SpoC: An Authenticated Cipher
Submission to the NIST LWC Competition

SUBMITTERS/DESIGNERS:
Riham AlTawy[†],[1] Guang Gong[†], Morgan He[†],
Ashwin Jha[◇], Kalikinkar Mandal[†],
Mridul Nandi[◇], and Raghvendra Rohit[†,⋆]

[*]Corresponding submitter:
Email: rsrohit@uwaterloo.ca
Tel: +1-519-888-4567 x45650

[†]COMMUNICATION SECURITY LAB
Department of Electrical and Computer Engineering
University of Waterloo
200 University Avenue West
Waterloo, ON, N2L 3G1, CANADA

[◇]Indian Statistical Institute
203 Barrackpore Trunk Road
Kolkata 700108
West Bengal, INDIA

https://uwaterloo.ca/communications-security-lab/lwc/spoc

September 26, 2019

[1]Currently with Department of Electrical and Computer Engineering, University of Victoria, 3800 Finnerty Rd, Victoria, BC, V8P 5C2, CANADA

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a nutshell, Sponge with masked Capacity, or SpoC (pronounced as *Spock*), is a permutation based mode of operation for authenticated encryption with associated data (henceforth "AEAD") functionality. The high level design is inspired by the Beetle [9, 10] mode of operation. It offers higher security guarantee with smaller states as compared to some of the previous AEAD designs based on the Sponge paradigm [7, 8, 11, 12]. In what follows, we briefly highlight the features of SpoC and then introduce the notations used throughout the document.

- Novelty. Capacity is masked with data blocks instead of rate which improves the security and allows larger rate value per permutation call.
- Underlying permutation. Adopts a lightweight permutation sLiSCP-light [4] which is efficient in both hardware and software because of bitwise and cyclic shift operations.
- Security. 128-bit security with state sizes 192 and 256, and rate values 64 and 128 bits, respectively.
- Control bits. A 4-bit control signal which distinguishes different phases and does not require an extra call of permutation in the case of empty and partial data blocks.
- Hardware footprint. Smaller instance has an area of 2329 GE in ASIC 65nm and achieves a throughput of 58.3 kbps for 1KB message, which fits the requirements of constrained devices.

1.1 Notations

In the following, n denotes a non-negative integer. We use $\{0,1\}^+$ and $\{0,1\}^n$ to denote the set of all non-empty (binary) strings, and n-bit strings, respectively. $\perp$ denotes the empty string and $\{0,1\}^* = \{0,1\}^+ \cup \{\perp\}$. For any string $B \in \{0,1\}^+$, $|B|$ denotes the number of bits in B and by $(B_0,\ldots,B_{\ell-1}) \overset{n}{\leftarrow} B$, we refer to the n-bit *block parsing* of B into $(B_0,\ldots,B_{\ell-1})$, where $|B_i| = n$ for $0 \le i \le \ell-2$, and $1 \le |B_{\ell-1}| \le n$. Moreover, $B_i[j]$ denotes the j-th byte of B_i (starting from left). For $A, B \in \{0,1\}^+$, and $|A| = |B|$, $A \oplus B$ (resp. $A \odot B$) denotes the "bitwise XOR" (resp. "bitwise AND") operation on A and B, and $A\|B$ denotes the "string concatenation" operation. For $x \in \{0,1\}^n$, $\mathsf{L}^{\mathsf{i}}(x)$ denotes the left cyclic shift operator, i.e., $\mathsf{L}^{\mathsf{i}}(x) = (x_i, x_{i+1},\ldots,x_{n-1},x_0,x_1,\ldots,x_{i-1})$.

We fix positive even integers b, t and n to denote the *state size*, *tag size* and *nonce size*, respectively in bits. $\mathsf{\Pi}$ denotes a permutation over $\{0,1\}^b$. We fix integers r and $c := b - r$ as the *rate* and *capacity* of the permutation $\mathsf{\Pi}$. With regard to rate and capacity, we use the following conventions:

- *Rate denotes the size of the amount of key stream bits (also called rate bits) generated per call of* $\mathsf{\Pi}$. By definition, capacity denotes the difference between block size and rate.
- For simplicity, we fix the r *least significant bits as the rate bits* and the remaining $c = b - r$ *most significant bits as the capacity bits.*

1.2 Outline

The rest of the document is organized as follows. In Chapter 2, we present the complete specification of SpoC along with its underlying permutation and recommended instances. We summarize the security claims of SpoC in Chapter 3 and provide the detailed security analyis in Chapter 4. In Chapter 5, we present the rationale of our design choices. Finally, we conclude by providing the details of hardware implementation in ASIC 65nm and 130nm, and performance results in Chapter 6.

Chapter 2

Specification

In this chapter, we present the specifications of SpoC along with its underlying permutation. We also give a detailed overall algorithmic description of the whole cipher and then list the recommended instances of SpoC.

2.1 SpoC Parameters

SpoC is primarily parameterized by the rate r of the underlying permutation, where $r \in \{64, 128\}$. We simply write SpoC-r to denote SpoC with the particular choice of r. The secondary parameters are set as follows.

- SpoC-64: In this version, $c = 128$, i.e., $b = 192$; $n = 128$; and $t = 64$.
- SpoC-128: In this version, $c = 128$, i.e., $b = 256$; $n = 128$; and $t = 128$.

In both versions, we set the key size κ same as the capacity, i.e., $\kappa = c$.

2.2 Description of SpoC

The SpoC's state can be viewed as an (c, r)-bit concatenated string $Y \| Z$ (sometimes, also viewed as (Y, Z)) made up of c-bit string Y (also called the Y-state) and r-bit string Z (also called the Z-state). In Algorithms 1 and 2, we present the complete algorithmic description of the mode, and Figure 2.1 illustrates the major components of the encryption/decryption process. We now give a high level description of the main modules (given in Algorithm 2) used in the encryption/decryption (described in Algorithm 1) process.

- init: The major task of this module is to create the initial state using the public nonce $N = N_0 || N_1$ and the secret key $K = K_0 || K_1$. We denote this state by $Y_0 || Z_0$ and is formally given as follows.

$$Y_0||Z_0 = \begin{cases} \Pi(\textsf{load-SpoC-64}(N_0, K)) \oplus (N_1||0^{b-r}) & \text{for SpoC-64,} \\ \textsf{load-SpoC-128}(N, K) & \text{for SpoC-128.} \end{cases}$$

  The function load-SpoC-r$(\cdot)$ depends on the choice of Π and assigns the nonce and key bytes to the particular byte positions of the state. We explicitily define this function in Section 2.6.

- proc_ad: This module is responsible for the associated data (AD) processing. During this phase, the control signal is set to 0010 to indicate a non-empty AD block. In case of partial block[1], the control signal changes to 0011. The entire AD processing step is given in "**function** proc_ad" of Algorithm 2.
- proc_pt: This module is responsible for the processing of plaintext (PT). During this phase, the control signal is set to 0100 to indicate a non-empty PT block. In case of partial block, the control signal changes to 0101. PT processing is similar to AD processing except for the fact that we squeeze out r-bit ciphertext and XOR the control signal (after extracting r-bit keystream). The entire PT processing step is given in "**function** proc_pt" of Algorithm 2.
- proc_ct: This module is responsible for ciphertext (CT) processing. It is symmetrical to proc_pt, and given in "**function** proc_ct" of Algorithm 2.
- proc_tg: This module is responsible for tag generation. At the tag generation call, the control signal is of the form $1xyz$, where the 3 least significant bits xyz depend on the previous modules. More details on the control signals is given below. We denote the process of extracting tag from state by tagextract-SpoC-r. More details are given in "**function** proc_tg" of Algorithm 2 and Section 2.7.

Algorithm 1 Encryption/Decryption algorithm in SpoC.

```
1: function SpoC-r_Π.enc($K, N, A, M$)
2:     $C \leftarrow \perp$
3:     $(Y_0, Z_0, a, m, \ell) \leftarrow$ init($K, N, A, M$)
4:     if $a \neq 0$ then
5:         $(X_a, Z_a) \leftarrow$ proc_ad($Y_0, Z_0, A$)
6:     if $m \neq 0$ then
7:         $(X_\ell, Z_\ell, C) \leftarrow$ proc_pt($X_a, Z_a, M$)
8:     $T \leftarrow$ proc_tg($X_\ell, Z_\ell$)
9:     return $(C, T)$
```

```
1: function SpoC-r_Π.dec($K, N, A, C, T$)
2:     $M \leftarrow \perp$
3:     is_auth $\leftarrow 0$
4:     $(Y_0, Z_0, a, m, \ell) \leftarrow$ init($K, N, A, C$)
5:     if $a \neq 0$ then
6:         $(X_a, Z_a) \leftarrow$ proc_ad($Y_0, Z_0, A$)
7:     if $m \neq 0$ then
8:         $(X_\ell, Z_\ell, M) \leftarrow$ proc_ct($X_a, Z_a, C$)
9:     $T' \leftarrow$ proc_tg($X_\ell, Z_\ell$)
10:    if $T' = T$ then
11:        is_auth $\leftarrow 1$
12:    else
13:        $M \leftarrow \perp$
14:    return (is_auth, $M$)
```

The Control Signal. Here we explain the 4-bit control signal that we use to separate the processing of various critical blocks. The control signal, denoted ctrl, can be viewed as a 4-bit string described below

$$\mathsf{ctrl} := \mathsf{ctrl}_{\mathsf{tag}}\,\mathsf{ctrl}_{\mathsf{pt}}\,\mathsf{ctrl}_{\mathsf{ad}}\,\mathsf{ctrl}_{\mathsf{par}}$$

Initially, all control bits are set to 0. The bits are set to 1 in the following manner:

1. $\mathsf{ctrl}_{\mathsf{ad}}$: The bit sets to 1 during the processing of associated data blocks. For empty AD it remains set to 0.
2. $\mathsf{ctrl}_{\mathsf{pt}}$: The bit sets to 1 during the processing of plaintext blocks. For empty messages it remains set to 0.
3. $\mathsf{ctrl}_{\mathsf{par}}$: The bit sets to 1 at the last AD (PT) block processing call if the last block is partial. For full last block it remains set to 0.

