Preimages and Collisions for Up to 5-Round Gimli-Hash Using Divide-and-Conquer Methods

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
willi.meier@fhnw.ch

Abstract. The Gimli permutation was proposed in CHES 2017 and the hash mode Gimli-Hash is now included in the Round 2 candidate Gimli in NIST's Lightweight Cryptography Standardization process. In the Gimli document, the security of the Gimli permutation has been intensively investigated. However, little is known about the security of Gimli-Hash. The designers of Gimli have claimed 2^{128} security against all attacks on Gimli-Hash, whose hash is a 256-bit value. Firstly, we present the trivial generic preimage attack on the structure of Gimli-Hash matching the 2^{128} security bound, both, in time and memory complexity. Following such a generic preimage attack framework, we then describe specific preimage attacks on 2/3/4/5 (out of 24) rounds of Gimli-Hash using divide-and-conquer methods. As will be shown, the application of the divide-and-conquer methods much benefits from the properties of the SP-box and the linear layer of Gimli. Therefore, this work can also be viewed to make the first step to exploit specific properties of the SP-box. Finally, the divide-and-conquer method was also applied to a collision attack on up to 5-round Gimli-Hash. As a support of our method, we provide a practical colliding four-block message pair for the full-state collision of 3-round Gimli-Hash. We hope our analysis can advance the understanding of Gimli-Hash.

Keywords: hash function · Gimli · Gimli-Hash · preimage attack · collision attack · divide-and-conquer

1 Introduction

As the demand for lightweight cryptographic primitives in industry increases, NIST is currently holding a public lightweight cryptography competition, aiming at selecting a lightweight cryptography standardization by combining the efforts from both academia and industry. Although such a competition started to call for submissions in 2018, considerable efforts have been put on the lightweight cryptography in academia since the publication of the ultra-lightweight block cipher PRESENT in CHES 2007 [8]. The last decade has also witnessed a lot of designs of lightweight cryptographic primitives, like PICCOLO [11], PHOTON [9], SIMON/SPECK [5], Midori [3], SKINNY [6], GIFT [4], and QARMA [2], etc.

Gimli was proposed by Bernstein et al. in CHES 2017 [7]. 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 [10] to explain how dangerous such a strategy would be. The attack described in [10] is for an ad-hoc mode and mainly exploits the fact that there is occasional 32-bit word communication among the 4 columns. As a response, the designers of Gimli claimed that such an ad-hoc mode has never appeared before and would never threaten the official authenticated encryption scheme and hash scheme based on Gimli.

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 scheme and hash scheme. As can be noted in the Gimli document [7], there has been an intensive scrutiny for the Gimli permutation. However, little is known about the AEAD and hash scheme. Thus, we are motivated to make the first step to look into the security of its hash scheme Gimli-Hash. Specifically, we would like to see whether it is still possible to exploit the fact that there is little communication between the 4 columns as done by Hamburg [10] to devise an attack on the AEAD scheme or the hash scheme.

As a result, the divide-and-conquer method starts to occur in our mind, which may fit well with the fact that there is little communication between the 4 columns. However, only exploiting such a fact is obviously insufficient. Thus, to make our divide-and-conquer method feasible and efficient, we further exploit the properties of the SP-box of Gimli and they are proven to be useful, as can be seen from our attacks.

Our Contributions. In this paper, we develop a divide-and-conquer method to analyze the security of Gimli-Hash. This method much benefits from the little communication between the 4 columns (linear layer) and the properties of the SP-box. While the property of the linear layer has been intensively exploited in Hamburg's attack [10], we are the first to investigate the properties of the SP-box and combine it with the linear layer to devise several attacks.

Specifically, we describe a trivial generic preimage attack on the structure of Gimli-Hash to match the claimed 2^{128} security bound at first. Following such a generic preimage attack framework, we can further devise specific improved preimage attacks on 2/3/4/5 rounds of Gimli-Hash with divide-and-conquer methods. Moreover, the divide-and-conquer method is also applied to a collision attack on up to 5-round Gimli-Hash and we provide a practical colliding four-block message pair for full-state collision of 3-round Gimli-Hash. Our results are summarized in Table 1.

Table 1: The analytical results of reduced Gimli-Hash

Method	Attack Type	Rounds	Memory	Time	Ref.
Meet-in-the-Middle	(second) preimage	arbitrary	2^{128}	2^{128}	
Divide-and-conquer	(second) preimage	2 (24~23)	2^{32}	2^{64}	Sec. 4.2
		3 (24~22)	2^{32}	2^{64}	Sec. 4.3
		4 (24~21)	2^{64}	2^{96}	Sec. 4.4
		5 (24~20)	2^{64}	2^{96}	Sec. 4.5
	collision	3 (24~22)	practical example		Sec. 5.2
		4 (24~21)	2^{64}	2^{65}	Sec. 5.1
		5 (24~20)	2^{64}	2^{65}	Sec. 5.1
Divide-and-conquer	(second) preimage	2 (2~1)	2^{32}	2^{64}	App. A.1
		3 (3~1)	2^{64}	2^{64}	App. A.2
		4 (4~1)	2^{64}	2^{96}	Sec. 4.4
	collision	4 (4~1)	2^{64}	2^{65}	Sec. 5.1

Organization. This paper is organized as follows. In Section 2, we introduce the notations, the Gimli permutation, some useful properties of the SP-box and the hash scheme Gimli-Hash. Then, the generic preimage attack on the structure of Gimli-Hash will be described in Section 3. Following such a generic framework, we present the preimage attacks and collision attacks using divide-and-conquer methods in Section 4 and Section 5 respectively. Finally, the paper is concluded in Section 6.

2 Preliminaries

In this section, we will present some notations, the description of the Gimli permutation and Gimli-Hash. Meanwhile, some useful properties of the SP-box will be discussed 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.

2.2 Description of Gimli

Gimli was proposed in CHES 2017 [7] and now 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.

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$

Figure 1: The Gimli state

The Gimli permutation is described in Algorithm 1. As can be seen from the description of the Gimli permutation, the permutation can be viewed as the following sequence of operations. For simplicity, we denote the SP-box, Small-Swap, Big-Swap and AddRoundConstant by SP, S_SW, B_SW and AC respectively.

$$\begin{aligned}
& (\mathrm{SP} \to \mathrm{S_SW} \to \mathrm{AC}) \to (\mathrm{SP}) \to (\mathrm{SP} \to \mathrm{B_SW}) \to (\mathrm{SP}) \\
\to\ & (\mathrm{SP} \to \mathrm{S_SW} \to \mathrm{AC}) \to (\mathrm{SP}) \to (\mathrm{SP} \to \mathrm{B_SW}) \to (\mathrm{SP}) \\
\to\ & (\mathrm{SP} \to \mathrm{S_SW} \to \mathrm{AC}) \to (\mathrm{SP}) \to (\mathrm{SP} \to \mathrm{B_SW}) \to (\mathrm{SP}) \\
\to\ & (\mathrm{SP} \to \mathrm{S_SW} \to \mathrm{AC}) \to (\mathrm{SP}) \to (\mathrm{SP} \to \mathrm{B_SW}) \to (\mathrm{SP}) \\
\to\ & (\mathrm{SP} \to \mathrm{S_SW} \to \mathrm{AC}) \to (\mathrm{SP}) \to (\mathrm{SP} \to \mathrm{B_SW}) \to (\mathrm{SP}) \\
\to\ & (\mathrm{SP} \to \mathrm{S_SW} \to \mathrm{AC}) \to (\mathrm{SP}) \to (\mathrm{SP} \to \mathrm{B_SW}) \to (\mathrm{SP}).
\end{aligned}$$

Therefore, when we attack r rounds of Gimli, we consider the sequence of first r operations in the main content and we believe it is meaningful. For example, when our target is 5-round Gimli, the sequence of operations is

$$\begin{aligned} & (\mathrm{SP} \to \mathrm{S\_SW} \to \mathrm{AC}) \to (\mathrm{SP}) \to (\mathrm{SP} \to \mathrm{B\_SW}) \to (\mathrm{SP}) \\ \to\; & (\mathrm{SP} \to \mathrm{S\_SW} \to \mathrm{AC}). \end{aligned}$$

To make this work complete, we also considered the sequence of the last r operations in Appendix. We suggest the readers consult the main content at first. Then, the content in the appendix will be easy to understand.

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**
    $x \leftarrow s_{0,j} \lll 24$
    $y \leftarrow s_{1,j} \lll 9$
    $z \leftarrow s_{2,j}$

    $s_{2,j} \leftarrow x \oplus z \ll 1 \oplus (y \wedge z) \ll 2$
    $s_{1,j} \leftarrow y \oplus x \oplus (x \vee z) \ll 1$
    $s_{0,j} \leftarrow z \oplus y \oplus (x \wedge y) \ll 3$
  end for

  if $r \bmod 4 = 0$ **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 $r \bmod 2 = 0$ **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 $r \bmod 4 = 0$ **then**
    $s_{0,0} \leftarrow s_{0,0} \oplus \mathtt{0x9e377900} \oplus r$
  end if
end for
return $(s_{i,j})$

2.3 SP-box

In this section, we present some useful properties of the SP-box in Gimli. The SP-box consists of three sub-operations: rotations of the first and second words; a 3-input nonlinear T-function; and a swap of the first and third words. Specifically, consider one column $(x, y, z) \in F^3_{2^{32}}$. Then the SP-box will update (x, y, z) as follows:

$$\begin{aligned} x &\leftarrow x \lll 24 \\ y &\leftarrow y \lll 9 \\ x &\leftarrow x \oplus z \ll 1 \oplus (y \wedge z) \ll 2 \\ y &\leftarrow y \oplus x \oplus (x \vee z) \ll 1 \\ z &\leftarrow z \oplus y \oplus (x \wedge y) \ll 3 \\ x &\leftarrow z \\ z &\leftarrow x \end{aligned}$$

Property 1. *Suppose the input to an SP-box is (x, y, z) and the corresponding output is (x', y', z'). Then, if $y[31 \sim 23] = 0$ and $y[19 \sim 0] = 0$, we can know that x' is independent of x.*

Proof. This can be easily proved by considering the expression to calculate x' as follows.

$$x' = z \oplus (y \lll 9) \oplus ((x \lll 24) \wedge (y \lll 9)) \ll 3.$$

□

Property 2. *Suppose the input to an SP-box is (x, y, z) and the corresponding output is (x', y', z'). Then, given (y, z, x'), the probability Pr that (y, z, x') is a valid tuple is 2^{-15} without knowing x.*

Proof. Based on the expression to calculate x', we already know that 3 bits of x' are independent of x, which are x'_j $(0 \leq j \leq 2)$. Moreover, supposing $y'' = y \lll 9$, if $y''_i = 0$ $(0 \leq i \leq 28)$, we can also compute x'_{i+3} without the knowledge of x.

$$x' = z \oplus y'' \oplus ((x \lll 24) \wedge y'') \ll 3.$$

Supposing y is uniformly distributed, Pr can be calculated as follows:

$$Pr = 2^{-3} \times \frac{\sum_{i=0}^{29}(C_{29}^{i} \times 2^{-i})}{2^{29}} \approx 2^{-15}.$$

To verify it, we first randomly generate a value for x'. Then, we randomly generate n pairs of (y, z) and determine the computable bits of x' for each pair. If these computable bits match those of x', we increase the counter cnt by 1. Experiments show that $\frac{cnt}{n}$ is close to 2^{-15} and slightly lower. □

Property 3. *Suppose the input to an SP-box is (x, y, z) and the corresponding output is (x', y', z'). Then, given (z', y, z), we can determine (x, x', y'). Moreover, given a random tuple (z', y', y, z), the probability that it is valid is 2^{-32}.*

Proof. Considering the expression to calculate z', it is easy to compute x if (z', y, z) are fixed, as shown below.

$$z' = (x \lll 24) \oplus z \ll 1 \oplus ((y \lll 9) \wedge z) \ll 2.$$
$$x = (z' \oplus z \ll 1 \oplus ((y \lll 9) \wedge z) \ll 2) \ggg 24.$$

After x is computed, (x, y, z) are all known and we can therefore compute (x', y').

