, $S^5[2][2]$, $S^{0.5}[0][2][30 \sim 0]$, $S^{0.5}[0][3]$) in a table denoted by T_8. Repeat this step until all the 2^{64} values of ($S^5[1][2]$, $S^5[2][2]$) are traversed.

Step 4: Similar to Step 3, guess ($S^5[1][3]$, $S^5[2][3]$) and compute ($S^3[0][3]$, $S^3[1][3]$, $S^3[2][3]$). If the computed $S^3[0][3]$ is not consistent with the one obtained at Step 2, guess another value. Otherwise, continue computing ($S^{0.5}[0][2]$, $S^{0.5}[1][3]$, $S^{0.5}[2][3]$). Based on Property 3, we can compute ($S^{0.5}[0][3][30 \sim 0]$, $S^0[1][3]$, $S^0[2][3][30 \sim 0]$) to match $S^0[0][3]$. Then, check whether the computed ($S^{0.5}[0][2][30 \sim 0]$, $S^{0.5}[0][3][30 \sim 0]$) is contained in T_8. If it is, record ($S^5[1][2]$, $S^5[2][2]$, $S^5[1][3]$, $S^5[2][3]$) and exit. Repeat this step until all 2^{64} values of ($S^5[1][3]$, $S^5[2][3]$) are traversed.

Complexity Evaluation. At Step 1, the time and memory complexity are both 2^{64}. At Step 2, it is necessary to match a 256-bit value of ($S^3[1][0]$, $S^3[2][0]$, $S^3[1][1]$, $S^3[2][1]$) based on a meet-in-the-middle method. Therefore, it is required to try 2^{64} possible values of ($S^5[1][0]$, $S^5[2][0]$, $S^5[1][1]$, $S^5[2][1]$). Thus, the time complexity at Step 2 is also 2^{64}. At step 3, 2^{64} values of ($S^5[1][2]$, $S^5[2][2]$) are traversed and each of it will be first filtered by $S^3[0][2]$ and then filtered according to Property 3. Thus, it is expected there will be 2^{31} elements in T_8. Similarly, at Step 4, there will be 2^{31} valid guesses of ($S^5[1][3]$, $S^5[2][3]$) left after filtering. For each valid guess, we need to manage a match in the 62-bit value of ($S^{0.5}[0][2][30 \sim 0]$, $S^{0.5}[0][3][30 \sim 0]$). Since there are in total 2^{62} possible pairs, one can expect one match. Consequently, the time and memory complexity to find a valid capacity part are both 2^{64}.

7.2 Matching the Capacity Part

Before describing how to match a given capacity part by utilizing two message blocks, we will pre-compute some tables in order to reduce the whole complexity.

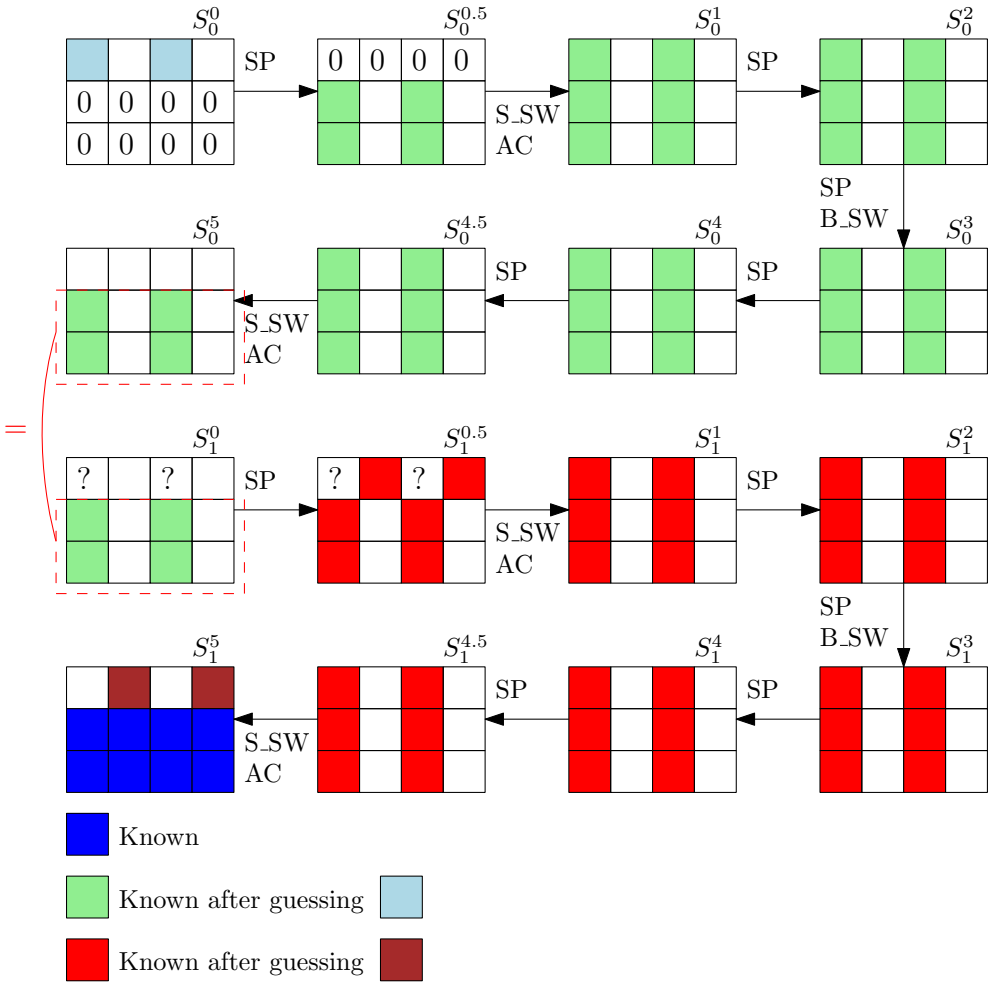

Fig. 10: Illustration of the preimage attack on 5-round Gimli-Hash

Pre-computing Tables. As shown in Figure 10, based on Property 1, the following facts can be observed:

- $(S_1^0[1][0], S_1^0[2][0], S_1^0[1][2], S_1^0[2][2])$ only depend on $(S_0^0[0][0], S_0^0[0][2])$, thus taking at most 2^{64} possible values.
- $(S_1^0[1][1], S_1^0[2][1], S_1^0[1][3], S_1^0[2][3])$ only depend on $(S_0^0[0][1], S_0^0[0][3])$, thus taking at most 2^{64} possible values.

Consequently, it is feasible to construct some mapping tables via precomputation. Specifically, exhaust all 2^{64} values of $(S_0^0[0][0], S_0^0[0][2])$ and compute the corresponding $(S_1^0[1][0], S_1^0[2][0], S_1^0[1][2], S_1^0[2][2])$. Store the 2^{64} values of $(S_0^0[0][0], S_0^0[0][2], S_1^0[1][0], S_1^0[2][0], S_1^0[1][2], S_1^0[2][2])$ in a table denoted by T_9, where the row number represents the value of $(S_1^0[1][0] + S_1^0[2][0] \times 2^{32})$.

Similarly, by exhausting all 2^{64} values of $(S_0^0[0][1], S_0^0[0][3])$, we can compute all the 2^{64} values of $(S_0^0[0][1], S_0^0[0][3], S_1^0[1][1], S_1^0[2][1], S_1^0[1][3], S_1^0[2][3])$ and store them in a table denoted by T_{10}, where the row number represents the value of $(S_1^0[1][1] + S_1^0[2][1] \times 2^{32})$.

Matching the Capacity Part. After preparing the tables, matching the capacity part by utilizing two message blocks can be described as follows. The corresponding illustration can be referred to Figure 10.

Step 1: Guess $(S_1^5[0][1], S_1^5[0][3])$ and compute the corresponding $(S_1^{0.5}[0][1], S_1^{0.5}[0][3], S_1^{0.5}[1][0], S_1^{0.5}[2][0], S_1^{0.5}[1][2], S_1^{0.5}[2][2])$. If all the 2^{64} values of $(S_1^5[0][1], S_1^5[0][3])$ are traversed, move to Step 3. Otherwise, move to Step 2.

