Spix: An Authenticated Cipher
Submission to the NIST LWC Competition

Submitters/Designers:
Riham AlTawy*,[1], Guang Gong, Morgan He,
Kalikinkar Mandal, and Raghvendra Rohit

*Corresponding submitter:
Email: raltawy@uvic.ca
Tel: +1-250-721-8639

Communication Security Lab
Department of Electrical and Computer Engineering
University of Waterloo
200 University Avenue West
Waterloo, ON, N2L 3G1, CANADA

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

September 27, 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, Spix is an authenticated encryption algorithm that supports both messages and associated data (AD). It targets lightweight applications that can guarantee nonce uniqueness for security. On the top level, Spix adopts the monkey duplex construction which supports two different calls for its underlying permutation, where each call invokes a specific number of rounds. In such sense, one may view Spix as a primitive that uses two instances of the permutation which in this case is sLiSCP-light [4]. In what follows, we briefly highlight the features of Spix and then introduce the notations that are used throughout the document.

- Monkey duplexed Sponge-based mode of operation which provides better throughput (81.8 Kbps for 1 KB messages) [12].
- A partial SPN 256-bit permutation that utilizes a hardware optimized nonlinear function (ASIC area of 2406 GE) [4].
- Provably secure mode with keyed initialization and finalization phases for added security.
- Small hardware footprint of 2611 GE for constrained devices: small state and simple mixing.
- 128-bit security with straight out parameters: key size = tag size = security level.
- Key agile, single pass, nonce-based and inverse-free decryption.

1.1 Notations

Notation	Description
$\oplus$, $\odot$, $\|\|$	Bitwise XOR, AND, and concatenation operator
$\|X\|$	Length of X in bits
$\mathbb{F}_2$	$\{0,1\}$
$\mathbb{F}_2^n$	n-dimensional vector space over $\mathbb{F}_2$
L^{i}	Left cyclic shift operator, i.e., for $x \in \mathbb{F}_2^n$, $\mathsf{L}^{\mathsf{i}}(x) = (x_i, x_{i+1}, \ldots, x_{n-1}, x_0, x_1, \ldots, x_{i-1})$
z^n	The concatenation of n z bits where $z \in \{0, 1\}$, e.g., $1^n = \underbrace{1, 1, \ldots, 1}_{n}$
ϕ	Empty string
r, c, b	Rate, capacity and permutation state size in bits
S	The state of sLISCP-light
S_r, S_c	r-bit rate part and c-bit capacity part of S
subblock	A 64-bit string
X, $X[u]$	A 64 bit subblock, u-th byte of X starting from left
X_j^i	The j-th subblock at i-th step of the permutation.
$x_{j,l}^i, x_{j,r}^i$	Upper and lower halves of subblock X_j^i, i.e., $X_j^i = x_{j,l}^i \|\| x_{j,r}^i$
ℓ_A	Length of vector A in subblocks

1.2 Outline

Chapter 2 gives a complete specification of the SPIX Authenticated Encryption (AE) scheme, along with its underlying operation mode and permutation. Confidentiality and integrity security goals are also listed in the following chapter. Our cryptanalysis of the underlying permutation and security claims of SPIX are given in Chapter 3. In Chapter 4, we present the rational behind our design features and justify our parameter choices. Chapter 5 provides our ASIC CMOS implementations and performance. Software efficiency including bit-sliced implementation of SPIX using SIMD instruction sets and microcontroller implementations, is discussed in Chapter 6. Finally, we conclude with references and appendices.

Chapter 2

Specification

In this chapter, we present the specifications of Spix along with its mode of operation and underlying permutations. We also give its parameter set and a detailed overall algorithmic description of the whole cipher.

2.1 Spix Overview

Spix is an authenticated cipher which adopts a modified monkey duplex mode of operation [12]. Spix offers 128-bit security and operates on two instances of a 256-bit sLiSCP-light permutation [4]. At its core, Spix defines the following two procedures.

- **Authenticated Encryption Algorithm** $\mathcal{E}$. This procedure takes as an input secret k-bit key K, a public n-bit nonce N, and a variable length message M and associated data AD (a.k.a header data). It outputs a ciphertext C with the same length as M and a k-bit tag T that authenticates N, M, and AD

$$\mathcal{E}(K, N, M, AD) = (C, T)$$

- **Verified Decryption Algorithm** $\mathcal{D}$. This procedure operates on the key K, the nonce N, the ciphertext C and associated data AD. It outputs the decrypted message M if the calculated tag is equal to the tag T, otherwise, it outputs $\perp$

$$\mathcal{D}(K, N, C, AD, T) \in \{M, \perp\}$$

2.1.1 Recommended parameter set

Spix uses 128-bit key and utilizes sLiSCP-light with state size 256 bits. In Table 2.1, we list the recommended parameter set for Spix. The length of each parameter is given in bits and D denotes the allowed data usage before re-keying is required.

2.1.2 Security claims

Spix assumes nonce respecting adversary and we do not claim security in the event of nonce reuse. If the verification procedure fails, the decrypted ciphertext and the

Table 2.1: Recommended parameter set for Spix

Rate (r)	Key (k)	Nonce (n)	Tag (t)	Data ($\log_2(D)$)
64	128	128	128	60

new tag should not be given as output. Moreover, we claim no security for reduced-round versions of Spix. We claim 128-bit security for both plaintext confidentiality and integrity where integrity covers the authenticity of plaintext, associated data and nonce.

2.2 The Mode of Operation

Spix uses the monkey duplex construction [12] as its operation mode. The adopted mode is depicted in Figure 2.2 and algorithmically specified in Algorithm 1. Spix's mode employs a keyed initialization and finalization phases that makes key recovery hard even if the internal state is recovered and also renders universal forgery with the knowledge of the internal state unattainable. The core algorithm of Spix is the sLiSCP-light-256 permutation (henceforth denoted by P or simply sLiSCP-light) of $b = r + c$ bits, where b in our case is 256 bits. We denote one iteration of P by step, and P^s denotes an s-step P. As depicted in Figure 2.1, the state is viewed as a concatenation of 32 bytes, i.e. $S = B_0, \ldots, B_{31}$, where $S_r = B_8, \ldots, B_{11}, B_{24}, \ldots, B_{27}$ are called the rate bytes and that is where data blocks are absorbed and squeezed. The remaining 24 bytes are called the capacity bytes and account for S_c. Each consecutive 8 bytes form a subblock as depicted in the 4-subblock state in Figure 2.1.