Since we can compute y' according to the knowledge of (z', y, z), it is natural to conclude that a random tuple (z', y', y, z) is valid with probability 2^{-32}. □

Property 4. *Suppose the input to an SP-box is (x, y, z) and the corresponding output is (x', y', z'). Then, given (z', y', x), it is a valid tuple with probability 2^{-1}. Once it is a valid tuple, we can determine $(x'[30 \sim 0], y, z[30 \sim 0])$.*

Proof. To prove this, let $x'' = x \lll 24$ and $y'' = y \lll 9$. Then, x'' is also known. Consider the expressions to calculate z' and y', as shown below.

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

Firstly, we can compute

$$\begin{aligned} y''[0] &= y'[0] \oplus x''[0], \\ z[0] &= z'[1] \oplus x''[1], \\ z[1] &= z'[2] \oplus x''[2], \\ y''[1] &= y'[1] \oplus x''[1] \oplus (x''[0] \vee z[0]). \end{aligned}$$

Then, we can recursively compute

$$\begin{aligned} y''[j] &= y'[j] \oplus x''[j] \oplus (x''[j-1] \vee z[j-1]), \\ z[k] &= z'[k+1] \oplus x''[k+1] \oplus (y''[k-1] \wedge z[k-1]). \end{aligned}$$

for $(2 \le j \le 31)$ and $(2 \le k \le 30)$. Thus, we can uniquely compute y and $z[30 \sim 0]$ if given (z', y', x). Then, according to the following expression to calculate x'

$$x' = z \oplus y'' \oplus ((x \lll 24) \wedge y'') \ll 3,$$

we can also determine $x'[30 \sim 0]$.

Moreover, note that $z'_0 = x''_0$. Thus, if given a random tuple (z', y', x), it is a valid tuple with probability 2^{-1}. □

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. In the rest part, we denote Small-Swap by S_SW and denote Big-Swap by B_SW.

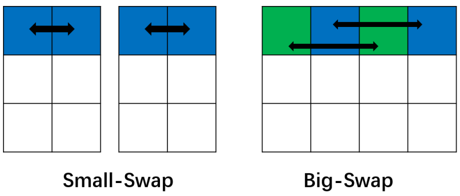

Figure 2: The linear layer

2.5 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 $M_0, M_1, \cdots$.

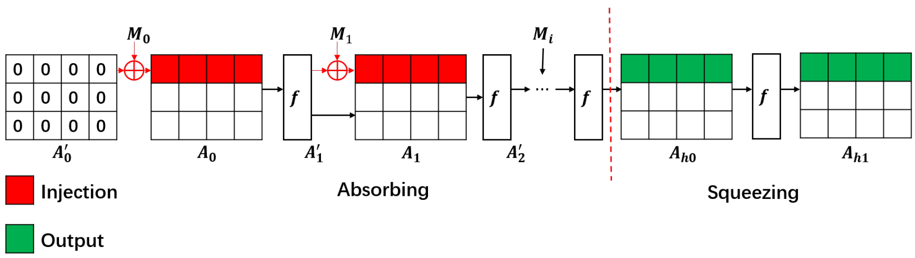

Figure 3: The process to compress the message

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 $0 \le b \le 15$. 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, for simplicity, we denote the initial state (all zero) by A'_0. The state after the first block message is added is denoted by A_0. Recursively, we denote the state before adding the i-th ($i \geq 0$) message block M_i by A'_i. After M_i is added, the state is denoted by A_i. Formally, we have the following relations:

$$\begin{aligned} A_i &= A'_i \oplus (M_i \| 0^{256}), \\ A'_{i+1} &= f(A_i). \end{aligned}$$

Finally, the last two states of the output are denoted by A_{h0} and A_{h1} respectively.

3 Generic Preimage Attack on Gimli-Hash

The designers of Gimli-Hash claim that it achieves 2^{128} security against all attacks. To have a better understanding, we show the generic preimage attack on Gimli-Hash to explain the claimed security bound. The attack is illustrated in Figure 4.

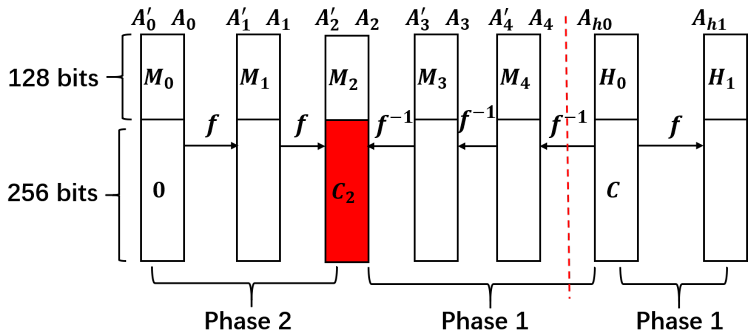

Figure 4: Generic preimage attack on Gimli-Hash

Specifically, given a hash value (H_0, H_1), the generic preimage attack procedure can be divided into two phases:

Phase 1: Set the rate part of A_{h0} to the value of H_0. Randomly choose a value for the capacity part of A_{h0}. In this way, A_{h0} is fully determined and we can compute $A_{h1} = f(A_{h0})$. It is expected to make the rate part of A_{h1} match with H_1 after trying 2^{128} random values for the capacity part of A_{h0}, Once a valid capacity part of A_{h0} is found, a valid value for the full state of A_{h0} is determined, thus making the application of f^{-1} to A_{h0} feasible. Then, we randomly choose 2^{128} values for (M_3, M_4) (note that M_4 can not take 2^{128} values due to the padding rule) and compute backward to obtain the capacity part of A_2 denoted by C_2. Store the corresponding 2^{128} values of C_2 in a table TA_0.

Phase 2: Similarly, randomly choose 2^{128} values of (M_0, M_1) and compute the capacity part of A'_2, which is also C_2. Store the 2^{128} values of C_2 in a table TA_1. Find a match between TA_0 and TA_1. Since there are $2^{128+128} = 2^{256}$ pairs and C_2 is a 256-bit value, it is expected that there is one match. Once the match is found, we can compute M_2 and therefore obtain the preimage.

Consequently, the time complexity and memory complexity of this generic preimage attack are both 2^{128}.

3.1 Discussion

As can be seen from the generic attack in Figure 4, it consists of two phases.

The first phase is to find a valid capacity part of the A_{h0}. After it is found, we apply f^{-1} to this state and obtain A_4. To satisfy the padding rule, we choose a random value of M_4, whose size is smaller than 16 bytes. Then, we can compute A'_4. Then, we can further apply f^{-1} to A'_4 and obtain another new state A_3. Next, we choose a random value for M_3 of size 16 bytes and compute A'_3. Finally, we apply f^{-1} to A'_3 and obtain the value of the capacity part of A'_2, i.e. C_2. At this phase, 2^{128} possible random values of (M_3, M_4) will be tried in order to collect 2^{128} possible values of C_2.

At the second phase, we choose 2^{128} random values of (M_0, M_1) and compute the corresponding C_2. Then, if we can find a match in C_2 which is computed by (M_0, M_1) and (M_3, M_4) respectively, we can always use the degree of freedom of M_2 to connect the choice for (M_0, M_1) and (M_3, M_4) and finally obtain the preimage.

What we want to emphasize here is that such a generic attack is irrelevant to the padding rule, which can be satisfied by choosing a non-full (smaller than 16 bytes) value for M_4.

4 Preimage Attacks with Divide-and-Conquer Methods

Inspired by the above generic preimage attack, we can devise preimage attacks on 2/3/4/5 rounds of Gimli-Hash. The main idea is to reduce the time complexity of the first and second phase of the generic attack respectively. To gain advantage over the generic attack, some properties of the SP-box and the linear layer will be exploited. We have to emphasize once again that in the main content, the r-round Gimli-Hash is treated as the sequence of the first r operations of the 24-round permutation, as stated in Section 2.2.

4.1 Overview

We extend the above generic preimage attack on Gimli-Hash illustrated in Figure 4 to specific preimage attacks on 2/3/4/5 rounds of Gimli-Hash. Our attack consists of two phases as well.

The first phase is to find a valid capacity part of A_{h0} as in the generic attack. Then, we properly choose just one (not 2^{128}) value for two message blocks (M_3, M_4) and compute backward to obtain the capacity part of A'_2, i.e. C_2. As explained in the generic attack, the padding rule can be satisfied by properly choosing M_4. Thus, the influence of the padding rule has been eliminated at this phase.

At the second phase, different from the generic attack which uses a meet-in-the-middle method to achieve the match in C_2, we will use a divide-and-conquer method to match the C_2 computed at the first phase. To achieve it, the degree of freedom of (M_0, M_1) will be utilized. Note that (M_0, M_1) can take 2^{256} possible values and C_2 is a 256-bit value. Therefore we can expect to find one solution of (M_0, M_1) to match C_2 . If it cannot be found, which happens with a negligible probability, we choose another proper value of (M_3, M_4) and repeat. We have to stress that it is expected to use only one value of (M_3, M_4). Once a solution of (M_0, M_1) is found, we can immediately compute M_2 as follows and obtain the preimage $(M_0, M_1, M_2, M_3, M_4)$ of the hash value (H_0, H_1).

$$(M_2 \| 0^{256}) = A_2 \oplus A'_2.$$

In this way, our preimage attacks are reduced to two subproblems. The first problem is how to find a valid capacity part of A_{h0} to match H_1 with complexity less than 2^{128}. The second problem is how to match a given capacity part with complexity less than 2^{128}. Thus, in the following description of our preimage attacks on 2/3/4/5 rounds of Gimli-Hash, we will separately explain how to find a valid capacity part of A_{h0} and how to match a given capacity part by utilizing the degree of freedom of (M_0, M_1).

4.2 Preimage Attack on 2-Round Gimli-Hash

We present the details of the preimage attack on 2-round Gimli-Hash in this part. As shown in Figure 5, we denote the hash value by

$$(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7),$$

where $h_i \in F_2^{32}$. Moreover, the capacity part of A_{h0} is denoted by $s_{i,j}$ $(1 \le i \le 2, 0 \le j \le 3)$.

4.2.1 Computing a Valid Capacity Part

Similar to the generic attack, we first generate a valid value for the capacity part of A_{h0}, as illustrated in Figure 5. The corresponding procedure is described as follows.

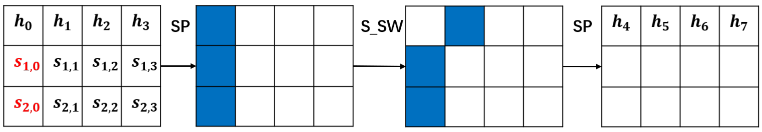

Figure 5: Generate a valid capacity part for the preimage attack on 2-round Gimli-Hash

Step 1: Randomly choose 2^{32} values of $(s_{1,0}, s_{2,0})$. Then, with the Property 2 of SP-box, we can find about $2^{32-15} = 2^{17}$ candidates for $(s_{1,0}, s_{2,0})$ which may match h_4. Store these values in a table CT_0.

Step 2: Similarly, we randomly choose 2^{32} values of $(s_{1,j}, s_{2,j})$ $(1 \le j \le 3)$ and partially match h_{j+4}. Store the candidates in table CT_j respectively.

Step 3: Exhaust all possible combinations between CT_0 and CT_1. For each combination, (h_4, h_5) can be fully computed and we compare it with the given hash value. It is expected that there is only one valid value of $(s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1})$ since there are totally 2^{64} random values for it.