[1]This is only possible if the current block is the last block.

Algorithm 2 Main modules of SpoC.

```
function init($K, N, A, M$)
    if $r = c$ then
        $Y_0||Z_0 \leftarrow$ load-SpoC-128($N, K$)
    else
        $Y_0||Z_0 \leftarrow \Pi$(load-SpoC-64($N_0, K$)) $\oplus$ ($N_1||0^{b-r}$)
    $a \leftarrow \lceil |A|/r \rceil$
    $m \leftarrow \lceil |M|/r \rceil$
    $\ell \leftarrow a + m$
    return ($Y_0, Z_0, a, m, \ell$)

function proc_pt($Y_a, Z_a, M$)
    ($M_0, \ldots, M_{m-1}$) $\leftarrow$ parse($M$)
    for $j = 0$ to $m - 2$ do
        $k \leftarrow a + j$
        $X_{k+1}\|Z_{k+1} \leftarrow \Pi(Y_k\|Z_k)$
        $C_j \leftarrow Z_{k+1} \oplus M_j$
        $Y_{k+1} \leftarrow X_{k+1} \oplus$ opt_oz_pad($M_j$)
        $Z_{k+1} \leftarrow Z_{k+1} \oplus 0100||0^{r-4}$
    $X_\ell\|Z_\ell \leftarrow \Pi(Y_{\ell-1}\|Z_{\ell-1})$
    $C_{m-1} \leftarrow$ chop($Z_\ell, |M_{m-1}|$) $\oplus M_{m-1}$
    $C \leftarrow (C_0, \ldots, C_{m-1})$
    $Y_\ell \leftarrow X_\ell \oplus$ opt_oz_pad($M_{m-1}$)
    if $r \nmid |M_{m-1}|$ then
        $Z_\ell \leftarrow Z_\ell \oplus 0101||0^{r-4}$
    else
        $Z_\ell \leftarrow Z_\ell \oplus 0100||0^{r-4}$
    return ($Y_\ell, Z_\ell, C$)

function proc_tg($Y_\ell, Z_\ell$)
    $Z_\ell \leftarrow Z_\ell \oplus 1000||0^{r-4}$
    if $r = c$ then
        $T \leftarrow$ tagextract-SpoC-128($\Pi(Y_\ell\|Z_\ell), r$)
    else
        $T \leftarrow$ tagextract-SpoC-64($\Pi(Y_\ell\|Z_\ell), r$)
    return $T$

function parse($I$)
    $\ell = \lceil |I|/r \rceil$
    if $\ell = 0$ then
        return $\perp$
    else
        $(I_0, \ldots, I_{\ell-1}) \xleftarrow{r} I$
        return $(I_0, \ldots, I_{\ell-1})$
```

```
function proc_ad($Y_0, Z_0, A$)
    ($A_0, \ldots, A_{a-1}$) $\leftarrow$ parse($A$)
    for $i = 0$ to $a - 2$ do
        $X_{i+1}\|Z_{i+1} \leftarrow \Pi(Y_i\|Z_i)$
        $Y_{i+1} \leftarrow X_{i+1} \oplus$ opt_oz_pad($A_i$)
        $Z_{i+1} \leftarrow Z_{i+1} \oplus 0010||0^{r-4}$
    $X_a\|Z_a \leftarrow \Pi(Y_{a-1}\|Z_{a-1})$
    $Y_a \leftarrow X_a \oplus$ opt_oz_pad($A_{a-1}$)
    if $r \nmid |A_{a-1}|$ then
        $Z_a \leftarrow Z_a \oplus 0011||0^{r-4}$
    else
        $Z_a \leftarrow Z_a \oplus 0010||0^{r-4}$
    return ($Y_a, Z_a$)

function proc_ct($Y_a, Z_a, C$)
    ($C_0, \ldots, C_{m-1}$) $\leftarrow$ parse($C$)
    for $j = 0$ to $m - 2$ do
        $k \leftarrow a + j$
        $X_{k+1}\|Z_{k+1} \leftarrow \Pi(Y_k\|Z_k)$
        $M_j \leftarrow Z_{k+1} \oplus C_j$
        $Y_{k+1} \leftarrow X_{k+1} \oplus$ opt_oz_pad($M_j$)
        $Z_{k+1} \leftarrow Z_{k+1} \oplus 0100||0^{r-4}$
    $X_\ell\|Z_\ell \leftarrow \Pi(Y_{\ell-1}\|Z_{\ell-1})$
    $M_{m-1} \leftarrow$ chop($Z_\ell, |C_{m-1}|$) $\oplus C_{m-1}$
    $M \leftarrow (M_0, \ldots, M_{m-1})$
    $Y_\ell \leftarrow X_\ell \oplus$ opt_oz_pad($M_{m-1}$)
    if $n \nmid |C_{m-1}|$ then
        $Z_\ell \leftarrow Z_\ell \oplus 0101||0^{r-4}$
    else
        $Z_\ell \leftarrow Z_\ell \oplus 0100||0^{r-4}$
    return ($Y_\ell, Z_\ell, M$)

function chop($I, \ell$)
    if $\ell > r$ then
        return $\perp$
    else
        $(I_0, \ldots, I_{|I|}) \xleftarrow{1} I$
        return $I_0\| \cdots \|I_{\ell-1}$

function opt_oz_pad($I$)
    if $r \nmid |I|$ then
        $\xi = r - (|I| \bmod r)$
        $I \leftarrow I||1||0^{\xi-1}$
    return $I||0^{c-r}$
```

4. ctrl$_{\text{tag}}$: The bit sets to 1 at tag generation call.

We XOR the ctrl to the four most significant bits of the rate bits (the Z-state). In message processing phase, this is done after the extraction of keystream bits. Table 2.1 enumerates all possible values for the control signal along with their meanings.

Table 2.1: Possible values for the control signal along with their meanings.

ctrl	Meaning
0000	Implicitly used in nonce processing.
0010	Full AD block processing.
0011	Partial AD block processing.
0100	Full PT/CT block processing.
0101	Partial PT/CT block processing.
1000	Tag generation in empty AD and PT/CT case.
1010	Tag generation in non-empty AD with full last block and empty PT/CT case.
1011	Tag generation in non-empty AD with partial last block and empty PT/CT case.
1100	Tag generation in (non-)empty AD and non-empty PT/CT with full last block case.
1101	Tag generation in (non-)empty AD and non-empty PT/CT with partial last block case.

2.3 The sLiSCP-light Permutation

sLiSCP-light [4] is a family of iterated permutations based on the partial Substitution Permutation Network (SPN) construction. In this section, we describe the design of the sLiSCP-light family of permutations.

2.3.1 Step function of the permutation

An s-step sLiSCP-light permutation takes an input of b bits from $\{0,1\}^b$ and produces an output of b bits after applying the step function s times sequentially where $b = 8 \times m$ and $m \in \{24, 32\}$. We denote by sLiSCP-light-$[b]$ a b-bit sLiSCP-light permutation. A high-level overview of the step function of sLiSCP-light is depicted in Figure 2.2.

The state of the permutation is divided into four $2m$-bit subblocks $(S_0^i, S_1^i, S_2^i, S_3^i)$, where i denotes the step number and $0 \le i \le s-1$. In each step, the state is updated by a sequence of three transformations: `SubstituteSubblocks` (SSb), `AddStepconstants` (ASc), and `MixSubblocks` (MSb), thus the step function is defined as

$$(S_0^{i+1}, S_1^{i+1}, S_2^{i+}, S_3^{i+1}) \leftarrow \mathsf{MSb} \circ \mathsf{ASc} \circ \mathsf{SSb}(S_0^i, S_1^i, S_2^i, S_3^i).$$

We now describe each transformation in detail.

`SubstituteSubblocks` (SSb)

This is a partial substitution layer of the SPN structure where the nonlinear operation is applied to the half of the state. It applies the u-round iterated unkeyed Simeck-$2m$ block cipher [17] (henceforth referred to as Simeck sbox or $\mathtt{SB}_u^{2m}$) to the odd indexed subblocks only. The SSb transformation is defined as

$$\mathsf{SSb}(S_0^i, S_1^i, S_2^i, S_3^i) = (S_0^i, \mathtt{SB}_u^{2m}(S_1^i), S_2^i, \mathtt{SB}_u^{2m}(S_3^i)).$$

Below we provide the details of Simeck sbox $\mathtt{SB}_u^{2m}$.