Step 2: Further guess $S_1^{0.5}[0][0]$ and compute $(S_1^0[1][0], S_1^0[2][0])$. Retrieve the corresponding values of $(S_0^0[0][0], S_0^0[0][2], S_1^0[1][2], S_1^0[2][2])$ from the $(S_1^0[1][0] + S_1^0[2][0] \times 2^{32})$-th row of T_9. Based on Property 4, verify the correctness of the tuple $(S_1^0[1][2], S_1^0[2][2], S_1^{0.5}[1][2], S_1^{0.5}[2][2])$. If it is valid, compute the corresponding $S^{0.5}[0][2]$ according to Property 4 and store the corresponding values of $(S_0^0[0][0], S_0^0[0][2], S_1^5[0][1], S_1^5[0][3], S_1^{0.5}[0][0], S_1^{0.5}[0][1], S_1^{0.5}[0][2], S_1^{0.5}[0][3])$ in a table denoted by T_{11}. Otherwise, try another value of $S_1^{0.5}[0][0]$. If all the 2^{32} values of $S_1^{0.5}[0][0]$ are traversed, go back to Step 1.

Step 3: Similarly, exhaust all the 2^{96} values of $(S_1^5[0][0], S_1^5[0][2], S_1^{0.5}[0][1])$. For each of its value, compute the corresponding $(S_1^{0.5}[0][0], S_1^{0.5}[0][2], S_1^0[1][1], S_1^0[2][1], S_1^{0.5}[1][3], S_1^{0.5}[2][3])$. Retrieve $(S_0^0[0][1], S_0^0[0][3], S_1^0[1][3], S_1^0[2][3])$ from the $(S_1^0[1][1] + S_1^0[2][1] \times 2^{32})$-th row of T_{10} and check the validity of the tuple $(S_1^0[1][3], S_1^0[2][3], S_1^{0.5}[1][3], S_1^{0.5}[2][3])$ based on Property 4. If it is valid, compute $S^{0.5}[0][3]$ and check whether the obtained value of $(S_1^{0.5}[0][0], S_1^{0.5}[0][1], S_1^{0.5}[0][2], S_1^{0.5}[0][3])$ at Step 3 also exists in T_{11}. If it does, exit and a solution of the rate part of S_0^0 and S_1^5 is found to match the given capacity part of S_1^5.

Complexity Evaluation. The time complexity at Step 1 is 2^{64} since all the 2^{64} values of $(S_1^5[0][1], S_1^5[0][3])$ need to be traversed. At Step 2, for each guessed value of $(S_1^5[0][1], S_1^5[0][3])$, all the 2^{32} values of $S_1^{0.5}[0][0]$ will be traversed. After

the 2^{32} values of $S_1^{0.5}[0][0]$ are traversed, one can expect one valid solution of ($S_1^{0.5}[0][0]$, $S_1^{0.5}[0][1]$, $S_1^{0.5}[0][2]$, $S_1^{0.5}[0][3]$) due to the influence of Property 4. As a result, Step 2 is will be carried out for 2^{96} times and there will be 2^{64} elements in T_{11}. As for Step 3, since all the 2^{96} values of ($S_1^{5}[0][0]$, $S_1^{5}[0][2]$, $S_1^{0.5}[0][1]$) will be traversed and each guessed value is valid with probability of 2^{-32} based on Property 4, one can expect 2^{64} solutions of($S_1^{0.5}[0][0]$, $S_1^{0.5}[0][1]$, $S_1^{0.5}[0][2]$, $S_1^{0.5}[0][3]$) in total. Thus, it is expected that there will be one match between the values of ($S_1^{0.5}[0][0]$, $S_1^{0.5}[0][1]$, $S_1^{0.5}[0][2]$, $S_1^{0.5}[0][3]$) obtained at Step 3 and those stored in T_{11}. As for the pre-computation, the time complexity and memory complexity are 2^{64} and $2^{64+1} = 2^{65}$, respectively. Consequently, taking the complexity to find a valid capacity part into account, the time complexity and memory complexity of the preimage attack on 5-round Gimli-Hash are 2^{96} and $2^{64} \times 3 = 2^{65.6}$, respectively.

8 Preimage Attacks on Round-Reduced Gimli-XOF-128

When the above preimage attack on Gimli-Hash is extended to more rounds, we are faced with an obstacle caused by the degrees of freedom, i.e. at least two message blocks are needed and should be traversed in less than 2^{128} time to match a given hash value. As can be observed in our method, benefiting from the weak diffusion of the linear layer of Gimli, we can efficiently exploit the divide-and-conquer technique to divide the space of two message blocks into several smaller ones and find solutions in each smaller space via exhaustive search. Finally, the solutions in each smaller space are combined and further verified to match the given hash value. When it comes to more rounds, it is difficult to divide the space of two message blocks into smaller ones. Thus, turning the exhaustive search into a smaller scale cannot be applied anymore. In addition, to control two consecutive message blocks when the number of rounds of the Gimli permutation is reduced to n, the difficulty is almost equivalent to an attack on $2n$ rounds of the Gimli permutation, by allowing the attacker to control a 128-bit value in the intermediate state.

To test how far our divide-and-conquer method can go for reduced Gimli, we consider another application of the Gimli permutation to hashing, namely the "extendable one-way function", which has been specified in the submitted Gimli document. Considering the existing preimage attacks on SHAKE-128 [8] and Ascon-XOF-64 [7], we believe it meaningful to investigate the preimage resistance of Gimli-XOF-128. In addition, since the size of one message block is 128 bits when neglecting the padding rule, the attacker only needs to focus on how to efficiently exhaust one message block rather than two message blocks in less than 2^{128} time. In other words, the attack on n rounds of Gimli-XOF-128 is equivalent to an attack on n rounds of the Gimli permutation.

Similar to the method to turn the 6-round semi-free-start collisions into collisions in [11], to efficiently mount the preimage attack on reduced Gimli-XOF-128, some conditions will be added. Specifically, when the target is n rounds

of Gimli, an equivalent problem to find the preimage of Gimli-XOF-128 can be described as below:

If

$$\begin{cases} (S^0[1][0] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^0[1][1] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^0[1][2] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^0[1][3] \lll 9) \wedge \texttt{0x1fffffff} = 0, \end{cases} \tag{8}$$

how to find a solution of ($S^0[0][0]$, $S^0[0][1]$, $S^0[0][2]$, $S^0[0][3]$) *to match a given value of* ($S^n[0][0]$, $S^n[0][1]$, $S^n[0][2]$, $S^n[0][3]$)*?*

It should be emphasized that the initial value of Gimli-XOF-128 satisfies Equation 8. In addition, due to the padding rule, there are at most $2^{128-8} = 2^{120}$ possible values of ($S^0[0][0]$, $S^0[0][1]$, $S^0[0][2]$, $S^0[0][3]$). Therefore, to mount the preimage attack on n rounds of Gimli-XOF-128, it is expected that 2^8 different values of the capacity part of S^0 are tried. For each of them, check whether there is a solution of ($S^0[0][0]$, $S^0[0][1]$, $S^0[0][2]$, $S^0[0][3]$) to match the given hash value under the conditions as specified in Equation 8.

Consequently, our attack is divided into two phases. The first phase called **Fulfilling Conditions** is to collect 2^8 different values of the capacity part which can satisfy Equation 8. The second phase called **Matching the Rate Part** is to exhaust the 2^{120} possible values of ($S^0[0][0]$, $S^0[0][1]$, $S^0[0][2]$, $S^0[0][3]$) in less than 2^{120} time under the conditions as specified in Equation 8. As will be shown, the main idea to finish the two tasks is almost the same. Therefore, in our description, we will start from **Matching the Rate Part** and then move to **Fulfilling Conditions**.

8.1 The Preimage Attack on 9-Round Gimli-XOF-128

Due to the page limit, the preimage attack on 8-round Gimli-XOF-128 can be referred to Appendix C. The two phases of the preimage attack on 9-round Gimli-XOF-128 will be described in this section. Different from the preimage attack on 8-round Gimli-XOF-128, a pre-computed table will be utilized to reduce the whole time complexity. An illustration of our preimage attack on 9-round Gimli-XOF-128 is shown in Figure 11.

Matching the Rate Part For the given value of ($S^9[0][0]$, $S^9[0][1]$, $S^9[0][2]$, $S^9[0][3]$), compute the corresponding ($S^{8.5}[0][0]$, $S^{8.5}[0][1]$, $S^{8.5}[0][2]$, $S^{8.5}[0][3]$) by reversing the AC and S_SW operations. Then, pre-compute four mapping tables as follows:

Constructing T_{18}. Exhaust all 2^{96} possible values of ($S^7[0][0]$, $S^7[1][0]$, $S^7[2][0]$) and compute the corresponding $S^{8.5}[0][0]$. Check whether it is consistent with the given value. If it is, store the corresponding ($S^7[0][0]$, $S^7[1][0]$, $S^7[2][0]$) in a table denoted by T_{18}.