Step 4: Similarly, we can obtain the value of $(s_{1,2}, s_{2,2}, s_{1,3}, s_{2,3})$ to match (h_6, h_7).

The time complexity can be evaluated as $2^{32} + 2^{17+17} = 2^{34}$ times of 2-round Gimli permutation. In this way, we can find a valid capacity part of A_{h0}.

4.2.2 Matching the Capacity Part

We expand on how to match a given capacity part by utilizing the degree of freedom of the first two blocks. To have a better understanding, it is better to refer to Figure 6 for the meaning of the notations in the following description. Specifically, $(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3})$ and $(b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$ can be randomly chosen. The goal is to match a given

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

The procedure to achieve the goal is described below.

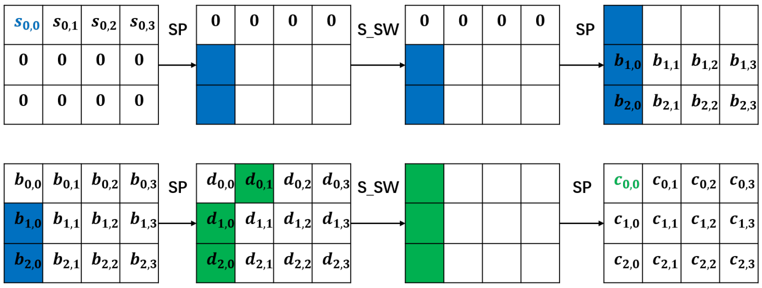

Figure 6: Preimage attack on 2-round Gimli-Hash

Step 1: Exhaust all 2^{64} possible values of $(c_{0,0},\ s_{0,0})$. Then, the tuple $(b_{1,0}, b_{2,0}, d_{1,0}, d_{2,0})$ can be computed for each guess of $(c_{0,0},\ s_{0,0})$. According to the Property 3 of SP-box, the tuple $(b_{1,0}, b_{2,0}, d_{1,0}, d_{2,0})$ is valid with probability 2^{-32}. Thus, we expect to obtain 2^{64-32} possible values of $(c_{0,0},\ s_{0,0})$ to match $(c_{1,0}, c_{2,0})$. For these 2^{32} valid values, we will collect 2^{32} possible values of $(d_{0,0}, d_{0,1})$. Note that according to the Property 3, $d_{0,0}$ can be computed when $(b_{1,0}, b_{2,0}, d_{1,0}, d_{2,0})$ is a valid tuple. Store all the 2^{32} valid values of the tuple $(d_{0,0}, d_{0,1}, c_{0,0}, s_{0,0})$ in the table GA_0.

Step 2: Similarly, exhaust all 2^{64} possible values of $(c_{0,1},\ s_{0,1})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,0}, d_{0,1}, c_{0,1}, s_{0,1})$ and store them in the table GA_1.

Step 3: Similarly, exhaust all 2^{64} possible values of $(c_{0,2},\ s_{0,2})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,2}, d_{0,3}, c_{0,2}, s_{0,2})$ and store them in the table GA_2.

Step 4: Similarly, exhaust all 2^{64} possible values of $(c_{0,3},\ s_{0,3})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,2}, d_{0,3}, c_{0,3}, s_{0,3})$ and store them in the table GA_3.

After obtaining GA_0, GA_1, GA_2 and GA_3, we can use GA_0 and GA_1 and expect to find a match in $(d_{0,0}, d_{0,1})$ since there are 2^{64} pairs in total. Similarly, we can use GA_2 and GA_3 to find a match in $(d_{0,2}, d_{0,3})$. Once the match is found, we get the solution of

$$(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$$

which will correspond to the given capacity part

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

In conclusion, the time and memory complexity of the preimage attack on 2-round Gimli-Hash are 2^{64} and 2^{32}, respectively.

4.3 Preimage Attack on 3-Round Gimli-Hash

We present the details of the preimage attack on 3-round Gimli-Hash in this part. As shown in Figure 7, we denote the hash value by

$$(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7),$$

where $h_i \in F_2^{32}$. Moreover, the capacity part of A_{h0} is denoted by $s_{i,j}$ $(1 \le i \le 2, 0 \le j \le 3)$.

4.3.1 Computing a Valid Capacity Part

The main idea to compute a valid capacity part of A_{h0} for the preimage attack on 3-round Gimli-Hash is illustrated in Figure 7. The procedure can be divided into two parallel computations, as shown below.

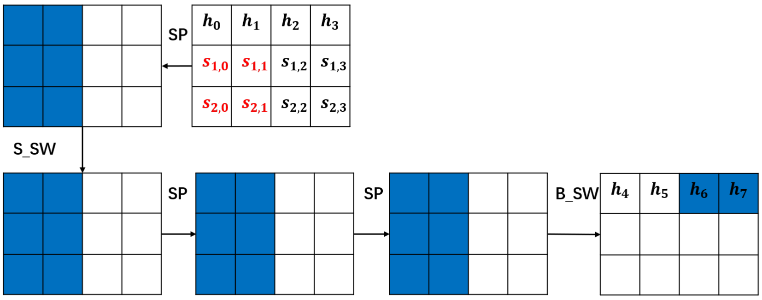

Figure 7: Generate a valid capacity part for the preimage attack on 3-round Gimli-Hash

Parallel-1: Randomly choose 2^{64} values of $(s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1})$. Then, we can compute (h_6, h_7). Compare the computed (h_6, h_7) with the given hash value. It is expected that there will be one value of $(s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1})$ to match the given (h_6, h_7).

Parallel-2: Randomly choose 2^{64} values of $(s_{1,2}, s_{2,2}, s_{1,3}, s_{2,3})$. Then, we can compute (h_4, h_5). Compare the computed (h_4, h_5) with the given hash value. It is expected that there will be one value of $(s_{1,2}, s_{2,2}, s_{1,3}, s_{2,3})$ to match the given (h_4, h_5).

Hence, we can find a valid capacity part of A_{h0} with 2^{64} time complexity.

4.3.2 Matching the Capacity Part

As shown in Figure 8, $(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3})$ and $(b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$ can be randomly chosen. The goal is to match a given

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

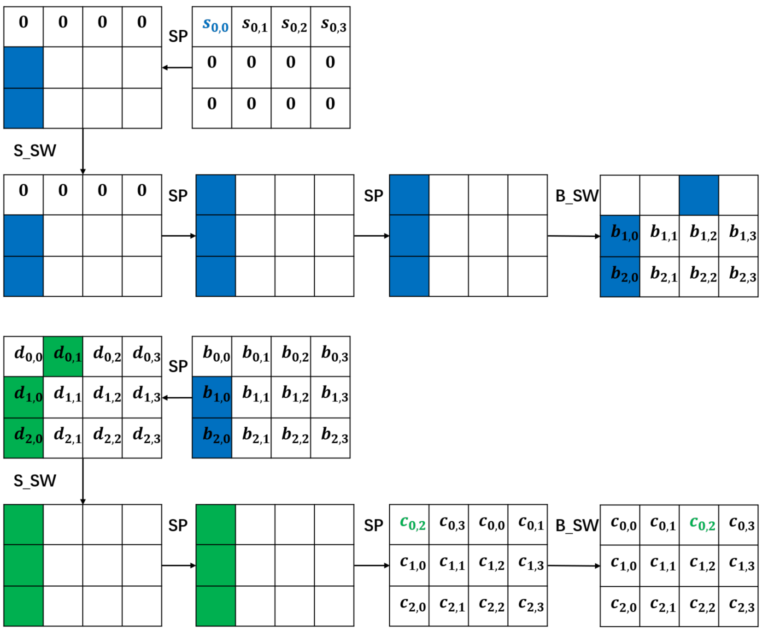

Figure 8: Preimage attack on 3-round Gimli-Hash

The procedure to gain this goal is as follows. To have a better understanding, we suggest to refer to Figure 8 for the meaning of the notations in the following description.

Step 1: Exhaust all 2^{64} possible values of $(s_{0,0}, c_{0,2})$. Note that we can compute backward to obtain $(d_{1,0}, d_{2,0})$ for each guess of $c_{0,2}$. Moreover, we can compute forward to obtain $(b_{1,0}, b_{2,0})$ for each guess of $s_{0,0}$. In other words, for each value of $(s_{0,0}, c_{0,2})$, we can obtain a tuple $(d_{1,0}, d_{2,0}, b_{1,0}, b_{2,0})$. Thanks to the Property 3 of SP-box, the obtained tuple $(d_{1,0}, d_{2,0}, b_{1,0}, b_{2,0})$ is valid with probability 2^{-32}. Consequently, we will finally obtain 2^{32} valid values of $(s_{0,0}, c_{0,2})$. Each valid value of $(s_{0,0}, c_{0,2})$ will suggest a valid value of $(d_{0,0}, d_{0,1})$, where $d_{0,0}$ is computed according to the valid tuple $(d_{1,0}, d_{2,0}, b_{1,0}, b_{2,0})$. Finally, we can collect 2^{32} values of the tuple $(d_{0,0}, d_{0,1}, s_{0,0}, c_{0,2})$ and store them in MT_0.

Step 2: Similarly, exhaust all 2^{64} possible values of $(s_{0,1}, c_{0,3})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,0}, d_{0,1}, s_{0,1}, c_{0,3})$ and store them in MT_1.

Step 3: Similarly, exhaust all 2^{64} possible values of $(s_{0,2}, c_{0,0})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,2}, d_{0,3}, s_{0,2}, c_{0,0})$ and store them in MT_2.

Step 4: Similarly, exhaust all 2^{64} possible values of $(s_{0,3}, c_{0,1})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,2}, d_{0,3}, s_{0,3}, c_{0,1})$ and store them in MT_3.

After obtaining MT_i $(0 \leq i \leq 3)$, we can use MT_0 and MT_1 to find a match in $(d_{0,0}, d_{0,1})$. Since there are 2^{64} such pairs and they match with each other with probability 2^{-64}, we expect to find one match. Similarly, we can use MT_2 and MT_3 to find a match in $(d_{0,2}, d_{0,3})$. After finding the match, we can obtain the final valid tuple

$$(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, c_{0,0}, c_{0,1}, c_{0,2}, c_{0,3}),$$

which can be used to compute

$$(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3}).$$

Hence, the time and memory complexity for the preimage attack on 3-round Gimli-Hash are 2^{64} and 2^{32} respectively.

4.4 Preimage Attack on 4-Round Gimli-Hash

We present the details of the preimage attack on 4-round Gimli-Hash in this part. As shown in Figure 9, we denote the hash value by

$$(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7),$$

where $h_i \in F_2^{32}$. Moreover, the capacity part of A_{h0} is denoted by $s_{i,j}$ $(1 \leq i \leq 2, 0 \leq j \leq 3)$.

4.4.1 Computing a Valid Capacity Part

The main idea to compute a valid capacity part of A_{h0} for the preimage attack on 4-round Gimli-Hash is illustrated in Figure 9. The procedure can be divided into three steps, as shown below.

Step 1: Randomly choose 2^{64} values of $(s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1})$. Then, according to Property 2 of SP-box, we can partially compute (h_4, h_5). Compare the computable bits of (h_4, h_5) with the given hash value. It is expected there will be $2^{64-15\times 2} = 2^{34}$ valid values of $(s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1})$ left. Store these values in the table LT_0.

Step 2: Randomly choose 2^{64} values of $(s_{1,2}, s_{2,2}, s_{1,3}, s_{2,3})$. Then, according to Property 2 of SP-box, we can partially compute (h_6, h_7). Compare the computable bits of (h_6, h_7) with the given hash value. It is expected there will be $2^{64-30} = 2^{34}$ valid values of $(s_{1,2}, s_{2,2}, s_{1,3}, s_{2,3})$ left. Store these values in the table LT_1.

