# A Preimage Attack on Reduced Gimli-Hash

Yongseong Lee[1], Jinkeon Kang[2], Donghoon Chang[2,3], and Seokhie Hong[1]

[1] Institute of Cyber Security & Privacy(ICSP), Korea University, Seoul, Republic of Korea
yslee0804@korea.ac.kr, shhong@korea.ac.kr
[2] National Istitute of Standards and Technology(NIST), Gaithersburg, Maryland, USA
jinkeon.kang@gmail.com
[3] Department of Computer Science, Indraprastha Institute of Information Technology Delhi(IIIT-Delhi), Delhi, India
donghoon@iiitd.ac.in

**Abstract.** In CHES 2017, Bernstein et al. proposed Gimli, a 384-bit permutation with 24 rounds, which aims to provide high performance on various platforms. In 2019, the full-round(24 rounds) Gimli permutation was used as an underlying primitive for building AEAD Gimli-Cipher and hash function Gimli-Hash. They were submitted to the NIST Lightweight Cryptography Standardization process and selected as one of the second-round candidates. In ToSC 2021, Liu et al. presented a preimage attack with a divide-and-conquer method on round-reduced Gimli-Hash, which uses 5-round Gimli. In this paper, we present preimage attacks on a round-reduced variant of Gimli-Hash, in which the message absorbing phase uses 5-round Gimli and the squeezing phase uses 9-round Gimli. We call this variant as 5-9-round Gimli-Hash. Our first preimage attack on 5-9-round Gimli-Hash requires $2^{96.44}$ time complexity and $2^{97}$ memory complexity. This attack requires the memory for storing several precomputation tables in Gimli SP-box operations. In our second preimage attack, we take a time-memory trade-off approach, reducing memory requirements for precomputation tables but increasing computing time for solving SP-box equations by SAT solver. This attack requires $2^{66.17}$ memory complexity and $2^{96+\epsilon}$ time complexity, where $\epsilon$ is a time complexity for solving SP-box equations. Our experiments using CryptoMiniSat SAT solver show that the maximum time complexity for $\epsilon$ is about $2^{20.57}$ 9-round Gimli.

**Keywords:** hash function, preimage attack, Gimli, Gimli-Hash

## 1 Introduction

Gimli [3] is a 384-bit permutation proposed at CHES 2017 by Bernstein et al. The number of rounds in Gimli permutation is 24. Gimli was designed for high performance in a broad range of platforms from 8-bit AVR microcontrollers to 64-bit Intel/AMD server CPUs. In 2019, AEAD Gimli-Cipher and hash function Gimli-Hash were developed using Gimli with 24 rounds as an underlying

primitive and submitted to the NIST Lightweight Cryptography Standardization process and selected as one of the second-round candidates [2]. Gimli-Hash produces 256-bit fixed-length output but can be used as an extendable one-way function (XOF). Gimli-Cipher is based on a duplex mode of operation, and Gimli-Hash uses the sponge mode with a rate of 128 bits and a capacity of 256 bits.

Since the Gimli was proposed in 2017, it has been investigated by a number of analyses. Hamburg [6] analyzed 22.5-round Gimli with meet-in-the-middle attack and pointed out the weak diffusion of Gimli. However, this attack is not applicable to AEAD or hashing scheme. In ToSC 2021, Liu et al. [9] performed the full-round (24 rounds) distinguishing attack with $2^{52}$ time complexity using zero-internal-differential method. In Asiacrypt 2020, Flórez-Gutiérrez et al. [5] performed 24-round and 28-round distinguishing attacks against round-shifted Gimli with time complexities of $2^{32}$ and $2^{64}$, respectively. Although many rounds of round-shifted Gimli were analyzed by the distinguishing attacks, there is no relation to the security of Gimli-Cipher or Gimli-Hash.

Zong et al. [14] proposed the collision attack on 6-round Gimli-Hash using differential characteristics with $2^{113}$ time complexity. In Crypto 2020, Liu et al. [8] performed the collision attack on 6-round Gimli-Hash with a time complexity of $2^{64}$. They also presented the semi-free-start collision with a time complexity of $2^{64}$ for a round-shifted 8-round Gimli-Hash. In Asiacrypt 2020, Flórez-Gutiérrez et al. [5] performed the collision and semi-free-start collision attacks on round-shifted Gimli-Hash, reaching up to 12 and 18 rounds, respectively. Both require $2^{96+\epsilon}$ time complexity, where $\epsilon$ is about $2^{10}$ Gimli operations, experimentally obtained from the SAT solver.

As we can see in analyzing collision attacks on Gimli-Hash by Flórez-Gutiérrez et al.[5], SAT solver is actively used for cryptanalysis. The SAT solver can determine whether a given boolean satisfiability problem (SAT problem) is satisfiable or unsatisfiable. Some cryptographic problems, such as finding differential trails or linear trails with minimal probability, are difficult to solve. To solve the complex cryptographic problems, these problems are transformed into SAT problems. For example, [11] made use of SAT solver to find differential trails on ARX cipher Salsa20. Using this method, differential cryptanalysis of the block cipher SIMON was performed [7]. Moreover, [10] was applied SAT solver to find linear trails on SPECK and Chaskey. Additionally, [13] analyzed integral cryptanalysis for ARX ciphers using SAT-solver of division property.

In ToSC 2021, Liu et al. [9] proposed preimage attacks on Gimli-Hash and Gimli-XOF-128 with divide-and-conquer method. Due to the weak diffusion of Gimli, the preimage attack on reduced Gimli-Hash can reach up to 5 rounds. It utilizes five message blocks that produce a specific 256-bit hash value of Gimli-Hash.

Because of the weak diffusion of Gimli, the divide-and-conquer method of dividing manipulable space into smaller spaces was effective on 5-round Gimli-Hash. However, as the number of rounds increases, the effect of diffusion increases. Hence, the preimage attacks on Gimli using the divide-and-conquer

method do not apply more than 5-round. Instead, a preimage attack on 9-round Gimli-XOF-128 was possible. Since the output size of Gimli-XOF-128 is 128bit, the preimage attack on Gimli-XOF-128 has fewer constraints than Gimli-Hash which has the 256-bit output size. Moreover, because some results of SP function could be known under certain conditions, the effect of the first Small-swap operation disappeared. Therefore, the preimage attack on 9-round Gimli-XOF performed with $2^{104}$ time complexity and $2^{70}$ memory complexity.

**Our Contributions.** In this paper, we present preimage attacks on 5-9-round Gimli-Hash. Gimli-Hash uses 24-round Gimli, and Liu et al. proposed preimages attacks on 5-round Gimli-Hash with the round-reduced Gimli of 5 rounds (out of 24). As shown in Figure 5, the 5-9-round Gimli-Hash uses 5-round Gimli in the message absorbing phase and 9-round Gimli in the squeezing phase under the sponge construction. Gimli-Hash has absorbed all message blocks into the state with 5-round Gimli. In the squeezing phase of Gimli-Hash, the first 128 bits of the state are returned as output blocks, interleaved with applications of 9-round Gimli.

Given a 256-bit hash value of Gimli-Hash, half of the hash value can make the other half with some appropriate 256-bit value and Gimli. In the preimage attack on 5-round Gimli-Hash[9], the divide-and-conquer method made the halves of the hash values overlap by weak diffusion of Gimli. However, when the number of rounds increases, additional connections are needed to find the relation in the hash value, so the idea of Liu et al. no longer works. In this paper, we propose a new collision-finding trail to overcome the limitation of Liu et al.'s approach, and show that the preimage attack on 5-9-round Gimli-Hash is possible with $2^{96.44}$ time complexity and $2^{97}$ memory complexity by using precomputation tables.

In addition, to reduce memory complexity, we propose equations that can be used instead of the precomputation tables. During the preimage attack, we find solutions on the fly by solving the equations instead of the precomputation tables. Using this method, the time complexity increases to $2^{96+\epsilon}$, but the memory complexity decreases to $2^{66.17}$, where $\epsilon$ is the time complexity of solving equations. To solve equations, we use CryptoMiniSat[12], one of the SAT solvers. Because our equations can be transformed to conjunctive normal form(CNF), we introduce how to transform given equations to CNF. Hence, the problem of finding solutions to the given our equations change to a SAT problem. Moreover, we conduct an experiment to measure the average time required to solve equations[4]. As the result of the experiments, the average time to find the solution of the most complex equation was $2^{19.15}$ times that of the full-round Gimli($\approx 2^{20.57}$ 9-round Gimli). Since the $\epsilon \approx 20.57$, the time complexity of solving equation method does not exceed $2^{128}$. A summary of our results can be seen in Table 1.