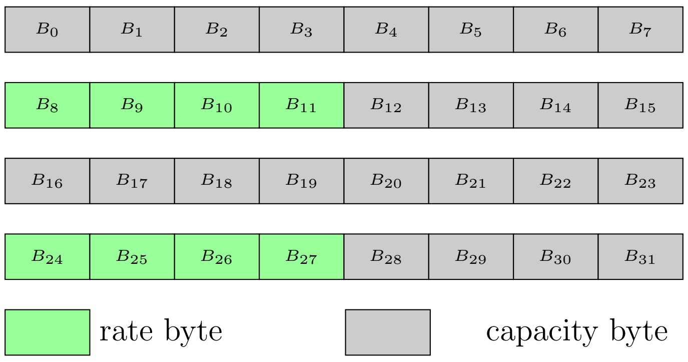
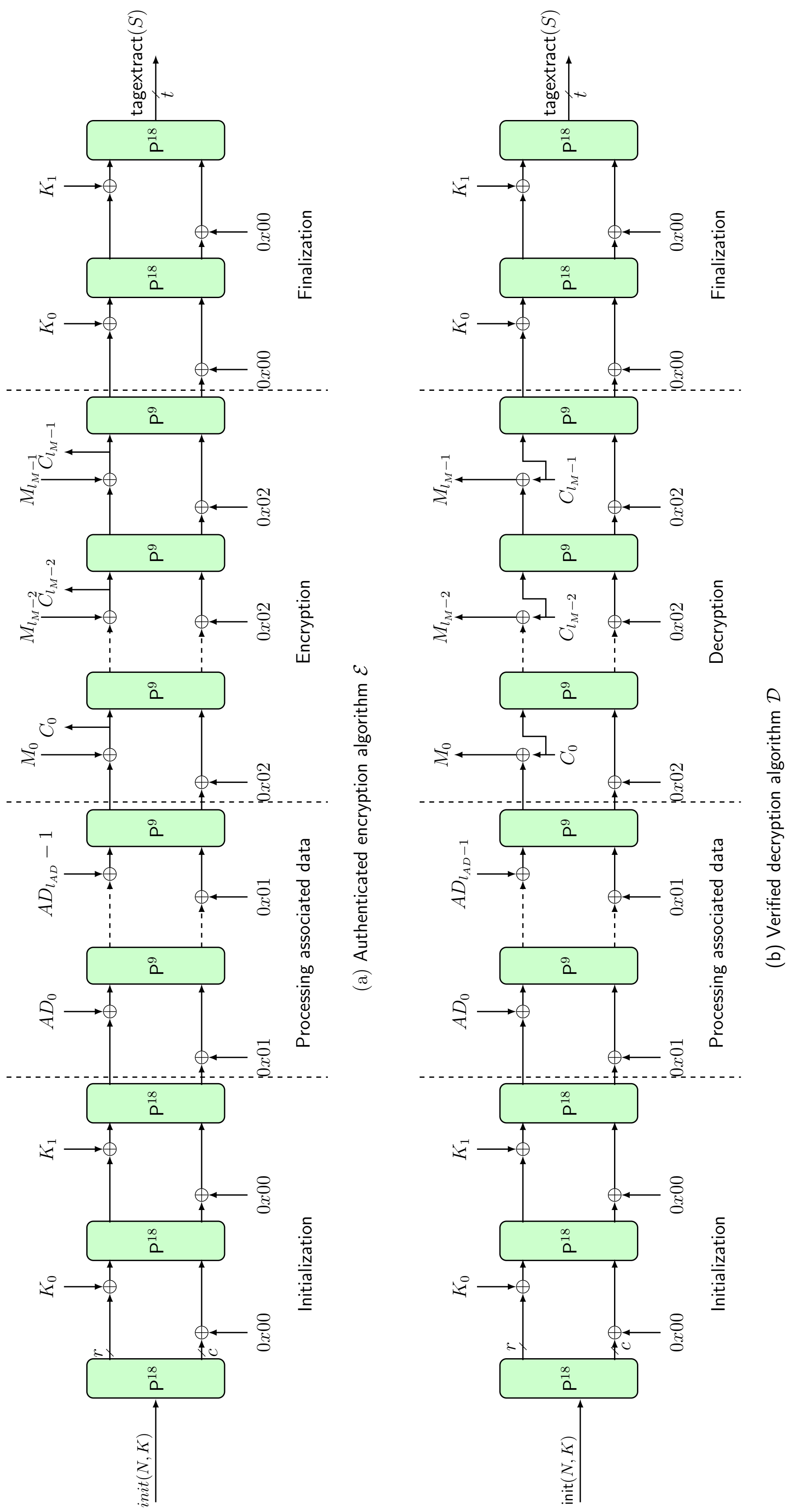

B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7
B_8 B_9 B_{10} B_{11} B_{12} B_{13} B_{14} B_{15}
B_{16} B_{17} B_{18} B_{19} B_{20} B_{21} B_{22} B_{23}
B_{24} B_{25} B_{26} B_{27} B_{28} B_{29} B_{30} B_{31}
rate byte
capacity byte

Figure 2.1: Visualization of the internal state of the permutation in Spix.

As depicted in Figure 2.2, $\mathcal{E}$ operates in four phases: initialization, processing associated data, encryption, and finalization. Analogously, $\mathcal{D}$ runs initialization, processing associated data, decryption, and finalization stages. Both initialization and finalization stages call P^{18}, while associated data processing, encryption and decryption invoke P^9.

2.2.1 Domain separation

Spix adopts a lightweight domain separation mechanism where a different 2-bit



(a) Authenticated encryption algorithm $\mathcal{E}$

(b) Verified decryption algorithm $\mathcal{D}$

Figure 2.2: Schematic diagram of Spix $\mathcal{E}$ and $\mathcal{D}$ algorithms.

constant (see Table 2.2) is XORed to byte B_{31} when a new phase starts. Their mixing with the state is illustrated in Figure 2.2, and their concrete integration is described in Algorithm 1.

Table 2.2: Domain separation constants in SPIX

Initialization	Proc. AD	Enc. & Dec.	Finalization
$0x00$	$0x01$	$0x02$	$0x00$

2.2.2 Padding

Padding is necessary when the length of the processed data is not a multiple of the rate r value. Since the key size is a multiple of r, we get two key blocks K_0 and K_1, so no padding is needed. Afterwards, the padding rule (10^*) which denotes a single 1 followed by required number of 0's is applied to the message M so that its length after padding is a multiple of r. The resulting padded message is divided into ℓ_M r-bit blocks $M_0\|\cdots\|M_{\ell_M-1}$. A similar procedure is carried out on the associated data AD which results in ℓ_{AD} r-bit blocks $AD_0\|\cdots\|AD_{\ell_{AD}-1}$. In the case where no associated data is present, no processing is necessary. We summarize the padding rules for the message and associated data below.

$$\begin{aligned}
\mathsf{pad_r}(M) &\leftarrow M\|1\|0^{r-1-(|M| \bmod r)}\\
\mathsf{pad_r}(AD) &\leftarrow \begin{cases} AD\|1\|0^{r-1-(|AD| \bmod r)} & \text{if } |AD|>0\\ \phi & \text{if } |AD|=0\end{cases}
\end{aligned}$$

Note that in case of AD or M whose length is a multiple of r, an additional r-bit padded block is appended to AD or M to distinguish between the processing of partial and complete blocks.