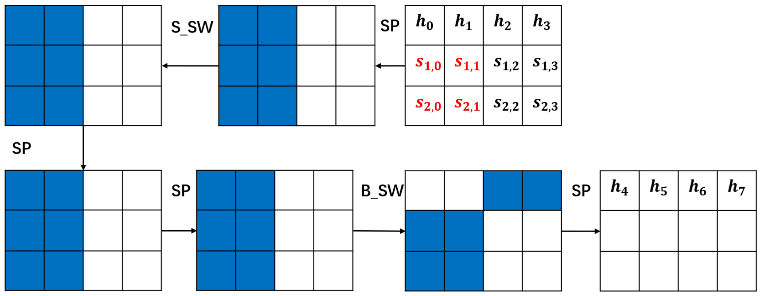

Figure 9: Generate a valid capacity part for the preimage attack on 4-round Gimli-Hash

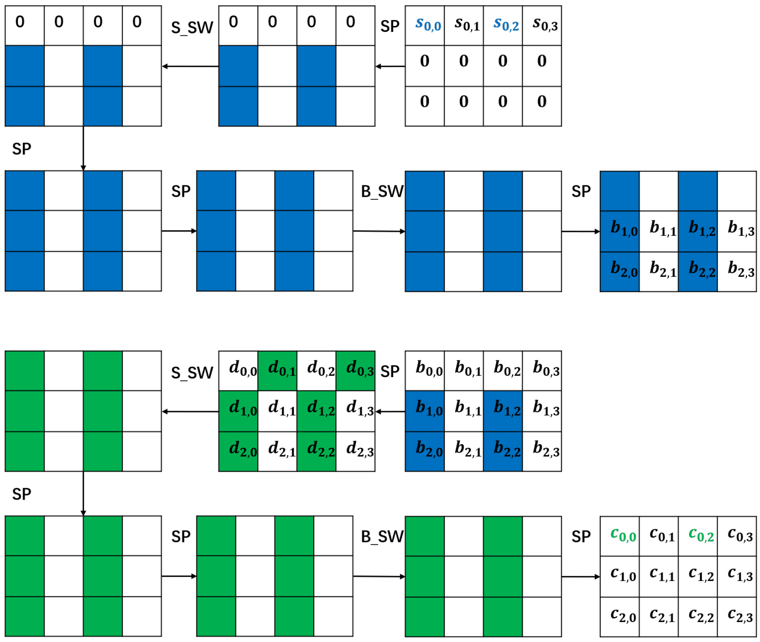

Figure 10: Preimage attack on 4-round Gimli-Hash

Step 3: Exhaust all the $2^{34+34} = 2^{68}$ possible combinations for $s_{i,j}$ $(1 \leq i \leq 2, 0 \leq j \leq 3)$ between LT_0 and LT_1. For each combination, we can compute the complete (h_4, h_5, h_6, h_7) and compare it with the given hash value. Since we tried 2^{128} possible values for $s_{i,j}$ $(1 \leq i \leq 2, 0 \leq j \leq 3)$, it is expected that one of them will match the given hash value.

Hence, with 2^{68} time and 2^{34} memory, we can find a valid capacity part of A_{h0}.

4.4.2 Matching the Capacity Part

As shown in Figure 10, $(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3})$ and $(b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$ can be randomly chosen. The goal is to match a given

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

For a better understanding, we suggest to refer to Figure 10 for the meaning of the notations in the following description.

Pre-computing some tables. Before explaining the details, we firstly introduce some tables. According to Figure 10, we can easily observe that

- $(b_{1,0}, b_{2,0}, b_{1,2}, b_{2,2})$ only depends on $(s_{0,0}, s_{0,2})$, thus taking at most 2^{64} possible values.
- $(b_{1,1}, b_{2,1}, b_{1,3}, b_{2,3})$ only depends on $(s_{0,1}, s_{0,3})$, thus taking at most 2^{64} possible values.

Thus, we can pre-compute a table of size 2^{64} to store the above mapping relations. Specifically, by exhausting all 2^{64} possible values of $(s_{0,0}, s_{0,2})$, we can obtain 2^{64} values of the tuple

$$(b_{1,0}, b_{2,0}, b_{1,2}, b_{2,2}, s_{0,0}, s_{0,2}).$$

Store the 2^{64} values in a table ST_0 of size 2^{64}, where the $(b_{1,0} \times 2^{32} + b_{2,0})$-th row of ST_0 stores the value of $(b_{1,2}, b_{2,2}, s_{0,0}, s_{0,2})$.

Similarly, by exhausting all 2^{64} possible values of $(s_{0,1}, s_{0,3})$, we can obtain 2^{64} values of the tuple

$$(b_{1,1}, b_{2,1}, b_{1,3}, b_{2,3}, s_{0,1}, s_{0,3}).$$

Store the 2^{64} values in a table ST_1 of size 2^{64}, where the $(b_{1,1} \times 2^{32} + b_{2,1})$-th row of ST_1 stores the value of $(b_{1,3}, b_{2,3}, s_{0,1}, s_{0,3})$.

Starting using the tables. After preparing the above two tables, we now describe how to match a given capacity part by utilizing the first two message blocks. We suggest the readers to refer to Figure 10 for a better understanding of our following attack procedure.

Step 1: Exhaust 2^{64} possible values of $(c_{0,0}, c_{0,2})$. For each guess of $(c_{0,0}, c_{0,2})$, $(d_{1,0}, d_{2,0}, d_{1,2}, d_{2,2})$ will be determined. Then, for each such guess, we further exhaust 2^{32} possible values of $d_{0,0}$. For each guessed value of $(c_{0,0}, c_{0,2}, d_{0,0})$, $(d_{0,0}, d_{1,0}, d_{2,0})$ can be fully determined and we can therefore compute $(b_{1,0}, b_{2,0})$. According to the computed value of $(b_{1,0}, b_{2,0})$, we retrieve the $(b_{1,0} \times 2^{32} + b_{2,0})$-th row of ST_0 and obtain the corresponding value of $(b_{1,2}, b_{2,2}, s_{0,0}, s_{0,2})$. At this point, $(b_{1,2}, b_{2,2}, d_{1,2}, d_{2,2})$ is determined. According to the Property 3 of SP-box, the obtained tuple $(b_{1,2}, b_{2,2}, d_{1,2}, d_{2,2})$ is valid with probability 2^{-32}. Once it is valid, we can obtain the corresponding value of $d_{0,2}$. In other words, each guessed value of $(c_{0,0}, c_{0,2}, d_{0,0})$ is correct with probability 2^{-32}. Thus, only 2^{64} possible values of $(c_{0,0}, c_{0,2}, d_{0,0})$ will survive. Thus, we can finally obtain 2^{64} possible values of the following tuple

$$(d_{0,0}, d_{0,1}, d_{0,2}, d_{0,3}, c_{0,0}, c_{0,2}, s_{0,0}, s_{0,2}).$$

Store the 2^{64} values in a table FT_0.

Step 2: Exhaust 2^{64} possible values of $(c_{0,1}, c_{0,3})$. For each guess of $(c_{0,1}, c_{0,3})$, $(d_{1,1}, d_{2,1}, d_{1,3}, d_{2,3})$ will be determined. Then, for each such guess, we further exhaust 2^{32} possible values of $d_{0,1}$. For each guessed value of $(c_{0,1}, c_{0,3}, d_{0,1})$, $(d_{0,1}, d_{1,1}, d_{2,1})$ can be fully determined and we can therefore compute $(b_{1,1}, b_{2,1})$. According to the computed value of $(b_{1,1}, b_{2,1})$, we retrieve the $(b_{1,1} \times 2^{32} + b_{2,1})$-th row of ST_1 and obtain the corresponding value of $(b_{1,3}, b_{2,3}, s_{0,1}, s_{0,3})$. At this point, $(b_{1,3}, b_{2,3}, d_{1,3}, d_{2,3})$ is determined. According to the Property 3 of SP-box, the obtained tuple $(b_{1,3}, b_{2,3}, d_{1,3}, d_{2,3})$ is valid with probability 2^{-32}. Once it is valid, we can obtain the corresponding value of $d_{0,3}$. In other words, each guessed value of $(c_{0,1}, c_{0,3}, d_{0,1})$ is correct with probability 2^{-32}. Thus, only 2^{64} possible values of $(c_{0,1}, c_{0,3}, d_{0,1})$ will survive. Thus, we can finally obtain 2^{64} possible values of the following tuple

$$(d_{0,0}, d_{0,1}, d_{0,2}, d_{0,3}, c_{0,1}, c_{0,3}, s_{0,1}, s_{0,3}).$$

Store the 2^{64} values in a table FT_1.

After obtaining FT_0 and FT_1, find a match in $(d_{0,0}, d_{0,1}, d_{0,2}, d_{0,3})$ between the table FT_0 and FT_1. Since there are 2^{128} pairs and the probability that they match each other is 2^{-128}, we expect to find one match. For this match, we can know the corresponding $(s_{0,0}, s_{0,2}, c_{0,0}, c_{0,2}, s_{0,1}, s_{0,3}, c_{0,1}, c_{0,3})$, which can be used to compute the tuple $(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$.

Consequently, the time complexity of the preimage attack on 4-round Gimli-Hash is 2^{96} while the memory complexity is 2^{64}.

4.5 Preimage Attack on 5-Round Gimli-Hash

We present the details of the preimage attack on 5-round Gimli-Hash in this part. As shown in Figure 11, we denote the hash value by

$$(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7),$$

where $h_i \in F_2^{32}$. Moreover, the capacity part of A_{h0} is denoted by $s_{i,j}$ $(1 \leq i \leq 2, 0 \leq j \leq 3)$.

4.5.1 Computing a Valid Capacity Part

The main idea to compute a valid capacity part of A_{h0} for the preimage attack on 5-round Gimli-Hash is illustrated in Figure 11. The procedure can be divided into 6 steps, as shown below.

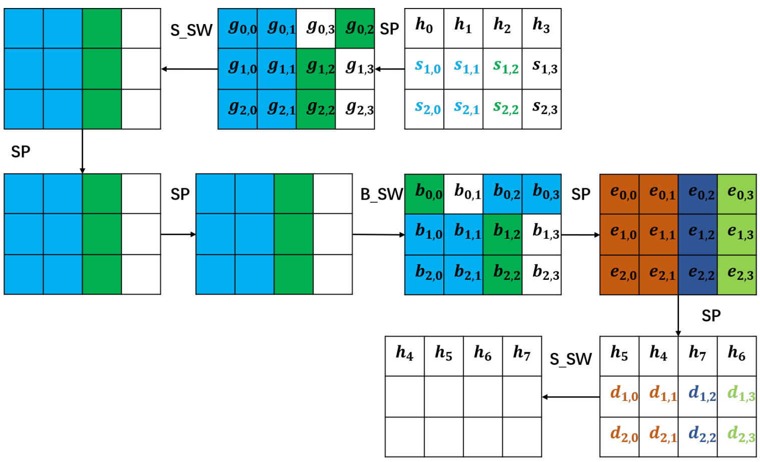

Figure 11: Generate a valid capacity part for the preimage attack on 5-round Gimli-Hash

Step 1: Randomly choose 2^{64} values of $(s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1})$. For each value of $(s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1})$, we can compute the corresponding $(b_{1,0}, b_{2,0}, b_{1,1}, b_{2,1})$. Store the 2^{64} values of the tuple

$$(b_{1,0}, b_{2,0}, b_{1,1}, b_{2,1}, s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1})$$

in table denoted by BT_0.

Step 2: Randomly choose 2^{64} values of $(d_{1,0}, d_{2,0}, d_{1,1}, d_{2,1})$. For each value of $(d_{1,0}, d_{2,0}, d_{1,1}, d_{2,1})$, we can compute $(e_{0,0}, e_{1,0}, e_{2,0}, e_{0,1}, e_{1,1}, e_{2,1})$ and therefore can compute $(b_{0,0}, b_{1,0}, b_{2,0}, b_{0,1}, b_{1,1}, b_{2,1})$. Store the 2^{64} values of the tuple

$$(b_{1,0}, b_{2,0}, b_{1,1}, b_{2,1}, b_{0,0}, b_{0,1}, d_{1,0}, d_{2,0}, d_{1,1}, d_{2,1})$$

in table denoted by BT_1.

