International Journal of Security and Its Applications
Vol. 11, No. 8 (2017), pp.45-58
http://dx.doi.org/10.14257/ijsia.2017.11.8.04

Double Length Sponge Construction DLP-Sponge

Baraa Tareq Hammad[1], Norziana Jamil[2], Mohd Ezanee Rusli[1]and Muhammad Reza Zaba [3]

[1] Universiti Tenaga Nasional, College of Computer Science and Information Technology, Malaysia.
[2] Universiti Tenaga Nasional, Center of Information and Network Security, Malaysia.
[3]MIMOS Berhad, Malaysia.
E-mail: omrami82@yahoo.com, {norziana, ezanee}@uniten.edu.my, reza.zaba@mimos.my

Abstract

In this paper, a new sponge construction called DLP-Sponge is proposed, which takes an arbitrary length of input and yields an output of random length. We prove that this construction is resistant against generic attacks such as multicollision attacks with a better complexity $2^{2(c+3)/2}$ even when a small capacity is used, i.e. lightweight cryptography, where c is the capacity. Furthermore, it is used in building other cryptographic primitives such as block cipher, cryptographic hash functions and Message Authentication Codes (MACs).

Keywords: *Sponge Construction, DLP-Sponge, Hash function, Double Length, Duplex Construction.*

1. Introduction

A cryptographic hash function is a function that takes an input of arbitrary length and produces a fixed-length output called a message, without involving any secret parameters. It typically consists of two parts, i.e. internal part known as compression function *f* (which is iterated sufficiently many times) and external part known as hash construction such as Merkle-Damg°ard [1] and sponge [10], to name a few.

It seems impossible to have an iterated hash function behaving like a random oracle. Similar shortcoming can be observed in sponge construction due to the involved inner collision [11]. Some researchers have reported that sponge construction shows weakness when the capacity equals to the output, *i.e.* $c = n$, with inner collision of $2^{(c+3)\backslash 2}$ which is similar to that of Random Oracle [5, 11].

Examples of hash function designs based on sponge construction are Keccak [12], PHOTON [13], Spongent [14], quark [15] and LHash [16]. These primitive designs are established based on permutations of many variants. Most of them are designed for resource-constrained devices with $c = n$.

To have a more secure and efficient construction, we consider a new double length construction. Previously, Hirose [23, 24] employed a Double Block Length construction by using two different block ciphers. They found that the collision resistance is $2^{n/2}$, which is more secure than Merkle-Damg°ard. Nandi [25] used a 2/3-rate double length compression function which takes n inputs and produces $2n$ outputs with collision resistance of $2^{2n/3}$. All these constructions were designed to solve the security issues of Merkle-Damg°ard construction. In this paper, a new sponge construction is proposed by creating two permutation lines running in parallel where the input length in every permutation line is doubled with capacity $c= 2n$. We refer this construction as DLP-Sponge.

We propose many applications of DLP-Sponge such as hash functions, MAC, block cipher and Authenticated Encryption. The main goal of DLP-Sponge functions is to facilitate designers in formulating primitive according to their requirements.

In this paper, general descriptions of sponge construction are firstly provided. In section 3 we give a description of our construction DLP-Sponge. Next, we discuss the security analysis of DLP-Sponge, the conclusion of our work is given in section 6.

2. Sponge Construction

Sponge construction is an iterative construction designed by Bertoni, G. et.al. [10, 11] that maps a variable length input to a variable length output. This feature renders this construction suitable for many applications such as hash function, stream cipher, mask generation function and Message Authentication Code (MAC) [17]. The length state $b = r+c$ is fixed, where r denotes the bitrate and c represents the capacity determined through a function f that generates a transformation or permutation of b. Sponge construction operates in two phases as shown in Figure1.

- The absorbing phase: Firstly, message M is padded with 1 followed by many 0s to make the total length a multiple of r. Then, the message is divided into r-bit blocks. Each input block m_i is XORed with the r part of internal state S. Finally, process S is iterated until all blocks are exhausted. The absorbing phase is denoted by $S = S_f(S_r \oplus m_i, S_c)$, where S_r and S_c refer to the outer and inner parts of S, respectively.
- The squeezing phase: The state progresses according to S_f; however, the r parts of the state are returned as output blocks after each iteration. The size of final output is determined by the user. The squeezing phase is denoted by $z_j = S_r$ and $S = S_f(S)$.

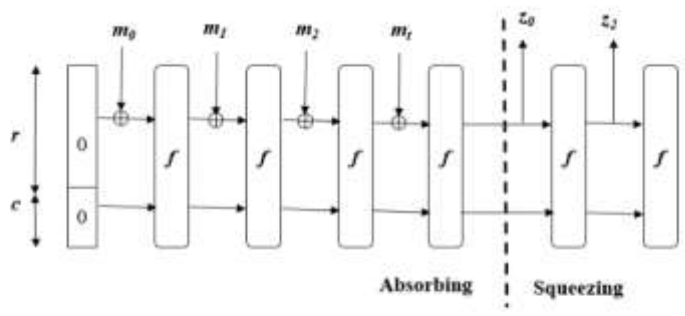

Figure 1. Sponge Construction

2.1 Duplex Sponge

Duplex construction as depicted in Figure2 is very similar to the design of the sponge construction. The main difference between the two designs is that in the former, there is no squeezing phase before the final digest is produced [20].