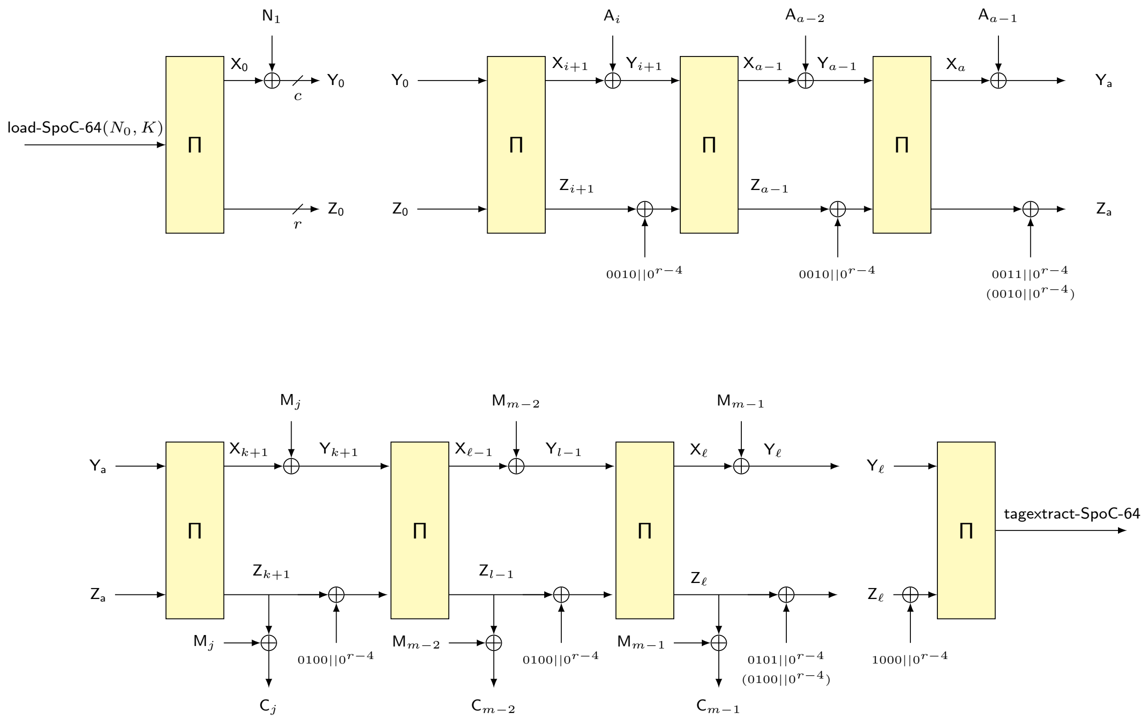

Figure 2.1: Schematic diagram of different modules used in the encryption algorithm of SpoC-64 for non empty AD and PT. From left to right and top to bottom, we have the following modules: init, proc_ad, proc_pt, and proc_tg. SpoC-128 is identical to SpoC-64 except for the init module. In which case we define $Y_0\|Z_0$ as load-SpoC-128(N, K). Moreover, we apply tagextract-SpoC-128 to generate the tag. See Algorithms 1 and 2 for more details.

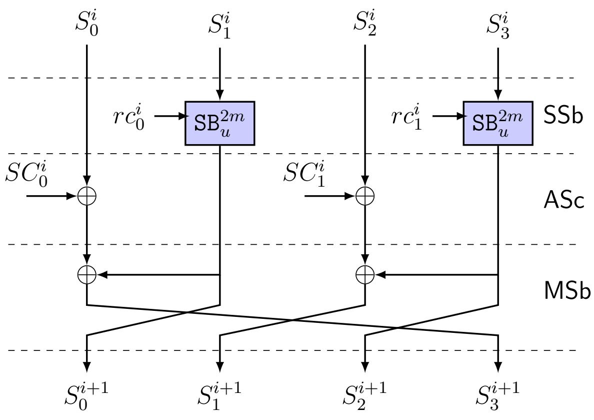

Figure 2.2: Step function of sLiSCP-light permutation

Definition 1 (Simeck sbox SB_u^{2m} [3]) *Let $u > 0$ and $rc = (q_{u-1}, \ldots, q_0)$ where $q_j \in \{0, 1\}$ and $0 \le j \le u - 1$. A Simeck sbox is a permutation of $2m$-bit input constructed by iterating the Simeck-2m block cipher for u rounds with round constant addition $\gamma_j = 1^{m-1}||q_j$ in place of key addition.*

An illustrated description of the Simeck sbox is shown in Figure 2.3 and is given by:

$$(x_{u+1}||x_u) \leftarrow \mathsf{SB}_u^{2m}(x_1||x_0, rc)$$

where

$$x_j \leftarrow f_{(5,0,1)}(x_{j-1}) \oplus x_{j-2} \oplus \gamma_{j-2},\ 2 \le j \le u + 1 \text{ and}$$

$f_{(5,0,1)} : \{0, 1\}^m \to \{0, 1\}^m$ given by $f_{(5,0,1)}(x) = (\mathsf{L}^5(x) \odot x) \oplus \mathsf{L}^1(x)$.

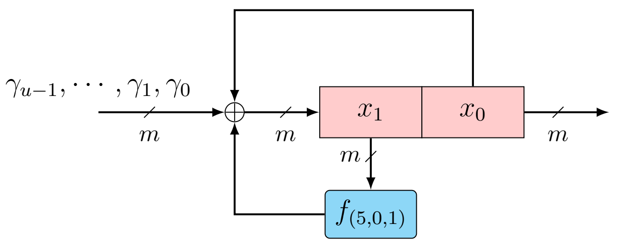

Figure 2.3: Simeck sbox SB_u^{2m}

`AddStepconstants` (ASc)

In this layer, the step constants SC_0^i and SC_1^i are XORed with the two even indexed subblocks S_0^i and S_2^i, respectively, $i = 0, 1, \ldots s-1$. Each SC_j^i is an $2m$-bit constant of the form $1^{2m-8}||0^2||sc_j^i$ (resp. $1^{2m-8}||sc_j^i$) for $(u, m) = (6, 24)$ (resp. $(u, m) = (8, 32)$), where sc_j^i is 6 (resp. 8)-bit constant generated by an LFSR. The ASc transformation is given by

$$\mathsf{ASc}(S_0^i, \mathsf{SB}_u^{2m}(S_1^i), S_2^i, \mathsf{SB}_u^{2m}(S_3^i)) = (S_0^i \oplus SC_0^i, \mathsf{SB}_u^{2m}(S_1^i), S_2^i \oplus SC_1^i, \mathsf{SB}_u^{2m}(S_3^i)).$$

`MixSubblocks` (MSb)

This layer applies the linear transformation that is used in the Type-2 GFS [15] to the subblocks of the state. More precisely, each even indexed subblock is replaced by the XOR of its initial value with its neighboring odd indexed subblock. Then a subblock cyclic left shift is applied. The MSb transformation is given by

$$(S_0^{i+1}, S_1^{i+1}, S_2^{i+1}, S_3^{i+1}) \leftarrow \mathsf{MSb}(S_0^i \oplus SC_0^i, \mathsf{SB}_u^{2m}(S_1^i), S_2^i \oplus SC_1^i, \mathsf{SB}_u^{2m}(S_3^i)),$$

where

$$\begin{aligned} S_0^{i+1} &= \mathsf{SB}_u^{2m}(S_1^i), & S_1^{i+1} &= S_2^i \oplus \mathsf{SB}_u^{2m}(S_3^i) \oplus SC_1^i, \\ S_2^{i+1} &= \mathsf{SB}_u^{2m}(S_3^i), & S_3^{i+1} &= S_0^i \oplus \mathsf{SB}_u^{2m}(X_1^i) \oplus SC_0^i. \end{aligned}$$

2.3.2 sLiSCP-light permutation instances

sLiSCP-light offers two lightweight instances, named sLiSCP-light-[192] and sLiSCP-light-[256], with state sizes 192 and 256 bits, respectively. Table 2.2 presents the recommended parameters for two lightweight instances of the sLiSCP-light permutation.

Table 2.2: Recommended parameter set for sLiSCP-light[192] and sLiSCP-light-[256] permutations.

Permutation	m	Rounds u	Steps s	Total # rounds $(u \cdot s)$
sLiSCP-light-[192]	24	6	18	108
sLiSCP-light-[256]	32	8	18	144

2.3.3 sLiSCP-light constants

As depicted in Figure 2.2, the step funtion of sLiSCP-light is parametrized by two sets of constants (rc_0^i, rc_1^i) and (sc_0^i, sc_1^i). We call them round and step constants, respectively. The round constants are used within the Simeck sboxes while step constants are XORed to the even subblocks as described earlier. In Table 2.3 and 2.4, we list the hexadecimal values of constants for both the instances of sLiSCP-light. More details on how to generate these constants can be found in [3, 4, 5].