2.2.3 Initialization

In this phase, the state S is initialized with a 16-byte public nonce $N = N_0||N_1$ and a 16-byte key $K_0||K_1$. The nonce and key bytes are loaded in the odd and even subblocks, respectively (see Figure 2.1). Such function is denoted by $init(K,N)$. Afterwards, the 18-step sLiSCP-light permutation, P^{18} is applied on the state and the key blocks are absorbed into the state. The initialization steps are described below.

$$\begin{aligned}
B_0,\ldots,B_7 &\leftarrow N_0[0]\ldots,N_0[7]\\
B_{16},\ldots,B_{23} &\leftarrow N_1[0]\ldots,N_1[7]\\
B_8,\ldots,B_{15} &\leftarrow K_0[0]\ldots,K_0[7]\\
B_{24},\ldots,B_{31} &\leftarrow K_1[0]\ldots,K_1[7]
\end{aligned}$$

$$\begin{aligned}
S &\leftarrow \mathsf{P}^{18}(S)\\
S &\leftarrow \mathsf{P}^{18}(S_r\oplus K_i, S_c),\ i=0,1
\end{aligned}$$

2.2.4 Processing associated data

Each r-bit block AD_i, $i = 0, \ldots, \ell_{AD} - 1$ is XORed to the S_r part of the internal state S and $0x01$ constant domain separator is XORed to of B_{31}. Then, the 9-step sLiSCP-light permutation is applied on the whole state.

$$S \leftarrow \mathsf{P}^9(S_r \oplus AD_i, S_c \oplus (0^{c-2}\|01)),\ i = 0, \ldots, \ell_{AD} - 1$$

2.2.5 Encryption

Similar to the processing of associated data, however, with a different domain separator, each r-bit message block M_i, $i = 0, \ldots, \ell_M - 1$ is XORed to the S_r part of the internal state S resulting in the corresponding ciphertext C_i which is extracted from the S_r part of the state. After the computation of each C_i, the whole internal state S is permuted by applying 9-step sLiSCP-light.

$$\begin{aligned} C_i &\leftarrow S_r \oplus M_i, \\ S &\leftarrow \mathsf{P}^9(C_i, S_c \oplus (0^{c-2}\|10)),\ i = 0, \cdots, \ell_M - 1 \end{aligned}$$

To keep a minimal overhead, the last ciphertext block $C_{\ell_M - 1}$ is truncated so that its length is equal to that of the last unpadded message block. The details of encryption procedure is given in Algorithm 1.

2.2.6 Finalization

In this phase, the domain separator is changed to $0x00$ indicating the start of finalization phase and the key blocks are absorbed in the state, followed by an application of P^{18}. Finally, the $tagextract(S)$ function is evaluated where 16-byte tag $T = T_0||T_1$ is extracted as follows: $T_0[0], \ldots, T_0[7] = B_7, \ldots, B_{15}$, $T_1[0], \ldots, T_1[7] = B_{24}, \ldots, B_{31}$. The finalization steps are given below and illustrated in Algorithm 1.

$$\begin{aligned} S &\leftarrow \mathsf{P}^{18}(S_r \oplus K_i, S_c),\ i = 0, 1, \\ T_0 &= B_7|| \ldots ||B_{15}, \\ T_1 &= B_{24}|| \ldots ||B_{31} \end{aligned}$$

2.2.7 Decryption

The decryption procedure is symmetrical to encryption procedure and illustrated in Algorithm 1.

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 what follows, we give the specifications of two instances of 256-bit sLiSCP-light permutation where each instance runs for s steps and $s \in \{9, 18\}$, we denote a given instance by P^s. An algorithmic description of sLiSCP-light is provided in Algorithm 2.

Algorithm 1 SPIX algorithm

```
Authenticated encryption($K, N, AD, M$):
    $S \leftarrow$ Initialization($N, K$)
    if $|AD| \neq 0$ then:
        $S \leftarrow$ Processing-Associated-Data($S, AD$)
    $(S, C) \leftarrow$ Encyption($S, M$)
    $T \leftarrow$ Finalization($S, K$)
    return $(C, T)$

Initialization($N, K$):
    $S \leftarrow init(N, K)$
    $S \leftarrow P^{18}(S)$
    for $i = 0$ to $1$ do:
        $S \leftarrow (S_r \oplus K_i, S_c)$
        $S \leftarrow P^{18}(S)$
    return $S$

Processing-Associated-Data($S, AD$):
    $(AD_0||\cdots||AD_{\ell_{AD}-1}) \leftarrow \mathsf{pad_r}(AD)$
    for $i = 0$ to $\ell_{AD} - 1$ do:
        $S \leftarrow (S_r \oplus AD_i, S_c \oplus 0^{c-2}||01)$
        $S \leftarrow P^9(S)$
    return $S$

Encryption($S, M$):
    $(M_0||\cdots||M_{\ell_M-1}) \leftarrow \mathsf{pad_r}(M)$
    for $i = 0$ to $\ell_M - 1$ do:
        $C_i \leftarrow M_i \oplus S_r$
        $S \leftarrow (C_i, S_c \oplus 0^{c-2}||10)$
        $S \leftarrow P^9(S)$
    $C_{\ell_M-1} \leftarrow$ trunc-msb($C_{\ell_M-1}, |M| \bmod r$)
    $C \leftarrow (C_0, C_1, \ldots, C_{\ell_M-1})$
    return $(S, C)$

pad_r($X$):
    $X \leftarrow X||10^{r-1-(|X| \bmod r)}$
    return $X$

trunc-lsb($X, n$):
    return $(x_{r-n}, x_{r-n+1}, \ldots, x_{r-1})$

Verified decryption($K, N, AD, C, T$):
    $S \leftarrow$ Initialization($N, K$)
    if $|AD| \neq 0$ then:
        $S \leftarrow$ Processing-Associated-Data($S, AD$)
    $(S, M) \leftarrow$ Decyption($S, C$)
    $T' \leftarrow$ Finalization($S, K$)
    if $T' \neq T$ then:
        return $\perp$
    else:
        return $M$

Decryption($S, C$):
    $(C_0||\cdots||C_{\ell_C-1}) \leftarrow \mathsf{pad_r}(C)$
    for $i = 0$ to $\ell_C - 2$ do:
        $M_i \leftarrow C_i \oplus S_r$
        $S \leftarrow (C_i, S_c \oplus 0^{c-2}||10)$
        $S \leftarrow P^9(S)$
    $M_{\ell_C-1} \leftarrow S_r \oplus C_{\ell_C-1}$
    $C_{\ell_C-1} \leftarrow$ trunc-msb($C_{\ell_C-1}, |C| \bmod r$)||trunc-lsb($M_{\ell_C-1}, r - |C| \bmod r$)
    $M_{\ell_C-1} \leftarrow$ trunc-msb($M_{\ell_C-1}, |C| \bmod r$)
    $M \leftarrow (M_0, M_1, \ldots, M_{\ell_C-1})$
    $S \leftarrow P^9(C_{\ell_C-1}, S_c \oplus 0^{c-2}||10)$
    return $(S, M)$

Finalization($S, K$):
    for $i = 0$ to $1$ do:
        $S \leftarrow P^{18}(S_r \oplus K_i, S_c)$
    $T \leftarrow$ tagextract($S$)
    return $T$

trunc-msb($X, n$):
    if $n = 0$ then:
        return $\phi$
    else:
        return $(x_0, x_1, \ldots, x_{n-1})$
```

2.3.1 Step function of the permutation

An s-step sLISCP-light permutation takes an input of 256 bits and produces an output of 256 bits after applying the step function s times sequentially where $b = 8 \times 32$. We denote by P^s a 256-bit sLISCP-light permutation where the step function is applied s times. A high-level overview of the step function of sLISCP-light is depicted in Figure 2.3.

The state of the permutation is divided into four 64-bit subblocks $(X_0^i, X_1^i, X_2^i, X_3^i)$, where i denotes the step number and $0 \leq i \leq s-1$. 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

$$(X_0^{i+1}, X_1^{i+1}, X_2^{i+}, X_3^{i+1}) \leftarrow \mathsf{MSb} \circ \mathsf{ASc} \circ \mathsf{SSb}(X_0^i, X_1^i, X_2^i, X_3^i)$$

We now describe each transformation in detail.