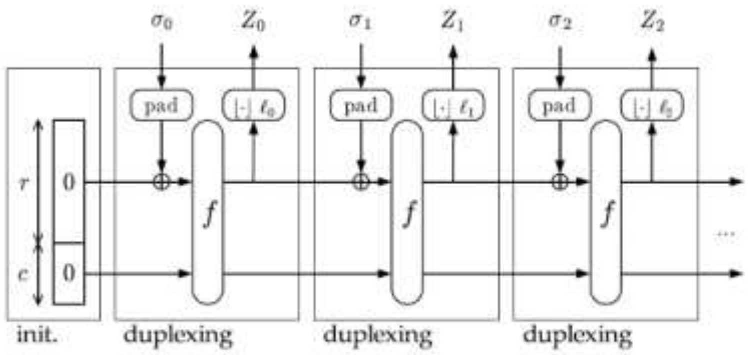
σ_0
Z_0
σ_1
Z_1
σ_2
Z_2
pad
$\lfloor\cdot\rfloor_{\ell_0}$
$\lfloor\cdot\rfloor_{\ell_1}$
$\lfloor\cdot\rfloor_{\ell_2}$
r
c
0
0
f
...
init.
duplexing
duplexing
duplexing

Figure 2. The Duplex Construction [20]

2.2 SpongeWrap

The SpongeWrap construction [20] is designed for authenticated encryption as shown in Figure 3. Firstly, the key *k* is initialized and loaded into the state. Next, the header *A* is padded and absorbed into the state. The message *M* is padded and divided into *p* blocks, and then the encryption (or decryption) runs in duplex mode. The resulted tag *T* is then compared with the recited tag to check the validity of this tag [18].

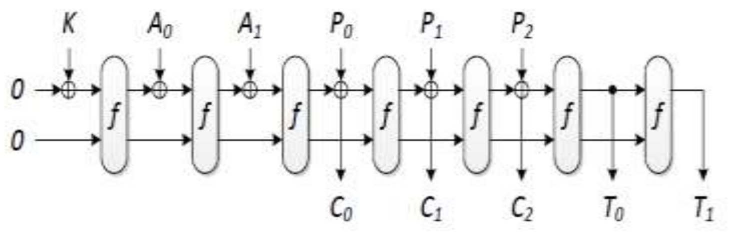

Figure 3. SpongeWrap Authenticated Encryption [18]

3. DLP-Sponge Construction

In this Section, a new construction based on sponge construction, i.e. DLP-Sponge, is proposed. In DLP-Sponge construction, problems of small keys and the resistance against generic attacks of sponge construction when the capacity *c* equals to the output *n* are addressed. Most of the previous work showed that sponge construction is similar to random oracle when $c = n$ [11, 20, 21].

The DLP construction uses two *b*-bit compression functions *f*. Two strings are kept similar during the absorbing process. Thereafter, two inputs are taken from the two strings to the compression function *f* as *r* and *c* for the issuance of the output as shown in Figure4. The procedures of DLP-Sponge construction phase are described below.

1- The message *M* is padded by *1* bit and *0* bits whereby the length of padded *M* must be a multiple of *r*-bit.
2- Initialize the capacity *c* and the bitrate *r* with zeros.
3- Divide *M* into *r*-bit blocks and process sequentially.
4- The absorbing phase: Each input block m_i is XORed with the *r* part of internal state *S*. The DLP-Sponge function has an internal state $S = (S_r, S_c) \in 2^r \times 2^c$, where the initial value is $(0, 0)$. Here, $0 \in 2^r$ is the neutral element of *r* and $0 \in 2^c$. S_0 *and* S_1 are iterated until all blocks are exhausted. Where S_0 is for the above pipeline state *and* S_1 for the bellow state.

$S_1 = S_1 \oplus (m_{I+1} \,||\, 0^{(b-r)})$

$S_0 = f(S_0)$
$S_1 = f`(S_1)$
$S = S_0 \| S_1$

5- The squeezing phase: The state progresses depending depends on S_f; however, the output block is the result of the r parts of the states after each iteration. $Z_j = S_r$ and $S = S_f(S)$.

Algorithm 1:- DLP-Sponge construction DLPS [*f*, *pad*, *r*]

```
Require: $r < b$
        Interface $Z$= DBLS ($M, \ell$) with $M \in Z^2$, integer $\ell > 0$ and $Z \in Z^2$.
        $m = M \| \text{pad}[r]([M])$.
        Let $M = m_0 \| m_1 \| \dots \| m_w$ with $[m_i] = r$.
                $s = 0^b$.
        for $i = 0$ to $w$ do
                $s = s \oplus (m_i \| 0^c)$
                $s` = s` \oplus (m`_i \| 0^c)$
                $s = f(s)$
                $s` = f`(s`)$
                $S = s \| s`$
        end for
        $Z = [S]^r$
        While $[z] < \ell$ do
                $s = f(S)$
                $Z = Z \| [S]_r$
        end While
return $\lceil Z \rceil^{\ell}$
```