Table 2.3: Round and step constants for sLiSCP-light-[192]

step i	(rc_0^i, rc_1^i)	(sc_0^i, sc_1^i)
0 - 5	(7, 27), (4, 34), (6, 2e), (25, 19), (17, 35), (1c, f)	(8, 29), (c, 1d), (a, 33), (2f, 2a), (38, 1f), (24, 10)
6 - 11	(12, 8), (3b, c), (26, a), (15, 2f), (3f, 38), (20, 24)	(36, 18), (d, 14), (2b, 1e), (3e, 31), (1, 9), (21, 2d)
12 - 17	(30, 36), (28, d), (3c, 2b), (22, 3e), (13, 1), (1a, 21)	(11, 1b), (39, 16), (5, 3d), (27, 3), (34, 2), (2e, 23)

Table 2.4: Round and step constants for sLiSCP-light-[256]

step i	(rc_0^i, rc_1^i)	(sc_0^i, sc_1^i)
0 - 5	(f, 47), (4, b2), (43, b5), (f1, 37), (44, 96), (73, ee)	(8, 64), (86, 6b), (e2, 6f), (89, 2c), (e6, dd), (ca, 99)
6 - 11	(e5, 4c), (b, f5), (47, 7), (b2, 82), (b5, a1), (37, 78)	(17, ea), (8e, 0f), (64, 04), (6b, 43), (6f, f1), (2c, 44)
12 - 17	(96, a2), (ee, b9), (4c, f2), (f5, 85), (7, 23), (82, d9)	(dd, 73), (99, e5), (ea, 0b), (0f, 47), (04, b2), (43, b5)

2.4 Recommended Instantiations

We instantiate SpoC with sLiSCP-light permutation to provide two lightweight AEAD instances which offer 128-bit security. Table 2.5 presents the recommended parameter sets for two lightweight instances of SpoC. The list is sorted in priority, i.e., SpoC-64_sLiSCP-light-[192] is the primary recommendation and SpoC-128_sLiSCP-light-[256] is the secondary one.

Table 2.5: Recommended parameter sets of SpoC

Instance	b	r	κ	n	t	Data (in bytes)
SpoC-64_sLiSCP-light-[192]	192	64	128	128	64	2^{50}
SpoC-128_sLiSCP-light-[256]	256	128	128	128	128	2^{50}

2.5 Positions of Rate and Capacity

In this section, we illustrate the exact positions of the state which are used for r-bit keystream and for masking r-bits of capacity. We view each subblock of sLiSCP-light state as a sequence of bytes, i.e., $S_i = S_i[0]||\cdots||S_i[j]$ where $0 \leq i \leq 3$ and $j = 5$ (resp. 7) for sLiSCP-light-[192] (resp. sLiSCP-light-[256]). Similary, we write Y and Z. The rate and masked capacity byte positions of SpoC-64_sLiSCP-light-[192] are given by:

$$\begin{aligned} \textbf{rate: } & S_0[0], S_0[1], S_0[2], S_0[3], S_2[0], S_2[1], S_2[2], S_2[3] \\ \textbf{masked capacity: } & S_1[0], S_1[1], S_1[2], S_1[3], S_3[0], S_3[1], S_3[2], S_3[3]. \end{aligned}$$

For SpoC-128_sLiSCP-light-[256], even (resp. odd) indexed sub-blocks constitute the rate (resp. masked capacity) part of the state. Figure 2.4 depicts the above positions for both instances of SpoC and 1-1 correspondence between SpoC and sLiSCP-light state.

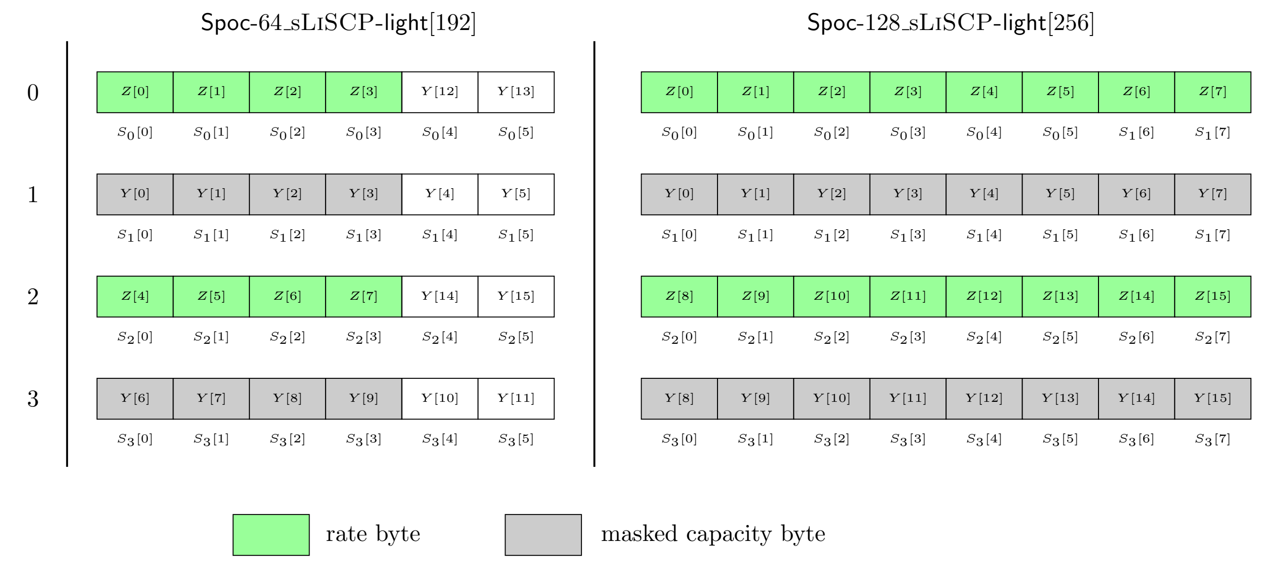
Spoc-64_sLiSCP-light[192]
Spoc-128_sLiSCP-light[256]
0
1
2
3
Z[0] Z[1] Z[2] Z[3] Y[12] Y[13]
S0[0] S0[1] S0[2] S0[3] S0[4] S0[5]
Y[0] Y[1] Y[2] Y[3] Y[4] Y[5]
S1[0] S1[1] S1[2] S1[3] S1[4] S1[5]
Z[4] Z[5] Z[6] Z[7] Y[14] Y[15]
S2[0] S2[1] S2[2] S2[3] S2[4] S2[5]
Y[6] Y[7] Y[8] Y[9] Y[10] Y[11]
S3[0] S3[1] S3[2] S3[3] S3[4] S3[5]
Z[0] Z[1] Z[2] Z[3] Z[4] Z[5] Z[6] Z[7]
S0[0] S0[1] S0[2] S0[3] S0[4] S0[5] S1[6] S1[7]
Y[0] Y[1] Y[2] Y[3] Y[4] Y[5] Y[6] Y[7]
S1[0] S1[1] S1[2] S1[3] S1[4] S1[5] S1[6] S1[7]
Z[8] Z[9] Z[10] Z[11] Z[12] Z[13] Z[14] Z[15]
S2[0] S2[1] S2[2] S2[3] S2[4] S2[5] S2[6] S2[7]
Y[8] Y[9] Y[10] Y[11] Y[12] Y[13] Y[14] Y[15]
S3[0] S3[1] S3[2] S3[3] S3[4] S3[5] S3[6] S3[7]
rate byte
masked capacity byte

Figure 2.4: Rate and capacity part of SpoC

2.6 Loading Key and Nonce

In this section, we describe the postions where the 128-bit key $K = K_0||K_1$ and 128-bit nonce $N = N_0||N_1$ are loaded in the state. In particular, we define the functions load-SpoC-128(N, K) and load-SpoC-64(N_0, K).

For SpoC-128_sLiSCP-light-[256] we load the key and nonce in odd and even subblocks, respectively. Formally, on calling load-SpoC-128(N, K), the state is loaded as follows.

$$S_1[j] \leftarrow K_0[j]; S_3[j] \leftarrow K_1[j]; S_0[j] \leftarrow N_0[j] \text{ and } S_2[j] \leftarrow N_1[j], \text{ for } 0 \leq j \leq 7.$$

The load-SpoC-64(N_0, K) function for SpoC-64_sLiSCP-light-[128] is given by:

$$\begin{aligned}
S_1[0], \cdots, S_1[5] &\leftarrow K_0[0], \cdots, K_0[5] \\
S_3[0], \cdots, S_3[5] &\leftarrow K_1[0] \cdots, K_1[5] \\
S_0[0] \cdots, S_0[3] &\leftarrow N_0[0], \cdots, N_0[3] \\
S_2[0], \cdots, S_2[3] &\leftarrow N_0[4], \cdots, N_0[7] \\
S_0[4], S_0[5] &\leftarrow K_0[6], K_0[7] \\
S_2[4], S_2[5] &\leftarrow K_1[6], K_1[7].
\end{aligned}$$

2.7 Tag Generation

In this section, we show the procedure to compute the tag for SpoC-64_sLiSCP-light-[192] and SpoC-128_sLiSCP-light-[256]. We denote this process by tagextract-SpoC-r for $r \in \{64, 128\}$.