**Organization** This paper is organized as follows. In section 2, we describe Gimli and Gimli-Hash with notations. In section 3, we describe the general approach of preimage attack on Gimli-Hash introduced in [9]. Our improved preimage attack on Gimli-Hash is in section 4. The method using precompu-

[4] The test code is on `https://github.com/yslee0804/Gimli_hash_9r_test`.

**Table 1.** Summary of attacks on Gimli

| Attack | Round | Technique | Time[(a)] | Memory[(a),(b)] | Reference |
|---|---|---|---|---|---|
| Preimage on Gimli-Hash (256-bit output) | 2 | Divide-and-Conquer | 42.4 | 32 | [9] |
| | 5 | | 93.68 | 66 | [9] |
| | 5-9[(c)] | | 96.44 | 97 | Sec. 4.2 |
| | | | 96+$\epsilon_1$[(e)] | 66.17 | Sec. 4.3 |
| Preimage on Gimli-XOF-128 (128-bit output) | 9 | Divide-and-Conquer | 104 | 70 | [9] |
| Distinguisher on Gimli | 24 | Zero-Internal-Differential | 52 | negligible | [9] |
| | 24[(d)] | Symmetry | 32 | negligible | [5] |
| | 28[(d)] | | 64 | negligible | [5] |
| Collision on Gimli-Hash (256-bit output) | 6 | Divide-and-Conquer | 64 | 64 | [8] |
| | 12[(d)] | Symmetry | 96+$\epsilon_2$[(e)] | negligible | [5] |
| | 14[(d)] | Quantum | 64+$\epsilon_2$[(e)] | negligible | [5] |
| Semi-free start Collision on Gimli-Hash (256-bit output) | 8[(d)] | Symmetry | 64 | negligible | [8] |
| | 18[(d)] | | 96+$\epsilon_2$[(e)] | 64 | [5] |
| | 20[(d)] | Quantum | 64+$\epsilon_2$[(e)] | 64 | [5] |

(*a*) The time and memory complexity is expressed in log scale.
(*b*) The memory complexity measured base on 128-bit blocks.
(*c*) The last two permutations of Gimli-Hash use 9-round Gimli. The remaining permutations of Gimli-Hash use 5-round Gimli.
(*d*) This attack is on the round shifted Gimli : starting from third round.
(*e*) $\epsilon_1$ and $\epsilon_2$ are the experimentally measured complexity($\epsilon_1 \approx 20.57$ and $\epsilon_2 \approx 10$). $\epsilon_1$ is in Section 4.3 and $\epsilon_2$ is in [5].

tation tables is in section 4.2 and using equations with SAT-solver is in section 4.3. The conclusion of this paper is in section 5. Additionally, Liu et al.'s preimage attack on 5-round Gimli-hash, which motivated this paper, is included in appendix A.

## 2 Preliminaries

In this section, we describe the Gimli and Gimli-Hash. We describe the structure and notation of Gimli in section 2.1, and Gimli-Hash in section 2.2.

### 2.1 Description of Gimli

The permutation Gimli consists of non-linear layer SP-box, linear layer Small-swap(S_SW) and Big-swap(B_SW), and constant addition. SP-box operation

is performed preferentially every round, whereas other operations are applied a particular round of Gimli. For $0 \leq i \leq 6$, Small-swap and constant addition occurs every $(4i+1)$-th round, and Big-swap operation occur every $(4i+3)$-th round. In other words, the Gimli repeats the following 4 round process six times.

$$\underbrace{(SP \rightarrow S\_SW \rightarrow AC)}_{(4i+1)\text{-th round}} \rightarrow \underbrace{(SP)}_{(4i+2)\text{-th round}} \rightarrow \underbrace{(SP \rightarrow B\_SW)}_{(4i+3)\text{-th round}} \rightarrow \underbrace{(SP)}_{(4i+4)\text{-th round}}$$

The whole process of Gimli can be seen in Algorithm 1.

**Gimli State** The Gimli state $S$ can be seen as a two-dimensional array with 3 rows and 4 columns, where each element is a 32-bit word as shown in Figure 1. The notation $S[i][j]$ refers the 32-bit word in $i$-th row and $j$-th column where $0 \leq i \leq 2$ and $0 \leq j \leq 3$. In addition, the representation $S[i,k][j,l]$ means 4 words $S[i][j]$, $S[i][l]$, $S[k][j]$, and $S[k][l]$. For example, $S[1,2][0,2]$ indicate the words $S[1][0]$, $S[1][2]$, $S[2][0]$, and $S[2][2]$. Additionally, the representation $S[*][j]$ means the whole word in $j$-th column and $S[i][*]$ means the whole word in $i$-th row. In other words, $S[*][j]$ means $S[0,1,2][j]$ and $S[i][*]$ means $S[i][0,1,2,3]$.

To distinguish the state after each round, the notation $S^i$ indicates the state after $i$-th round. In particular, $S^0$ indicates the state before the first round. Moreover, the notation $S^{i.5}$ means the state after the SP operation and before Small-swap or Big-swap. For example, $S^{0.5}$ indicates the state in the first round after $SP$ operation but before Small-swap and constant addition.

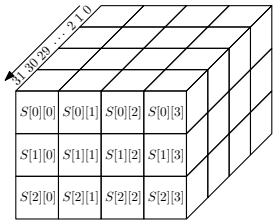


**Fig. 1.** The state of Gimli

**Non-linear Layer: SP-Box** The non-linear layer is the SP-Box on 96 bits, which is applied column-wise. The SP-Box is often treated as a function such that $SP : \mathbb{F}_2^{32\times 3} \rightarrow \mathbb{F}_2^{32\times 3}$. On three 32-bit input $x$, $y$, and $z$, $SP$ updates them as follows:

$$
\left.\begin{array}{l} x \leftarrow x \lll 24 \\ y \leftarrow y \lll 9 \end{array}\right\} \text{Compute in parallel}
$$

$$
\left.\begin{array}{l} x \leftarrow x \oplus (z \ll 1) \oplus ((y \wedge z) \ll 2) \\ y \leftarrow y \oplus x \oplus ((x \vee z) \ll 1) \\ z \leftarrow z \oplus y \oplus ((x \wedge y) \ll 3) \end{array}\right\} \text{Compute in parallel}
$$

$$
\left.\begin{array}{l} x \leftarrow z \\ z \leftarrow x \end{array}\right\} \text{Compute in parallel}
$$

**Linear Layers: Small-swap, Big-swap** The Gimli consists of two linear operations; Small-swap(S_SW) and Big-swap (B_SW). These operations do not occur every round. S_SW is executed on the $(4i+1)$ round, and B_SW is executed on the $(4i+3)$ round, where $0 \leq i \leq 5$.

S_SW and B_SW affect only the first row of Gimli state. S_SW operation swap the values $S[0][0]$ and $S[0][1]$, and swap the values $S[0][2]$ and $S[0][3]$. Similarly, B_SW operation swap the values $S[0][0]$ and $S[0][2]$, and swap the values $S[0][1]$ and $S[0][3]$. Figure 2 illustrates the S_SW and B_SW operations.

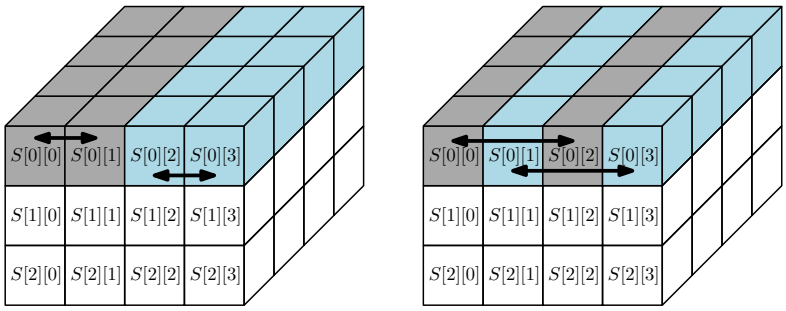


**Fig. 2.** Gimli Small-swap(a) and Big-swap(b) operations

**Constant Addition: *AC*** In constant addition, 32-bit round constant $rc_i = 0x9e377900 \oplus (24 - 4i)$ is xored to the state $S[0][0]$ in every fourth round where $0 \leq i \leq 5$.