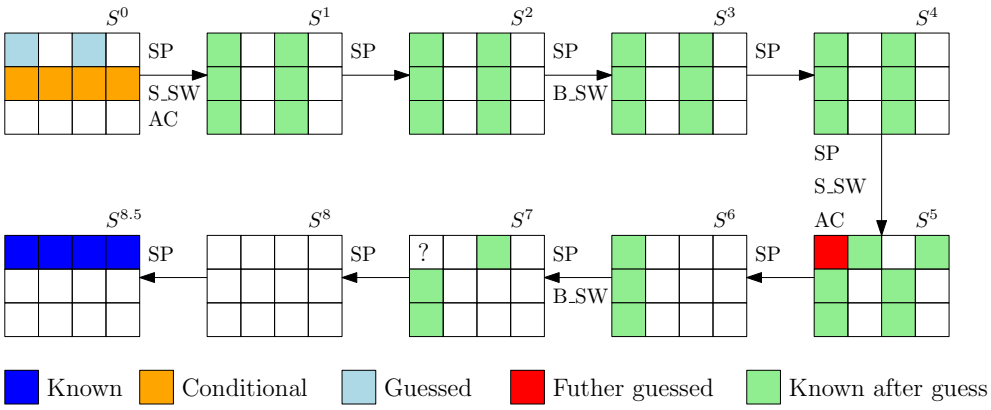

Fig. 11: Illustration of the preimage attack on 9-round Gimli-XOF-128

Constructing T_{19}. Exhaust all 2^{96} possible values of $(S^7[0][1], S^7[1][1], S^7[2][1])$ and compute the corresponding $S^{8.5}[0][1]$. Check whether it is consistent with the given value. If it is, store the corresponding $(S^7[0][1], S^7[1][1], S^7[2][1])$ in a table denoted by T_{19}.

Constructing T_{20}. Exhaust all 2^{96} possible values of $(S^7[0][2], S^7[1][2], S^7[2][2])$ and compute the corresponding $S^{8.5}[0][2]$. Check whether it is consistent with the given value. If it is, store the corresponding $(S^7[0][2], S^7[1][2], S^7[2][2])$ in a table denoted by T_{20}.

Constructing T_{21}. Exhaust all 2^{96} possible values of $(S^7[0][3], S^7[1][3], S^7[2][3])$ and compute the corresponding $S^{8.5}[0][3]$. Check whether it is consistent with the given value. If it is, store the corresponding $(S^7[0][3], S^7[1][3], S^7[2][3])$ in a table denoted by T_{21}.

Obviously, the time and memory complexity to construct the four tables are 2^{96} and $2^{64+2} = 2^{66}$, respectively. It should be emphasized that it is possible to construct each table with less time complexity via some algebraic methods, i.e. constructing an equation system and solving it with a guess-and-determine method. However, as will be shown, the time complexity of our preimage attack is not dominated by this pre-computation phase. Therefore, the simplest approach is exploited.

Matching the Rate Part. After this pre-computation phase, how to find a solution of the rate part of S^0 under the condition that S^0 satisfies Equation 8 will be described, as specified below.

Step 1: Exhaust all the 2^{64} values of $(S^0[0][0], S^0[0][2])$. Since S^0 satisfies Equation 8, based on Property 1, for each guess of $(S^0[0][0], S^0[0][2])$, $(S^5[0][1], S^5[0][3], S^5[1][0], (S^5[2][0], S^5[1][2], S^5[2][2])$ can be determined and we move to Step 2. If all possible values of $(S^0[0][0], S^0[0][2])$ are traversed, move to Step 4.

Step 2: Exhaust all the 2^{32} values of $S^5[0][0]$. For each guess of $S^5[0][0]$, ($S^7[1][0]$, $S^7[2][0]$, $S^7[0][2]$) become known. Retrieve $S^7[0][0]$ from T_{18} according to the value of ($S^7[1][0]$, $S^7[2][0]$). It is expected to obtain 2^{32} solutions of ($S^7[0][0]$, $S^7[0][2]$) after exhausting $S^5[0][0]$ for each guessed value of ($S^0[0][0]$, $S^0[0][2]$). Store all the solutions of ($S^5[0][0]$, $S^7[0][0]$, $S^7[0][2]$) in a table denoted by T_{22}. After exhausting $S^5[0][0]$, move to Step 3.

Step 3: Similarly, exhaust all the 2^{32} values of $S^5[0][2]$. For each guess of $S^5[0][2]$, ($S^7[1][2]$, $S^7[2][2]$, $S^7[0][0]$) become known. Retrieve $S^7[0][2]$ from T_{20} according to the value of ($S^7[1][2]$, $S^7[2][2]$). For solution of ($S^5[0][2]$, $S^7[0][0]$, $S^7[0][2]$), check whether ($S^7[0][0]$, $S^7[0][2]$) exists in T_{22}. If it does, a solution of ($S^5[0][0]$, $S^5[0][2]$) which can match ($S^{8.5}[0][0]$, $S^{8.5}[0][2]$) for the guessed value of ($S^0[0][0], S^0[0][2]$) is found. It is expected that there will be one solutions of ($S^5[0][0]$, $S^5[0][2]$) for each guessed value of ($S^0[0][0]$, $S^0[0][2]$) since one 64-bit value needs to be matched. Consequently, after exhausting ($S^0[0][0]$, $S^0[0][2]$), it is expected to collect 2^{64} possible values of ($S^0[0][0]$, $S^0[0][2]$, $S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$). Store these values in a table denoted by T_{23}.

Step 4: Exhaust all the 2^{56} values of ($S^0[0][1]$, $S^0[0][3]$). For each such guess, we first further exhaust $S^5[0][1]$ to collect 2^{32} solutions of ($S^7[0][1]$, $S^7[0][3]$) according to T_{19} which can match $S^{8.5}[0][1]$ and store them in a table denoted by T_{24}. Then, exhaust $S^5[0][3]$ to collect another 2^{32} solutions of ($S^7[0][1]$, $S^7[0][3]$) according to T_{21} which can match $S^{8.5}[0][3]$ and check whether the obtained ($S^7[0][1]$, $S^7[0][3]$) is in T_{24}. For each guessed value of ($S^0[0][1]$, $S^0[0][3]$), it should be noted that ($S^5[0][0]$, $S^5[0][2]$) are determined. In addition, after exhausting $S^5[0][1]$ and $S^5[0][3]$, one can expect a match in ($S^7[0][1]$, $S^7[0][3]$), which will correspond to solution of ($S^5[0][1]$, $S^5[0][3]$). For each solution of ($S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$) obtained in Step 4, check whether it also exist in T_{23}. If it does, output the corresponding value of ($S^0[0][0]$, $S^0[0][1]$, ($S^0[0][2]$, $S^0[0][3]$). Otherwise, repeat until all values of ($S^0[0][1]$, $S^0[0][3]$) are traversed.

Complexity Evaluation. One can easily observe that the above description is almost the same with that in the preimage attack on 8-round Gimli-XOF-128. The only difference is that we compute $S^7[0][i]$ ($i \in \{0, 1, 2, 3\}$) via tale look-ups in the 9-round preimage attack while it is based on Property 9 in the 8-round preimage attack. Thus, the time and memory complexity at this phase are 2^{104} and 2^{64}, respectively.

Fulfilling Conditions Similarly, we can start from S^0 satisfying Equation 8 and compute the solutions of ($S^0[0][0]$, $S^0[0][1]$, ($S^0[0][2]$, $S^0[0][3]$) which can also make the capacity part of $S^{8.5}$ satisfy Equation 42. For convenience, the 29

bits of $S^{8.5}[1][i]$ ($i \in \{0,1,2,3\}$) required to be 0 are called conditional bits.

$$\begin{cases} (S^{8.5}[1][0] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^{8.5}[1][1] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^{8.5}[1][2] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^{8.5}[1][3] \lll 9) \wedge \texttt{0x1fffffff} = 0. \end{cases} \tag{9}$$

The basic procedure is almost the same with that to find the rate part. Firstly, constructing 4 tables as follows:

Constructing T_{25}. Exhaust all 2^{96} possible values of ($S^7[0][0]$, $S^7[1][0]$, $S^7[2][0]$) and compute the corresponding $S^{8.5}[1][0]$. Check whether the 29 bit conditions on $S^{8.5}[1][0]$ hold. If it is, store the corresponding ($S^7[0][0]$, $S^7[1][0]$, $S^7[2][0]$) in a table denoted by T_{25}.

Constructing T_{26}. Exhaust all 2^{96} possible values of ($S^7[0][1]$, $S^7[1][1]$, $S^7[2][1]$) and compute the corresponding $S^{8.5}[1][1]$. Check whether the 29 bit conditions on $S^{8.5}[1][1]$ hold. If it is, store the corresponding ($S^7[0][1]$, $S^7[1][1]$, $S^7[2][1]$) in a table denoted by T_{26}.