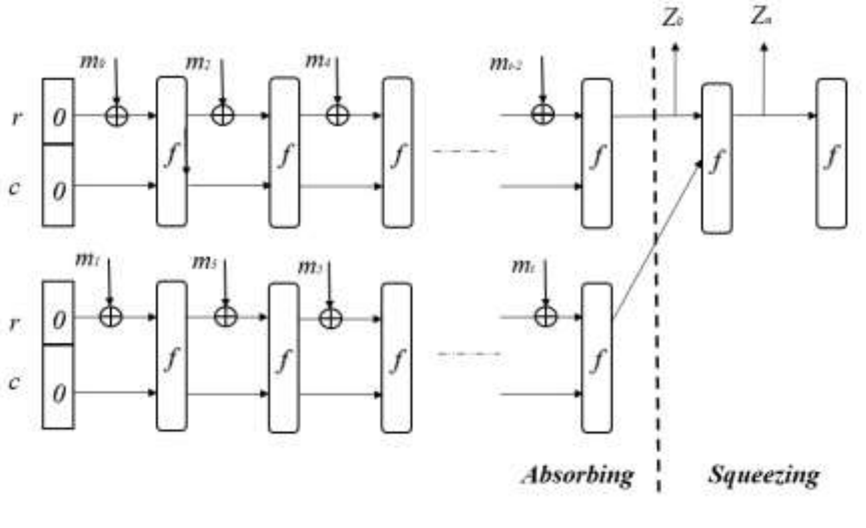

Figure 4. DLP-Sponge Construction

4. Security Analysis of DLP-Sponge

When a new cryptographic hash function is designed, one chooses a construction and a compression function to build a new function with input and output of arbitrary sizes [20]. When the security of this construction is proven, these primitives are guaranteed to be secure against generic attacks such as multicollision [4], herding [8], second pre-image

[6], faster multicollision [7] and length extension attacks [9]. It's important to know the most important security standard before designing a new hash function. Security standards for a cryptographic hash function are collision resistance, pre-image resistance and second pre-image resistance [11].

In a hash function, collision resistance is considered as the main security criterion. The birthday paradox theory shows that the effort to find two peoples with the same birth date is comparable to that of finding two messages with the same hash output. All iterated hash functions suffer from the same weakness. The birthday attack can happen with complexity ($2^{n/2}$).

In sponge construction, the security depends on the capacity *c* [10, 11]. Therefore, any attack on sponge focuses on capacity *c* rather than output *n*. When the capacity *c* is small, the collision is limited to *c* with a complexity of $2^{c/2}$. Nevertheless, this is not true in lightweight cryptography, where using $c=2n$ is considered as expensive in terms of area as found in PHOTON [13], quark [2, 15] and LHash [16].

In order to study the security of DLP-Sponge construction, birthday attack is analyzed. This attack can be applied to any iterated hash function which requires $2^{n/2}$ complexity. On the other hand, it requires $2^{c/2}$ in sponge construction. It is more susceptible to an attack if *c* is small. Therefore, DLP doubles the size of *c* by doubling the sponge length to provide better security.

When using DLP-Sponge, it must provide collision, second pre-image and pre-image resistances. The construction must not have any clear vulnerabilities.

We use the analysis of sponge construction as our reference [10]. For measuring the resistance of DLP-Sponge, the probability of success for these operations has been counted.

– Inner collision.
– Track to an inner state.
– Output cycles.
– Linking an output string to a state.

Firstly, we represent the DLP-Sponge graphically for better clarity. DLP-Sponge graph represents the information that the attacker gains from the experiments. This graph is known as the attacker's graph.

4.1 Graph Representation

A DLP-Sponge graph can be represented by 2^{2b} nodes and 2^{2b} edges. The nodes are the state values. For states' *S* and *T*, there is an edge going directly from *S* to *T*, where $T = f(S)$. For each node there are is one arriving edge and one exiting edge. The nodes can be divided into subsets of nodes according to the value of inner state *S*: inner parts and outer parts. There are 2^{2c} inner parts. The 2^{2r} nodes within the outer parts are determined by their S_R values.

From the root (*0*, *0*), input string *m* can be absorbed by the following edges beginning from the inner part *0*. Firstly, an edge is drawn from (m_0, *0*) and (m_1, *0*). This edge interacts with nodes with outer parts ($S_{R,f0}$ [m_0] and $S_{R,f1}$ [m_1]) as well as of inner parts ($S_{C,f0}$ [m_0] and $S_{C,f1}$ [m_1]). After that, an edge is drawn from the nodes within the inner part and outer parts ($S_{R,f0}$ [m_0]$\oplus m_2$ and $S_{R,f1}$ [m_1]$\oplus m_3$) they arrives at nodes S_{f0} [$m_0 m_1$] and S_{f1} [$m_0 m_1$]. For m_i, the nodes in outer parts ($S_{R},_{f0}$ [$m_0 m_2 \ldots m_{i-2}$] $\oplus$ m_{i-2} and S_R,f_1 [$m_1 m_3 \ldots m_{i-2}$] $\oplus$ m_{i-2}) and inner parts ($S_{C,f}$ [$m_0 m_1 \ldots m_{i-1}$] and $S_{C,f1}$ [$m_0 m_1 \ldots m_{i-1}$]) are linked. Graphically, track to an inner state is a track of edges running into the inner part (coming from the root). Due to the graphical representation, one can restore the value of *m*. The *i*-th character of the track *m* is specified by the edges from the *i*-th inner part on the track. Indeed, it is the outer part of the node where the outgoing edge minus the incoming edge.

The first symbols of tracks m_0 and m_1 are compatible with the root, which are equivalent to the outer part of the node where the first edge begins and there is no outgoing edge.