### 2.2 Gimli-Hash

Gimli-Hash was built using the sponge construction framework. Sponge construction framework is a way to make a function with arbitrary input/output length from a permutation with fixed input/output length [4].

**Algorithm 1** The permutation Gimli

**Input:** A 384-bit state $S$. $S[i][j]$ is 32-bit word, where $0 \leq i \leq 2$ and $0 \leq j \leq 3$
**Output:** **Gimli**$(S)$

1: **for** $r = 0$ to 23 **do**
2: **for** $j = 0$ to 3 **do**
3: $S[*][j] \leftarrow SP(S[*][j])$
4: **end for**
5: **if** $r$ mod $4 = 0$ **then**
6: Swap $S[0][0]$ and $S[0][1]$, swap $S[0][2]$ and $S[0][3]$
7: $S[0][0] \leftarrow S[0][0] \oplus (0\text{x}9e377900 \oplus (24 - r))$
8: **else if** $r$ mod $4 = 2$ **then**
9: Swap $S[0][0]$ and $S[0][2]$, swap $S[0][1]$ and $S[0][3]$
10: **end if**
11: **end for**
12: **return** $S$

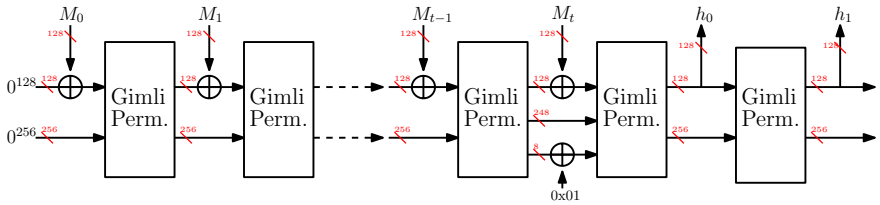


**Fig. 3.** The illustration of Gimli-Hash

Algorithm 2 describes the Gimli-Hash process. The Gimli-Hash initializes the Gimli state to 0. Then the input message of arbitrary length is read by dividing it into 16-byte blocks. Before reading the message, XOR 1 next to the last byte of input message. Then the input message is read by dividing it into 16-byte blocks. The remaining bytes in the last block are set to 0. This message modifying process is **pad**$(M)$ in Algorithm 2.

The 16-byte blocks are sequentially XOR to the 0-th row of Gimli state (which is $S[0][*]$) and perform Gimli. When the last block is XORed to the first row of the state, 1 is also XORed to the last byte of the state which the byte position is 47(line 5 in Algorithm 2). After the last message block is absorbed, perform the Gimli and squeeze out the first 16-byte of the state(which is $h_0 = S[0][*]$). This value is the first 16-byte of the hash value. To get the remaining 16-byte of the hash value, perform the Gimli again and then squeeze out the first 16-byte of the state(which is $h_1 = S[0][*]$). The final hash value is $h = (h_0, h_1)$(line 10-13 in Algorithm 2).

In this paper, we consider the separated state in the Gimli-Hash. That means the state after $M_0$ absorbed and the state after $M_1$ absorbed should be distinguishable. To distinguish these states, $S_i$ indicates the state after $M_i$ is absorbed. For example, if the message has $t + 1$ length$((M_0, M_1, \cdots, M_t))$, $S_2$ indicates the state **Gimli**(**Gimli**$(0^{384} \oplus (M_0 \| 0^{256})) \oplus (M_1 \| 0^{256})) \oplus (M_2 \| 0^{256})$

where the notation $\|$ is concatenation. Moreover, $S_{h_0}$ and $S_{h_1}$ indicate the states where $h_0$ and $h_1$ are squeezed, respectively. Note that $\mathbf{Gimli}(S_{h_0}) = S_{h_1}$.

**Algorithm 2** The GIMLI-HASH function

**Input:** $M \in \{0,1\}^*$
**Output:** 256-bit hash value $h = (h_0, h_1)$

The state S set to 0
$(M_0, M_1, \cdots, M_t) \leftarrow \mathbf{pad}(M)$
**for** $i = 0$ to $t$ **do**
    **if** $i = t$ **then**
        $S[2][3] \leftarrow S[2][3] \oplus 0\text{x}01000000$
    **end if**
    $S[0][*] \leftarrow S[0][*] \oplus M_i$
    $S \leftarrow \mathbf{Gimli}(S)$
**end for**
$h_0 \leftarrow S[0][*]$
$S \leftarrow \mathbf{Gimli}(S)$
$h_1 \leftarrow S[0][*]$
**return** $h = (h_0, h_1)$

## 3 The General Approach of Preimage Attack

In this section, we summarized the general approach of preimage attack on GIMLI-HASH, which Liu et al. performed in [9]. The overall preimage attack uses the meet-in-the-middle method and consists of four phases. It is also assumed to recover five message blocks $(M_0, M_1, M_2, M_3, M_4)$ for a given 256-bit hash value as shown in Figure 4. A brief description of each phase is as follows.

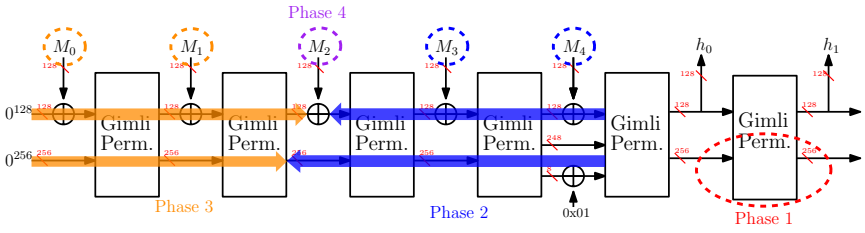


**Fig. 4.** The framework of preimage attack of GIMLI-HASH

**Phase 1** From given hash values, find a 256-bit valid capacity part such that $\mathbf{Gimli}(S_{h_0}) = S_{h_1}$. The 256-bit hash value can be seen as $(h_0, h_1)$ which $h_0$ and $h_1$ are 128-bit length, respectively. Moreover, $h_0$ and $h_1$ are rate part

of Gimli. Therefore, if the valid capacity part of $S_{h_1}$ or $S_{h_0}$ is known, the whole state data is revealed and can be computed in the backward direction of Gimli-Hash.

**Phase 2** Choose $M_3$ and $M_4$, and compute backward direction. Since $S_{h_0}$ was known, $S_4$ can be computed. Moreover, $S_3$ and $S_2$ can be known because $M_4$ and $M_3$ are chosen.

**Phase 3** Choose $M_0$ and $M_1$ such that the capacity part of $S_2$ collides. The attacker can only control the rate part of $S_0$, $S_1$, and $S_2$ by $M_0$, $M_1$, and $M_2$, respectively. Moreover, the capacity parts of $S_2$ and $S_0$ are fixed. The strategy of this phase is making collisions in the capacity part of $S_1$ by the rate parts of $S_0$ and $S_2$. If the capacity part of $S_1$ is collide, $M_1$ can be obtain from $M_1 = \mathbf{Gimli}(S_0)[0][*] \oplus S_1[0][*]$. Since the size of the controllable variable is 256-bit, one value is matched in the capacity part of $S_1$ as expected.

**Phase 4** From the found values in phase2 and phase3, $M_2$ can be obtain from $M_2 = \mathbf{Gimli}(S_1)[0][*] \oplus S_2[0][*]$.

**Phase 1** and **Phase 3** have most of the complexity for this preimage attack, while **Phase 2** and **Phase 4** are simple.