For SpoC-128_sLiSCP-light-[256], the tagextract-SpoC-128 function computes the 128-bit tag $T = T_0||T_1$ which is given by $T_0 \leftarrow S_1$ and $T_1 \leftarrow S_3$. Similarly, tagextract-SpoC-64 computes the 64-bit tag T of SpoC-64_sLiSCP-light-[192] as follows.

$$\begin{aligned}
T[0], \cdots, T[3] &\leftarrow S_1[0], \cdots, S_1[3] \\
T[4], \cdots, T[7] &\leftarrow S_3[0], \cdots, S_3[3].
\end{aligned}$$

Chapter 3

Security Claims

In Table 3.1, we list the security levels of two instances of SpoC. We assume a nonce-respecting adversary, i.e., for a fixed key, the public nonce value is never repeated for an encryption query. Moreover, the numbers are based on our security analysis of sLiSCP-light permutation which is modeled as a random permutation for 18 steps. Accordingly, we do not claim security for SpoC with round-reduced sLiSCP-light permutation.

Table 3.1: Security levels of the two AEAD algorithms based on SpoC using sLiSCP-light permutation. The two AEAD algorithms are secure while the prescribed data and time limit are respected.

AEAD algorithm	**Confidentiality**		**Integrity**	
	Time	Data (in bytes)	Time	Data (in bytes)
SpoC-64_sLiSCP-light[192]	2^{112}	2^{50}	2^{112}	2^{50}
SpoC-128_sLiSCP-light[256]	2^{112}	2^{50}	2^{112}	2^{50}

Chapter 4

Security Analysis

4.1 Security of SpoC

In this section, we present the security analysis of SpoC against generic attacks (assuming the underlying permutation is ideal, i.e., random permutation). First, we briefly explain possible attack strategies along with a rough lower bound estimate on the amount of data and time complexity required for each attack. Then, we substitute concrete parameters and compute the actual advantage to validate the recommended criteria listed in Table 3.1.
In the following discussion:

- D denotes the data complexity (or online queries). This parameter quantifies the online resource requirements, and includes the total number of blocks (among all messages and associated data) processed through the underlying permutation for a fixed key. For simplicity we also use D to account for the data complexity of forging attempts.

- T denotes the time complexity (or offline queries). This parameter quantifies the offline resource requirements, and includes the total time required to process the offline evaluations of the underlying permutation. Since one call of the permutation can be assumed to take a constant amount of time, we generally take T as the total number of offline calls to the permutation.

4.1.1 Key or internal state recovery

Key recovery. The adversary can try to guess the key using offline permutation queries. Once the key is known, the adversary can certainly distinguish, forge valid ciphertexts, or worse recover the plaintext. But, since the key is chosen uniformly in both SpoC-64 and SpoC-128, this strategy would require exhaustive search over the key space, i.e., $T \approx 2^c$ many offline queries, and a constant number of online queries, i.e., $D = O(1)$, to filter out the correct key.

State recovery. The adversary can try to guess the internal state for some encrypted block using a combination of offline and online queries. If the adversary guesses the state correctly then it can forge valid ciphertexts for the nonce value used in this encrypted block. In fact, given one internal state, the adversary can even recover the key. We show here that the internal state recovery is much harder than standalone key recovery. Recall that we denote the internal state by the tuple (Y, Z). Note that guessing just one of Y or Z is not enough as the other value is random. Further, guessing both Y and Z requires the product of data and time, $DT \approx 2^b$. This can be argued using list matching attack, i.e., the adversary creates a list $\mathcal{L}_T$ of T offline query-response tuples and a list $\mathcal{L}_D$ of D online query-response tuples (with one block PT). A matching between $\mathcal{L}_T$ and $\mathcal{L}_D$ happens with approx. $DT2^{-b}$ probability. So, to get a non negligible advantage, one must satisfy $DT \approx 2^b$.

4.1.2 Privacy of SpoC

In privacy attacks the adversary is concerned with distinguishing the SpoC mode with an ideal authenticated encryption scheme. In addition to access of the encryption algorithm, the adversary is also allowed offline evaluations of the underlying permutation. A trivial attack strategy is guessing the key or internal state (as discussed in Section 4.1.1). Non-trivially, the adversary can distinguish the mode from ideal if there is no randomness in some ciphertext (or tag) blocks. This is possible in the following two ways:

1. **Online-online block matching.** For a pair of distinct online (in this case encyrption) query blocks, the internal state matches. Then, the block that appears later will have non-random behavior. Note that this matching is only accidental and will happen with probability approx. 2^{-b} in both SpoC-64 and SpoC-128. Thus, it requires $D^2 \approx 2^b$.

2. **Online-offline block matching.** This is similar to the state recovery attack strategy of Section 4.1.1. Again this matching will happen accidentally with probability approx. 2^{-b}, which gives $DT \approx 2^b$.

4.1.3 Integrity of SpoC

Integrity violation means that the adversary can forge a new and valid (passes verification) ciphertext and tag pair. The adversary is allowed to make encryption queries to the encryption algorithm and forging queries to the decryption algorithm.

In a forgery attack, the adversary can apply previous strategies of key or state recovery as in Section 4.1.1. Other attack strategies are described below:

1. **Tag guessing.** This is a more direct attack, where the adversary makes arbitrary decryption attempts in the hope that the tag matches. This is equivalent to guessing r-bit output of a random permutation, which holds with approx. 2^{-r} probability for each query. So tag guessing attack would require $D \approx 2^r$.

2. **Decryption query matching with online chain.** This corresponds to the attack strategy where the adversary tries to match full state of some decryption query block to some previous encryption query block. This may lead to a forgery in the following manner:
   Suppose the adversary gets the response $(c'_1, c'_2, c'_3, c'_4, c'_5, t')$ for some encryption query, and then tries a decryption query of the form $(c_1, c_2, c_3, c'_4, c'_5, t')$ (with a different nonce). In this case the adversary forges with certainty, if the next internal state corresponding to the ciphertext block c_3 of decryption query matches with the one corresponding to c'_3 of encryption query. One can show that the probability of such event for all decryption query blocks is approx. $rD^2/2^b + D/2^{2r}$, which gives $D \approx \min(\sqrt{2^{b-\log_2 r}}, 2^{2r})$.

3. **Decryption query matching with offline chain.** This corresponds to the attack strategy where the adversary tries to create a valid forgery using offline queries. This would mean that the adversary constructed a chain of internal states (for some fixed ciphertext blocks followed by a fixed tag value) using offline queries and then matched some decryption query to such a chain. The chain can be constructed by making permutation queries in one of the two ways:

   - **Using forward only or backward only queries.** This corresponds to the event where the chain is created using forward only or backward only queries. In this case, the probability of successful forgery can be bounded by approx. $rDT/2^b + T/2^c$;

- **Using both forward and backward queries.** This corresponds to meet-in-the-middle kind of attack where the chain is created using both forward and backward calls to the permutation. In this case, the probability of successful forgery can be bounded by approx. $cDT^2/2^{b+c} + T^2/2^{2c}$;

On combining the two cases, we get $DT \approx \min(2^{b-\log_2 r}, 2^{b-\log_2 c})$ and $T \approx 2^c$.

4.1.4 Validation of security claims

The security claims given in Table 3.1 follow from the rough lower bounds on D, T, and DT, as discussed in Sections 4.1.1-4.1.3. Specifically, one can observe that SpoC-64 and SpoC-128 are secure as long as $D < 2^{50}$ bytes and $T < 2^{112}$.

4.2 Security of sLiSCP-light

In what follows, we present the results of our cryptanalysis of the two instances of sLiSCP-light permutation. Since the permutation is used in SpoC, we aim to provide evidence of how it is secure against various distinguishing attacks in an attempt to prove that its behavior is as close as possible to that of an ideal permutation. Our analysis focuses on providing results related to the diffusion, differential and linear, algebraic degree, self-symmetry properties of the permutation.

4.2.1 Diffusion

We investigate the following two properties to asses the diffusion behavior of sLiSCP-light. For the details and results of the adopted methodologies, the reader is referred to [4].

1. **Permutation full bit diffusion.** We evaluate the minimum number of steps required such that each bit in the state depends on all the input state bits. We find that using 6 (resp. 8) rounds of Simeck in sLiSCP-light-[192] (resp. sLiSCP-light-[256]), full bit diffusion is achieved after four steps.

2. **Avalanche effect.** We use a uniform random sampling method to evaluate the average number of flipped bits after four steps corresponding to flipping one bit in the input state. More precisely, for each bit position in the input state, we generate 1024 random input states and flip this bit once and count the number of changed bits in the output state. Then we compute the average number of changes per bit over these 1024 random samples. We found that the average numbers of flipped bits after 4 steps corresponding to flipping the individual 192 (resp. 256) bit positions for sLiSCP-light-[192] (resp. sLiSCP-light-[256]) spans between 95.13 and 96.56 (resp. 126.98 and 128.90).

Based on the above results, we claim that meet/miss-in-the middle distinguishers may not cover more than eight steps because eight steps guarantee full bit diffusion in both the forward and backward directions.

4.2.2 Differential and linear cryptanalysis

In order to evaluate the differential and linear behavior of sLiSCP-light, we firstly give our results of analyzing the differential and linear properties of the Simeck sboxes. Such results are generated using the SAT/SMT tools proposed in [13] coupled with an optimized differentials and linear masks exhaustive search. Secondly, we develop a MILP model for the sLiSCP-light permutation to bound the minimum number of differentially and linearly active Simeck sboxes in order to evaluate the expected maximum probabilities of differential and linear characteristics.