Constructing T_{27}. Exhaust all 2^{96} possible values of ($S^7[0][2]$, $S^7[1][2]$, $S^7[2][2]$) and compute the corresponding $S^{8.5}[1][2]$. Check whether the 29 bit conditions on $S^{8.5}[1][2]$ hold. If it is, store the corresponding ($S^7[0][2]$, $S^7[1][2]$, $S^7[2][2]$) in a table denoted by T_{27}.

Constructing T_{28}. Exhaust all 2^{96} possible values of ($S^7[0][3]$, $S^7[1][3]$, $S^7[2][3]$) and compute the corresponding $S^{8.5}[1][3]$. Check whether the 29 bit conditions on $S^{8.5}[1][3]$ hold. If it is, store the corresponding ($S^7[0][3]$, $S^7[1][3]$, $S^7[2][3]$) in a table denoted by T_{28}.

Obviously, it is expected that there will be $2^{64+3} = 2^{67}$ elements in T_i ($25 \leq i \leq 28$).

Fulfilling Conditions. The corresponding procedure can be simply summarized as follows:

Step 1: Exhaust 2^{64} values of ($S^0[0][0]$, $S^0[0][2]$). For each guess, first exhaust $S^5[0][0]$ and then exhaust $S^5[0][2]$. When exhausting $S^5[0][0]$, by retrieving T_{25}, collect all the solutions of ($S^7[0][0]$, $S^7[0][2]$) and store the values of ($S^7[0][0]$, $S^7[0][2]$, $S^5[0][0]$) in a table denoted by T_{29}, which is expected to contain 2^{35} values. When exhausting $S^5[0][2]$, by retrieving T_{27}, compute the solution of ($S^7[0][0]$, $S^7[0][2]$) and check whether it is in T_{29}. Once it is, record the corresponding value of ($S^0[0][0]$, $S^0[0][2]$, $S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$) in a table denoted by T_{30}. After exhausting ($S^0[0][0]$, $S^0[0][2]$), one can expect $2^{64+6} = 2^{70}$ elements stored in T_{30}.

Step 2: Exhaust 2^{64} values of ($S^0[0][1]$, $S^0[0][3]$). For each guess, first exhaust $S^5[0][1]$ and then exhaust $S^5[0][3]$. When exhausting $S^5[0][1]$, by retrieving T_{26}, collect all the solutions of ($S^7[0][1]$, $S^7[0][3]$) and store the values of ($S^7[0][1]$, $S^7[0][3]$, $S^5[0][1]$) in a table denoted by T_{31}, which is expected to contain 2^{35} values. When exhausting $S^5[0][3]$, by retrieving T_{28}, compute the solution of ($S^7[0][1]$, $S^7[0][3]$) and check whether it is in T_{31}. Once it is, record the corresponding value of ($S^0[0][0]$, $S^0[0][2]$, $S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$) and check whether ($S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$) exists in T_{30}. If it does, output the corresponding solution of ($S^0[0][0]$, $S^0[0][1]$, $S^0[0][2]$, $S^0[0][3]$). After exhausting ($S^0[0][1]$, $S^0[0][3]$), one can expect $2^{140-128} = 2^{12}$ solutions of ($S^0[0][0]$, $S^0[0][1]$, $S^0[0][2]$, $S^0[0][3]$).

Complexity Evaluation. The time complexity and memory complexity to construct T_i ($25 \leq i \leq 28$) are 2^{96} and $2^{67+2} = 2^{69}$, respectively. Regarding the time complexity to find a solution of ($S^0[0][0]$, $S^0[0][1]$, $S^0[0][2]$, $S^0[0][3]$), it can be evaluated as $2^{96+3} = 2^{99}$. For the memory complexity, it is dominated by constructing T_{30} and therefore is 2^{70}. In a word, the time and memory complexity to mount the preimage attack on 9-round Gimli-XOF-128 are 2^{104} and 2^{70}, respectively.

9 Conclusion

Due to Gimli's weak diffusion, a novel zero-internal-differential distinguisher is constructed for the full Gimli permutation. Although such a distinguisher can not threaten the security of the hash scheme or authenticated encryption scheme based on Gimli, it implies that there exist undesirable properties inside the design. Compared with the zero-sum distinguisher for full Keccak, our distinguisher has a much smaller ratio of the time complexity to the value space of the state, i.e. 2^{-255} V.S. 2^{-25}. In addition, it does not rely on the algebraic degree evaluation but much depends on the weak diffusion among the four columns of the Gimli state, i.e. the feature of the design. To further exploit the weak diffusion, we propose a divide-and-conquer method to accelerate the preimage finding procedure for both Gimli-Hash and Gimli-XOF-128. Benefiting from some new properties of the SP-box, we are able to mount a practical preimage attack on 2-round Gimli-Hash and the theoretical attack can reach up to 5 rounds of Gimli-Hash. As an extreme example, the preimage attack on Gimli-XOF-128 can reach up to 9 rounds. To the best of knowledge, this is the first distinguishing attack on the full Gimli permutation and our preimage attacks on reduced Gimli-Hash and Gimli-XOF-128 are the best thus far.

References

1. `https://csrc.nist.gov/projects/lightweight-cryptography/round-2-candidates`.

2. Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli : A cross-platform permutation. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 299–320, 2017.
3. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.
4. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011. `http://sponge.noekeon.org/CSF-0.1.pdf`.
5. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, 2011. `http://keccak.noekeon.org`.
6. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2, 2018. `https://ascon.iaik.tugraz.at/files/asconv12-nist.pdf`.
7. Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Preliminary analysis of Ascon-Xof and Ascon-Hash (version 0.1), 2019. `https://ascon.iaik.tugraz.at/files/Preliminary_Analysis_of_Ascon-Xof_and_Ascon-Hash_v01.pdf`.
8. Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced keccak. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 249–274, 2016.
9. Mike Hamburg. Cryptanalysis of 22 1/2 rounds of gimli. Cryptology ePrint Archive, Report 2017/743, 2017. `https://eprint.iacr.org/2017/743`.
10. Ting Li and Yao Sun. Preimage attacks on round-reduced keccak-224/256 via an allocating approach. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 556–584. Springer, 2019.
11. Fukang Liu, Takanori Isobe, and Willi Meier. Automatic verification of differential characteristics: Application to reduced Gimli. not published yet.
12. Pawel Morawiecki, Josef Pieprzyk, and Marian Srebrny. Rotational cryptanalysis of round-reduced keccak. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 241–262, 2013.
13. Hailun Yan, Xuejia Lai, Lei Wang, Yu Yu, and Yiran Xing. New zero-sum distinguishers on full 24-round keccak-f using the division property. *IET Information Security*, 13(5):469–478, 2019.
14. Rui Zong, Xiaoyang Dong, and Xiaoyun Wang. Collision attacks on round-reduced Gimli-Hash/Ascon-Xof/Ascon-Hash. Cryptology ePrint Archive, Report 2019/1115, 2019. `https://eprint.iacr.org/2019/1115`.

A Algorithm of Gimli

Algorithm 1 Description of Gimli permutation

Input: $\mathbf{S} = (S[i][j])$

for R from 24 down to 1 inclusive **do**
  for j from 0 to 3 inclusive **do**
    $IX \leftarrow S[0][j] \lll 24$ ▷ SP-box
    $IY \leftarrow S[1][j] \lll 9$
    $IZ \leftarrow S[2][j]$

    $S[2][j] \leftarrow IX \oplus IZ \ll 1 \oplus (IY \wedge IZ) \ll 2$
    $S[1][j] \leftarrow IY \oplus IX \oplus (IX \vee IZ) \ll 1$
    $S[0][j] \leftarrow IZ \oplus IY \oplus (IX \wedge IY) \ll 3$
  end for

  if R mod 4 =0 **then**
    $S[0][0], S[0][1], S[0][2], S[0][3] \leftarrow S[0][1], S[0][0], S[0][3], S[0][2]$ ▷ Small-Swap
  else if r mod 2 =0 **then**
    $S[0][0], S[0][1], S[0][2], S[0][3] \leftarrow S[0][2], S[0][3], S[0][0], S[0][1]$ ▷ Big-Swap
  end if

  if R mod 4 =0 **then**
    $S[0][0] \leftarrow S[0][0] \oplus \texttt{0x9e377900} \oplus r$ ▷ Constant Addition
  end if
end for
return $(S[i][j])$

B Proof of Property 8

Property 8. Suppose $(x_1, y_1, z_1) = SP(x_0, y_0, z_0)$ *and* $(x', y', z') = SP(x_2, y_1, z_1)$. *Given a random value of* (y_0, z_0, y', z'), *all feasible solutions of* (x_0, x_2) *can be recovered with time complexity of* $2^{10.4}$.