In this paragraph, we outline the attacker graph. Firstly, we assume that the attacker has no information about *f*. To get these information the attacker must send enquires to *f* (and f^{-1}). We suppose the attacker makes no calls compatible with the familiar edges. From here, a call to *f* and f^{1} is compatible to adding an edge beginning and arriving at familiar node.

In the attacker graph, an inner part T_C can be reached from an inner part S_C if there is a string of directed edges from S_C to T_C or $T = S$. We refer to the group of inner parts that can be reached from the root as rooted inner partsÅ, with $A = |Å|$. All nodes in a rooted inner part are referred as rooted nodes.

4.2 Inner Collision

If the attacker can find two tracks from the root to the same inner part then an inner collision occurs. First, we assume that there is no inner collision acquired yet. Next, the attacker makes i-th calls to *f* in order to calculate the probability that causes to an inner collision. We call the rooted inner part reachable in the attacker graph as Å. Their group is denoted as *V* with $V = |V|$, where $Å \subseteq V$. Before adding the i-th edge, $V = Å = \{0\}$ and $A = V = 1$. Where $i - 1$ edges and $A \le V \le i$.

Before adding the the i-th edge, $V = Å = \{0\}$ and $A = V = 1$. Where $i - 1$ edges and $A \le V \le i$.

The attacker can add edges and arrives to any selected nodes. As explained before in the graph representation each node has only one edge that comes in and one edge that goes out. So the attacker has different positions of adding edges to nodes. If the attacker adds edge to rooted node (*0*, *0*), then the probability of success is:

$$\frac{(2^{2r}-1)V+1}{2^{2b}-i}$$

If the added edge arrives to a selection node in *V*, the same probability is obtained:

$$\frac{(2^{2r}-1)A+1}{2^{2b}-i}$$

Here, *A* is a subset from *V* and $A \le V \le$ i. Then, the probability of the i-th call is:

$$\frac{(2^{2r}-1)i+1}{2^{2b}-i}$$

When an edge arrives to a selected node in *V* that does not have an inner collision, *A* is not influenced and therefore it leads to $A < i$. We get:

$$Pr(no\ IC) = \prod_{i=1}^{N}\left(1-\frac{(2^{2r}-1)i+1}{2^{2b}-i}\right) = \prod_{i=1}^{N}\frac{1-\frac{i}{2^{2c}}-\frac{1}{2^{2b}}}{1-\frac{i}{2^{2b}}}$$

Here, *N* refers to the number of calls that the attacker makes to *f* and f^{1}. The cost of attack is:

$$c_p(IC) \approx \sum_{i=1}^{N} -\frac{i-1}{2^{2b}}+\frac{i}{2^{2c}} = \frac{N(N+1)}{2^{2c+1}}-\frac{N(N-1)}{2^{2b+1}}$$

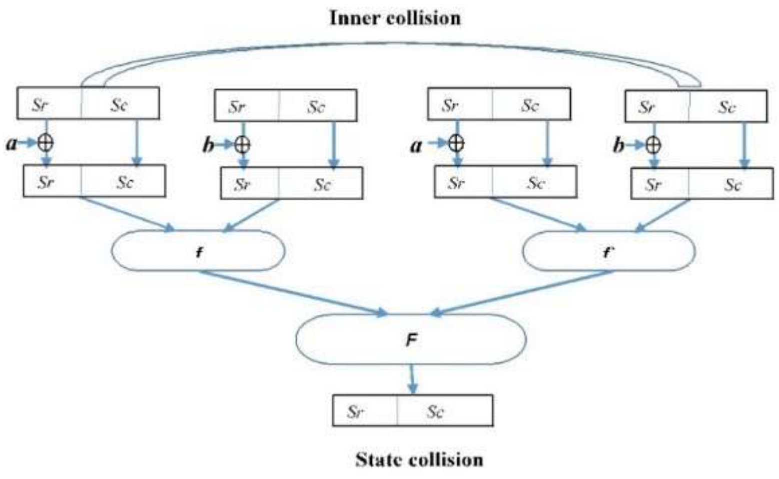

Figure 5. Inner Collision and State Collision in DLP-Sponge

4.3 Finding a Track to an Inner State

The attacker can find track p where $S_{C,f}\,[p] = T_C$. T_C is familiar to the attacker. We assume that no track is found yet; therefore, the attacker makes the i-th call to f and computes the probability. A rooted inner part must be linked to inner part and hence we can reach T_C. In the beginning, $V = \{T_C\}$, $A = \{0\}$ and $A = V = 1$. Right before the i-th quire, the graph includes $i - 1$ edges where $A \leq i$, $V \leq i$ and $A + V \leq i + 1$.

The probability of success if the edge starts from rooted selection node is:

$$\frac{(2^{2r}-1)V\ +1}{2^{2b}-i}$$

If the added edge arrives to the target node T_C, the attacker gets the same probability. We assume that there is no inner collision because it would increase the success probability.

$$\frac{(2^{2r}-1)A\ +1}{2^{2b}-i}$$

We add *1* to V if there is no inner collision and the new edge does not begin from a target-reaching node. We suppose that the two previous conditions are true. Subsequently, we check if this supposition is warranted. Anyway, if there are no inner collisions, the success probabilities would be affected.