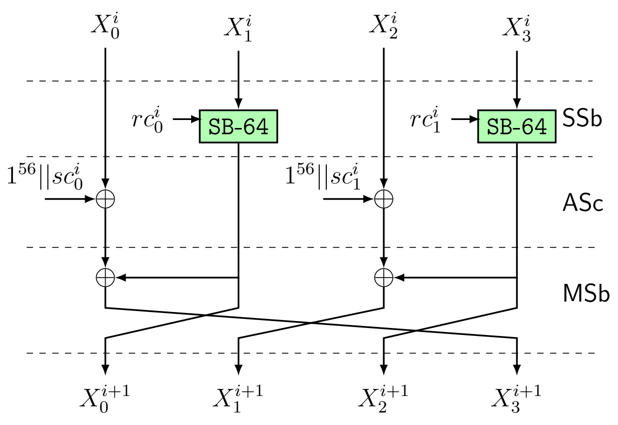


Figure 2.3: Step function of sLiSCP-light permutation

2.3.1.1 **`SubstituteSubblocks`** (SSb)

This is a partial substitution layer of the SPN structure where the nonlinear operation is applied to half of the state. It applies the 8-round iterated unkeyed Simeck-64 block cipher [19] (henceforth referred to as Simeck sbox or SB-64) to the odd indexed subblocks only. The SSb transformation is defined as

$$\mathsf{SSb}(X_0^i, X_1^i, X_2^i, X_3^i) = (X_0^i, \mathsf{SB\text{-}64}(X_1^i, rc_0^i), X_2^i, \mathsf{SB\text{-}64}(X_3^i, rc_1^i))$$

Below we provide the details of Simeck sbox SB-64.

Definition 1 (Simeck sbox SB-64 [5]) *Let $u = 8$ and $rc = (q_7, \ldots, q_0)$ where $q_j \in \mathbb{F}_2$ and $0 \leq j \leq 7$. A Simeck sbox is a permutation of 64-bit input constructed by iterating the Simeck-64 block cipher for 8 rounds with round constant addition $\gamma_j = 1^{31}||q_j$ in place of key addition.*

An illustrated description of the Simeck sbox as an nonlinear feedback shift register is shown in Figure 2.4 and is given by:

$$(x_9||x_8) \leftarrow \mathsf{SB\text{-}64}(x_1||x_0, rc)$$

where

$$x_j \leftarrow f_{(5,0,1)}(x_{j-1}) \oplus x_{j-2} \oplus \gamma_{j-2},\ 2 \leq j \leq 9 \text{ and}$$

$f_{(5,0,1)} : \mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$ given by $f_{(5,0,1)}(x) = (\mathsf{L}^5(x) \odot x) \oplus \mathsf{L}^1(x)$.

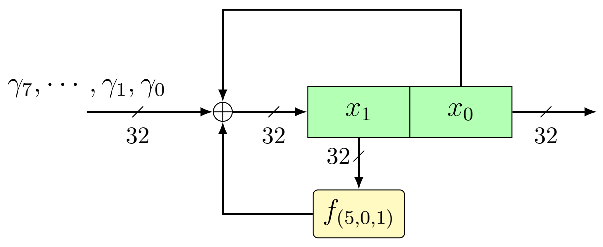


Figure 2.4: Simeck sbox SB-64

2.3.1.2 **AddStepconstants** (ASc)

In this layer, the step constants of the form $1^{56}||sc_0^i$ and $1^{56}||sc_1^i$ are XORed with the two even indexed subblocks X_0^i and X_2^i, respectively, for $i = 0, 1, \ldots s-1$. Each sc_j^i is a 8-bit constant generated by a 7-bit LFSR. The ASc transformation is given by

$$\mathsf{ASc}(X_0^i, \mathsf{SB\text{-}64}(X_1^i, rc_0^i), X_2^i, \mathsf{SB\text{-}64}(X_3^i, rc_1^i)) = (X_0^i \oplus 1^{56}||sc_0^i, \mathsf{SB\text{-}64}(X_1^i, rc_0^i), X_2^i \oplus 1^{56}||sc_1^i, \mathsf{SB\text{-}64}(X_3^i, rc_1^i))$$

2.3.1.3 **MixSubblocks** (MSb)

In this transformation, 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

$$(X_0^{i+1}, X_1^{i+1}, X_2^{i+1}, X_3^{i+1}) \leftarrow \mathsf{MSb}(X_0^i \oplus 1^{56}||sc_0^i, \mathsf{SB\text{-}64}(X_1^i, rc_0^i), X_2^i \oplus 1^{56}||sc_1^i, \mathsf{SB\text{-}64}(X_3^i, rc_1^i)),$$

where

$$\begin{aligned} X_0^{i+1} &= \mathsf{SB\text{-}64}(X_1^i), & X_1^{i+1} &= X_2^i \oplus \mathsf{SB\text{-}64}(X_3^i) \oplus 1^{56}||sc_1^i, \\ X_2^{i+1} &= \mathsf{SB\text{-}64}(X_3^i), & X_3^{i+1} &= X_0^i \oplus \mathsf{SB\text{-}64}(X_1^i) \oplus 1^{56}||sc_0^i \end{aligned}$$

Algorithm 2 sLiSCP-light permutation (P^s)

```
1:  Input : $S^0 = X_0^0||X_1^0||X_2^0||X_3^0$
2:  Output : $S^s = X_0^s||X_1^s||X_2^s||X_3^s$
3:  for $i = 0$ to $s-1$ do :
4:          $S^{i+1} \leftarrow \mathsf{P\text{-}step}(S^i)$
5:  return($S^s$)

6:  Function P-step($S^i$) :
7:          $X_0^{i+1} \leftarrow \mathsf{SB\text{-}64}(x_{1,l}^i||x_{1,r}^i, rc_0^i)$
8:          $X_1^{i+1} \leftarrow \mathsf{SB\text{-}64}(x_{3,l}^i||x_{3,r}^i, rc_1^i) \oplus X_2^i \oplus (1^{56}||sc_1^i)$
9:          $X_2^{i+1} \leftarrow \mathsf{SB\text{-}64}(x_{3,l}^i||x_{3,r}^i, rc_1^i)$
10:         $X_3^{i+1} \leftarrow \mathsf{SB\text{-}64}(x_{1,l}^i||x_{1,r}^i, rc_0^i) \oplus X_0^i \oplus (1^{56}||sc_0^i)$
11:         return($X_0^{i+1}||X_1^{i+1}||X_2^{i+1}||X_3^{i+1}$)

12: Function SB-64($x_1||x_0, rc$) :
13:     $rc = (q_7, q_6, \ldots, q_0)$
14:     for $j = 2$ to $9$ do
15:         $x_j \leftarrow (\mathsf{L}^5(x_{j-1}) \odot x_{j-1}) \oplus \mathsf{L}^1(x_{j-1}) \oplus x_{j-2} \oplus (1^{31}||q_{j-2})$
16:     return($x_9||x_8$)
```

2.3.2 sLiSCP-light **permutation instances**

In Spix, we use two instances of sLiSCP-light. Table 2.3 presents the recommended parameters for these two instances.

Table 2.3: Recommended parameter set for sLiSCP-light used in Spix.

Permutation	m	Rounds u	Steps s	Total # rounds $(u \cdot s)$
P^{18}	32	8	18	144
P^{9}	32	8	9	72

2.3.3 sLiSCP-light constants

As depicted in Figure 2.3, 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 sbox while step constants are XORed to the even subblocks as described earlier. In Table 2.4, we list the hexadecimal values of constants for sLiSCP-light. More details on how to generate these constants can be found in [5].