Proof. First of all, consider the generic time complexity to recover the pair (x_0, x_2). For each guessed value of x_0, (x_1, y_1, z_1) can be determined. Since (y', z') are known, based on Property 4, the correctness of the computed (y_1, z_1) can be immediately checked without knowing x_2. According to Property 4, the tuple (y_1, z_1, y', z') is valid with probability 2^{-32}. Since there are at most 2^{32} values of x_0, after all the possible values of x_0 are traversed, one can expect only one solution of x_0 which can make the tuple (y_1, z_1, y', z') valid. Once the tuple is valid, x_2 can be uniquely determined based on Property 4. Consequently, the generic method is a simple exhaustive search for x_0, which requires 2^{32} time.

In our following method, x_0 can be efficiently exhausted with the guess-and-determine technique.

For simplicity, let $v = x_0 \lll 24$. First of all, consider the relations between (x_0, y_0, z_0) and (y_1, z_1):

$$\begin{aligned} z_1 &= v \oplus z_0 \ll 1 \oplus ((y_0 \lll 9) \wedge z_0) \ll 2, \\ y_1 &= (y_0 \lll 9) \oplus v \oplus (v \vee z_0) \ll 1. \end{aligned}$$

It can be easily observed that when (y_0, z_0) are constants, each bit of (z_1, y_1) can be expressed as follows:

$$\begin{aligned} z_1[i] &= v[i] \oplus \gamma_i, \\ y_1[i] &= v[i] \oplus \mu_i v[i-1] \oplus \lambda_i, \end{aligned}$$

where γ_i, μ_i and λ_i $(0 \leq i \leq 31)$ are constants over $GF(2)$, which can be calculated according to (y_0, z_0).

For convenience, let $y = y_1 \lll 9$, $z = z_1$, $x = x_2 \lll 24$. Then, each bit of (z, y) can be expressed as follows:

$$\begin{aligned} z[i] &= v[i] \oplus \gamma_i, \\ y[i] &= v[i-9] \oplus \alpha_i v[i-10] \oplus \beta_i, \end{aligned}$$

where γ_i, α_i and β_i $(0 \leq i \leq 31)$ are constants over $GF(2)$, which can be calculated according to (y_0, z_0).

Consider the relations between (x, y, z) and (y', z'), as specified below:

$$\begin{aligned} z' &= x \oplus z \ll 1 \oplus (yz) \ll 2, \\ y' &= y \oplus x \oplus (x \vee z) \ll 1 = y \oplus x \oplus (xz \oplus x \oplus z) \ll 1. \end{aligned}$$

We rewrite the expression of y' as follows:

$$y' = y \oplus x \oplus (xz \oplus x \oplus z) \ll 1 = y \oplus (x \oplus z \ll 1) \oplus (xz \oplus x) \ll 1.$$

By involving z' into the expression of y', we can obtain that

$$\begin{aligned} y' &= y \oplus (x \oplus z \ll 1) \oplus (xz \oplus x) \ll 1 \\ &= y \oplus z' \oplus (yz) \ll 2 \oplus (x\overline{z}) \ll 1. \\ &\Downarrow \\ y' \oplus z' &= y \oplus (yz) \ll 2 \oplus (x\overline{z}) \ll 1. \end{aligned}$$

Note that

$$x = z' \oplus z \ll 1 \oplus (yz) \ll 2.$$

Thus, it can be derived that

$$y' \oplus z' = y \oplus (yz) \ll 2 \oplus (\overline{z}(z' \oplus z \ll 1 \oplus (yz) \ll 2)) \ll 1.$$

For simplicity, let $Y = y' \oplus z'$. Considering the expression from the bit level, we can derive the following 32 equations:

$$
\begin{aligned}
Y[0] &= y[0], && (10)\\
Y[1] &= y[1] \oplus z'[0]\overline{z[0]}, && (11)\\
Y[2] &= y[2] \oplus y[0]z[0] \oplus \overline{z[1]}(z'[1] \oplus z[0]), && (12)\\
Y[3] &= y[3] \oplus y[1]z[1] \oplus \overline{z[2]}(z'[2] \oplus z[1] \oplus y[0]z[0]), && (13)\\
Y[4] &= y[4] \oplus y[2]z[2] \oplus \overline{z[3]}(z'[3] \oplus z[2] \oplus y[1]z[1]), && (14)\\
Y[5] &= y[5] \oplus y[3]z[3] \oplus \overline{z[4]}(z'[4] \oplus z[3] \oplus y[2]z[2]), && (15)\\
Y[6] &= y[6] \oplus y[4]z[4] \oplus \overline{z[5]}(z'[5] \oplus z[4] \oplus y[3]z[3]), && (16)\\
Y[7] &= y[7] \oplus y[5]z[5] \oplus \overline{z[6]}(z'[6] \oplus z[5] \oplus y[4]z[4]), && (17)\\
Y[8] &= y[8] \oplus y[6]z[6] \oplus \overline{z[7]}(z'[7] \oplus z[6] \oplus y[5]z[5]), && (18)\\
Y[9] &= y[9] \oplus y[7]z[7] \oplus \overline{z[8]}(z'[8] \oplus z[7] \oplus y[6]z[6]), && (19)\\
Y[10] &= y[10] \oplus y[8]z[8] \oplus \overline{z[9]}(z'[9] \oplus z[8] \oplus y[7]z[7]), && (20)\\
Y[11] &= y[11] \oplus y[9]z[9] \oplus \overline{z[10]}(z'[10] \oplus z[9] \oplus y[8]z[8]), && (21)\\
Y[12] &= y[12] \oplus y[10]z[10] \oplus \overline{z[11]}(z'[11] \oplus z[10] \oplus y[9]z[9]), && (22)\\
Y[13] &= y[13] \oplus y[11]z[11] \oplus \overline{z[12]}(z'[12] \oplus z[11] \oplus y[10]z[10]), && (23)\\
Y[14] &= y[14] \oplus y[12]z[12] \oplus \overline{z[13]}(z'[13] \oplus z[12] \oplus y[11]z[11]), && (24)\\
Y[15] &= y[15] \oplus y[13]z[13] \oplus \overline{z[14]}(z'[14] \oplus z[13] \oplus y[12]z[12]), && (25)\\
Y[16] &= y[16] \oplus y[14]z[14] \oplus \overline{z[15]}(z'[15] \oplus z[14] \oplus y[13]z[13]), && (26)\\
Y[17] &= y[17] \oplus y[15]z[15] \oplus \overline{z[16]}(z'[16] \oplus z[15] \oplus y[14]z[14]), && (27)\\
Y[18] &= y[18] \oplus y[16]z[16] \oplus \overline{z[17]}(z'[17] \oplus z[16] \oplus y[15]z[15]), && (28)\\
Y[19] &= y[19] \oplus y[17]z[17] \oplus \overline{z[18]}(z'[18] \oplus z[17] \oplus y[16]z[16]), && (29)\\
Y[20] &= y[20] \oplus y[18]z[18] \oplus \overline{z[19]}(z'[19] \oplus z[18] \oplus y[17]z[17]), && (30)\\
Y[21] &= y[21] \oplus y[19]z[19] \oplus \overline{z[20]}(z'[20] \oplus z[19] \oplus y[18]z[18]), && (31)\\
Y[22] &= y[22] \oplus y[20]z[20] \oplus \overline{z[21]}(z'[21] \oplus z[20] \oplus y[19]z[19]), && (32)\\
Y[23] &= y[23] \oplus y[21]z[21] \oplus \overline{z[22]}(z'[22] \oplus z[21] \oplus y[20]z[20]), && (33)\\
Y[24] &= y[24] \oplus y[22]z[22] \oplus \overline{z[23]}(z'[23] \oplus z[22] \oplus y[21]z[21]), && (34)\\
Y[25] &= y[25] \oplus y[23]z[23] \oplus \overline{z[24]}(z'[24] \oplus z[23] \oplus y[22]z[22]), && (35)\\
Y[26] &= y[26] \oplus y[24]z[24] \oplus \overline{z[25]}(z'[25] \oplus z[24] \oplus y[23]z[23]), && (36)\\
Y[27] &= y[27] \oplus y[25]z[25] \oplus \overline{z[26]}(z'[26] \oplus z[25] \oplus y[24]z[24]), && (37)\\
Y[28] &= y[28] \oplus y[26]z[26] \oplus \overline{z[27]}(z'[27] \oplus z[26] \oplus y[25]z[25]), && (38)\\
Y[29] &= y[29] \oplus y[27]z[27] \oplus \overline{z[28]}(z'[28] \oplus z[27] \oplus y[26]z[26]), && (39)\\
Y[30] &= y[30] \oplus y[28]z[28] \oplus \overline{z[29]}(z'[29] \oplus z[28] \oplus y[27]z[27]), && (40)\\
Y[31] &= y[31] \oplus y[29]z[29] \oplus \overline{z[30]}(z'[30] \oplus z[29] \oplus y[28]z[28]). && (41)
\end{aligned}
$$

In the above equation system (Eq. 1∼32), (z', Y) are known and (y, z) are linear in the unknown x_0. Our aim is to recover (y, z) in order to recover the unknown (x_0, x_2).