Step 3: Find a match in $(b_{1,0}, b_{2,0}, b_{1,1}, b_{2,1})$ between the table BT_0 and BT_1. Since the matching probability is 2^{-128} and there are 2^{128} pairs, we expect to find one match. After the match is found, we record the corresponding valid value of the tuple

$$(s_{1,0}, s_{2,0}, s_{1,1}, s_{2,1}, d_{1,0}, d_{2,0}, d_{1,1}, d_{2,1}, b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3}).$$

Step 4: Exhaust 2^{64} values for $(d_{1,2}, d_{2,2})$. For each value of $(d_{1,2}, d_{2,2})$, we can compute $(e_{0,2}, e_{1,2}, e_{2,2})$ and therefore can compute $(b_{0,2}, b_{1,2}, b_{2,2})$. Compare the computed value $b_{0,2}$ with the one in the recorded tuple obtained at Step 3. It is expected only 2^{32} valid values of $(d_{1,2}, d_{2,2})$ will remain. Then, for each of the 2^{32} valid $(d_{1,2}, d_{2,2})$, we can compute backward to obtain $(g_{1,2}, g_{2,2})$. According to the Property 4 of SP-box, $(g_{1,2}, g_{2,2}, h_2)$ is a valid tuple with probability 2^{-1}. Thus, we will finally to obtain 2^{31} valid values of $(d_{1,2}, d_{2,2})$ and the corresponding valid value of $(g_{1,2}, g_{2,2}, h_2)$. We again use the Property 4 of SP-box to compute the corresponding $(s_{1,2}, s_{2,2}[30 \sim 0])$ with the valid tuple $(g_{1,2}, g_{2,2}, h_2)$. When $(s_{1,2}, s_{2,2}[30 \sim 0])$ is determined, we can compute $g_{0,3}[30 \sim 0]$. Note that we can also determine $g_{0,2}$ when computing backward. In other words, we will have 2^{31} valid values of $(g_{0,2}, g_{0,3}[31 \sim 0])$, each of which will correspond to a valid value of $(d_{1,2}, d_{2,2})$. Thus, we can store the 2^{31} valid values of $(d_{1,2}, d_{2,2}, g_{0,2}, g_{0,3}[30 \sim 0])$ in a table denoted by GT_0.

Step 5: Similar to dealing with $(d_{1,2}, d_{2,2})$, we can exhaust 2^{64} values for $(d_{1,3}, d_{2,3})$. For each guess of $(d_{1,3}, d_{2,3})$, $(e_{0,3}, e_{1,3}, e_{2,3})$ is determined and we can therefore compute $(b_{0,3}, b_{1,3}, b_{2,3})$. Compare the computed value of $b_{0,3}$ with the one in the recorded tuple obtain at Step 3. It is expected only 2^{32} valid values of $(d_{1,3}, d_{2,3})$ will remain. Then, for each of the valid tuple $(d_{1,3}, d_{2,3})$, we can compute $(g_{1,3}, g_{2,3}[30 \sim 0])$. According to Property 4 of SP-box, the tuple $(g_{1,3}, g_{2,3}, h_3)$ is a valid tuple with probability 2^{-1}. Once it is valid, we can obtain the corresponding $g_{0,2}[30 \sim 0]$. Note the we can determine $g_{0,3}$ when computing backward. Thus, we can finally obtain 2^{31} valid tuples $(d_{1,3}, d_{2,3}, g_{0,2}[30 \sim 0], g_{0,3})$, which will be stored in a table denoted by GT_1.

Step 6: Use GT_0 and GT_1 to find a match in $(g_{0,2}[30 \sim 0], g_{0,3}[30 \sim 0])$. Note there are 2^{62} pairs and the matching probability is 2^{-62}. Therefore, we can expect to find a match. Once a match is found, we can know the corresponding $g_{0,2}[31]$ according to GT_0 and the corresponding $g_{0,3}[31]$ according to GT_1. Then, we can compute the corresponding $(s_{1,2}, s_{2,2}, s_{1,3}, s_{2,3})$.

Hence, a valid capacity part of A_{h0} can be found in 2^{64} time. The memory complexity at this phase is 2^{64}.

4.5.2 Matching the Capacity Part

As shown in Figure 12, $(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3})$ and $(b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$ can be randomly chosen. The goal is to match a given

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

The procedure to reach this goal is the same with that of the preimage attack on 4-round Gimli-Hash. One only need to refer to Figure 12 when reading the contents in **Matching the Capacity Part** in the preimage attack on 4-round Gimli-Hash. In brief, we first compute two tables ST_0 and ST_1 to store the following two mappings.

$$(b_{1,0}, b_{2,0}) \to (b_{1,2}, b_{2,2}, s_{0,0}, s_{0,2}),$$
$$(b_{1,1}, b_{2,1}) \to (b_{1,3}, b_{2,3}, s_{0,1}, s_{0,3}).$$

Then, we obtain two tables FT_0 and FT_1 with 2^{96} time to store the candidate values of $(d_{0,0}, d_{0,1}, d_{0,2}, d_{0,3})$. Finally, find a match in $(d_{0,0}, d_{0,1}, d_{0,2}, d_{0,3})$ between the tables FT_0 and FT_1. Consequently, the time complexity of the preimage attack on 5-round Gimli-Hash is 2^{96} while the memory complexity is 2^{64}.

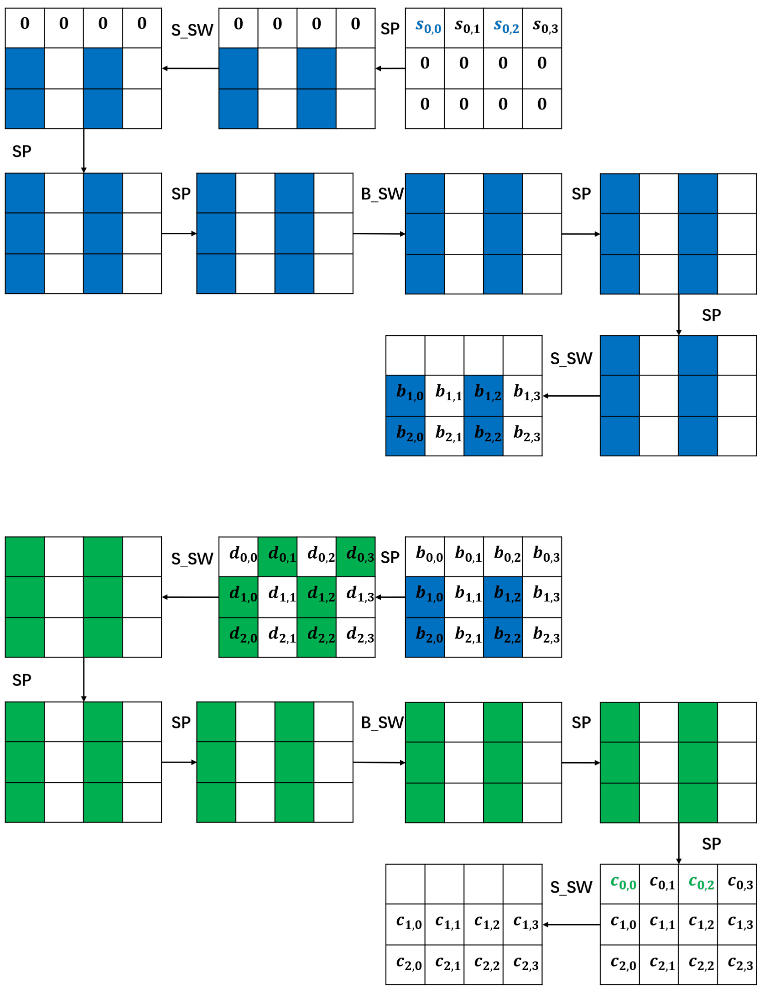

Figure 12: Preimage attack on 5-round Gimli-Hash

5 Collision Attack on Reduced Gimli-Hash

After the preimage attacks on 2/3/4/5 rounds of Gimli-Hash were presented, it is natural to ask whether it is possible to find a better collision attack than the preimage attack. This motivates us to devise the following collision attacks on 3/4/5-round Gimli-Hash. Especially, we can provide the first colliding message pair for 3-round Gimli-Hash.

Similar to the preimage attack, we will try to find a collision in the capacity part, which can then be easily converted into a valid collision for the reduced Gimli-Hash. Our collision attack procedure consists of two phases on the whole. The first phase is to find two different messages which satisfy a certain condition. The second phase is to utilize the degree of freedom of one-block message to generate a collision in the capacity part. We have to emphasize once again that in the main content, the r-round Gimli-Hash is treated as the sequence of the first r operations of the 24-round permutation, as stated in Section 2.2.

5.1 Collision Attacks on 4/5-round Gimli-Hash

As described at the beginning of this section, we will describe the two phases of the collision attack respectively.

5.1.1 The First Phase

At the first phase, we hope to find two random messages m and m'. Denote the state after m and m' are absorbed by $q = (q_{i,j})$ and $q' = (q'_{i,j})$ $(0 \leq i \leq 2, 0 \leq j \leq 3)$ respectively. Specially, we have the following conditions on q and q'.

$$(q_{1,1} \lll 9)[28 \sim 0] = (q_{1,3} \lll 9)[28 \sim 0] = 0,$$
$$(q'_{1,1} \lll 9)[28 \sim 0] = (q'_{1,3} \lll 9)[28 \sim 0] = 0.$$

Therefore, by trying $2^{29+29} = 2^{58}$ random values of m, we expect to obtain the q satisfying the condition. By trying 2^{58} random values of m', we expect to find the corresponding q' satisfying the condition. In other words, the time complexity to find a valid m and m' at this phase is $2^{59} = 2^{58} + 2^{58}$. After they are found, we move to the second phase.

5.1.2 The Second Phase

After the first phase, two states $q = (q_{i,j})$ and $q' = (q'_{i,j})$ $(0 \leq i \leq 2, 0 \leq j \leq 3)$ can be collected. Now, we explain how to use one more message block to achieve the collision attack. For a better understanding, we suggest to refer to Figure 13.

Once there is one more message block to be processed, the message will be first added to $(q_{0,0}, q_{0,1}, q_{0,2}, q_{0,3})$ and $(q'_{0,0}, q'_{0,1}, q'_{0,2}, q'_{0,3})$ respectively according to the specification of Gimli-Hash. Then, the Gimli permutation will be applied. To avoid introducing more notations and for simplicity, we treat $(q_{0,0}, q_{0,1}, q_{0,2}, q_{0,3})$ and $(q'_{0,0}, q'_{0,1}, q'_{0,2}, q'_{0,3})$ as the controllable variables by the attacker rather a constant value obtained at the first phase. Moreover, denote the state after the one more message block is absorbed by $c = (c_{i,j})$ $(0 \leq i \leq 2, 0 \leq j \leq 3)$, as shown in Figure 13. Then, the collision attack can be described as follows.

Step 1: Exhaust all 2^{64} possible values of $(q_{0,0}, q_{0,2})$. Thanks to the Property 1 of SP-box, we can compute $(c_{1,0}, c_{2,0}, c_{1,2}, c_{2,2})$ for each guessed value of $(q_{0,0}, q_{0,2})$, which is irrelevant to the value of $(q_{0,1}, q_{0,3})$. Therefore, we can store the 2^{64} values of

$$(q_{0,0}, q_{0,2}, c_{1,0}, c_{2,0}, c_{1,2}, c_{2,2})$$

in a table denoted by L_0.