Table 2.4: Round and step constants for sLiSCP-light

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)

Chapter 3

Security Analysis

In this section, we analyze the security of the Spix by first assessing the behavior of sLiSCP-light permutation against various distinguishing attacks. We primarily focus on the diffusion behavior, evaluation of the expected maximum probabilities of differential and linear characteristics, and algebraic properties of this new design.

3.1 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 a monkey duplex, we aim to provide evidence of how the instance that is used in the initialization and finalization phases 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. We also show that the other instance is secure against differential and linear cryptanalysis which are the most powerful attacks on authenticated ciphers. Our analysis focuses on providing results related to the diffusion, differential and linear, algebraic degree, self-symmetry properties of the permutation.

3.1.1 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 8 rounds of Simeck in sLiSCP-light 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 256 bit positions for sLiSCP-light spans between 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.

3.1.2 Differential and linear cryptanalysis

In order to evaluate the differential and linear behavior of sLiSCP-light, we first 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 [16] coupled with an optimized differentials and linear masks exhaustive search. Next, 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 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 the Simeck-64 sbox, SB-64 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 evaluate the tradeoff 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 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^{-256} and 2^{-128}, respectively for the permutation instance used in initialization and finalization, formally, $\delta m_s \leq -256$ and $\gamma m_s \leq -128$. For the permutation instance which is used in encryption and decryption, MEDCP and MELCSC should be upper bounded by 2^{-128} and 2^{-64}, respectively, formally, $\delta m_s \leq -128$ and $\gamma m_s \leq -64$.
2. For the ideal permutation, the total number of steps to be lower bounded by three times the number of steps required for full bit diffusion, formally, $s \geq d_s$.

For the reduced-round permutation, no inbound distinguishers exist.

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

In [4], the tradeoffs between u and s are considered and it has been found that for SB-64, when $u = 8$, $\delta = -15.9$, $\gamma = -15.6$, the above three conditions are optimally satisfied when $s = 18$ and accordingly $m_s = 18$. Moreover, when iterating sLiSCP-light in a monkey duplex to provide 128-bit security, 9 steps are enough to ensure resistance against differential and linear cryptanalysis that may be used to launch forgery attacks through injecting differences in either the associated data or message blocks. In other words, the expected bounds on the MEDCP and MELCSC are as follows:

$$P^{18}: \begin{array}{lll} (\mathrm{MDP}(\texttt{SB-64}))^{18} & = (2^{-15.9})^{18} & = 2^{-286.2} \\ (\mathrm{MLSC}(\texttt{SB-64}))^{18} & = (2^{-15.6})^{18} & = 2^{-280.8} \end{array}$$

$$P^{9}: \begin{array}{lll} (\mathrm{MDP}(\texttt{SB-64}))^{9} & = (2^{-15.9})^{9} & = 2^{-143.1} \\ (\mathrm{MLSC}(\texttt{SB-64}))^{9} & = (2^{-15.6})^{9} & = 2^{-140.4} \end{array}$$

3.1.3 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 [18, 7]. In [4], it is found that the algebraic degree of SB-64 is 36 which results in an upper bound on the algebraic degree of all component functions of sLiSCP-light of 247 after only 4 steps. Such numbers suggest that maximum degree for sLiSCP-light maybe reached after 6 steps of the permutation. In what follows, we give the results of our integral and zero-sum distinguishers.

Integral distinguishers. According to the cryptanalysis presented in [4], we report that for sLiSCP-light, there exists an 8-step integral distinguishers that can be found with data and time complexities of 2^{255}.

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. Such a distinguisher requires data and time complexities equal to that of the exhaustive search.

3.1.4 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, we employ a 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 $i \neq j$, $(sc_0^i, sc_1^i) \neq (sc_0^j, sc_1^j)$.

3.2 Security of SPIX

The security proofs of modes based on the Monkey duplex construction relies on the indistinguishability of underlying permutation from a random one [9, 11, 10, 15] in both initialization and finalization phases, as well as the assumption of nonce respecting adversary. In previous sections, we have shown that there are no distinguishers for 18 steps of sLISCP-light. Thus, the security bounds of the monkey duplex mode are applicable to SPIX.

We assume a nonce-respecting adversary, i.e., for a fixed K, nonce N is never repeated during encryption queries. Then considering a data limit of 2^d, k-bit security is achieved if $c \geq k + d + 1$ and $d << c/2$ [10]. The parameter set of SPIX (see Table 2.1) with actual effective capacity 190 (2 bits are lost for domain separation) satisfy this condition, and hence SPIX provides 128-bit security for confidentiality, integrity and authenticity when $d = 60$.

Note that we could use $r = 96$, $d = 50$ and obtain 112-bit security [15]. However, this would require an additional 32 XORs and is not efficient in software as $r = 64$.

Chapter 4

Design Rationale

In this chapter, we provide the rationale for our design choices and justify the design principles of each component of Spix.

4.1 Choice of the Mode: Monkey Duplex Sponge Mode

Our adopted mode is a variation of the sponge duplex construction. Monkey duplex is proposed to be used in keyed modes where nonce reuse can be effectively mitigated. Its most important advantage is that it utilizes two instances of the the permutation each with a different number of rounds. Such a feature enhances the performance of Spix because the number of rounds of the permutation during absorbing the message can be optimized by reducing it such that it resists differential attacks in this specific scenario. It has been shown that under certain conditions of round number choices, Monkey duplexing security can be reduced to the original sponge [12, 8]. In addition to the former reasons, the adopted mode offers the following features:

- **Key agility and reverse free decryption**. Simple and lightweight mode where neither key scheduling nor decryption algorithm implementation is required.
- **Keyed initialization and finalization phases**. Key recovery is hard even if the internal state is recovered. Universal forgery with the knowledge of the internal state is not practical.
- **Uniform domain separators**. Domain transitions run for all rounds and they are changed with each new transition because we found that it leads to a more efficient ASIC implementation. Such mechanism has been shown to be secure in [15].

4.2 Spix State Size

Following the NIST submission requirements, Spix needs to work with 128-bit keys and provide a minimum of $T = 2^{112}$ attack time complexity with data complexity of $D = 2^{50} - 1$ bytes. For b-bit state with $b = r + c$, r-bit rate and c-bit capacity, following the security bounds in [15, 14], to satisfy the security requirements, c should be at least $\log_2(D^2 + DT)$ where $D = 2^{47}$ blocks for $r = 64$ which means $c \geq 160$. Taking hardware area constraints in mind, we found that $b = 256$ and $r = 64$ is an efficient choice. More precisely, we needed r and b to be a multiple of 64 to have an efficient software implementation and follow the specifications of the sLiSCP-light structure. Accordingly, practical choices for state sizes would be $b = 256$, and 512, so we choose $b = 256$ as it provides the best trade-off among hardware and software requirements, security and efficiency. With this choice of b, Spix is efficiently implemented on wide range of platforms. Moreover, with our choice, we offer 128-bit security with 2^{60} data complexity.

4.3 Nonlinear Layer: Simeck Sbox

The Simeck sbox, is an unkeyed independently parameterized variant of the round function of the Simon round function [6]. Moreover, it has set a new record in terms of hardware efficiency and performance on almost all platforms. In what follows, we list the reasons that motivated our adoption of Simeck sboxes as the nonlinear function of sLiSCP-light permutation.

- Simeck has a hardware friendly round function which consists of simple bitwise XOR, AND and cyclic shift operations. Moreover, the resulting footprint grows linearly with its input size.

- It is practical to evaluate the Sbox maximum (expected) differential probability and maximum (expected) linear squared correlation which are $2^{-15.8}$ and $2^{-15.6}$, respectively. Accordingly, we can provide an expected bounds against differential and linear cryptanalysis.