Liu et al. performed a preimage attack on 5-round Gimli-Hash based on a general approach with $2^{93.68}$ time complexity and $2^{66}$ memory complexity [9]. The 5-round attack by Liu et al. can be seen in appendix A.

## 4 Preimage Attack on 5-9-Round Gimli-Hash

The Liu et al.'s preimage attack on 5-round Gimli-Hash has a big difference in complexity between **phase 1** and **phase 3**. The time complexity of **phase 1** is $2^{64}$. On the other side, the time complexity of **phase 3** is $2^{93.68}$. Therefore, **phase 1** of the preimage attack on Gimli-Hash seems to be able to analyze more rounds. However, extending the round in **phase 1** is not easy in a direct way. This is because when the round is extended, it does not overlap in the intermediate state. In appendix A, the *step 2* of **phase 1** checks whether the calculated values collide with the $T_8$ created in *step 1*. Hence, it cannot be extended by approaching [9] method. On the other hand, we extend the rounds by making precomputation tables and solving equations.

We describe the 5-9-round Gimli-Hash in 4.1. Then, we will show finding a valid inner part(**phase 1**) by making precomputation tables in 4.2. Moreover, 4.3 describes how to reduce memory by solving equations with SAT-solver instead of using precomputation tables.

### 4.1 Description 5-9-round Gimli-Hash

The 5-9-round Gimli-Hash is a reduced Gimli-Hash that uses two types of permutations. One of the permutations consists of a 5-round Gimli, and the other consists of a 9-round Gimli. The 5-round Gimli is used in the message

absorbing stage. On the other hand, the 9-round GIMLI is used in the hash squeezing stage after the message absorbing stage. Figure 5 shows the 5-9-round GIMLI-HASH.

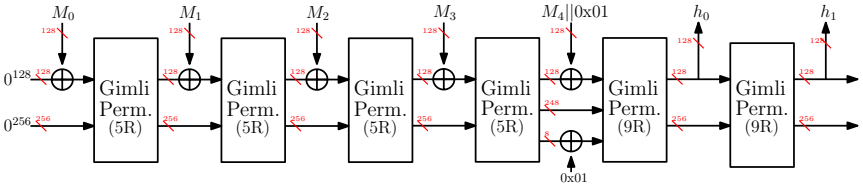


**Fig. 5.** The illustration 5-9-Round GIMLI-HASH with 5 message blocks

The strategy of using two kinds of permutations within an algorithm is not unique. This strategy can take advantage of message processing speed, or it can provide additional security when transitioning from the absorbing phase to the squeezing phase. For example, the hash scheme of SPARKLE[1], one of the NIST lightweight cryptography competition finalists, has two kinds of permutations. One is the 11-round permutation located after message absorbing, and the other is 7-round permutation consisting of the remaining permutation.

### 4.2 Finding Valid Inner Part on 9-round GIMLI(Phase 1)

To find valid capacity part, we make precomputation table $T_1$,$T_2$,$T_3$,$T_4$, and $T_5$ preferentially.

**Precomputation** Given a 256-bit hash output $h = (h_0 \| h_1)$, let $h_0 = (h_{0,0} \| h_{0,1} \| h_{0,2} \| h_{0,3})$ and $h_1 = (h_{1,0} \| h_{1,1} \| h_{1,2} \| h_{1,3})$ where $h_0$ and $h_1$ are 128-bit values and $h_{i,j} (0 \leq i \leq 1, 0 \leq j \leq 3)$ is a 32-bit value. For all $(a, b, x)$, we compute SP-box operation as follows.

$$SP(h_{0,3}, a, b) = (c, d, e)$$
$$SP(SP(x, d, e)) = (f, g, h)$$

In the calculation, store $(a, b)$ in table $T_1[(c \| x \| f)]$. It requires $2^{96}$ 64-bit memory complexity. For all $(a, b)$, compute $(c, d, e) = SP(h_{0,2}, a, b)$ and store $(a, b, c)$ in table $T_2[(d \| e)]$. Consider the $h_{1,0}$, for all $(a, b)$, compute $(c, d, e) = SP^{-1}(SP^{-1}(h_{1,0}, a, b))$ and store $(a, b, c)$ in table $T_3[(d \| e)]$. The tables $T_2$ and $T_3$ require $2^{64}$ 64-bit memory complexity, respectively. For all $(a, b, c)$, compute $(d, e, f) = SP(SP(a, b, c))$. Then $(a, e, f)$ are stored in $T_4[(b \| c \| d)]$ and $(b, c, d)$ are stored in $T_5[(a \| e \| f)]$. Each table has $2^{96}$ 96-bit memory complexity. The Algorithm 3 shows a description of generating precomputation tables.

**Algorithm 3** Generating precomputation tables for finding the valid inner part

**Input:** Hash values $h_0$ and $h_1$
**Output:** Precomputation table $T_1$, $T_2$, $T_3$, $T_4$, and $T_5$

```
for all a and b do
    (c, d, e) ← SP(h_{0,3}, a, b)
    for all x do
        (f, g, h) ← SP(SP(x, d, e))
        T_1[(c‖x‖f)] ← (a, b)
    end for
    (c, d, e) ← SP(h_{0,2}, a, b)
    T_2[(d‖e)] ← (a, b, c)
    (c, d, e) ← SP^{-1}(SP^{-1}(h_{1,0}, a, b))
    T_3[(d‖e)] ← (a, b, c)
    for all c do
        (d, e, f) ← SP(SP(a, b, c))
        T_4[(b‖c‖d)] ← (a, e, f)
        T_5[(a‖e‖f)] ← (b, c, d)
    end for
end for
```

**Finding the valid inner part** The 9-round preimage attack starts from the guess 128-bit capacity part of $S^0$ and $S^9$. It is a meet-in-the-middle attack, but it does not overlap in the Gimli state. The precomputation table is helpful to fill the gap in the middle. After making the connection, we guess the left of the inner part and find the valid values.

*Step 1* (line 1-8 in Algorithm 4): Choose random values for $S^0[1,2][0,1]$ and compute $(S^3[1,2][0,1],\ S^3[0][2,3])$. Store $(S^0[1,2][0,1],\ S^3[1,2][0,1],\ S^3[0][2,3])$ in the table $T_6$ and repeat $2^{64}$. Then choose random values for $(S^9[1,2][0,2])$ and compute $(S^{4.5}[1,2][0,2],\ S^{4.5}[0][1,3])$. Store $(S^9[1,2][0,2],\ S^{4.5}[1,2][2],\ S^{4.5}[0][1,3])$ in the table $T_7[(S^{4.5}[1][0])\|\ (S^{4.5}[2][0])]$ and repeat $2^{64}$.

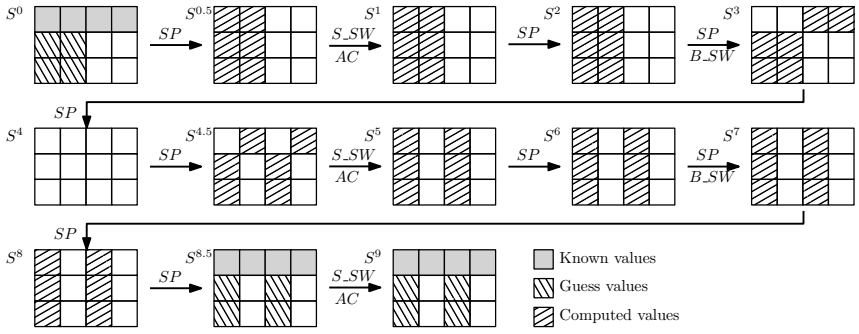


**Fig. 6.** Finding the valid inner part of 9-round Gimli (*step 1*)

*Step 2* (line 9-15 in Algorithm 4): Given the values ($S^0[1,2][0,1]$, $S^3[1,2][0,1]$, $S^3[0][2,3]$) from $T_6$, choose $2^{32}$ values for $S^3[0][0]$ and find the values ($S^9[1,2][0,2]$, $S^{4.5}[1,2][2]$, $S^{4.5}[0][1,3]$) in $T_7[(S^{4.5}[1][0])\|(S^{4.5}[2][0])]$. For each matched value, determine the value of ($S^3[0][1]$, $S^{4.5}[1,2][1]$) in $T_4[(S^3[1][1])\|(S^3[1][1])\|(S^{4.5}[0][1])]$. Similarly, determine the value of ($S^3[1,2][2]$, $S^{4.5}[0][2]$) in $T_5[(S^3[0][2])\|(S^{4.5}[1][2])\|(S^{4.5}[2][2])]$.