The procedure to solve the above equation system is described as follows:

Step 1: Guess $(z[0], z[1], z[2], z[3], z[4])$. For each such guess, $v[i]$ $(0 \leq i \leq 4)$ becomes known. Based on Eq. 1∼6, we can also uniquely compute

$$(y[0], y[1], y[2], y[3], y[4], y[5]).$$

Note that we need to compute $y[i]$ before computing $y[i+1]$ $(0 \leq i \leq 4)$.

Step 2: Note that the expression of $y[i]$ is as follows:

$$y[i] = v[i-9] \oplus \alpha_i v[i-10] \oplus \beta_i.$$

Since $(y[0], y[1], y[2], y[3], y[4], y[5])$ are known, we can uniquely determine $v[i]$ $(22 \leq i \leq 28)$ by guessing $v[22]$.

Step 3: Guess $(y[22], y[23], y[24])$. Since $v[i]$ $(22 \leq i \leq 28)$ have been determined at Step 2, we can compute the corresponding $z[i]$ $(22 \leq i \leq 28)$. Then, based on Eq. 26∼30, we can uniquely compute

$$(y[25], y[26], y[27], y[28], y[29]).$$

Then

$$(y[22], y[23], y[24], y[25], y[26], y[27], y[28], y[29])$$

become determined. Therefore, we can uniquely determine $v[i]$ $(12 \leq i \leq 20)$ by guessing $v[12]$.

Step 4: At this step, only $v[i]$ $(i \in \{5, 6, 7, 8, 8, 10, 11, 21, 29, 30, 31\})$ are unknown. We can compute $(y[11], y[12], y[13])$ according to the knowledge of $(v[1], v[2], v[3], v[4])$. Observing Eq. 15, when $z[13] = 1$ or $y[11] = 0$, we can uniquely compute $y[14]$ since the unknown $z[11]$ will not influence the calculation of $y[14]$ anymore. After $y[14]$ is obtained, based on Eq. 16∼21, we can uniquely compute

$$(y[15], y[16], y[17], y[18], y[19], y[20]).$$

Then, the values of $v[i]$ $(i \in \{5, 6, 7, 8, 8, 10, 11\})$ are determined.
If $z[13] = 0$ and $y[11] = 1$, which occurs with probability 2^{-2}, similarly, we simply guess $z[11]$ and then obtain the value of

$$(y[14], y[15], y[16], y[17], y[18], y[19], y[20]),$$

which will correspond to a solution of $v[i]$ $(i \in \{5, 6, 7, 8, 8, 10, 11\})$. Compare the value of $v[11]$ with its guessed value (we can obtain $v[11]$ from $z[11]$). If they are consistent, we find a correct solution of $v[i]$ $(i \in \{5, 6, 7, 8, 8, 10, 11\})$. Otherwise, it is wrong.
In conclusion, whatever the case is, we could only get one solution of $v[11]$ $(i \in \{5, 6, 7, 8, 8, 10, 11\})$. The average cost at this step can be estimated as $\frac{3}{4} + \frac{1}{4} \times 2 \approx 2^{0.4}$.

Step 5: Since $(v[5], v[6], v[7])$ are determined, we can compute $(z[5], z[6], z[7])$. Then, based on Eq. 7∼9, we can uniquely compute $(y[6], y[7], [8])$, thus determining $(v[29], v[30], v[31])$ and $(z[29], z[30], z[31])$. Then, we can compute $y[30]$ based on Eq. 31 because $z[29]$ becomes known. After $y[30]$ is computed, we can uniquely determine $v[21]$. Until this phase, (v, y, z) are fully determined and we can check the correctness by checking the validity of the tuple (y, z, y', z') according to Property 4.

The time complexity of our guess-and-determine method to solve the above equation system can be evaluated in this way. At Step 1, $(z[0], z[1], z[2], z[3], z[4])$ are guessed. At Step 2, $v[22]$ is guessed. At Step 3, $(y[22], y[23], y[24], v[12])$ are guessed. At Step 4, the cost of guessing can be evaluated as $2^{0.4}$. As a result, the time complexity to traverse all solutions of the above equation system is $2^{5+1+4+0.4} = 2^{10.4}$. On the other hand, we do not construct any coefficient matrix nor use Gauss elimination when solving the above equation system. The unknown variables can be calculated step by step by considering the corresponding expressions, which is very efficient.

As explained at the beginning of the proof, since x_0 can be exhausted in $2^{10.4}$ time, (x_0, x_2) can be recovered in $2^{10.4}$ time and the expected number of solutions is 1.

C The Preimage Attack on 8-Round Gimli-XOF-128

The corresponding two phases of the preimage attack on 8-round Gimli-XOF-128 will be specified below.

Matching the Rate Part The attack procedure to exhaust all the 2^{120} possible values of ($S^0[0][0]$, $S^0[0][1]$, $S^0[0][2]$, $S^0[0][3]$) to match a given 128-bit ($S^8[0][0]$, $S^8[0][1]$, $S^8[0][2]$, $S^8[0][3]$) is described as follows. The corresponding illustration can be referred to Figure 12.

Step 1: Exhaust all the 2^{64} values of ($S^0[0][0]$, $S^0[0][2]$). Since S^0 satisfies Equation 8, based on Property 1, for each guess of ($S^0[0][0]$, $S^0[0][2]$), ($S^5[0][1]$, $S^5[0][3]$, $S^5[1][0]$, $S^5[2][0]$, $S^5[1][2]$, $S^5[2][2]$) can be determined and we move to Step 2. If all possible values of $(S^0[0][0], S^0[0][2])$ are traversed, move to Step 4.

Step 2: Exhaust all the 2^{32} values of $S^5[0][0]$. For each guess of $S^5[0][0]$, ($S^7[1][0]$, $S^7[2][0]$, $S^7[0][2]$) become known. According to Property 9, it is expected to obtain 2^{32} solutions of ($S^7[0][0]$, $S^7[1][0]$, $S^7[2][0]$) to match $S^8[0][0]$ after traversing all values of $S^5[0][0]$. Store all the solutions of ($S^5[0][0]$, $S^7[0][0]$, $S^7[0][2]$) in a table denoted by T_{12}. After exhausting $S^5[0][0]$, move to Step 3.

Step 3: Similarly, exhaust all the 2^{32} values of $S^5[0][2]$. For each guess of $S^5[0][2]$, ($S^7[1][2]$, $S^7[2][2]$, $S^7[0][0]$) become known. According to Property 9, it is expected to obtain 2^{32} solutions of ($S^7[0][2]$, $S^7[1][2]$, $S^7[2][2]$) to

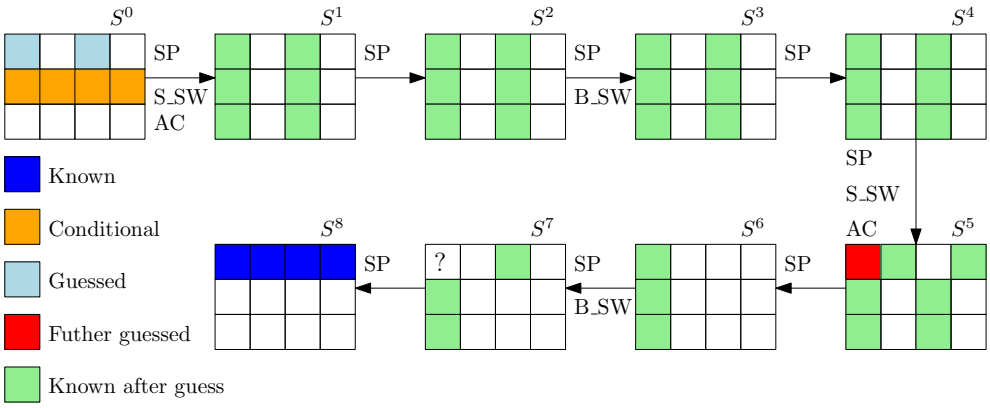

Fig. 12: Illustration of the preimage attack on 8-round Gimli-XOF-128

match $S^8[0][2]$ after traversing all values of $S^5[0][2]$. For each feasible value of ($S^5[0][2]$, $S^7[0][0]$, $S^7[0][2]$) which can match $S^8[0][2]$, check whether ($S^7[0][0]$, $S^7[0][2]$) exists in T_{12}. If it does, a solution of ($S^5[0][0]$, $S^5[0][2]$) which can match ($S^8[0][0]$, $S^8[0][2]$) for the guessed value of ($S^0[0][0], S^0[0][2]$) is found. It is expected that there will be one solution of ($S^5[0][0]$, $S^5[0][2]$) for each guessed value of ($S^0[0][0]$, $S^0[0][2]$) since one 64-bit value needs to be matched. Consequently, after exhausting ($S^0[0][0]$, $S^0[0][2]$), it is expected to collect 2^{64} possible values of ($S^0[0][0]$, $S^0[0][2]$, $S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$). Store these values in a table denoted by T_{13}.