Differential analysis of Simeck sbox. Generally, one may derive estimates for the expected Maximum Differential Probabilities (MDP) and maximum linear squared correlation (MLSC) of Simeck sboxes by adopting the Markov assumption, hence ignoring the effect of the constants similar to keyed ciphers (cf. Sec. 5.1 in [3]). Tighter estimates for the MDP of the constant-based Simeck sboxes can be obtained by considering the differential effect along with an exhaustive search for the exact probabilities associated with the expected optimal differentials as has been shown in [4].

Expected maximum probabilities of differential and linear characteristics. According to our diffusion and ideal permutation criteria, we wanted to find the optimal number of rounds, u, for $\mathtt{SB}_u^{48}$ and $\mathtt{SB}_u^{64}$ sboxes so that we achieve the expected ideal differential and linear behavior in the minimum number of steps, s. Additionally, we constrained the lower bound on s such that we run the permutation for at least three times the number of steps required for full bit diffusion. Such analysis has been carried out simultaneously with bounding the minimum number of active Simeck sboxes in order to to evaluate the trade off between the number of Simeck rounds, u, and permutation steps, s. More formally, for a Simeck sbox that is iterated for u rounds, let δ and γ denote the $\log_2$ scaled MDP and MLSC, respectively. Given an s-step iterated b-state sLiSCP-light permutation, let m_s be the minimum number of active Simeck sboxes and d_s denote the number of steps required to achieve full bit diffusion. Then we require the following three conditions to hold:

1. The maximum expected differential characteristic probability (MEDCP) and the maximum expected linear characteristic squared correlation (MELCSC) to be upper bounded by 2^{-b} and $2^{-b/2}$, respectively. Formally, $\delta m_s \leq -b$ and $\gamma m_s \leq -b/2$.

2. The total number of steps to be lower bounded by three times the number of steps required for full bit diffusion. Formally, $s \geq d_s$.

3. The total number of rounds, $u \times s$ in the permutation is minimized as this directly translates to better performance.

In [4], the trade-offs between u and s are considered and it has been found that for $\mathtt{SB}_u^{48}$ (resp. $\mathtt{SB}_u^{64}$), $u = 6$ (resp. $u = 8$), $\delta = -10.7$ (resp. $\delta = -15.9$), $\gamma = -10.8$ (resp. $\gamma = -15.6$) the above three conditions are optimally satisfied when $s = 18$ and accordingly $m_s = 18$. In other words, the expected bounds on the MEDCP and MELCSC are as follows:

$$\text{sLiSCP-light-[192]:}\quad \begin{array}{lll} (\mathrm{MDP}(\mathtt{SB}_6^{48}))^{18} & = (2^{-10.7})^{18} & = 2^{-192.6} \\ (\mathrm{MLSC}(\mathtt{SB}_6^{48}))^{18} & = (2^{-10.8})^{18} & = 2^{-194.4} \end{array}$$

$$\text{sLiSCP-light-[256]:}\quad \begin{array}{lll} (\mathrm{MDP}(\mathtt{SB}_8^{64}))^{18} & = (2^{-15.9})^{18} & = 2^{-286.2} \\ (\mathrm{MLSC}(\mathtt{SB}_8^{64}))^{18} & = (2^{-15.6})^{18} & = 2^{-280.8}. \end{array}$$

4.2.3 Algebraic distinguishers

The algebraic degree of sLiSCP-light can be upper bounded using a tweaked version of the division property that is employed to find the degree of an s-step sLiSCP-light permutation [16, 6]. In [4], it is found that the algebraic degree of $\mathtt{SB}_6^{48}$ (resp. $\mathtt{SB}_8^{64}$) is 19 (resp. 36) which results in an upper bound on the algebraic degree of all component functions of sLiSCP-light-[192] (resp. sLiSCP-light-[256]) of 189 (resp. 247) after only 5 (resp. 4) steps. Such numbers suggest that maximum degree for sLiSCP-light-[192] (resp. sLiSCP-light-[256]) maybe reached after

7 (resp. 6) steps of the permutation. In what follows, we give the results of our integral and zero-sum distingushers.

Integral distinguishers. According to the cryptanalysis presented in [4], we report that for both instances of sLiSCP-light, there exists an 8-step integral distinguishers that can be found with data and time complexities of 2^{b-1}.

Zero-sum distinguishers. It is found that the maximum number of steps covered by zero-sum distinguishers in one direction is at most 7 [4]. Thus following a start from the middle approach, a 14-step zero-sum distinguisher exists for sLiSCP-light-[192] (resp. sLiSCP-light-[256]). Such a distinguisher require data and time complexities equal to that of the exhaustive search.

4.2.4 Rotational, slide and invariant subspace distinguishers

A cryptographic permutation where the internal steps can not be distinguished can exhibit undesired self-symmetry properties. To thwart such properties in sLiSCP-light-[192] (resp. sLiSCP-light-[256]), we employ a 6-bit (resp. 7-bit) LFSR to generate a tuple of two round constants (rc_0^i, rc_1^i), and a tuple of two step constants, (sc_0^i, sc_1^i) (see Chapter 2 for details). In order to mitigate rotational, slide, and invariant subspace distingusihers, we ensure that the following conditions hold:

- For $0 \leq i \leq 17$, $sc_0^i \neq sc_1^i$
- For $0 \leq i \leq 17$, $(rc_0^i, rc_1^i) \neq (sc_0^i, sc_1^i)$
- For $0 \leq i, j \leq 17$ and $i \neq j$, $(rc_0^i, rc_1^i) \neq (rc_0^j, rc_1^j)$
- For $0 \leq i, j \leq 17$ and $i \neq j$, $(sc_0^i, sc_1^i) \neq (sc_0^j, sc_1^j)$.

Chapter 5

Design Rationale

5.1 Novelty of SpoC

The design of SpoC is inspired by the Beetle mode of operation which offers better security as compared to the traditional Sponge duplex mode. Our goal is to achieve better security and smaller state size than both Sponge duplex and Beetle, while keeping the overheads as low as possible. We achieve this goal by simply masking the capacity of the permutation state with the input AD/PT block. This is quite different from Sponge duplex and Beetle, where instead of capacity, rate is masked by input. In the decryption phase of Sponge duplex the input rate bits of any permutation call is completely controlled by the adversary, as the input rate bits are same as the ciphertext bits, which leads to a loss in security. SpoC avoids such security loss by masking the capacity instead of the rate bits.

In summary, the capacity masking helps SpoC in achieving significantly higher security for a smaller state size, as compared to both Sponge duplex and Beetle (see Section 4.1). For instance among the three modes, only SpoC achieves significant security up to data complexity of $\approx 2^{60}$ blocks and time complexity of $\approx 2^{120}$, when instantiated with a 192-bit permutation. SpoC also has some advantages over Beetle's internal state update function in terms of XOR counts per block and some minor shift operations.

5.2 Choice of the Permutation: sLiSCP-light

Our goal is to choose a permutation which, to the best of our knowledge, offers the lowest hardware footprint for SpoC. Although, SpoC has higher security than traditional Sponge duplex and Beetle mode of operation, it still requires an extra r-bit XORs and r-bit MUXs for the Z part of state. Thus, to have an overall lightweight AEAD scheme, permutation alone should be lightweight.

Our choice of Π, i.e., $\Pi :=$ sLiSCP-light-$[b]$ for $b \in \{192, 256\}$ has the lowest hardware footprint among all other permutations of similar state sizes [4]. It adopts two of the well analyzed cryptographic primitives in literature, namely (tweaked) Type II Generalized Feistel Structure and round-reduced unkeyed Simeck block cipher as its components. In addition, it has a simple security analysis and offers good bounds against the generic distinguishers (see Section 4.2).

5.3 Choice of Rate and Capacity Positions

The choice of rate and capacity positions for SpoC depends on the underlying permutation. Since, we instantiate SpoC with sLiSCP-light permutation, we have followed a similar strategy in choosing the rate and capacity positions as the one that has been used in sLiSCP [3]. The data block to be processed is absorbed in the odd subblocks and keystream is taken from the even indexed subblocks (see Section 2.5 for exact byte positions). Such capacity positions allow the input bits to be processed by the Simeck sboxes as soon as possible so we achieve faster diffusion. Also, our choice forces any injected difference to activate Simeck sboxes in the first step which also enhances sLiSCP-light's resistance to differential and linear cryptanalysis. This observation has also been confirmed by a third party cryptanalysis of sLiSCP [14].

5.4 Statement

The authors declare that there are no hidden weaknesses in SpoC-64_sLiSCP-light-[192] and SpoC-128_sLiSCP-light-[256].

Chapter 6

Hardware Implementation

In this chapter, we provide the details of our ASIC implementation of sLiSCP-light-192, sLiSCP-light-256 permutations, SpoC-64 and SpoC-128. The reported implementations are in ASIC $65nm$ and $130nm$ technologies.

6.1 ASIC Implementation

SpoC is highly hardware optimized and has a very efficient ASIC implementations particularly because its core permutation sLiSCP-light employs partial layers. More precisely, the SB_u^{2m} sboxes, step constant addition, and linear mixing are all applied on half of the state. Additionally, each SB_u^{2m} sbox is itself a very efficient unkeyed Feistel round function. The datapath of the round-based ASIC parallel architecture implementation of a given sLiSCP-light instance is depicted in Figure 6.1.