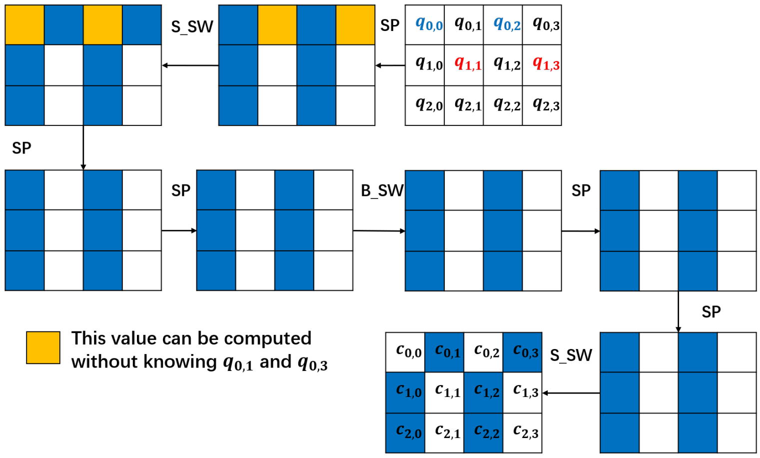

Figure 13: Collision attack on 5-round Gimli-Hash

Step 2: Exhaust all 2^{64} possible values of $(q'_{0,0}, q'_{0,2})$. Thanks to the Property 1 of SP-box, we can also compute $(c_{1,0}, c_{2,0}, c_{1,2}, c_{2,2})$ for each guessed value of $(q'_{0,0}, q'_{0,2})$, which is irrelevant to the value of $(q'_{0,1}, q'_{0,3})$. Therefore, we can store the 2^{64} values of

$$(q'_{0,0}, q'_{0,2}, c_{1,0}, c_{2,0}, c_{1,2}, c_{2,2})$$

in a table denoted by L'_0.

Step 3: Find a match in $(c_{1,0}, c_{2,0}, c_{1,2}, c_{2,2})$ between L_0 and L'_0. Since there are $2^{64+64} = 2^{128}$ pairs, we expect to find a match. After the match is found, $(q_{0,0}, q_{0,2}, q'_{0,0}, q'_{0,2})$ becomes a fixed constant.

Step 4: Exhaust all 2^{64} possible values of $(q_{0,1}, q_{0,3})$. Since $(q_{0,0}, q_{0,2})$ has been fixed, the full state of q is known for each guess of $(q_{0,1}, q_{0,3})$ and we can compute $(c_{1,1}, c_{2,1}, c_{1,3}, c_{2,3})$. Store the 2^{64} values of

$$(q_{0,1}, q_{0,3}, c_{1,1}, c_{2,1}, c_{1,3}, c_{2,3})$$

in a table denoted by L_1.

Step 5: Exhaust all 2^{64} possible values of $(q'_{0,1}, q'_{0,3})$. Since $(q'_{0,0}, q'_{0,2})$ has been fixed, the full state of q' is known for each guess of $(q'_{0,1}, q'_{0,3})$ and we can compute $(c_{1,1}, c_{2,1}, c_{1,3}, c_{2,3})$. Store the 2^{64} values of

$$(q'_{0,1}, q'_{0,3}, c_{1,1}, c_{2,1}, c_{1,3}, c_{2,3})$$

in a table denoted by L'_1.

Step 6: Find a match in $(c_{1,1}, c_{2,1}, c_{1,3}, c_{2,3})$ between L_1 and L'_1. Since there are $2^{64+64} = 2^{128}$ pairs, we expect to find a match. After the match is found, $(q_{0,1}, q_{0,3}, q'_{0,1}, q'_{0,3})$ becomes a fixed constant.

Complexity Evaluation. After the above procedure, we know that $f(q)$ and $f(q')$ will share the same capacity part. Then, we use two different one-block messages to eliminate the difference at the rate part of $f(q)$ and $f(q')$. In this way, we can obtain a full-state collision. Finally, we use another non-full one-block message to satisfy the padding rule, which will make the collision valid. Obviously, the time and memory complexity of our collision attack are 2^{65} and 2^{64} respectively.

5.1.3 Collision Attack on 4-round Gimli-Hash

The above collision attack procedure can be directly applied to the collision attack on 4-round Gimli-Hash. The first phase is the same. As for the second phase, one only need to refer to Figure 14 when reading the above attack procedure of the collision attack on 5-round Gimli-Hash. Thus, the time and memory complexity of the collision attack on 4-round Gimli-Hash are also 2^{65} and 2^{64} respectively.

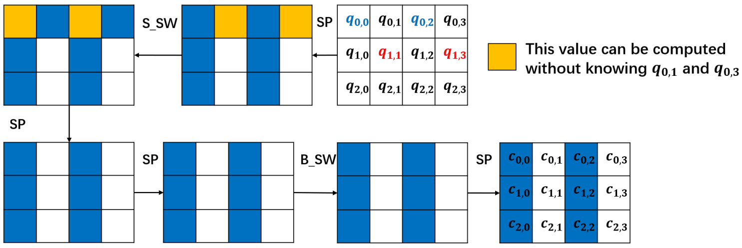

Figure 14: Collision attack on 4-round Gimli-Hash

5.2 Practical Collision Attack on 3-Round Gimli-Hash

Like the collision attack on 4/5-round Gimli-Hash, we will also explain the two phases of the collision attack on 3-round Gimli-Hash respectively.

5.2.1 The First Phase

Similar to the attack on 5-round Gimli-Hash, at this phase, we will generate two different messages which satisfy a certain condition after they are absorbed. Denote the two messages by m and m' respectively. Moreover, after m and m' are absorbed, denote their corresponding state by $q = (q_{i,j})$ and $q' = (q'_{i,j})$ $(0 \leq i \leq 2, 0 \leq j \leq 3)$ respectively. Especially, we constrain that both m and m' are a two-block message, although such a constraint is indeed not necessary. Different from the first phase for the 5-round collision attack, we only need to add the condition on one 32-bit word of q and q' as follows.

$$(q_{1,2} \lll 9)[28 \sim 0] = 0,$$
$$(q'_{1,2} \lll 9)[28 \sim 0] = 0.$$

In addition, we have the following conditions on the first two columns of q and q', i.e. they are the same.

$$q_{u,v} = q'_{u,v},$$

where $(1 \leq u \leq 2, 0 \leq v \leq 1)$.

Now, we expand on how to generate such a pair of (m, m'). For a better understanding, we refer the readers to Figure 15, especially for the notations used in the following description.

Step 1: Randomly choose a value of $(s_{0,2}, s_{0,3}, b_{0,2}, b_{0,3})$. Note that after $(s_{0,2}, s_{0,3})$ are fixed, we can always compute $(b_{1,2}, b_{2,2}, b_{1,3}, b_{2,3})$, which is irrelevant to the value of $(s_{0,0}, s_{0,1})$. Then, after fixing $(b_{0,2}, b_{0,3})$, the last two columns of the state

$$(b_{0,2}, b_{0,3}, b_{1,2}, b_{2,2}, b_{1,3}, b_{2,3})$$

are all known, thus making the computation of $(q_{1,2}, q_{2,2}, q_{1,3}, q_{2,3})$ feasible, which is irrelevant to the value of $(b_{0,0}, b_{0,1})$. Since $q_{1,2}$ has to satisfy the following condition

$$(q_{1,2} \lll 9)[28 \sim 0] = 0,$$

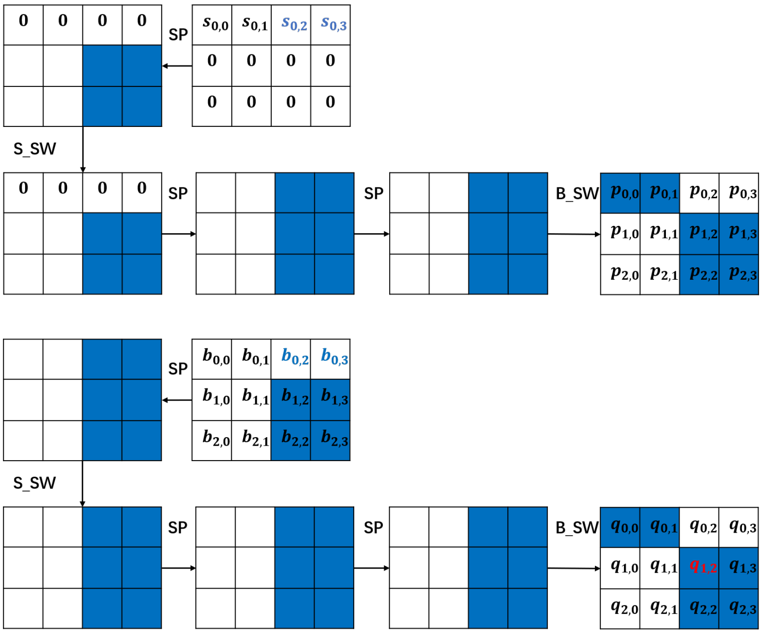

Figure 15: Generate candidates for the first two blocks for the collision attack on 3-round Gimli-Hash

we expect to obtain two values of $(s_{0,2}, s_{0,3}, b_{0,2}, b_{0,3})$ which can make this condition hold after trying $2^{29+1} = 2^{30}$ possible values. For a better understanding and simplicity, we denote the two values by

$$(s_{0,2}, s_{0,3}, b_{0,2}, b_{0,3}),$$
$$(s'_{0,2}, s'_{0,3}, b'_{0,2}, b'_{0,3}),$$

which will make

$$(q_{1,2} \lll 9)[28 \sim 0] = 0,$$
$$(q'_{1,2} \lll 9)[28 \sim 0] = 0.$$

hold respectively.

Step 2: Randomly choose a fixed value $(c_0, c_1) \in F^2_{2^{32}}$ for $(s_{0,0}, s_{0,1})$ and $(s'_{0,0}, s'_{0,1})$, i.e.

$$s_{0,0} = s'_{0,0} = c_0,$$
$$s_{0,1} = s'_{0,1} = c_1.$$

In this way, the first message block of m and m' denoted by m_0 and m'_0 are fixed as follows.

$$m_0 = (c_0, c_1, s_{0,2}, s_{0,3}),$$
$$m'_0 = (c_0, c_1, s'_{0,2}, s'_{0,3}).$$

Then, we can compute

$$p = (p_{i,j}) = f(m_0 \| 0^{256}),$$
$$p' = (p'_{i,j}) = f(m'_0 \| 0^{256}),$$

where $(0 \le i \le 2, 0 \le j \le 3)$. For such a value of (m_0, m'_0), we can know that

$$p_{u,v} = p'_{u,v},$$

where $(1 \le u \le 2, 0 \le v \le 1)$.

Step 3: Randomly choose a fixed value $(c_2, c_3) \in F_{2^{32}}^2$ for $(b_{0,0}, b_{0,1})$ and $(b'_{0,0}, b'_{0,1})$, i.e.

$$b_{0,0} = b'_{0,0} = c_2,$$
$$b_{0,1} = b'_{0,1} = c_3.$$

In this way, the second message block of m and m' denoted by m_1 and m'_1 are fixed as follows.

$$m_1 = (c_2 \oplus p_{0,0}, c_3 \oplus p_{0,1}, b_{0,2} \oplus p_{0,2}, b_{0,3} \oplus p_{0,3}),$$
$$m'_1 = (c_2 \oplus p'_{0,0}, c_3 \oplus p'_{0,1}, b'_{0,2} \oplus p'_{0,2}, b'_{0,3} \oplus p'_{0,3}).$$

With the first message block, we have ensured that

$$b_{u,v} = b'_{u,v},$$

where $(1 \le u \le 2, 0 \le v \le 1)$.

With the second message block, we can therefore ensure that

$$q_{u,v} = q'_{u,v},$$

where $(1 \le u \le 2, 0 \le v \le 1)$.