- The Sbox has an algebraic degree of 36 and the output component functions depend on either 61 or 55 input state bits which enables us to provides guarantees against algebraic and diffusion-based attacks.

- Each Simeck sbox is independently parameterized by the associated set of round constants which suggests that the reported expected bounds against differential and linear cryptanalysis are not tight, thus, better security is expected.

4.4 Round and Step Constants

We use the following set of constants to mitigate the self-symmetry distinguishers.

- **Three 8-bit unique step constants** (sc_0^i, sc_1^i, sc_2^i). The 3-tuple constant value is unique across all steps, hence it destroys any symmetry between the steps of the permutation. Accordingly, we mitigate slide distinguishers [13]. We also require that for any step i, $sc_0^i \neq sc_1^i \neq sc_2^i$ in order to destroy any symmetry between word shuffle.

- **Three 8-bit unique round constants** (rc_0^i, rc_1^i, rc_2^i). One bit of each round constant is XORed with the state of the Simeck sbox in each round to destroy the preservation of any rotational properties. Moreover, we append 31 '1' bits to each one bit constant which results in a lot of inversions, and accordingly breaks the propagation of the rotational property in one step.

Our choice of the utilized LFSR polynomial ensures that each tuple of such constants does not repeat due to the periodicity of the 8-tuple sequence constructed from the decimated m-sequence of period 127.

4.5 Number of Rounds and Steps

Our rationale for choosing the number of rounds u and number of steps s of Spix is based on achieving the best trade-off between security and efficiency. In both instances of sLiSCP-light, we require the value $u \times s$ to be minimized while satisfying the following two conditions:

- For sLiSCP-light instance that is used in initialization and finalization, we aim that it is indistinguishable from a random permutation.
- For sLiSCP-light instance that is used in AD processing, encryption and decryption, we require that the expected maximum differential probability over the permutation is less than 2^{-128} and there exist no start from the middle distinguishers for it.

4.5.1 P^{18} sLiSCP-light instance

Diffusion. Our first criteria is that s should be at least 3×4 where 4 steps are required to achieve full bit diffusion in state. This choice adds 33% security margin against meet/miss-in-the-middle distinguishers, as in 8 steps, full bit diffusion is achieved in both forward and backward directions. Hence, $s \geq 12$.

MEDCP(P^s) $< 2^{-b}$. Our second criteria is to push the MEDCP value of sLiSCP-light to below 2^{-256}. This value depends on the MEDCP of u-round Simeck sbox and the minimum number of such active sboxes in s steps. An in depth automated analysis was conducted in [4] and it was found that the expected maximum differential probability of an 8-round iterated Simeck sbox is $2^{-15.8}$ and that iterating the permutation step function for 18 steps results in a minimum of 18 active sboxes. Accordingly, we have $18 \times -15.8 = -284.4 < -256$.

4.5.2 P^9 sLiSCP-light instance

Inbound distinguishers. Following the terminology of the rebound attack, we denote distingushers that are constructed from input (resp. output) to output (resp. input) by inbound distinguishers. In the cryptanalysis of sLiSCP-light it was found the best distinguisher that does not require start from the middle approach may cover up to 8 steps. Accordingly, choosing $s > 8$ was a safe choice for sLiSCP-light during AD processing, encryption, and decryption.

MEDCP(P^s) $< 2^{-k}$. The number of steps in this instance is chosen such that in addition to being larger than 8, it also should result in an MEDCP value less than 2^{-128}. For $s = 9$ steps, we get an MEDCP of $9 \times -15.8 = -142.2 < -128$.

4.6 Choice of Rate Positions

We absorb message blocks in subblocks X_1 and X_3. Such rate 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 differences to activate Simeck sboxes in the first step which also enhances Spix's resistance to differential and linear cryptanalysis. This observation has also been confirmed by a third party cryptanalysis of sLiSCP [17].

4.7 Statement

The authors declare that there are no hidden weaknesses in Spix and sLiSCP-light.

Chapter 5

Hardware Design

In this chapter, we provide the details of our ASIC hardware implementation of sLiSCP-light permutation and Spix. The reported implementations are carried out using CMOS 65 *nm* and 130 *nm* technologies.

5.1 ASIC Implementation

Spix is highly hardware optimized and has very efficient ASIC implementations particularly because its core permutation sLiSCP-light employ partial layers. More precisely, the SB-64 boxes, step constant addition, and linear mixing are all applied on half of the state. Additionally, each SB-64 box is itself a very efficient unkeyed Feistel round function. The datapath of the round-based ASIC parallel architecture implementation of sLiSCP-light is depicted in Figure 5.1.

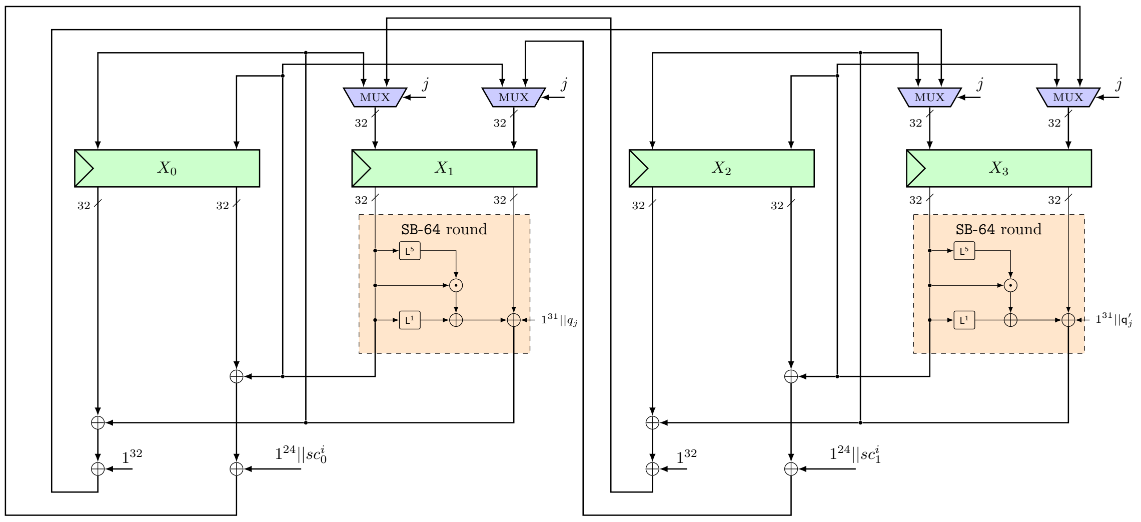


Figure 5.1: Parallel datapath of the sLiSCP-light-256 permutation step function.

The implementation of Spix in ASIC is carried out using STMicroelectronics CMOS 65 *nm* CORE65LPLVT library and IBM CMOS 130 *nm* library. As depicted in Table 5.1, the parallel implementations in CMOS 65 *nm* show that the area of sLiSCP-light is 2397 GE. The area in CMOS 130 *nm* is 2500 GE. Throughput is calculated by $\frac{256}{latency} \times 100$, where latency denotes the number of clock cycles for one permutation call and is equal to the total number of permutation rounds, $s \times 8$.

Table 5.1: Parallel hardware implementation results of sLiSCP-light instances P^{18} and P^{9}. Throughput and power are given at a frequency of 100 kHz.