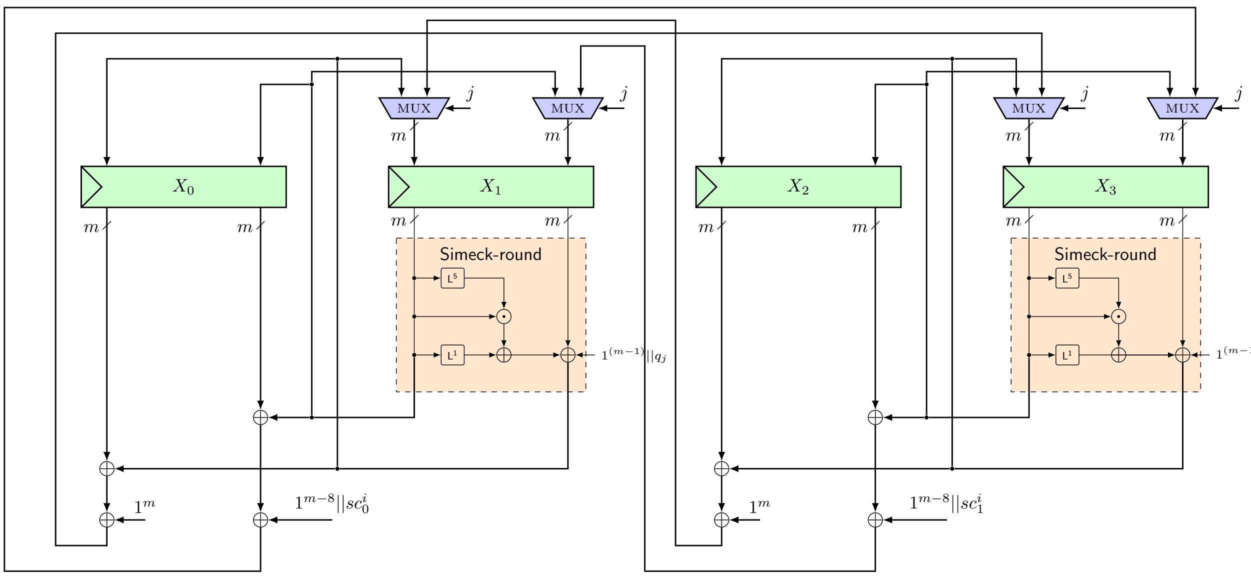

Figure 6.1: Parallel datapath of the sLiSCP-light-256 permutation step function ($m = 32$). The datapath of sLiSCP-light-192 is idential to sLiSCP-light-256 except addition of step constants.

The implementations of both sLiSCP-light and SpoC in ASIC are carried out using STMicroelectronics CMOS $65nm$ CORE65LPLVT library and IBM CMOS $130nm$ library. As depicted in Table 6.1, the parallel implementations in CMOS $65nm$ show that the area of sLiSCP-light-192 (resp. sLiSCP-light-256) is 1820 (resp. 2397) GE. Their areas in CMOS $130nm$ are 1892 GE and 2500 GE, respectively. Throughput is calculated by $\frac{b}{latency} \times 100$, where b denotes the state size and latency denotes the number of clock cycles for one permutation call and is equal to the total number of permutation rounds, $s \times u$.

Table 6.1: Parallel hardware implementation results of sLiSCP-light-192 and sLiSCP-light-256. Throughput and power are given at a frequency of 100 kHz.

Instance	**ASIC Technology** [nm]	**Latency** [cycles]	**Area** [GE]	**Power** [μW]	**Throughput** [kbps]
sLiSCP-light-192	65	108	1820	3.97	177.7
	130		1892	5.05	
sLiSCP-light-256	65	144	2397	4.77	
	130		2500	7.27	

Design flow and metrics. The Synopsys Design Compiler Version D-2010.03-SP4 is used to synthesize the RTL of the designs into netlist based on the STMicroelectronics CMOS 65nm CORE65LPLVT_1.20V and IBM CMOS 130nm CMR8SF-LPVT Process SAGE v2.0 standard cell libraries with both having a typical 1.2V voltage. Cadence SoC Encounter v09.12-s159_1 is used to finalize the place and route phase in order to generate the layout of the designs. We use Mentor Graphics ModelSim SE 10.1a to conduct functional simulation of the designs and perform timing simulation by using the timing delay information generated from SoC Encounter. We provide the areas and power consumption of both sLiSCP-light instances and SpoC after the logic synthesis.

We determine the power consumption based on the activity information generated from the timing simulation with a frequency of 100 kHz, and a duration time of 0.1s using SoC Encounter v09.12-s159_1. We specifically use 100 kHz clock frequency because it is widely used for benchmarking purpose in resource constrained applications and 0.1 s is long enough to provide an accurate activity information for all the signals.

6.2 Round-based Implementation of sLiSCP-light

Our round-based implementation executes one step of the permutation in u clock cycles, where $u = 6$ or 8, and requires the components as given in Table 6.2. As depicted in Figure 6.1, all four $2m$-bit registers are divided into two parts to accommodate the Feistel execution of the Simeck sboxes. Two counters i and j of 5 and 3 bits, respectively are utilized, where i $(0 \leq i \leq s-1)$ controls the permutation step function and j $(0 \leq j \leq u-1)$ controls the round function of Simeck.

During each clock cycle when $0 \leq j < u-1$, we first XOR the right half of registers X_1 (resp. X_3) with $\mathbf{1}^{m-1}||q_j$ (resp. $\mathbf{1}^{m/2-1}||q'_j$) where q_j, q'_j are round constant bits (see Section 2.3.3). Next, the right half output of the Simeck round function (dashed box) on registers X_1 and X_3 is fed back to the left half of the registers, and the left half of the registers is shifted to the right half. When j equals $u-1$, the left half of the register X_3 is replaced by the XORed value of the right half of register X_1, left half of register X_0 and $\mathbf{1}^m$. At the same time, the left half of the register X_1 is XORed with the right half of the register X_0, and then is XORed with $\mathbf{1}^{m-8}||sc_0^i$.

In particular, for sLiSCP-light-192, the $(m-8)$ bits are first padded with two 0's followed by padding the 6-bit constant sc_0^i. The generated new value is then shifted to the right half of the register X_3. The same process takes place between X_2 and X_3 to update the value of X_1. At the same time, the values of registers X_1 and X_3 are shifted into the registers X_0 and X_2 respectively. Multiplexers are used at the inputs of X_1 and X_3 to make a selection between the output of the Simeck sboxes when $j = u-1$ and the cyclically shifted registers. Finally, a new permutation step begins where i is incremented by 1 and j is reset to 0.

Table 6.2: Breakdown of the number of discrete components in both instances of sLiSCP-light, where XOR is 1-bit xor operation and MUX is 2-1 1-bit multiplexer.

Permutation block	Discrete component	sLiSCP-light-192	sLiSCP-light-256
State	Registers	4×48	4×64
	MUX	96	128
SB_u^{2m} sboxes	AND	2×24	2×32
	XOR	2×49	2×65
Step constants	XOR	2×6	2×8
Mix Subblocks	XOR	2×48	2×64
LFSR	Registers	6	7
	XOR	6	9

6.3 SpoC Implementation Results

To implement SpoC-r, we implement the outer mode structure for capacity masking, plaintext/ciphertext bits and XORing 4-bit control signals, then include the sLiSCP-light-b module in it. In Table 6.3, we give a numerical summary of SpoC-r that covers its implementation results in CMOS 65nm and 130nm technologies, performance, and recommended parameters. Throughput in Table 6.3 is given for 1 KB messages and no associated data.

Throughput for processing an l-block data of length lr bits and n-bit nonce is given by:

$$\frac{lr}{su(n/r + l)} \times 100,$$

where one permutation call is needed for initialization, one permutation call for absorbing the rest of the nonce when $n = 2r$, and l calls for data authenticated encryption.

Table 6.3: Parallel hardware implementation results of SpoC-r. Throughput is given at a frequency of 100 kHz for processing 1 KB message with no AD.

Instance	ASIC Technology	Parameters						Latency	Area	Throughput
	[nm]	s	u	n	r	c	t	[cycles]	[GE]	[kbps]
SpoC-64	65	18	6	128	64	128	64	14040	2329	58.3
	130								2389	
SpoC-128	65	18	8	128	128	128	128	9360	3054	87.5
	130								3125	

Bibliography

[1] Aagaard, M., AlTawy, R., Gong, G., Mandal, K., and Rohit, R. ACE: An authenticated encryption and hash algorithm. Submission to NIST-LWC (announced as round 2 candidate on August 30, 2019).

[2] AlTawy, R., Gong, G., He, M., Mandal, K., and Rohit, R. SPIX: An authenticated cipher. Submission to NIST-LWC (announced as round 2 candidate on August 30, 2019).

[3] AlTawy, R., Rohit, R., He, M., Mandal, K., Yang, G., and Gong, G. sLiSCP: Simeck-based Permutations for Lightweight Sponge Cryptographic Primitives. In *SAC* (2017), C. Adams and J. Camenisch, Eds., Springer, pp. 129–150.