Obviously, the time complexity of the first phase is dominated by Step 1 in the above attack procedure. Therefore, the time complexity of the first phase is 2^{30}.

5.2.2 The Second Phase

After obtaining two potential messages m and m', we can further utilize the degree of freedom of one more message block m_2 (resp. m'_2) to generate a collision in the capacity part, as in the collision attack on 5-round Gimli-Hash. Now, we expand on how to use one more message block to achieve the collision attack. For a better understanding, we suggest to refer to Figure 16.

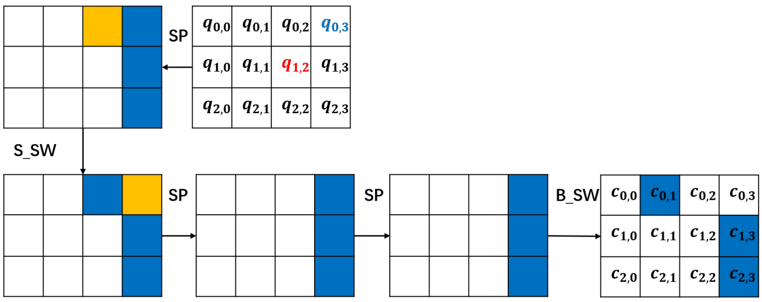

Figure 16: Collision attack on 3-round Gimli-Hash

Once there is one more message block to be processed, the message will be first added to $(q_{0,0}, q_{0,1}, q_{0,2}, q_{0,3})$ and $(q'_{0,0}, q'_{0,1}, q'_{0,2}, q'_{0,3})$ respectively. Then, the Gimli permutation will be applied. To avoid introducing more notations and for simplicity, let us treat $(q_{0,0}, q_{0,1}, q_{0,2}, q_{0,3})$ and $(q'_{0,0}, q'_{0,1}, q'_{0,2}, q'_{0,3})$ as the controllable variables by the attacker rather a constant value obtained at the first phase. Moreover, denote the state after the one more message block m_2 is absorbed by $c = (c_{i,j})$ $(0 \le i \le 2, 0 \le j \le 3)$, as shown in Figure 16.

Similar to the first phase, we can adjust two words of m_2 and m'_2 to keep

$$q_{0,0} = q'_{0,0},$$
$$q_{0,1} = q'_{0,1}.$$

Moreover, we have ensured at the first phase that

$$q_{u,v} = q'_{u,v},$$

where $(1 \leq u \leq 2, 0 \leq v \leq 1)$. In this way, we have already made the collision occur in

$$(c_{1,0}, c_{2,0}, c_{1,1}, c_{2,1}).$$

Therefore, the main target at the second phase is to find the value of $(q_{0,2}, q_{0,3})$ and $(q'_{0,2}, q'_{0,3})$ which can make the collision occur in

$$(c_{1,2}, c_{2,2}, c_{1,3}, c_{2,3}).$$

The corresponding attack procedure is described below. Once again, we refer the readers to Figure 16 for a clear understanding.

Step 1: Exhaust all 2^{32} possible values of $q_{0,3}$. Thanks to the Property 1 of SP-box, we can compute $(c_{1,3}, c_{2,3})$ for each guessed value of $q_{0,3}$, which is irrelevant to the value of $q_{0,2}$. Therefore, we can store the 2^{32} values of

$$(q_{0,3}, c_{1,3}, c_{2,3})$$

in a table denoted by LI_0.

Step 2: Exhaust all 2^{32} possible values of $q'_{0,3}$. Thanks to the Property 1 of SP-box, we can compute $(c_{1,3}, c_{2,3})$ for each guessed value of $q'_{0,3}$, which is irrelevant to the value of $q'_{0,2}$. Therefore, we can store the 2^{32} values of

$$(q'_{0,3}, c_{1,3}, c_{2,3})$$

in a table denoted by LI'_0.

Step 3: Find a match in $(c_{1,3}, c_{2,3})$ between LI_0 and LI'_0. Since there are $2^{32+32} = 2^{64}$ pairs, we expect to find a match. After the match is found, $(q_{0,3}, q'_{0,3})$ becomes a fixed constant.

Step 4: Exhaust all 2^{32} possible values of $q_{0,2}$. Since $q_{0,3}$ has been fixed at Step 3, we can compute $(c_{1,2}, c_{2,2})$. Store the 2^{32} values of

$$(q_{0,2}, c_{1,2}, c_{2,2})$$

in a table denoted by LI_1.

Step 5: Exhaust all 2^{32} possible values of $q'_{0,2}$. Since $q'_{0,3}$ has been fixed at Step 3, we can compute $(c_{1,2}, c_{2,2})$. Store the 2^{32} values of

$$(q'_{0,2}, c_{1,2}, c_{2,2})$$

in a table denoted by LI'_1.

Step 6: Find a match in $(c_{1,2}, c_{2,2})$ between LI_1 and LI'_1. Since there are $2^{32+32} = 2^{64}$ pairs, we expect to find a match. After the match is found, $(q_{0,2}, q'_{0,2})$ becomes a fixed constant.

After the above procedure, we know that $f(q)$ and $f(q')$ will share the same capacity part. Then, we use two different one-block messages (m_3, m'_3) to eliminate the difference at the rate part of $f(q)$ and $f(q')$. In this way, we can obtain a full-state collision. Finally, we use another non-full one-block message to satisfy the padding rule, which will make the collision valid. Obviously, the time and memory complexity of our collision attack are 2^{33} and 2^{32} respectively.

Experimental Verification. Due to the practical time and memory complexity, we have implemented the collision attack on 3-round Gimli-Hash. After the whole attack procedure (the first and second phase) is repeated twice, we obtained the following four-block message pair that can lead a full-state collision, as listed in Table 2. By appending another arbitrary non-full message block and considering the padding rule, we can generate an arbitrary valid collision.

Table 2: Four-block message pair for full-state collision of 3-round Gimli-Hash

m_0	0xb28d37cb	0xf45c55d6	0xde66f7c3	0x311b4daf
m_1	0xff2ecb4b	0xad17efea	0x72cd23ee	0xd9b8184
m_2	0xe6c17a12	0x4e6b8149	0x6bcf4f78	0xb2bb53c3
m_3	0x41dc5ce8	0x556eee8c	0xe2a8eec	0xc6f2b830
m'_0	0xb28d37cb	0xf45c55d6	0x6385d8fc	0x2c337f96
m'_1	0xe2d9e2fb	0xd86356a7	0xb6e4ad39	0x23205c31
m'_2	0x1ded3fee	0xc29968a4	0x3a53f26	0x8e721abb
m'_3	0xa7604db7	0x271cc14a	0xe2a8eec	0xc6f2b830
Full-state Value	0xb058f51	0x7bdae866	0x9d91e603	0x2990292f
	0x3fc4504a	0x72dcd367	0xf28ddd2f	0x68af4c32
	0x28015655	0x7c507696	0x5f998b7f	0xb8638e53

6 Conclusion

Following the generic preimage attack framework for Gimli-Hash, specific preimage attacks on 2/3/4/5 rounds of Gimli-Hash with divide-and-conquer methods are developed. The divide-and-conquer methods much rely on the properties of the SP-box and the linear layer. Moreover, to obtain better collision attacks, the divide-and-conquer methods are further extended to a practical collision attack on 3-round Gimli-Hash and theoretical collision attacks on 4/5-round Gimli-Hash. It is natural to ask whether it is possible to extend the divide-and-conquer method to attack more rounds.

References

[1] https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates.

[2] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[3] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 411–436, 2015.

[4] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 321–345, 2017.

[5] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: block ciphers for the internet of things. *IACR Cryptology ePrint Archive*, 2015:585, 2015.

[6] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153, 2016.

[7] 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.

[8] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, pages 450–466, 2007.

[9] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON family of lightweight hash functions. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 222–239, 2011.

[10] Mike Hamburg. Cryptanalysis of 22 1/2 rounds of gimli. Cryptology ePrint Archive, Report 2017/743, 2017. https://eprint.iacr.org/2017/743.

[11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 342–357, 2011.

A Preimage Attacks on Another Reduced Version of Gimli-Hash

As shown in Section 2, the sequence of operations for the 24-round permutation is as follows:

$$\begin{aligned}
& (\text{SP} \rightarrow \text{S_SW} \rightarrow \text{AC}) \rightarrow (\text{SP}) \rightarrow (\text{SP} \rightarrow \text{B_SW}) \rightarrow (\text{SP}) \\
\rightarrow \quad & (\text{SP} \rightarrow \text{S_SW} \rightarrow \text{AC}) \rightarrow (\text{SP}) \rightarrow (\text{SP} \rightarrow \text{B_SW}) \rightarrow (\text{SP}) \\
\rightarrow \quad & (\text{SP} \rightarrow \text{S_SW} \rightarrow \text{AC}) \rightarrow (\text{SP}) \rightarrow (\text{SP} \rightarrow \text{B_SW}) \rightarrow (\text{SP}) \\
\rightarrow \quad & (\text{SP} \rightarrow \text{S_SW} \rightarrow \text{AC}) \rightarrow (\text{SP}) \rightarrow (\text{SP} \rightarrow \text{B_SW}) \rightarrow (\text{SP}) \\
\rightarrow \quad & (\text{SP} \rightarrow \text{S_SW} \rightarrow \text{AC}) \rightarrow (\text{SP}) \rightarrow (\text{SP} \rightarrow \text{B_SW}) \rightarrow (\text{SP}) \\
\rightarrow \quad & (\text{SP} \rightarrow \text{S_SW} \rightarrow \text{AC}) \rightarrow (\text{SP}) \rightarrow (\text{SP} \rightarrow \text{B_SW}) \rightarrow (\text{SP}).
\end{aligned}$$

In this section, we present the preimage attacks on 2/3 rounds of Gimli-Hash by choosing the sequence of operations for 2-round permutation and 3-round permutation as follows:

$$\begin{aligned}
2-\text{round}: \quad & (\text{SP} \rightarrow \text{B_SW}) \rightarrow (\text{SP}), \\
3-\text{round}: \quad & (\text{SP}) \rightarrow (\text{SP} \rightarrow \text{B_SW}) \rightarrow (\text{SP}).
\end{aligned}$$

In other words, we choose the sequence of last 2/3 operations. Note that such a way to choose in the reduced version for 4-round permutation is the same with the main content, we therefore omit it in the appendix. In addition, we can not attack 5-round Gimli-Hash for such a way to choose in the reduced version.

Although the reduced version for 2/3-round Gimli-Hash is different from that in the main content, the procedure of the preimage attack is the same, which consists of **Computing a Valid Capacity Part** and **Matching the Capacity Part**. As for the collision attack, we however could not find a better one than the preimage attack. Thus, we only focus on the preimage attacks here.

Although the collision attack on 3-round Gimli-Hash in the main content can be trivially applied to the new reduced 2-round Gimli-Hash, it is actually difficult to find a match due to the

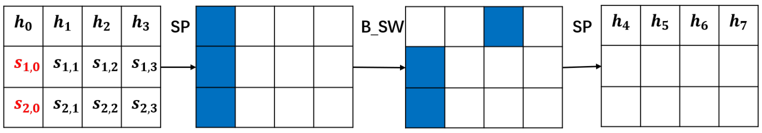

Figure 17: Generate a valid capacity part for the preimage attack on 2-round Gimli-Hash

non-randomness of the elements in two tables according to our experiments, which are generated by only exhausting just one 32-bit word. The reason why there is a match for 3-round Gimli-Hash in the main content, we believe, is that the randomness will increase as the number of rounds increases.

A.1 Preimage Attacks on 2-round Gimli-Hash

We present the details of the preimage attack on 2-round Gimli-Hash in this part. As shown in Figure 17, we denote the hash value by

$$(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7),$$

where $h_i \in F_2^{32}$. Moreover, the capacity part of A_{h0} is denoted by $s_{i,j}$ $(1 \le i \le 2, 0 \le j \le 3)$.