Instance	**Steps**	**ASIC Technology**	**Latency**	**Area**	**Throughput**	**Power**
	[s]	[nm]	[Cycles]	[GE]	[kbps]	[μW]
P^{18}	18	65	144	2396	177.7	4.77
		130		2500		7.27
P^{9}	9	65	72	2396	355.5	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 65 nm CORE65LPLVT_1.20V and IBM CMOS 130 nm 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 and Spix 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 0.1s is long enough to provide an accurate activity information for all the signals.

5.2 Round-based implementation of sLiSCP-light

Our round-based implementation executes one step of the permutation in u clock cycles, where $u = 8$, and requires the components as given in Table 5.2. As depicted in Figure 5.1, all four 64-bit registers are divided into two parts to accommodate the Feistel execution of the SB-64 boxes. Two counters i and j of 5 and 3 bits, respectively are utilized, where i ($0 \leq i \leq 17$ or 8) controls the permutation step function and j ($0 \leq j \leq 7$) controls the round function of SB-64 box.

During each clock cycle when $0 \leq j < 7$, we first XOR the right half of registers X_1 (resp. X_3) with $1^{31}||q_j$ where q_j is a round constant bit (see Section 2.3.3). Next, the right half output of the SB-64 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 7, 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 1^{32}. 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 $1^{24}||sc_0^i$.

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 SB-64 boxes when $j = 7$ and the cyclically shifted registers. Finally, a new permutation step begins where i is incremented by 1 and j is reset to 0.

Table 5.2: Breakdown of the number of discrete components in both sLISCP-light and SPIX where XOR is 1-bit xor operation and MUX is 2-1 1-bit multiplexer.

Block	**Discrete component**	sLISCP-light	SPIX
Mode	Mux	-	64
	XOR	-	64
State	Registers	4×64	
	MUX	128	
SB-64 boxes	AND	2×32	
	XOR	2×65	
Add step constants	XOR	2×8	
Mix Subblocks	XOR	2×64	
LFSR	Registers	7	
	XOR	9	

5.3 SPIX by the Numbers

In Table 5.3, we give a numerical summary of SPIX that covers its implementation results in CMOS 65 *nm* and 130 *nm* technologies, performance, power, and recommended parameters. Throughput in Table 5.3 is given for 1 KB messages and no AD.

Throughput for processing an l-block data of length $64l$ bits is given by:

$$\frac{64l}{5(144) + 72(l+1)} \times 100,$$

where five P^{18} permutation calls are needed for initialization and finalization, l calls of P^9 are invoked for data authenticated encryption, and an extra P^9 call for the padding data block. Note that if the data length is not a multiple of 64, then no extra padding block is processed and thus 72 clock cycles should be deducted from the total cycles in the denominator of the above equation. Analogously, if associated data is processed, where its size is multiple of 64, then an extra 72 clock cycles should be added to the total cycles in the denominator of the above equation.

Table 5.3: Parallel hardware implementation results of Spix. Throughput and power are given at a frequency of 100 kHz for processing 1 KB message with no AD.

Functionality	**ASIC Technology**	**Parameters**			**Latency**	**Area**	**Throughput**	**Power**
	[nm]	r	c	t	[Cycles)	[GE]	[kbps]	[μW]
Spix	65	64	192	128	10008	**2611**	81.8	4.77
	130					**2742**		7.27

Chapter 6

Software Implementations

In this chapter, we present a bit-sliced implementation of sLiSCP-light using SIMD instruction sets, then augment it with the implementation of the monkey duplex mode to implement Spix. We also provide microcontroller implementation results.

6.1 Bit-sliced Implementation of sLiSCP-light

We use different instructions in the SSE2 and AVX2 instruction sets to implement the sLiSCP-light where the SSE2 and AVX2 instruction sets support 128-bit and 256-bit SIMD registers, known as XMM and YMM, respectively. The operations used in the sLiSCP-light permutation are bitwise XOR, AND and left cyclic shift. In our implementation, packing and unpacking of data are two salient tasks which are performed at the beginning and end, and during the execution of the permutation.

Recall that in the SSb layer, the Simeck sbox is applied only on the odd indexed subblocks. That is, the operations on every block are not homogeneous. The key idea for our software implementation of the sLiSCP-light permutation is to separate the state of the permutation among different registers for performing the homogeneous operations. For instance, when four parallel instances of sLiSCP-light are evaluated using YMM registers, we pack data for the Simeck sbox operation into two YMM registers and other subblocks are stored in two other YMM registers. This allows us to perform the same kind of operations in different registers to achieve efficiency in the implementation.

Packing and Unpacking for sLiSCP-light**.** There are two different types of packing and unpacking operations in our implementation: 1) one pair is performed at the beginning and end of the permutation execution; and 2) the other one is performed at the beginning and end of the SSb layer in each step. We start by describing the first one. Let us denote the sLiSCP-light state by $S_i = s^i_0 s^i_1 s^i_2 s^i_3 s^i_4 s^i_5 s^i_6 s^i_7$ where each s^i_j is a 32-bit word, $0 \leq i \leq 3$ and $0 \leq j \leq 7$. We first load four independent states S_0, S_1, S_2, S_3 of sLiSCP-light into four 256-bit registers and then pack as follows:

$$\begin{aligned}
&\text{PACK}(R_0, R_1, R_2, R_3):\\
&R_0 \leftarrow s_0^0 s_1^0 s_4^0 s_5^0 s_0^1 s_1^1 s_4^1 s_5^1;\\
&R_1 \leftarrow s_0^2 s_1^2 s_4^2 s_5^2 s_0^3 s_1^3 s_4^3 s_5^3;\\
&R_2 \leftarrow s_2^0 s_3^0 s_6^0 s_7^0 s_2^1 s_3^1 s_6^1 s_7^1;\\
&R_3 \leftarrow s_2^2 s_3^2 s_6^2 s_7^2 s_2^3 s_3^3 s_6^3 s_7^3;
\end{aligned}$$

The unpacking operation, denoted by UNPACK(), is performed in the reverse order of the packing operation. Both operations are implemented using `vpermd` and `vperm2i128`, `vpunpcklqdq` and `vpunpckhqdq` instructions. Let us assume that we wish to apply the Simeck sbox on disjoint 64 bits (i.e., $a_{2i}a_{2i+1}$) in the registers $A = a_0a_1a_2a_3a_4a_5a_6a_7$ and $B = b_0b_1b_2b_3b_4b_5b_6b_7$. Due to the Feistel nature of the Simeck sbox, we regroup the data in A and B for the homogeneity of operations in the round function of the Simeck sbox. For this, we need the second pair of packing and unpacking operation for the SSb layer, which are given by

$$\begin{aligned}
&\text{PACK\_SSb}(A, B):\\
&A \leftarrow a_0a_2a_4a_6b_0b_2b_4b_6;\\
&B \leftarrow a_1a_3a_5a_7b_1b_3b_5b_7;
\end{aligned}
\qquad\qquad
\begin{aligned}
&\text{UNPACK\_SSb}(A, B):\\
&A \leftarrow a_0b_0a_1b_1a_2b_2a_3b_3;\\
&B \leftarrow a_4b_4a_5b_5a_6b_6a_7b_7;
\end{aligned}$$

Translating 4-Parallel sLiSCP-light **Instances.** In sLiSCP-light's design, the construction of Simeck sbox is based on the Feistel structure consisting of constant-distance left cyclic shift, bitwise AND and XOR operations. We create an instruction for one round execution of the Simeck sbox, denoted by ROAX, which is given by

$$\begin{aligned}
&\text{ROAX}(A, B, t_1, t_2):\\
&\mathsf{tmp} \leftarrow A;\;\; C \leftarrow \texttt{0xfffffffe};\\
&A \leftarrow (\text{ROT5}(A) \& A) \oplus \text{ROT1}(A);\\
&A \leftarrow A \oplus B \oplus (C \oplus t_1, C \oplus t_2, \cdots, C \oplus t_1, C \oplus t_2);\\
&B \leftarrow \mathsf{tmp};
\end{aligned}$$

where A and B are either a XMM or YMM register, ROT5(A) (resp. ROT1(A)) denotes the left cyclic shift by 5 (resp. 1) on every a_i in A, which is implemented using `vpslld` and `vpsrld` instructions. When $A = a_0a_1a_2a_3a_4a_5a_6a_7$, the swap block operation is defined as SWAPBLK(A) $= a_2a_3a_0a_1a_6a_7a_4a_5$. The evaluation of four parallel instances of sLiSCP-light is given in Algorithm 3.