[4] Altawy, R., Rohit, R., He, M., Mandal, K., Yang, G., and Gong, G. Sli scp-light: Towards hardware optimized sponge-specific cryptographic permutations. *ACM Transactions on Embedded Computing Systems (TECS) 17*, 4 (2018), 81.

[5] AlTawy, R., Rohit, R., He, M., Mandal, K., Yang, G., and Gong, G. Towards a cryptographic minimal design: The sliscp family of permutations. *IEEE Transactions on Computers 67*, 9 (2018), 1341–1358.

[6] Bernstein, D. J., Kölbl, S., Lucks, S., Massolino, P. M. C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.-X., Todo, Y., and Viguier, B. Gimli: a cross-platform permutation. In *CHES* (2017), Springer, pp. 299–320.

[7] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers* (2011), pp. 320–337.

[8] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. Duplexing the sponge: single-pass authenticated encryption and other applications. *IACR Cryptology ePrint Archive 2011* (2011), 499.

[9] Chakraborti, A., Datta, N., Nandi, M., and Yasuda, K. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2018*, 2 (2018), 218–241.

[10] Chakraborti, A., Datta, N., Nandi, M., and Yasuda, K. Beetle Family of Lightweight and Secure Authenticated Encryption Ciphers. *IACR Cryptology ePrint Archive 2018* (2018), 805.

[11] Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. Cryptanalysis of Ascon. In *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the*

RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings (2015), pp. 371–387.

[12] Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. Cryptanalysis of Ascon. *IACR Cryptology ePrint Archive 2015* (2015), 30.

[13] Kölbl, S., Leander, G., and Tiessen, T. Observations on the Simon block cipher family. In *CRYPTO* (2015), R. Gennaro and M. Robshaw, Eds., Springer, pp. 161–185.

[14] Liu, Y., Sasaki, Y., Song, L., and Wang, G. Cryptanalysis of reduced sliscp permutation in sponge-hash and duplex-AE modes. In *SAC* (2018), C. Cid and J. Michael J. Jacobson, Eds., vol. 11349, Springer, pp. 92–114.

[15] Nyberg, K. Generalized feistel networks. In *ASIACRYPT* (1996), K. Kim and T. Matsumoto, Eds., Springer, pp. 91–104.

[16] Todo, Y., and Morii, M. Bit-based division property and application to simon family. In *FSE* (2016), Springer, pp. 357–377.

[17] Yang, G., Zhu, B., Suder, V., Aagaard, M. D., and Gong, G. The simeck family of lightweight block ciphers. In *CHES* (2015), T. Güneysu and H. Handschuh, Eds., Springer, pp. 307–329.

Appendix A

Other NIST-LWC Submissions Adopting sLiSCP-light Family of Permutations

In Table A.1, we list our other NIST-LWC submissions whose underlying permutation adopts a similar design as the sLiSCP-light [4] family of permutations. SpoC is an authenticated cipher that enables tighter bound on the underlying state size to offer same security as other generic AE schemes, thus allowing larger rate size. SpoC adopts sLiSCP-light-192 and sLiSCP-light-256 to enable different performance and hence different target applications. Spix adopts sLiSCP-light in a monkey duplex to offer higher throughput than generic Sponge-based AE schemes. ACE is an all in one primitive that utilizes a generalized version of sLiSCP-light with state size 320-bit and a different linear layer to offer both hashing and authenticated encryption functionalities. In Table A.1, the submissions are classified based on their functionalities, mode of operation parameters and hardware area in ASIC $65nm$.

Table A.1: Submissions with sLiSCP-light like permutations

Algorithm	Permutation	Functionality	Parameters (in bits)			Mode of operation	Area
			State	Rate	Security		[GE]
ACE-$\mathcal{AE}$ and ACE-$\mathcal{H}$ [1]	ACE	AEAD & Hash	320	64	128	Unified sLiSCP sponge	4286
Spix [2]	sLiSCP-light-256	AEAD	256	64	128	Monkey Duplex	2611
SpoC-64	sLiSCP-light-192	AEAD	192	64	128	SpoC	2329
SpoC-128	sLiSCP-light-256	AEAD	256	128	128	SpoC	3054

Appendix B

Test Vectors

B.1 sLiSCP-light-[192]

Table B.1: Test vector for sLiSCP-light-[192] permutation

Step	State
0	000000000000 000000000000 000000000000 000000000000
1	FFFF9AFFFFFC 0000640000D5 FFFF9BFFFFFC 0000650000F4
2	D29A66FE8E7D C77A57FE2B66 C77A33FE2B87 D29A03FE8E8D
3	BB3EF980467D 663C22251B97 5EB9EE24CF23 965B6081370A
4	AB8B315855D7 944A50C96FAC 350C41125FA5 EF4A3727EC85
5	D78C6BE71947 5D5A8B2BCDE0 97A935C66D5A 83F8A540B3A8
6	A1E3DDD4BB98 CF92CB8D9493 A7C401B406D9 899049CC5DFB
7	D559023ED894 3CE33CEE4058 64D8C2A5B999 8B4520159C3A
8	B5A5F36ECB25 E373D6FA348B 7854EBA07206 9F030EAFECBC
9	0981F319A9B0 D068FBF3E53D 57C3EFAC6825 43DBFF889DBE
10	69F4002F1B2D B231C1D1E58B 1A0DD182729F 9F8A0CC94DA3
11	DDA85151C3F3 A837984FB5D6 4DC5B6323840 4BA3AE8127DF
12	B667F4B427DB 32B6F61B8396 808CBFD644FB 94305A1A1B09
13	186368B04BD2 5F6FF8A84957 201CB881F2B7 51FB63FB9318
14	FE146C662788 2B34ECFAB10E F4D7AB84BCAF 1988FB299363
15	67711D11ECDB 74487A1BA653 7F602E60E5C1 669A8E883456
16	0681008DCE57 62E55DE5568F E27A8C7A4C4D 9E0FE263DDAB
17	AD451841DF99 D9B6D181F062 C433A204432D 543BE733EEFA
18	2DCACA3466FA 126D47F0E142 29A11A0B5D4C 7F702D8A464D

B.2 SpoC-64_sLiSCP-light-[192]

Key	00111122335588DD 00111122335588DD
Nonce	111122335588DD00 111122335588DD00
Associated data	1122335588DD0011 1122335588DD00
Plaintext	335588DD00111122 335588DD001111
Ciphertext	B11663DA2A4B955F B0499BCAB9AD6F
Tag	F447B954EF852CC1

B.3 sLiSCP-light-[256]

Table B.2: Test vector for sLiSCP-light-[256] permutation

Step	State
0	0000000000000000 0000000000000000 0000000000000000 0000000000000000
1	00000C6F00000426 FFFFE3C3FFFFF348 00001C3C00000C2C FFFFF390FFFFFB2E
2	1DE1A7CF6E2DEA09 62A63FBB4C7F5233 9D59DC78B380A174 E21E545F91D211A9
3	2F11A3C5964A2121 EE5762E6E896794D 8CF14161A4E92756 CD0FFBF5079834CA
4	88343AFEBA25720B 9DBD1D318AFC04E4 EEB3A3AFD1EADC9E 58DA66C4D390ACA3
5	43505BFAD90F2156 73381560F8362948 62744930D6230A0B 349B9EFB9CD5ACBB
6	209FDCD6B4BC6E7B C944D232D517F4EB 54CF64FDFCCB0179 9C3078D3924CB0E7
7	CDF5132B02768F42 0C645033E732AA5D A754CB31E40654CE 1295300249351E2E
8	E3551361FF666A96 11E7A1F154C787FD 494C953F4F3E2C3C D15FFFB502EF1A5A
9	5BD8FE9BE803B316 F11CA614E5E599A6 47AFCCD455244A9E 47721205E89A26E4
10	F197723AA428B1A2 FC546679B9B26621 440455521369D3FC 55B0735EB3D4FDDF
11	8F17F61709A80DEE C96925615D4B740C 72928FCCB1DD5801 817F7BD2527F4323
12	2B858D69E03F180C 96536EBDE32B1437 1B3E1E8EAD09B372 5B6D84811668EACE
13	32D1FCDF790EF884 FE7457572B23191C 1AB5B62679D5551D E6AB8E4966CE1F55
14	DCEF74D18CA6AA09 60A8131451FF0FD5 85E25ACDD7D5A52D 11C177F10A57AD14
15	4D930DEA642F22ED A3E7DE93A806267E D9FA7BA1802C7C58 6E8386C41776770E
16	40D8429E26CB7CC2 C299816562B93DAE E49C053B1D6ABEB1 F2B4B08BBD1BA120
17	87994BD3E40B3A9E 9EB7A40050ADB69C 85D45EC4B238F79F 38BEF6B23D3FB958
18	C14FD32FDD8C4F91 3D7CD37CE4C0FC40 47577247A907F46A B9296703C6788A4C

B.4 SpoC-128_sLiSCP-light-[256]

Key	00111122335588DD 00111122335588DD
Nonce	111122335588DD00 111122335588DD00
Associated data	1122335588DD0011 1122335588DD00
Plaintext	335588DD00111122 335588DD001111
Ciphertext	A1F2FE57A1956C02 55C6B9B225ED39
Tag	745D95285F4BE3BE 99CC0ADA3EF9521B