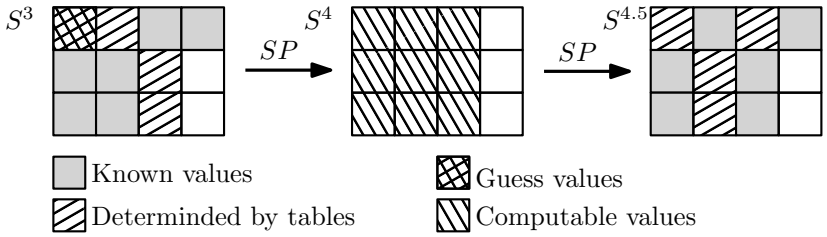


**Fig. 7.** Finding the valid inner part of 9-round Gimli (*step 2*)

*Step 3* (line 16-18 in Algorithm 4): From the found values, compute the forward and backward direction as possible rounds. Then $S^{0.5}[1,2][2]$, $S^{0.5}[0][3]$, $S^7[1,2][1]$, and $S^7[0][3]$ can be computed from the state $S^3$ and $S^{4.5}$ respectively. Determine the $S^0[1,2][2]$, and $S^{0.5}[0][2]$ in the table $T_2[(S^{0.5}[1][2])\|(S^{0.5}[2][2])]$. Moreover, determine the $S^7[0][1]$, and $S^{8.5}[1,2][1]$ in the table $T_3[(S^7[1][1])\|(S^7[2][1])]$.

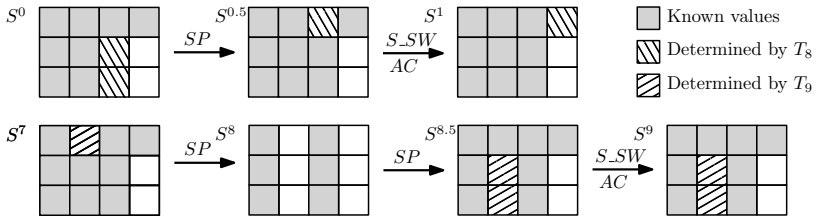


**Fig. 8.** Finding the valid inner part of 9-round Gimli (*step 3*)

*Step 4* (line 19-23 in Algorithm 4): Let the values $c = S^{0.5}[0][3]$, $x = S^1[0][3]$, and $f = S^3[0][1]$ Then determine the value $S^0[1,2][3]$ using the table $T_1[(c\|x\|f)]$. To verify the capacity part, compute from $S^0$ to $S^9$ and checking the values $S^{4.5}[0][3]$, $S^7[0][3]$, and $S^{8.5}[0][3]$ are matched. If it is matched, return the $S^9[1,2][*]$. If not, go to *step 2* and try the other values in $T_6$.

**Complexity** In the precomputation, five precomputation tables were created. Table $T_1$ can be made with all $a, b$, and $x$. Therefore, $T_1$ has $2^{96}$ 64-bit memory complexity and $3\times2^{96}$ SP operations. Similar way, the memory complexity of

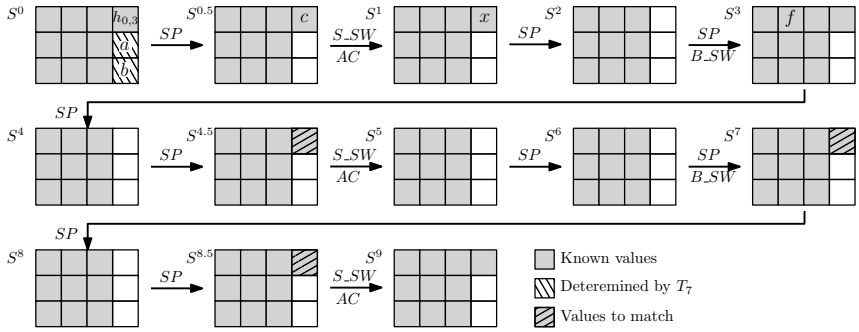


**Fig. 9.** Finding the valid inner part of 9-round Gimli (*step 4*)

$T_2, T_3, T_4$, and $T_5$ is $2^{64}$ 64-bit, $2^{64}$ 64-bit, $2^{96}$ 96-bit, and $2^{96}$ 96-bit, respectively. To create the tables, $T_2$ and $T_3$ needs $2^{64}$ and $2 \times 2^{64}$ SP operation, respectively. When creating tables $T_4$ and $T_5$, the equation $(d, e, f) = SP(SP(a, b, c))$ can fill the tables $T_4$ and $T_5$ with one operation. Therefore, making $T_4$ and $T_5$ requires only $2 \times 2^{96}$ SP operations in total. Overall, precomputation has $2^{97}$ 128-bit memory complexity and ($5 \times 2^{96}$ SP operation) $\approx$ ($2^{93.152}$ 9-round Gimli) time complexity.

In the Step 1, $T_6$ can be made by $2^{64}$ random variables for $S^0[1, 2][0, 1]$, and each variable performs 6 SP operations. $T_7$ is similar except that each random variable performs 8 SP operations. Therefore, the time complexity is $14 \times 2^{64}$ SP operations in Step 1. In terms of memory complexity, $T_6$ has $2^{64}$ 320-bit memory complexity and $T_7$ has $2^{64}$ 256-bit memory complexity. At Step 2, each value in $T_6$ need $2^{32}$ random variables, 2 SP operations, 2 table lookup. If one table lookup is considered as one SP operation, the time complexity of each value in $T_6$ is $2^{32} \times 4$ SP operations. At Step 3, it is 6 SP operations that fill some unknown values from the found values in Step 2. Then 2 table lookup need to find ($S^0[1, 2][2]$,$S^{0.5}[0][2]$) and ($S^7[0][1]$,$S^{8.5}[1, 2][1]$). Step 4 only has 1 table lookup and 36 SP operations. Therefore, the time complexity of the main computation is $2^{64} \times 14 + 2^{64} \times 2^{32} \times (4 + 8 + 37) \approx 2^{96.44}$ 9-round Gimli. The memory complexity of the main computation is ($2^{64} \times$ 576-bit) $\approx$ ($2^{66.17}$ 128-bit).

Overall, the time complexity is $2^{96.44}$ 9-round Gimli, and the memory complexity is $2^{97}$ 128-bit block.

### 4.3 Reducing precomputation Memory using SAT Solver

To find the valid capacity part of 9-round Gimli, Algorithm 4 uses five sorted precomputation tables, $T_1, T_2, T_3, T_4$, and $T_5$. These tables are used to find unknown values from the given values. If we find unknown values another way, we should reduce the memory complexity for precomputation tables. The solutions of the following equations replace the precomputation table.

**Algorithm 4** Finding the valid inner part of 9-round GIMLI using precomputation tables

**Input:** Hash value $h = (h_0, h_1)$ and precomputation tables $T_1$, $T_2$, $T_3$, $T_4$, $T_5$
**Output:** Valid capacity part of $S_{h_1}$