Step 4: Exhaust all the 2^{56} values of ($S^0[0][1]$, $S^0[0][3]$). For each such guess, we first further exhaust $S^5[0][1]$ to collect 2^{32} solutions of ($S^7[0][1]$, $S^7[0][3]$) which can match $S^8[0][1]$ and store them in a table denoted by T_{14}. Then, exhaust $S^5[0][3]$ to collect another 2^{32} solutions of ($S^7[0][1]$, $S^7[0][3]$) which can match $S^8[0][3]$ and check whether the obtained ($S^7[0][1]$, $S^7[0][3]$) is in T_{14}. For each guessed value of ($S^0[0][1]$, $S^0[0][3]$), it should be noted that ($S^5[0][0]$, $S^5[0][2]$) are determined. In addition, after exhausting $S^5[0][1]$ and $S^5[0][3]$, one can expect a match in ($S^7[0][1]$, $S^7[0][3]$), which will correspond to solution of ($S^5[0][1]$, $S^5[0][3]$). For each solution of ($S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$) obtained in Step 4, check whether it also exist in T_{13}. If it does, output the corresponding value of ($S^0[0][0]$, $S^0[0][1]$), ($S^0[0][2]$, $S^0[0][3]$). Otherwise, repeat until all values of ($S^0[0][1]$, $S^0[0][3]$) are traversed.

Complexity Evaluation. At step 1, the time complexity is 2^{64}. For each guess of ($S^0[0][0]$, $S^0[0][2]$), $S^5[0][0]$ and $S^5[0][2]$ need to be exhausted at Step 2 and Step 3, respectively. As a result, the time complexity at Step 2 and Step 3 are both 2^{96}. As for Step 4, we need to exhaust 2^{56} values of ($S^0[0][1]$, $S^0[0][3]$). For each of it, $S^5[0][1]$ and $S^5[0][3]$ needs to be exhausted, respectively. Thus, the time complexity at Step 4 is $2^{56+32} = 2^{88}$.

Moreover, there will be 2^{32} elements stored in T_{12}, 2^{64} elements stored in T_{13} and 2^{32} elements stored in T_{14}. Therefore, the memory complexity at this phase is 2^{64}.

In addition, since there are only $2^{64+56} = 2^{120}$ pairs of ($S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$), a correct solution of ($S^0[0][0]$, $S^0[0][1]$, ($S^0[0][2]$, $S^0[0][3]$) is found with probability of 2^{-8}, indicating that the whole time complexity to match the given hash value is $2^{96+8} = 2^{104}$ and the memory complexity is 2^{64} if there are 2^8 different values of the capacity part of S^0 satisfying Equation 8.

Fulfilling Conditions As shown in the above time complexity evaluation, it is expected that there are 2^8 different values of the capacity part of S^0 satisfying Equation 8 in order to match a given 128-bit hash value. In this part, how to finish this task will be introduced.

First of all, it should be emphasized that the initial value of Gimli-XOF-128 satisfies Equation 8. However, one such value is insufficient. Consequently, it is necessary to overcome the obstacle of how to efficiently generate many preferred capacity parts. By exploiting the fact that the initial value satisfies the conditions, we can start from S^0 satisfying Equation 8 and compute the solutions of ($S^0[0][0]$, $S^0[0][1]$, ($S^0[0][2]$, $S^0[0][3]$) which can also make the capacity part of S^8 satisfy Equation 42. For convenience, the 29 bits of $S^8[1][i]$ ($i \in \{0, 1, 2, 3\}$) required to be 0 are called conditional bits.

$$\begin{cases} (S^8[1][0] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^8[1][1] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^8[1][2] \lll 9) \wedge \texttt{0x1fffffff} = 0, \\ (S^8[1][3] \lll 9) \wedge \texttt{0x1fffffff} = 0. \end{cases} \tag{42}$$

The corresponding attack procedure is almost the same with that to match the rate part. Specifically, it can be summarized as follows:

Step 1: Exhaust all the 2^{64} values of ($S^0[0][0]$, $S^0[0][2]$). Since S^0 satisfies Equation 8, based on Property 1, for each guess of ($S^0[0][0]$, $S^0[0][2]$), ($S^5[0][1]$, $S^5[0][3]$, $S^5[1][0]$, $S^5[2][0]$, $S^5[1][2]$, $S^5[2][2]$) can be determined and we move to Step 2. If all possible values of ($S^0[0][0], S^0[0][2]$) are traversed, move to Step 4.

Step 2: Exhaust all the 2^{32} values of $S^5[0][0]$. For each guess of $S^5[0][0]$, ($S^7[1][0]$, $S^7[2][0]$, $S^7[0][2]$) become known. According to Property 5, it is expected to obtain $2^{32+3} = 2^{35}$ solutions of ($S^7[0][0]$, $S^7[1][0]$, $S^7[2][0]$) to match the 29 conditions bits of $S^8[1][0]$ after traversing all values of $S^5[0][0]$. Store all the solutions of ($S^5[0][0]$, $S^7[0][0]$, $S^7[0][2]$) in a table denoted by T_{15}. After exhausting $S^5[0][0]$, move to Step 3.

Step 3: Similarly, exhaust all the 2^{32} values of $S^5[0][2]$. For each guess of $S^5[0][2]$, ($S^7[1][2]$, $S^7[2][2]$, $S^7[0][0]$) become known. According to Property 5, it is expected to obtain $2^{32+3} = 2^{35}$ solutions of ($S^7[0][2]$, $S^7[1][2]$, $S^7[2][2]$) to match the 29 conditional bits of $S^8[0][2]$ after traversing all values

of $S^5[0][2]$. For each feasible value of ($S^5[0][2]$, $S^7[0][0]$, $S^7[0][2]$) which can match the conditional 29 bits of $S^8[0][2]$, check whether ($S^7[0][0]$, $S^7[0][2]$) exists in T_{15}. If it does, a solution of ($S^5[0][0]$, $S^5[0][2]$) which can match ($S^8[0][0]$, $S^8[0][2]$) for the guessed value of ($S^0[0][0], S^0[0][2]$) is found. It is expected that there will be 2^6 solutions of ($S^5[0][0]$, $S^5[0][2]$) for each guessed value of ($S^0[0][0]$, $S^0[0][2]$) since one 58-bit value needs to be matched. Consequently, after exhausting ($S^0[0][0]$, $S^0[0][2]$), it is expected to collect $2^{64+6} = 2^{70}$ possible values of ($S^0[0][0]$, $S^0[0][2]$, $S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$). Store these values in a table denoted by T_{16}.

Step 4: Exhaust all the 2^{64} values of ($S^0[0][1]$, $S^0[0][3]$). For each such guess, we first further exhaust $S^5[0][1]$ to collect 2^{35} solutions of ($S^7[0][1]$, $S^7[0][3]$) which can match the 29 conditional bits of $S^8[0][1]$ and store them in a table denoted by T_{17}. Then, exhaust $S^5[0][3]$ to collect another 2^{35} solutions of ($S^7[0][1]$, $S^7[0][3]$) which can match the 29 conditional bits of $S^8[0][3]$ and check whether the obtained ($S^7[0][1]$, $S^7[0][3]$) is in T_{17}. For each guessed value of ($S^0[0][1]$, $S^0[0][3]$), ($S^5[0][0]$, $S^5[0][2]$) are determined. In addition, after exhausting $S^5[0][1]$ and $S^5[0][3]$, one can expect 2^6 matches in ($S^7[0][1]$, $S^7[0][3]$), which will correspond to 2^6 solutions of ($S^5[0][1]$, $S^5[0][3]$). For each solution of ($S^5[0][0]$, $S^5[0][1]$, $S^5[0][2]$, $S^5[0][3]$) obtained in Step 4, check whether it also exist in T_{16}. If it does, output the corresponding value of ($S^0[0][0]$, $S^0[0][1]$, ($S^0[0][2]$, $S^0[0][3]$). Otherwise, repeat until all values of ($S^0[0][1]$, $S^0[0][3]$) are traversed.