Computing a Valid Capacity Part. At first, we generate a valid value for the capacity part of the first output block, as illustrated in Figure 17.

The procedure can be described as follows. Please refer to Figure 17 to understand the meaning of notations.

Step 1: Randomly choose 2^{32} values of $(s_{1,0}, s_{2,0})$. Then, with the Property 2 of SP-box, we can find about $2^{32-15} = 2^{17}$ candidates for $(s_{1,0}, s_{2,0})$ which may match h_4. Store these values in a table CT_0.

Step 2: Similarly, we randomly choose 2^{32} values of $(s_{1,j}, s_{2,j})$ $(1 \le j \le 3)$ and partially match h_{j+4}. Store the candidates in table CT_j respectively.

Step 3: Exhaust all possible combinations between CT_0 and CT_2. For each combination, (h_4, h_6) can be fully computed and we compare it with the given hash value. It is expected that there is only one valid value of $(s_{1,0}, s_{2,0}, s_{1,2}, s_{2,2})$ since there are totally 2^{64} random values for it.

Step 4: Similarly, we can obtain the value of $(s_{1,1}, s_{2,1}, s_{1,3}, s_{2,3})$ to match (h_5, h_7).

The time complexity can be evaluated as $2^{32} + 2^{17+17} = 2^{34}$ times of 2-round Gimli permutation. In this way, we can find a valid capacity part of A_{h0}.

Matching the Capacity Part We expand on how to match a given capacity part by utilizing the degree of freedom of the first two blocks. To have a better understanding, it is better to refer to Figure 18 for the meaning of the notations in the following description. Specifically, $(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3})$ and $(b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$ can be randomly chosen. The goal is to match a given

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

The procedure to achieve the goal is described below.

Step 1: Exhaust all 2^{64} possible values of $(c_{0,0}, s_{0,0})$. Then, the tuple $(b_{1,0}, b_{2,0}, d_{1,0}, d_{2,0})$ can be computed for each guess of $(c_{0,0}, s_{0,0})$. According to the Property 3 of SP-box, the tuple $(b_{1,0}, b_{2,0}, d_{1,0}, d_{2,0})$ is valid with probability 2^{-32}. Thus, we expect to obtain 2^{64-32}

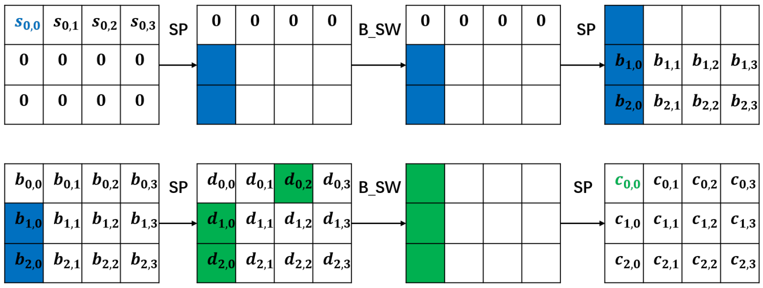

Figure 18: Preimage attack on 2-round Gimli-Hash

possible values of $(c_{0,0},\ s_{0,0})$ to match $(c_{1,0}, c_{2,0})$. For these 2^{32} valid values, we will collect 2^{32} possible values of $(d_{0,0}, d_{0,2})$. Note that according to the Property 3, $d_{0,0}$ can be computed when $(b_{1,0}, b_{2,0}, d_{1,0}, d_{2,0})$ is a valid tuple. Store all the 2^{32} valid values of the tuple $(d_{0,0}, d_{0,2}, c_{0,0}, s_{0,0})$ in the table GA_0.

Step 2: Similarly, exhaust all 2^{64} possible values of $(c_{0,1},\ s_{0,1})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,1}, d_{0,3}, c_{0,1}, s_{0,1})$ and store them in the table GA_1.

Step 3: Similarly, exhaust all 2^{64} possible values of $(c_{0,2},\ s_{0,2})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,0}, d_{0,2}, c_{0,2}, s_{0,2})$ and store them in the table GA_2.

Step 4: Similarly, exhaust all 2^{64} possible values of $(c_{0,3},\ s_{0,3})$. In this way, we can obtain 2^{32} valid values of the tuple $(d_{0,1}, d_{0,3}, c_{0,3}, s_{0,3})$ and store them in the table GA_3.

After obtaining GA_0, GA_1, GA_2 and GA_3, we can use GA_0 and GA_2 and expect to find a match in $(d_{0,0}, d_{0,2})$ since there are 2^{64} pairs in total. Similarly, we can use GA_1 and GA_3 to find a match in $(d_{0,1}, d_{0,3})$. Once the match is found, we get the solution of

$$(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$$

which will correspond to the given capacity part

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

In conclusion, the time and memory complexity of the preimage attack on 2-round Gimli-Hash are 2^{64} and 2^{32}, respectively.

A.2 Preimage Attacks on 3-round Gimli-Hash

The preimage attack on 3-round Gimli-Hash will be discussed in this part. As shown in Figure 19, we denote the hash value by

$$(h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7),$$

where $h_i \in F_2^{32}$. Moreover, the capacity part of A_{h0} is denoted by $s_{i,j}$ $(1 \le i \le 2, 0 \le j \le 3)$.

Computing a Valid Capacity Part. The main idea to compute a valid capacity part for the preimage attack on 3-round Gimli-Hash is illustrated in Figure 19. The procedure can be divided into 4 steps, as shown below. Please refer to Figure 19 for the meaning of the notations.

Step 1: Randomly choose 2^{32} values of $(s_{1,0}, s_{2,0})$. Then, with the Property 2 of SP-box, we can find about $2^{32-15} = 2^{17}$ candidates for $(s_{1,0}, s_{2,0})$ which may match h_4. Store these values in a table CT_0.

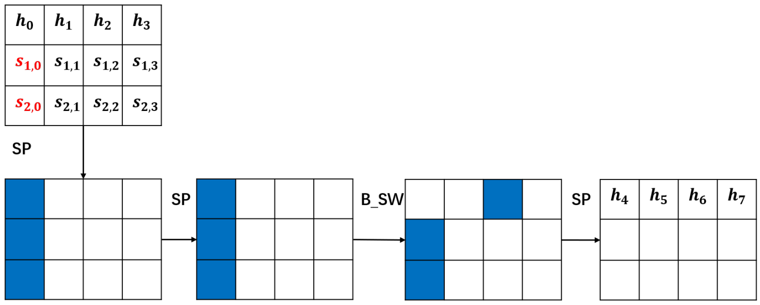

Figure 19: Generate a valid capacity part for preimage attack on 3-round Gimli-Hash

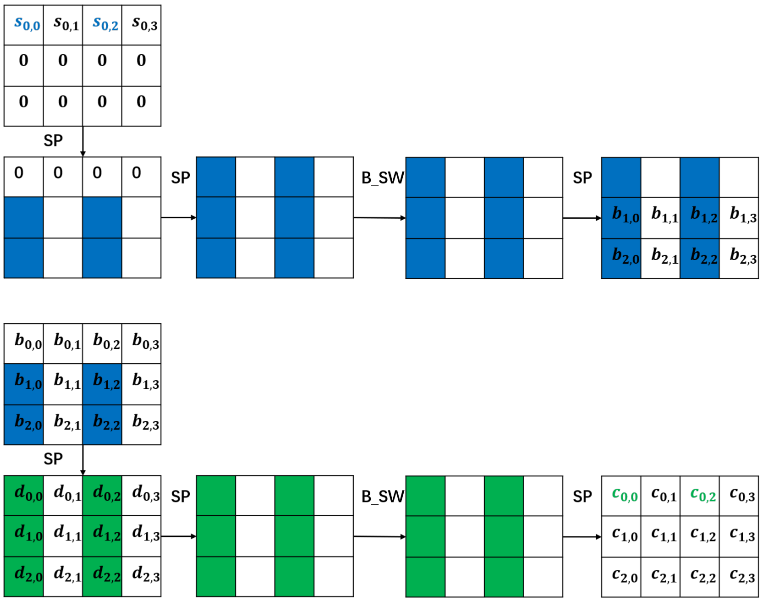

Figure 20: Preimage attack on 3-round Gimli-Hash

Step 2: Similarly, we randomly choose 2^{32} values of $(s_{1,j}, s_{2,j})$ $(1 \leq j \leq 3)$ and partially match h_{j+4}. Store the candidates in table CT_j respectively.

Step 3: Exhaust all possible combinations between CT_0 and CT_2. For each combination, (h_4, h_6) can be fully computed and we compare it with the given hash value. It is expected that there is only one valid value of $(s_{1,0}, s_{2,0}, s_{1,2}, s_{2,2})$ since there are totally 2^{64} random values for it.

Step 4: Similarly, we can obtain the value of $(s_{1,1}, s_{2,1}, s_{1,3}, s_{2,3})$ to match (h_5, h_7).

Hence, with $2^{17+17} = 2^{34}$ time, we can find a valid capacity part for the first output block.

Matching the Capacity Part. Now we describe how to match a given capacity part. It is better to refer to Figure 20 for the meaning of the notations in the following description. Specifically, $(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3})$ and $(b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$ can be randomly chosen. The goal is to match a given

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

The attack procedure can be found below.

Step 1: Exhaust all possible values of $(s_{0,0}, s_{0,2})$. Then we can collect 2^{64} values of

$$(b_{1,0}, b_{2,0}, b_{1,2}, b_{2,2}, s_{0,0}, s_{0,2}).$$

Store these values in a table MT_0 of size 2^{64}.

Step 2: Exhaust all possible values of $(c_{0,0}, c_{0,2})$. Then we collect 2^{64} values of

$$(b_{1,0}, b_{2,0}, b_{1,2}, b_{2,2}, c_{0,0}, c_{0,2}).$$

Store these values in a table MT_1 of size 2^{64}.

Step 3: Exhaust all possible values of $(s_{0,1}, s_{0,3})$. Then we can collect 2^{64} values of

$$(b_{1,1}, b_{2,1}, b_{1,3}, b_{2,3}, s_{0,1}, s_{0,3}).$$

Store these values in a table MT_2 of size 2^{64}.

Step 4: Exhaust all possible values of $(c_{0,1}, c_{0,3})$. Then we collect 2^{64} values of

$$(b_{1,1}, b_{2,1}, b_{1,3}, b_{2,3}, c_{0,1}, c_{0,3}).$$

Store these values in a table MT_3 of size 2^{64}.

Step 5: Find a match in $(b_{1,0}, b_{2,0}, b_{1,2}, b_{2,2})$ between the tables MT_0 and MT_1. Since there are 2^{128} such pairs and they match with each other with probability 2^{-128}. Therefore, it is expected to find only one match. Record the corresponding $(s_{0,0}, s_{0,2}, c_{0,0}, c_{0,2})$.

Step 6: Use MT_2 and MT_3 to find one match in $(b_{1,1}, b_{2,1}, b_{1,3}, b_{2,3})$. Record the corresponding $(s_{0,1}, s_{0,3}, c_{0,1}, c_{0,3})$.

After the above procedure, we can obtain the solution of

$$(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, c_{0,0}, c_{0,1}, c_{0,2}, c_{0,3}),$$

which can be used to compute the corresponding

$$(s_{0,0}, s_{0,1}, s_{0,2}, s_{0,3}, b_{0,0}, b_{0,1}, b_{0,2}, b_{0,3})$$

and will match the given capacity part

$$(c_{1,0}, c_{1,1}, c_{1,2}, c_{1,3}, c_{2,0}, c_{2,1}, c_{2,2}, c_{2,3}).$$

Hence, the time and memory complexity for preimage attack on 3-round Gimli-Hash are both 2^{64}.