**for** 0 to $2^{64} - 1$ **do**
  Choose random values for $S^0[1,2][0,1]$
  Compute ($S^3[1,2][0,1]$, $S^3[0][2,3]$)
  Store ($S^0[1,2][0,1]$,$S^3[1,2][0,1]$, $S^3[0][2,3]$) in $T_6$
  Choose random values for $S^9[1,2][0,2]$
  Compute ($S^{4.5}[1,2][0,2]$, $S^{4.5}[0][1,3]$)
  $T_7[(S^{4.5}[1][0])\|(S^{4.5}[2][0])] \leftarrow (S^9[1,2][0,2]$, $S^{4.5}[1,2][2]$, $S^{4.5}[0][1,3])$
**end for**
**for** 0 to $2^{64} - 1$ **do**
  Get ($S^0[1,2][0,1]$,$S^3[1,2][0,1]$, $S^3[0][2,3]$) from $T_6$
  **for all** $S^3[0][0]$ **do**
    Compute $S^{4.5}[*][0] = SP(S^3[*][0])$
    Get ($S^9[1,2][0,2]$, $S^{4.5}[1,2][2]$, $S^{4.5}[0][1,3]$) from $T_7[(S^{4.5}[1][0])\|(S^{4.5}[2][0])]$
    Get ($S^3[0][1]$, $S^{4.5}[1,2][1]$) from $T_4[(S^3[1][1])\|(S^3[2][1])\|(S^{4.5}[0][1])]$
    Get ($S^3[1,2][2]$, $S^{4.5}[0][2]$) from $T_5[(S^3[0][2])\|(S^{4.5}[1][2])\|((S^{4.5}[2][2]))]$
    Compute ($S^{0.5}[1,2][2], S^{0.5}[0][3]$) and ($S^7[1,2][1], S^7[0][3]$) from $S^3, S^{4.5}$
    Get ($S^0[1,2][2]$, $S^{0.5}[0][2]$) from $T_2[(S^{0.5}[1][2])\|(S^{0.5}[2][2])]$
    Get ($S^7[0][1]$, $S^{8.5}[1,2][1]$) from $T_3[(S^7[1][1])\|(S^7[2][1])]$
    Get $S^0[1,2][3]$ from $T_1[(S^{0.5}[0][3])\|(S^1[0][3])\|(S^3[0][1])]$
    Recover the state $S^0$ and compute all states to $S^9$
    **if** $S^{4.5}[0][3]$, $S^7[0][3]$, and $S^{8.5}[0][3]$ are collide **then**
      **return** $S^9[1,2][*]$
    **end if**
  **end for**
**end for**

$$\text{Given } h_{0,3}, c, f, \text{ and } x, \text{ find } a, b \text{ such that } SP(h_{0,3}, a, b) = (c, d, e) \text{ and } SP^2(x, d, e)_x = f. \tag{1}$$

$$\text{Given } h_{0,2}, d, e, \text{ find } c \text{ such that } SP(h_{0,2}, a, b) = (c, d, e). \tag{2}$$

$$\text{Given } b, c, d, \text{ find } a, e, f \text{ such that } SP^2(a, b, c) = (d, e, f). \tag{3}$$

$$\text{Given } a, e, f, \text{ find } b, c, d \text{ such that } SP^2(a, b, c) = (d, e, f). \tag{4}$$

The equation (1), (2), (3), and (4) replace precomputation table $T_1$, $T_2$, $T_4$, and $T_5$ in 4.2 respectively. $T_3$ can be replaced by equation (3) because $T_3$ is specific case of $T_4$. The equation (1), (2), (3), and (4) are expected to have 1 solution. For the case of equation (2), $h_{0,2}, d, e$ are fixed variables and the cardinal number of range of variable $c$ is $2^{32}$. Since the SP-box operation is a permutation, we can expect that there is 1 solution in the range of variable $c$. Similarly since $SP^2$ is also a permutation, equation (3) and (4) are expected to have 1 solution. For the case equation (1), $h_{0,3}, a, b, x$ are fixed variables and the the number of candidate of $a, b$ is $2^{64}$. Because $c$ is a fixed 32-bit variable, there are

$2^{32}$ candidates of $a, b, d, e$. In addition, $f$ is also fixed 32-bit variable, so equation (1) can be expected to have 1 solution.
There are a lot of ways to solve the equation (1), (2), (3), and (4). One of the fast ways is transforming the equations to SAT problems and using SAT solver. Some of the cryptographic equations are too complex to transform into SAT problems. For example, the S-boxes with a high algebraic degree are hard to convert to CNF, so sometimes it is easier to use the truth table. However, CNF of Gimli SP function is not complex because the SP-box of Gimli has a low algebraic degree.
The SP-box of the Gimli consists of bitwise AND($\wedge$), OR($\vee$), XOR($\oplus$) operations. When converting the SP operation to CNF, the AND and OR operations should be considered as basic operations first. For example, the equation in which the intermediate variable $x_3$ is calculated by AND operation of $x_1$ and $x_2$ is as follows.

$$x_1 \wedge x_2 = x_3$$

This equation can be treated as a logical expression such that $x_1$ is **true** if $x_1$ is '1' and $x_1$ is **false** if $x_1$ is '0'. Then the equation can be transformed to CNF equation as follow.

$$(\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee \neg x_3) = \textbf{true}$$

Similarly, the equation $x_1 \vee x_2 = x_3$ can be transformed to CNF equation as follow.

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3) = \textbf{true}$$

To complete the SP-box, 3 variables must be XORed to the output variable. In other words,

$$x_1 \oplus x_2 \oplus x_3 = x_4.$$

This XOR equation can be transformed to CNF equation with 8 clauses.

$$\begin{aligned}(x_1 \vee x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge \\ (x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3 \vee x_4) \wedge \\ (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge \\ (\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4) = \textbf{true}\end{aligned}$$

Rotation and bit shift operations can be treated by changing the variable index. The SP-box operation can be transformed to CNF combining the above equations.

To solve equations (1), (2), (3), and (4), we use CryptoMiniSat library[12] (`https://github.com/msoos/cryptominisat`). We construct a test to measure the average number of solutions and the operation time in the **Ubuntu 16.04.4 LTS** with **g++ 5.5.0** compiler using **C++** code. Our test code is on `https://github.com/yslee0804/Gimli_hash_9r_test`.

When the input variables are uniformly random, equation (1), (2), (3), (4) have on average 1 solution as expected. Table 2 shows the result of the average

solving time for equations. Based on full round Gimli, the solving time ranged from $2^{7.74}$ to $2^{19.15}$. Equation (2) finds the solution the fastest because it has simpler operations and fewer intermediate variables than other equations (only 1 SP-box operation). On the other hand, equation (1) has complex operations and many intermediate variables, so the solution is found the slowest. Equation (3) and (4) are very similar, but there is difference in solving speed. This is presumed to be due to the structure of SP-box. The forward direction of SP-box can be treat variables independently. That means any intermediate variable can be calculated with only input variables in the forward direction. However, when calculated in the backward direction, intermediate variables must be calculated sequentially. Since equation (4) is restricted by the given output variables, the intermediate variables were woven as if they were operating in the backward direction. On the other hand, equation (3) has more free output variables.

**Table 2.** Time complexity to solve equations

| | Gimli(full round) | eq (1) | eq (2) | eq (3) | eq (4) |
|---|---|---|---|---|---|
| Time complexity | 1 | $2^{19.15}$ | $2^{7.74}$ | $2^{10.34}$ | $2^{18.15}$ |

The Algorithm 5 shows the process of finding the valid inner part of 9-round Gimli using equations instead of precomputation tables. Except for the line 14 to 19 of Algorithm 5, the rest are the same as Algorithm 4.

**Complexity** In 4.2, time and memory complexity is $2^{96.44}$ and $2^{97}$ respectively. Because of $T_6$ and $T_7$, The memory complexity of 4.3 is equal to $2^{66.17}$. However, 5 table lookups in 4.2 change to equation (1) (4). Hence, the time complexity of 4.3 is $2^{96.29+\epsilon}$ where $2^{\epsilon}$ is the complexity of solving equations. Referring to the Table 2, the worst case time complexity of solving the equation(which is equation (1)) is about $2^{20.57}$ 9-round Gimli. The memory complexity decreased from $2^{97}$ to $2^{66.17}$ compared to Section 4.2.