Complexity Evaluation. As can be observed, the attack procedure is almost the same with that to match the rate part. The only difference is that Property 5 is utilized in **Fulfilling Conditions** while Property 9 is utilized in **Matching the Rate Part**. It should be noted that there are in total $29 \times 4 = 116$ bit conditions on S^8 and there are 2^{128} possible values of the rate part of S^0. Therefore, after exhausting all the 2^{128} values, one can expect $2^{128-116} = 2^{12}$ solutions of ($S^0[0][0]$, $S^0[0][1]$, ($S^0[0][2]$, $S^0[0][3]$) which can make the capacity part of S^8 satisfy Equation 42. The method to evaluate the time complexity is almost the same with that to match the rate part. Obviously, the memory complexity to exhaust all the 2^{128} values is 2^{70} consumed by T_{16}. The time complexity is $2^{96+3} = 2^{99}$ since only 29 bits of $S^8[1][i]$ ($i \in \{0, 1, 2, 3\}$) are conditional. In conclusion, the time and memory complexity to mount the preimage attack on 8-round Gimli-XOF-128 are 2^{104} and 2^{70}, respectively.

D Illustrations of the Hybrid ZID Distinguisher for 18-Round Gimli

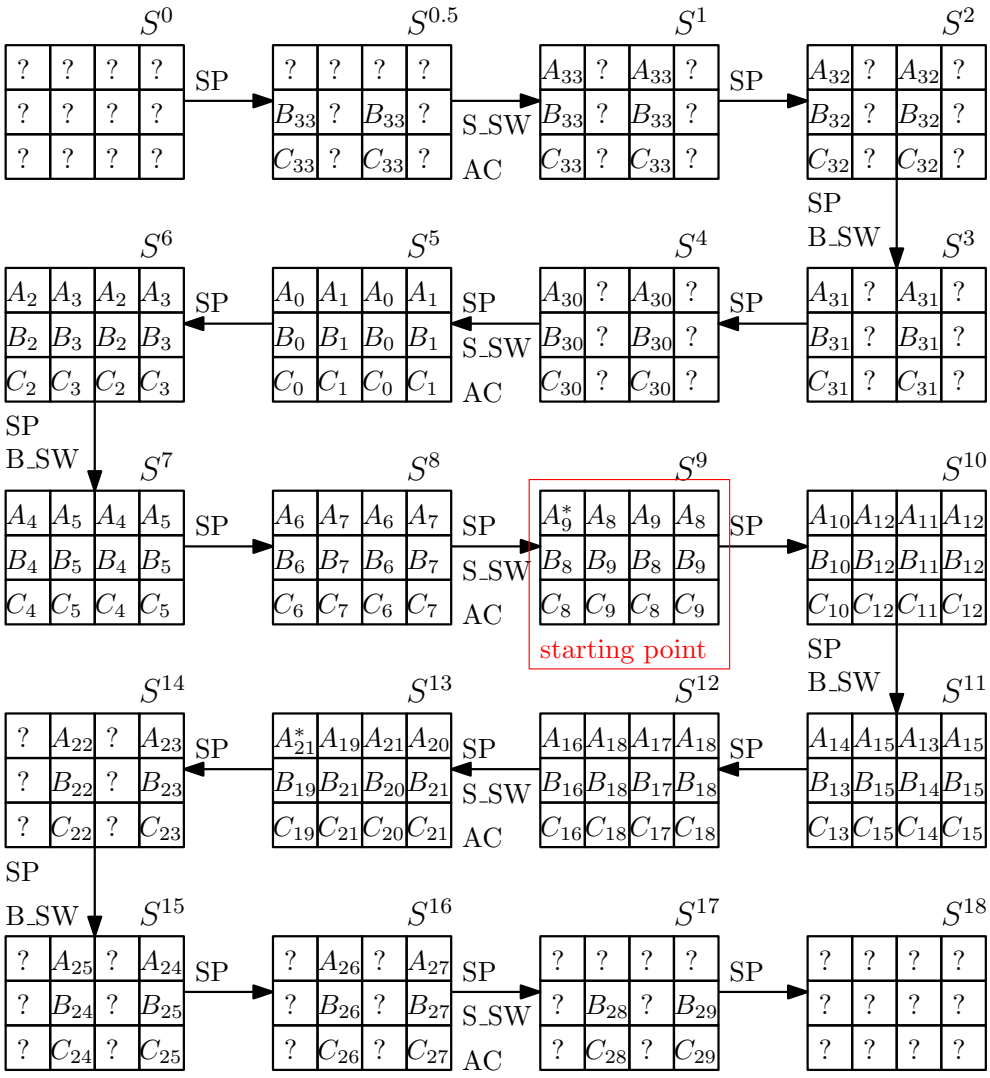

Fig. 13: Evolution of the internal difference for one input

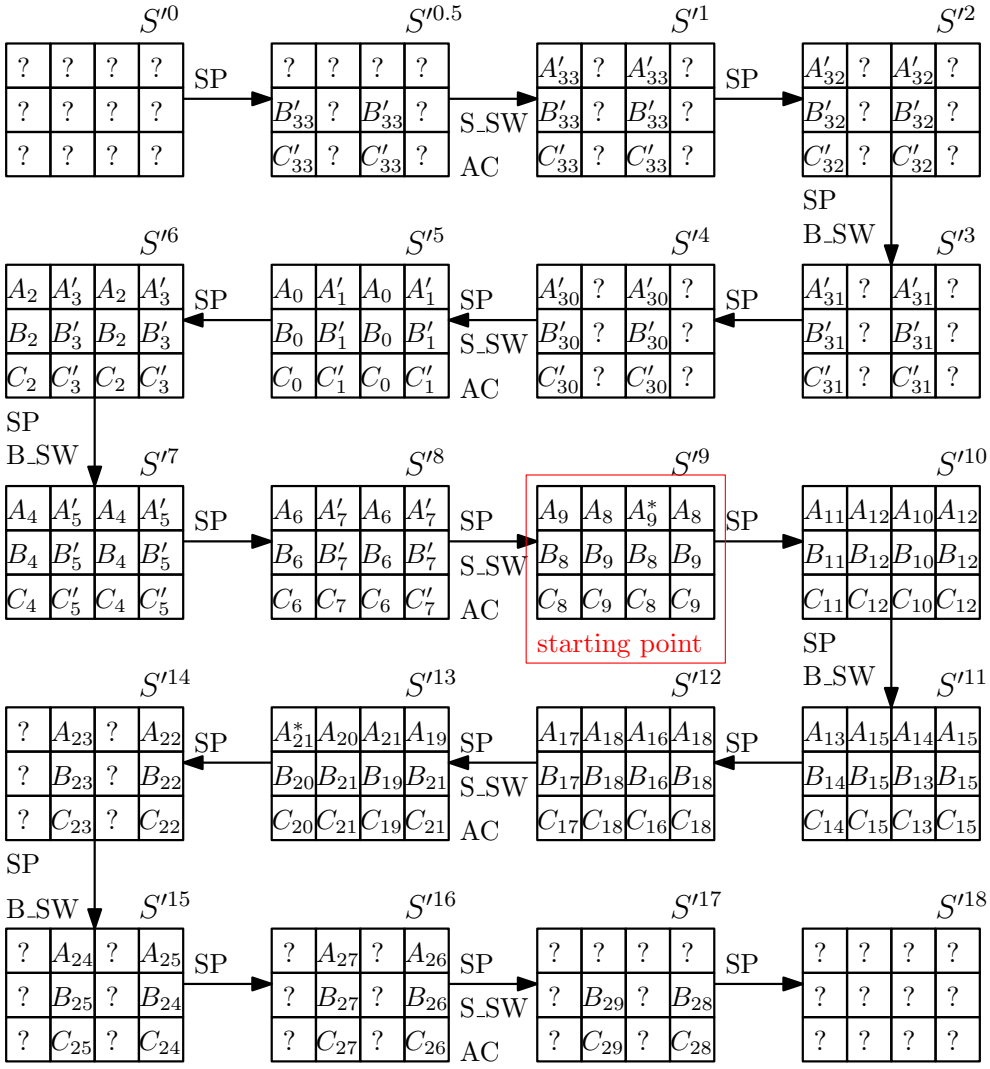

Fig. 14: Evolution of the internal difference for another input