To find a track to T_C, the edge must start from the rooted nodes. Adding edge on the rooted nodes ensures better success probability. So, $\delta i = 1$ is introduced if the edge starts from rooted nodes. Otherwise it is -1. The probability of this case is:

$$1-\frac{(2^{2r}-1)\left(\frac{1+\delta i}{2}Vi+\frac{1-\delta i}{2}Ai\right)+1}{2^{2b}-i}$$
$$=\frac{1-\frac{i+1}{2^{2b}}-\frac{2^{2r}-1}{2^{2b+1}}(Vi+Ai-\delta i(Ai-Vi))}{1-\frac{i}{2^{2b}}}$$

Then the cost function is:

$$c_p(path)\approx\sum_{i=1}^{N}\left(\frac{1}{2^{2b}}+\frac{2^{2r}-1}{2^{2b+1}}\big(Vi+Ai-\delta i(Ai-Vi)\big)\right)$$

We have $Vi + Ai \leq i + 1$ by assuming that $Vi + Ai = i + 1$. Moreover, we have $Ai - Vi = \sum_{j=1}^{i=1} \delta i$. This gives:

$$c_p(path) \approx \frac{N}{2^{2b}} + \frac{2^{2r}-1}{2^{2b+2}}\left(N^2 + 3N + 1 - \sum_{i=1}^{N}\sum_{j=1}^{i-1} 2\delta i\delta j)\right)$$

We can now derive the last term by using:

$$\sum_{i=1}^{N}\sum_{j=1}^{i-1} 2\delta i\delta j = \sum_{i=1}^{N}\sum_{j=1}^{N} \delta i\delta j - \sum_{i=1}^{N} \delta i\delta i = (A_{N+1} - V_{N+1})^2 - N$$

This gives:

$$c_p(path) \approx \frac{N(N+4) - (A_{N+1} - V_{N+1})^2}{2^{2b}} - \frac{N^2 - (A_{N+1} - V_{N+1})^2}{2^{2b+2}}$$

It can be maximized if N *is* even. Then $A_{N+1} = V_{N+1}$. Otherwise, $|A_{N+1} = V_{N+1}| = 1$. It is better by adding edges starting from and arriving to the selected nodes of A in a rotational manner to ensure $(A_N - V_N)2 \leq 1$.:

$$c_p(path) \approx \frac{N(N+4)}{2^{2c+2}} - \frac{N^2}{2^{2b+2}}$$

4.4 Detecting Cycles in the Output

By taking two inputs p, q and an integer d the attacker can absorb them to yield a nodes $S_f[p]$ and $S_f[q]$. The outputs $S_{R,f}[p|0^d]$ and $S_{R,f1}[q|0^d]$ are created by following a set of nodes linked by edges, i.e., $S_f[p|0^d] = f(S_f[p|0^{d-1}]$ and $S_f[q|0^d] = f(S_f[q|0^{d-1}]$, where we locate a string as a series of nodes linked by directed edges. The first node in the string is $U = Sf[p_0] + p|p|{-1}$ and $V = S_f[q_0] + q|q|{-1}$ with p_0 and q_0 equal to p and q, respectively. Consequently, the last characters $p|p|{-1}$ and $q|q|{-1}$ are removed.

By adding edges to the last nodes, the attacker can find a cycle by creating nodes in such strings until the new edge arrives to a node in the string. The shortest standing input string has a length of *1* that includes a non-zero character. Before adding the *i*-th edge, the string includes *i* nodes.

The probability that the new edge arrives to a node in the string is subsequently $1/(2^b)$.

$$cp(output\ cycle) \approx \frac{N}{2^b}$$

4.5 Linking an Output String to a State

Given a string Z, we intend to find a state S such that $Z = z_0, z_1, z\ldots z_t$. DLP-Sponge generates Z as an output, where $S_R = z_0$, $f_R(S) = z_1$, $f_R(f(S)) = z_2 \ldots f_R(f_{m-1}(S)) = z_t$. Here, the inner part S_C is unknown.

The probability, over f and over $S_C \in 2^{2c}$, is:

$$f_R\left(f^{i-1}(z_0, S_C)\right) = z_i, \forall i \in \{1\ldots m\}, \qquad (1)$$

The effect of z_i values on the success probability is described here. z_i would be similar if we have identical values of S_C or there is a cycle. If S_R is small, then the effect is more pronounced. If there is no cycle, the probability of success for the *i*-th check is $1/2^{2r}$. This probability is only $\frac{C-q}{2^{2b}-1}$, where q is the number of zj with $j < i$. The impact is significant if zi is unfair. It could be neglected for large 2^{2c}, and the success probability can be approximated by 2^{2r-1}. The probability that S can detect a cycle of period d is $1/2^{2b}$. Meanwhile, the probability of this cycle to find the correct output is about $1/2^r\ d^{-1}$. The probability of a cycle to locate a specific inner state is $1/2^{2b}$ and the probability that there is no cycle found is $1 - 1/2^{2r}$.

To find a cycle such as $f_R(z_0, S_C) = z_1$ the attacker must keep on guessing. Then, he/she can estimate $f_R(f(z_0, S_C))$ and check if this value is equal to z_2. If yes, the process is continued until the attacker reaches the last character. Else, the process is restarted from a new guess for S_C. If there is no cycle, the attacker has a probability of 2^{r-1}. The attacker proceeds with the next guess for S_C. At the moment a wrong character is found, the number of calls to f is 2^{2r-1}.