## 5 Conclusion

In this paper, we analyzed preimage attack on 5-9-round Gimli-Hash, which extends the last two permutations of Gimli-Hash to 9-round. In the structure which the last two permutations extended to 9-round, it is difficult to apply the Liu et al.'s preimage attack. This is because there is a gap between the states $S^3$ and $S^{4.5}$ in the process of finding the valid inner part of 9-round Gimli. We bridge this gap using precomputation tables. As a result, finding the valid inner part has $2^{96.44}$ time-complexity and $2^{97}$ memory complexity. Moreover, equations that can replace the precomputation tables are presented to lower the high memory complexity. We proposed a method of transforming the equations to CNF that can be solved with SAT-solver. The solutions were found experimentally with the time complexity up to $2^{20.57}$ 9-round Gimli. If we use these equations instead of precomputation tables, the memory complexity is reduced to $2^{66.17}$ instead

**Algorithm 5** Finding the valid inner part of 9-round Gimli using equations

**Input:** Hash values $h_0$ and $h_1$
**Output:** Valid capacity part

**Find a valid capacity part**
**for** 0 to $2^{64}-1$ **do**
  Choose random values for $S^0[1,2][0,1]$
  Compute ($S^3[1,2][0,1]$, $S^3[0][2,3]$)
  Store ($S^0[1,2][0,1]$, $S^3[1,2][0,1]$, $S^3[0][2,3]$) in $T_6$
  Choose random values for $S^9[1,2][0,2]$
  Compute ($S^{4.5}[1,2][0,2]$, $S^{4.5}[0][1,3]$)
  $T_7[(S^{4.5}[1][0])\|(S^{4.5}[2][0])] \leftarrow (S^9[1,2][0,2]$, $S^{4.5}[1,2][2]$, $S^{4.5}[0][1,3])$
**end for**
**for** 0 to $2^{64}-1$ **do**
  Get ($S^0[1,2][0,1]$,$S^3[1,2][0,1]$, $S^3[0][2,3]$) from $T_6$
  **for all** $S^3[0][0]$ **do**
    Compute $S^{4.5}[*][0] = SP(S^3[*][0])$
    Get ($S^9[1,2][0,2]$, $S^{4.5}[1,2][2]$, $S^{4.5}[0][1,3]$) from $T_7[(S^{4.5}[1][0])\|(S^{4.5}[2][0])]$
    Solve the equation (3) from ($S^3[1,2][1], S^{4.5}[0][1]$)
    From the solution of equation (3), determine ($S^3[0][1]$, $S^{4.5}[1,2][1]$)
    Solve the equation (4) from ($S^3[0][2], S^{4.5}[1,2][2]$)
    From the solution of equation (4), determine ($S^3[1,2][2]$, $S^{4.5}[0][2]$)
    Compute ($S^1[*][2]$), ($S^7[1,2][1], S^7[0][3]$) start from $S^3, S^{4.5}$
    Solve the equation (2) from ($S^{0.5}[1,2][2]$)
    From the solution of equation (2), determine ($S^0[1,2][2]$, $S^{0.5}[0][2]$)
    Solve the equation (3) from ($S^7[1,2][1], S^{8.5}[0][1]$)
    From the solution of equation (3), determine ($S^7[0][1]$, $S^{8.5}[1,2][1]$)
    Solve the equation (1) from ($h_{0,3}, S^{0.5}[0][3], S^1[0][3], S^3[0][1]$)
    From the solution of equation (1), determine $S^0[1,2][3]$
    Recover the state $S^0$ and compute all states to $S^9$
    **if** $S^{4.5}[0][3]$, $S^7[0][3]$, and $S^{8.5}[0][3]$ are collide **then**
      **return** $S^9[1,2][*]$
    **end if**
  **end for**
**end for**

of $2^{96+\epsilon}$ time complexity, where $\epsilon$ is the time complexity of finding solutions of equations. A future work is preimage attack to 9-round Gimli-Hash. If two states $S_0$ and $S_1$ can be connected effectively, the preimage attack of 9-round Gimli-Hash may be possible.

**Acknowledgments**

This work was supported as part of Military Crypto Research Center(UD210027 XD) funded by Defense Acquisition Program Administration(DAPA) and Agency for Defense Development(ADD).

## References

1. Beierle, C., Biryukov, A., Cardoso dos Santos, L., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: Lightweight AEAD and hashing using the sparkle permutation family. IACR Transactions on Symmetric Cryptology **2020**(S1), 208–261 (2020). https://doi.org/0.13154/tosc.v2020.iS1.208-261
2. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., Viguier, B.: Gimli. Submission to the NIST Lightweight Cryptography Standardization Process (2019), `https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates`
3. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., Viguier, B.: Gimli : A Cross-Platform Permutation. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. Lecture Notes in Computer Science, vol. 10529, pp. 299–320. Springer, Heidelberg, Germany, Taipei, Taiwan (Sep 25–28, 2017). https://doi.org/10.1007/978-3-319-66787-4_15
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT hash workshop. vol. 2007 (2007)
5. Flórez-Gutiérrez, A., Leurent, G., Naya-Plasencia, M., Perrin, L., Schrottenloher, A., Sibleyras, F.: New Results on Gimli: Full-Permutation Distinguishers and Improved Collisions. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020, Part I. Lecture Notes in Computer Science, vol. 12491, pp. 33–63. Springer, Heidelberg, Germany, Daejeon, South Korea (Dec 7–11, 2020). https://doi.org/10.1007/978-3-030-64837-4_2
6. Hamburg, M.: Cryptanalysis of 22 1/2 rounds of gimli. Cryptology ePrint Archive, Report 2017/743 (2017), `https://eprint.iacr.org/2017/743`
7. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON Block Cipher Family. In: Gennaro, R., Robshaw, M.J.B. (eds.) Advances in Cryptology – CRYPTO 2015, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 161–185. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015). https://doi.org/10.1007/978-3-662-47989-6_8
8. Liu, F., Isobe, T., Meier, W.: Automatic Verification of Differential Characteristics: Application to Reduced gimli. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 219–248. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2020). https://doi.org/10.1007/978-3-030-56877-1_8

9. Liu, F., Isobe, T., Meier, W.: Exploiting weak diffusion of gimli: Improved distinguishers and preimage attacks. IACR Transactions on Symmetric Cryptology **2021**(1), 185–216 (2021). https://doi.org/10.46586/tosc.v2021.i1.185-216
10. Liu, Y., Wang, Q., Rijmen, V.: Automatic Search of Linear Trails in ARX with Applications to SPECK and Chaskey. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 16: 14th International Conference on Applied Cryptography and Network Security. Lecture Notes in Computer Science, vol. 9696, pp. 485–499. Springer, Heidelberg, Germany, Guildford, UK (Jun 19–22, 2016). https://doi.org/10.1007/978-3-319-39555-5_26
11. Mouha, N., Preneel, B.: Towards finding optimal differential characteristics for arx: Application to salsa20. Cryptology ePrint Archive, Report 2013/328 (2013), `https://eprint.iacr.org/2013/328`
12. Soos, M.: Cryptominisat 5.8.0, `https://github.com/msoos/cryptominisat/`
13. Sun, L., Wang, W., Wang, M.: Automatic Search of Bit-Based Division Property for ARX Ciphers and Word-Based Division Property. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017, Part I. Lecture Notes in Computer Science, vol. 10624, pp. 128–157. Springer, Heidelberg, Germany, Hong Kong, China (Dec 3–7, 2017). https://doi.org/10.1007/978-3-319-70694-8_5
14. Zong, R., Dong, X., Wang, X.: 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 The Preimage Attack of Liu et al.'s on 5-Round Gimli-Hash

The 5-round Gimli-hash is a modification of the internal Gimli of Gimli-hash to 5 rounds. The preimage attack on 5-round Gimli-hash was analyzed in [9]. The aim of this attack is to find five 128-bit message blocks for a given 256-bit hash value. The attack process follows the general approach in section 3.

**Phase 1: Finding a Valid Inner Part** For the given hash value $h$=($h_0$,$h_1$), $h_0$ and $h_1$ are the 128-bit values of the rate part of state $S_{h_0}$ and $S_{h_1}$. Thus, the states $S^0[0][*] = h_0$ and $S^5[0][*] = h_1$ are set. Then, the following steps can be found an appropriate capacity part that $\mathbf{Gimli}(S^0) = S^5$.

*Step 1* : Choose $2^{64}$ random values for $S^0[1,2][0,1]$. Since $S^0[0][*]$ was known by $h_0$, two columns $S^0[*][0,1]$ can compute the values of $S^3[1,2][0,1]$ and $S^3[0][2,3]$. Store the $S^3[1,2][0,1]$ and $S^3[0][2,3]$ in the table $T_8$.