If $2^{2rm} < 2^{2c}$ the success probability is:

$$Pr(success\ with\ guess) = 1 - (1 - 2^{2rm}).$$

The cost is:

$$c_p(state\ binding) \approx \frac{2^{2r} - 1}{2^{2r}} N/2^{2rm}.$$

When $rm > c$ the expected number for N increases. Meanwhile, numerous trials within S_C
$\in 2^{2c}$ must be attempted by the attacker. Therefore, the probability is:

$$c_p(success\ with\ guess) \approx \frac{1 - 2^{2r-1}}{2^{2r}} \frac{N}{2^{2c}}.$$

5. DLP-Sponge as a Hash Function

In this section we will discuss some of the generic attacks on DLP-Sponge and show the relation between the previous operation and each attack. The resistance of DLP-Sponge is then compared with that of sponge construction.

5.1 Output Collision

The differences of inner collisions, state collision and output collision are shown in Figure6. Inner collision occurs if we have two different messages $M \neq M`$ processed by the same inner state: $S_{c,f}[M] = S_{c,f}[M`]$. If a state collision exists, then an inner collision exists as well. However, the reverse is not true.

We get state collision if we have two different messages $M \neq M`$ processed by the same state $S_f[M] = S_f[M`]$. The state collisions obtained during the absorbing phase may produce similar hash function values $S_f[M] = S_f[M`]$. The squeezing phase produces the same output values $S_f[M|0^j] = S_f[M`|0^j]$ for all j. When inner collisions p and q occur, then we can have a state collision with $p|a$ and $q|b$ for any a and b that satisfy $S_{R,f}[p]+a = S_{R,f}[q]+b$ [10]. Then, any two inputs $p|a|m$ and $q|b|m$ produce an output collision.

In DLP-Sponge, the complexity to get an output collision is $2^{2(c+3)/2}$. Meanwhile, the complexity in a random oracle is $2^{(n+3)/2}$. For sponge construction, the complexity is $2^{(c+3)/2}$. Hence, both sponge construction and random oracle require the same effort to generate output collisions. On the other hand, finding output collision in DLP-Sponge requires additional efforts.

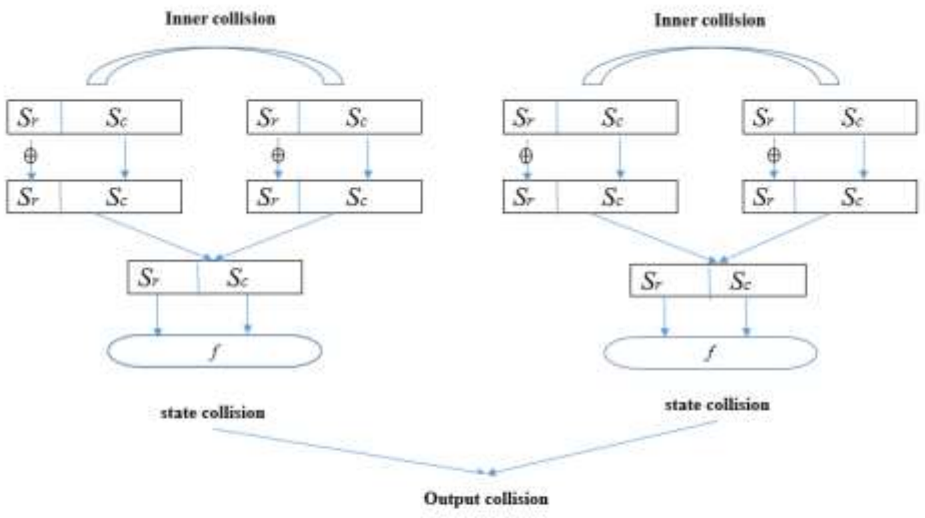

Figure 6. Output Collision in DLP-Sponge

5.2 Analysis against Multicollision Attacks

Joux showed that identifying ordinary collisions is no more difficult than producing one collision.

The multicollision attack on DLP-Sponge construction is not as easy as that on classical sponge. The concatenation of two compression function f and $f`$, $F = f(H)|| f`(H) = n(i + 1)$, $i > 2$, else the compression function will not work on the input, so. The complexity of birthday attack is 2^c instead of $2^{c/2}$.

The compression function f takes input as $f = \{0, 1\}^b \times \{0, 1\}^b \rightarrow \{0, 1\}^n$, assuming that f is resistant to collision. If we have inner collisions p, q and v, w, then we have output collisions $p|a|m$, $q|b|m$ and $v|a|m$, $w|b|m$, with complexity $2^{2(c+3)/2}$. The complexity of Joux multicollision attack is $t.2^{c/2}$ when $c \geq 2n$. In Joux attack [4], the attacker uses algorithm to find collision called machine C. It could be any collision finding attack like birthday or brute force attack in order to produce collisions for f with each call.

In order to launch Jouxs attack on DLP-Sponge construction, we use the birthday attack such as machine C. The attack works in two steps: Birthday Attack1 (BD1) and Birthday Attack2 (BD2). Firstly, Birthday Attack1 (BD1) is used for messages M_1, $M`_1$, $M_{11}$ and $M`_{11}$, and the initial value is $W_1 = 0$. This value is inputted into f and f_1 along with h_0. The outputs of BD1 are X_1, $X`_1$, $X_{11}$ and $X`_{11}$ where $f(h, X_1) = f_1(h, X`_1) = f(h, X_{11}) = f1(h, X`_{11}) = d_0$. Secondly, the values of X_1, $X`_1$, $X_{11}$ and $X`_{11}$ become 0 for the next iteration of f and f_1 and the outputs are W_2 and h_1. These outputs are the inputs for the next iteration, along with the generic Birthday Attack2 (BD2) for messages M_2, $M`_2$, $M_{22}$ and $M`_{22}$. The outputs of BD2 are X_2, $X`_2$, $X_{22}$ and $X`_{22}$ where $f(h, X_2) = f_1(h, X`_2) = f(h, X_{22}) = f1(h, X`_{22}) = d_1$. Figure7 shows the details of this attack.

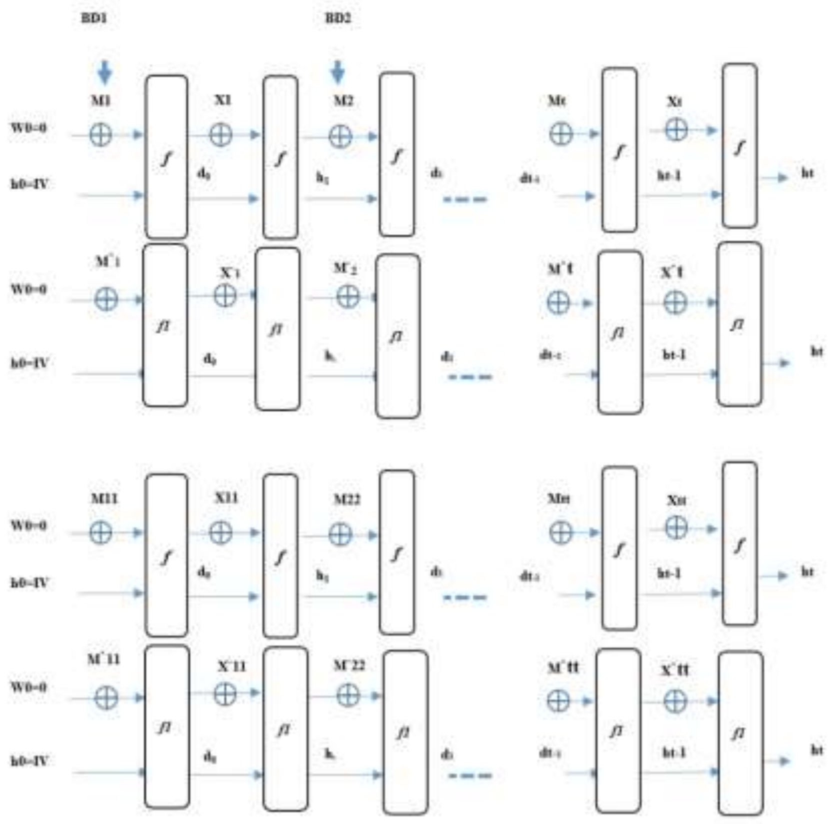

Figure 7. Multicollision Attack on DLP-Sponge

5.3. Analysis Against Second Pre-image Attack and Herding Attack

The second pre-image attack [6] and Herding attack [8] work on Merkle-Damg°ard construction. The sponge construction would be more resistant to these attacks by having a size of inner state which is at least twice of that of the output, i.e. $c \geq 2n$ in order to get a complexity of $2^{2(c+3)/2}$.

By using "expandable message-pattern", Kelsey and Schneier [6] launched a second pre-image attack on Merkle-Damg°ard. Their attack are different from Joux's attack because the cost of attack is no longer dependent on *t*.

Herding [8] attacked the Merkle-Damg°ard hash function and found that any attacker who can find multicollision [4] on the hash function is able to herd any prefix by choosing appropriate suffix. Herding attack focuses on the CTFP (chosen target forced prefix pre-image resistance) properties of hash function which is also known as "target value resistance". Based on this property, Merkle-Damg°ard and RO can be differentiated.

For a message *p* we search for the second pre-image of length ℓ. In DLP-Sponge, we have a second pre-image if we can find a second track to the inner state $T = S_{C,f}[p_0]$ (see section 4.4), where p_0 is the prefix of *p*. Given *q* as the second track, we have $S_f[q|x] = S_f[p]$ with $x = -S_{R,f}[q] + S_{R,f}[\mathrm{p}_0] + p`-1$.

Generally, sponge construction and random oracle have the same level of resistance against second pre-image if $c > n$, as the second pre-image includes an inner collision and the complexity of finding an inner collision is $2^{(c+3)/2}$. We obtain higher complexity than those obtained in [6] and [8], as the second pre-image includes inner collision. So, for small value of *M*, the complexity is ~ 2^c. In Random Oracle, the expected work to find the second pre-image is 2^n. In DLP-Sponge construction, the sizes of *n* and *c* are no longer a concern [10].

5.4 Analysis Against Pre-image Attack

In DLP-Sponge, we obtain the pre-image by linking an output string (see Section 4.5). We count $T = f^{-1}(S)$ and find a track p to TC. This detects a track to a given S by finding the track to T, viz $p|(-S_{R,f}[p] + T_R)$. The complexity of finding a pre-image for f is $2^{n-r} + 2^{2c}$ if $n < c$. Otherwise, there is no pre-image. If there is a pre-image, the complexity is $2^{c-1} + 2^{2c}$. The complexity to find a pre-image in random oracle is 2^n.