*Step 2* : Similar to step 1, choose a random value for $S^5[1,2][0,1]$. Then the value $S^3[*][0,1]$ can be obtained from two columns $S^5[*][0,1]$ with backward direction computation. Check whether the values $S^3[1,2][0,1]$ are in the table $T_8$. If the values are in the table $T_8$, hold all selected values and go to the next step. Otherwise, repeat this step. This step may be repeated $2^{64}$ times.

*Step 3* : Note that $S^3[0][*]$, $S^3[1,2][0,1]$ and $S^5[1,2][0,1]$ are fixed in *Step 2*. For all values of $S^5[1,2][2]$, $S^3[*][2]$ can be obtained with backward direction computation. Check that computed $S^3[0][1]$ matches the fixed $S^3[0][1]$. If it

matched, then store the values $(S^5[1,2][2], S^3[1,2][2])$ in the table $T_9$. The expected number of stored values is about $2^{32}$. Similar to the previous process, for all values of $S^5[1,2][3]$, $S^3[*][3]$ can be obtained. Check if it matches the fixed $S^3[0][3]$ and store $(S^5[1,2][3], S^3[1,2][3])$ in the table $T_{10}$

*Step 4* : Select each value in $T_9$ and $T_{10}$ and then compute $S^0[*][2,3]$ in backward direction. Then check the values $S^0[0][2,3]$ matches the given $S^0[0][2,3]$ which is the part of hash value $h_0$. If it is matched, return $S^5[1,2][*]$ which is capacity part of $S_{h_1}$. Else, repeat this step. Because all possible number of combinations is about $2^{64}$, we can expect the one will be matched.

**Complexity** The time complexity of Step 1 is $2^{64} \times 6$ SP operations and the memory complexity is $2^{64} \times 192$ bits $\approx 2^{64.58}$ 128-bit block. Since Step 2 may be repeat $2^{64}$ times, the time complexity is $2^{64} \times 4$ SP operations. In the Step 3, all $S^5[1,2][2]$ values and all $S^5[1,2][3]$ are searched. Also, in each search, the time complexity is $2^{64} \times 2$ SP operations and the memory complexity is $2^{32}$. In the Step 4, $2^{64}$ $T_9$, $T_{10}$ combination are possible, and each pair need 6 SP operation. Since 5-round GIMLI consist of 20 SP operations, total time complexity is $(2^{64} \times (6+4+2+2+6)$ SP operations$) \approx (2^{64}$ 5-round GIMLI), and total memory complexity is about $2^{64.58}$.

**Phase 2: Choosing $M_3$ and $M_4$** The capacity part of $S_{h_1}$ was found by **phase 1**, the full state of $S_{h_1}$ and $S_{h_0}$ can be recovered. Thus, the state $S_2$ can be obtained by selecting any $M_3$ and $M_4$. Therefore, choose random values for $M_3$ and $M_4$ and compute backward direction to get $S_2$.

**Phase 3 : Matching the Inner Part** In this phase, select the appropriate $M_0$ and $M_1$ connecting the capacity part of $S_0$ and $S_2$. The capacity part of $S_0$ was set to 0 and the rate part of $S_0$ can be controlled by $M_0$. Additionally, the capacity part of $S_1^5$ can be known because $S_2$ was known and the rate part of $S_1^5$ can be controlled by $M_2$. Since the capacity part of $S_0$ is 0, the precomputation tables can be created. After creating precomputation tables, the part of $S_1^5$ would be guessed and connected it with the precomputation tables. In this process, the following property is used for checking validation.

*Property 1 ([8]).* Suppose $(OX, OY, OZ) = SP(IX, IY, IZ)$. 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 $2^{-32}$.

The following steps show the way to find $M_0$ and $M_1$ which capacity part of states are connected.

*Step 0* (precomputation step): Since $S_0^0[1,2][*]$ is all zero, $S_0^{0.5}[0][*]$ is also zero. Therefore, for all values of $S_0^0[0][0,2]$, the values $S_1^0[1,2][0,2]$ can be computed. ($S_0^0[0][0,2]$, $S_1^0[1,2][2]$) is stored in the table $T_{10}[(S_1^0[1][0] \| S_1^0[2][0])]$. Similar to before process, for all values of $S_0^0[0][1,3]$, calculate $S_1^0[1,2][1,3]$. ($S_0^0[0][1,3]$, $S_1^0[1,2][3]$) is stored in the table $T_{11}[(S_1^0[1][1] \| S_1^0[2][1])]$.

*Step 1* : For a guessed value $S_1^5[0][1,3]$, the values ($S_1^{0.5}[0][1,3]$, $S_1^{0.5}[1,2][0,2]$) can be calculated in the backward direction. Then, guess all values for $S_1^{0.5}[0][0]$, and calculate $S_1^0[1,2][0]$. From $T_{10}[(S_1^0[1][0]\|S_1^0[2][0])]$, get the values for ($S_0^0[0][0,2]$, $S_1^0[1,2][2]$). The value $S_1^0[1,2][2]$ in the $T_{10}$ and calculated value $S_1^{0.5}[1,2][2]$ are valid with probability $2^{32}$ by property 1. Hence, there is one valid pair ($S_1^0[1,2][2]$, $S_1^{0.5}[1,2][2]$) for all $S_1^{0.5}[0][0]$ as expected. Moreover, if ($S_1^0[1,2][2]$, $S_1^{0.5}[1,2][2]$) is valid, it can determine $S_1^{0.5}[0][2]$ by 1. From the valid $S_1^{0.5}[0][0]$, store ($S_0^0[0][0,2]$, $S_1^5[0][1,3]$, $S_1^{0.5}[0][*]$) in the table $T_{12}$. Repeat this step for all $S_1^5[0][1,3]$.

*Step 2* : Similar to Step 1, for all values $S_1^5[0][0,2]$, calculate the values ($S_1^{0.5}[0][0,2]$, $S_1^{0.5}[1,2][1,3]$). Then guess $S_1^{0.5}[0][1]$ for all possible value and calculate $S_1^0[1,2][1]$. From $T_{11}[(S_1^0[1][1]\|S_1^0[2][1])]$, get the values for ($S_0^0[0][1,3]$, $S_1^0[1,2][3]$). By property 1, verify ($S_1^0[1,2][3]$, $S_1^{0.5}[1,2][3]$) and recover valid $S_1^{0.5}[0][3]$. Check the values $S_1^{0.5}[0][*]$ are in the table $T_{12}$. If it in, move to step 3, else repeat this step.

*Step 3* : At the step 2, $S_0^0[0][1,3]$ was guessed and $S_0^0[0][0,2]$ was found in the table $T_{12}$. Therefore, the message $M_0 = S_0^0[0][*]$ can be recovered, and then the values $S_0^5[0][*]$ can be calculated. Moreover, $S_1^0[0][*]$ can be calculated because $S_1^5[0][0,2]$ and $S_1^5[0][1,3]$ were also known at the step 2. As a result, the message $M_1$ can be obtained by $S_0^5[0][*] \oplus S_1^0[0][*]$.

**Complexity** In the precomputation, the time complexity is $2 \times 2^{64} \times 10$ SP operation and the memory complexity is $2 \times 2^{64}$. At the step 1, there are $2^{64}$ $S_1^5[0][0,2]$. Each $S_1^5[0][0,2]$ performs $8 + 2^{32} + 2^{32}$ SP operations and store one ($S_0^0[0][0,2]$, $S_1^5[0][1,3]$, $S_1^{0.5}[0][*]$) in $T_{12}$ as an expected value. Thus, the time and memory complexity of step 1 is $2^{96}$ SP operations and $2^{64}$ 256-bit blocks, respectively. Similarly, the time complexity of step 2 is $2^{96}$ SP operations. The time complexity of step 3 is just 2 Gimli. Therefore, total time complexity is ($2 \times 2^{97}$ SP operations) $\approx$ ($2^{93.68}$ 5-round Gimli), and total memory complexity is $2^{66}$ 128-bit blocks.

**Phase 4: Finding the Valid $M_2$** At the **phase 2**, the state $S_2^0$ was recovered. Additionally, at the **phase 3**, the state $S_1^5$ was also recovered. Therefore, the message $M_2$ can be obtained by $S_2^0[0][*] \oplus S_1^5[0][*]$.