We need to link the output to T in order to count the state $T = f^{-1}(S)$. In order to estimate the inner part of the state which is identical to z_0, we need to estimate T by finding $f_R(T) = z_0$. Therefore, the number of paths increases by 2^{2r} and the complexity becomes 2^n for $n < b$ and 2^b for $n > b$. Meanwhile, the complexities are 2^n+2^{2c-1} for $n < b$ and $2^{2b}+2^{2c-1}$ for $n > b$.

6. Conclusion

In this paper, we have proposed a new construction of hash function named as DLP-Sponge. This new construction can be used as a keyed hash in authenticated encryption and MAC to solve the issues related to sponge security when $c = n$. The security analysis shows that DLP-Sponge is resistant against many sophisticated attacks such as multicollision attack and herding attack. We have proven that the DLP-Sponge construction is indifferentiable from RO. The security of DLP-Sponge depends on size of the internal state b especially on the capacity c not on the output size n. Our Indifferentiability bounds in terms of the capacity c permit to express up to which output length n such a hash function may offer the expected resistance.

References

[1] W. Diffie and M. Hellman, “New directions in cryptography”, IEEE transactions on Information Theory, vol. 22, no. 6, **(1976)**, pp.644-654.
[2] J.P. Aumasson, S. Knellwolf and W. Meier, “Heavy Quark for secure AEAD”, DIAC-Directions in Authenticated Ciphers, **(2012)**.
[3] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, “Permutation-based encryption, authentication and authenticated encryption”, Directions in Authenticated Ciphers, **(2012)**.
[4] A. Joux, “Multicollisions in iterated hash functions”, Application to cascaded constructions, Annual International Cryptology Conference, **(2004)**, pp. 306-316.
[5] M.A. Alahmad, I.F. Alshaikhli and M. Nandi, “Multicollisions in Sponge construction”, Informatics and Creative Multimedia (ICICM), **(2013)**, pp. 215-219.
[6] J. Kelsey and B. Schneier, “Second preimages on n-bit hash functions for much less than 2 n work”, Annual International Conference on the Theory and Applications of Cryptographic Techniques, , **(2005)**, pp. 474-490.
[7] J.P. Aumasson, “Faster multicollisions”, In International Conference on Cryptology in India, **(2008)**, pp. 67-77.
[8] J. Kelsey and T. Kohno, “Herding hash functions and the Nostradamus attack”, Annual International Conference on the Theory and Applications of Cryptographic Techniques, **(2006)**, pp. 183-200.
[9] D. Gligoroski, “Length extension attack on narrow-pipe SHA-3 candidates”, International Conference on ICT Innovations, **(2010)**, pp. 5-10.
[10] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, “Sponge functions”, ECRYPT hash workshop, **(2007)**.
[11] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, “Cryptographic sponges”, **(2011)**, Online http://sponge. noekeon. org.
[12] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, “Keccak sponge function family main document”, Submission to NIST (Round 2), vol. 3, **(2009)**, p. 30.
[13] J. Guo, T. Peyrin and A. Poschmann, “The PHOTON family of lightweight hash functions”, Annual Cryptology Conference, **(2011)**, pp. 222-239.
[14] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varıcı and I. Verbauwhede, **(2011)**, “SPONGENT: A lightweight hash function”, International Workshop on Cryptographic Hardware and Embedded Systems (pp. 312-325). Springer Berlin Heidelberg.
[15] J.P. Aumasson, L. Henzen, W. Meier and M. Naya-Plasencia, “Quark: A lightweight hash”, International Workshop on Cryptographic Hardware and Embedded Systems, **(2010)**, pp. 1-15.

[16] W. Wu, S. Wu, L. Zhang, J. Zou and L. Dong, "LHash: A Lightweight Hash Function (Full Version)", IACR Cryptology ePrint Archive, **(2013)**, p.867.
[17] M. Borowski, "The sponge construction as a source of secure cryptographic primitives", Military Communications and Information Systems Conference (MCC), **(2013)**, pp. 1-5.
[18] T. Yalçın and E.B. Kavun, "On the implementation aspects of sponge-based authenticated encryption for pervasive devices" International Conference on Smart Card Research and Advanced Applications, **(2012)**, pp. 141-157.
[19] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "On the indifferentiability of the sponge construction", Annual International Conference on the Theory and Applications of Cryptographic Techniques, **(2008)**, pp. 181-197.
[20] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "Duplexing the sponge: single-pass authenticated encryption and other applications", International Workshop on Selected Areas in Cryptography, **(2011)**, pp. 320-337.
[21] B. T. Hammad, N. Jamil, M. R. Zaba, M. E. Rusli and I. T. Ahmed, "Faster Multicollision attack on sponge construction", International Conference on Computer, Communication and Control Technology (I4CT), **(2016)**.
[22] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "On the security of the keyed sponge construction", SKEW, **(2011)**.
[23] S. Hirose, "Some plausible constructions of double-block-length hash functions", International Workshop on Fast Software Encryption, **(2006)**, pp. 210-225.
[24] S. Hirose, "Provably secure double-block-length hash functions in a black-box model", International Conference on Information Security and Cryptology, **(2004)**, pp. 330-342.
[25] M. Nandi, W. Lee, K. Sakurai and S. Lee, "Security analysis of a 2/3-rate double length compression function in the black-box model", International Workshop on Fast Software Encryption, **(2005)**, pp. 243-254.