For the SSE2 implementation, we use four 128-bit XMM registers to pack two parallel instances of the sLiSCP-light permutation. As the SSE2 implementation of sLiSCP-light is similar to the AVX2 implementation, the details are omitted. The implementation results of the two instances of sLiSCP-light on both platforms are presented in Table 6.1.

Algorithm 3 Four parallel instances of sLiSCP-light computation

```
1: Input: $(R_0, R_1, R_2, R_3)$
2: Output: $(R_0, R_1, R_2, R_3)$
3: $(R_0, R_1, R_2, R_3) \leftarrow \mathrm{PACK}(R_0, R_1, R_2, R_3)$;
4: for $j$ from 0 to $s-1$ do
   //SSb layer
5:     $R_2, R_3 \leftarrow \mathrm{PACK\_SSb}(R_2, R_3)$;
6:     for $i$ from 0 to $u-1$ do
7:         $R_2, R_3 \leftarrow \mathrm{ROAX}(R_2, R_3, rc_0^j[i], rc_1^j[i])$;        ▷ $rc_0^j[i]$ : $i$-th lsb of $rc_0^j$
8:     end for
9:     $R_2, R_3 \leftarrow \mathrm{UNPACK\_SSb}(R_2, R_3)$;
   //ASc layer
10:    $C \leftarrow$ 0xffffff00; $D \leftarrow$ 0xffffffff;
11:    $R_0 \leftarrow R_0 \oplus (D, C \oplus sc_0^j, D, C \oplus sc_1^j, D, C \oplus sc_0^j, D, C \oplus sc_1^j)$;
12:    $R_1 \leftarrow R_1 \oplus (D, C \oplus sc_0^j, D, C \oplus sc_1^j, D, C \oplus sc_0^j, D, C \oplus sc_1^j)$;
   //MSb layer
13:    tmp0 $\leftarrow R_0$; tmp1 $\leftarrow R_1$;
14:    $R_0 \leftarrow R_2$;
15:    $R_1 \leftarrow R_3$;
16:    $R_2 \leftarrow \mathrm{SWAPBLK}(R_2 \oplus \mathsf{tmp0})$;
17:    $R_3 \leftarrow \mathrm{SWAPBLK}(R_3 \oplus \mathsf{tmp1})$;
18: end for
19: return $(R_0, R_1, R_2, R_3) \leftarrow \mathrm{UNPACK}(R_0, R_1, R_2, R_3)$;
```

6.2 Spix Optimized Implementation

We implement the Spix in C using SSE2 and AVX2 instruction sets and measure its performance on two different Intel processors Skylake and Haswell. The code is compiled using gcc 5.4.0 on 64-bit machines with the compiler flags `-O2 -funroll-all-loops -march=native`. For AVX2 and SSE2 implementations, we evaluate 4 and 2 parallel instances of Spix, respectively, and compute the throughput for 1024-bit message and 128-bit associated data. Table 6.1 presents the performance results in cycles per byte for both implementations. In our implementation, we include the costs for all packing and unpacking operations.

6.3 Spix Microcontroller Implementation

We implement the 18 and 9-step sLiSCP-light (P^{18}, P^{9}) permutations and Spix on three distinct microcontroller platforms. For Spix, we implement only encryption as decryption is the same as encryption, except updating the rate with ciphertext. Our codes were written in assembly language to achieve optimal performances. We choose: 1) the Atmel ATmega128, an 8-bit mocrocontroller with 128 Kbytes of programmable flash memory, 4.448 Kbytes of RAM, and 32 general purpose registers of 8 bits, 2) MSP430F2370, a 16-bit mocrocontroller from Texas Instruments with 2.3 Kbytes of programmable flash memory, 128 bytes of RAM, and 12 general purpose

Table 6.1: Benchmarking the results for the two instances of the sLiSCP-light permutation and Spix. The throughput is measured in cycles per byte (cpb).

Cryptographic primitive	Speed [cpb]	Instruction Set	CPU Name Spec.
P^{18}	16.40	SSE2	Skylake
	10.19	AVX2	`Intel i7-6700 CPU@3.40GHz`
	19.28	SSE2	Haswell
	10.52	AVX2	`Intel i7-4790 CPU@3.60GHz`
P^{9}	8.63	SSE2	Skylake
	5.12	AVX2	`Intel i7-6700 CPU@3.40GHz`
	9.58	SSE2	Haswell
	5.47	AVX2	`Intel i7-4790 CPU@3.60GHz`
Spix	87.29	SSE2	Skylake
	46.87	AVX2	`Intel i7-6700 CPU@3.40GHz`
	94.00	SSE2	Haswell
	56.00	AVX2	`Intel i7-4790 CPU@3.60GHz`

registers of 16 bits, and 3) ARM Cortex M3 LM3S9D96, a 32-bit microcontroller with 524.3 Kbytes of programmable flash memory, 131 Kbytes of RAM, and 13 general purpose registers of 32 bits. We focus on four key performance measures, namely throughput (cycles/bit), code size (Kbytes), energy (nJ), and RAM (Kbytes) consumptions.

Spix is instantiated with a random 128-bit key and 128-bit nonce. Table 6.2 presents the performance of Spix on the previously mentioned platforms [20].

Table 6.2: Performance of Spix on microcontrollers at a clock frequency of 16 MHz

Cryptographic primitive	Platform	$\#AD$ blocks	$\#M$ blocks	Memory usage [Bytes]		Setup [Cycles]	Throughput [Kbps]	Energy/bit [nJ]
				SRAM	Flash			
P^{18}	8-bit ATmega128	-	-	161	1262	128377	31.91	3879
	16-bit MSP430F2013			24	1409	52294	78.33	211
	32-bits LM3S9D96			352	946	10900	375.78	887
Spix	8-bit ATmega128	0	16	175	1550	1667042	9.83	12591
	16-bit MSP430F2013			50	1845	677818	24.17	685
	32-bits LM3S9D96			408	1210	139569	117.39	2839
	8-bit ATmega128	2	16	175	1644	1795322	9.13	13560
	16-bit MSP430F2013			50	1891	730340	22.43	738
	32-bits LM3S9D96			424	1326	150313	109.00	3058

Acknowledgment

The submitters would like to thank Yunjie Yi for the microcontroller implementation of Spix.

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., Jha, A., Mandal, K., Nandi, M., and Rohit, R. SpoC: 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. SLISCP-light: Towards hardware optimized sponge-specific cryptographic permutations. *ACM Trans. Embeded Computing Systems 17*, 4 (2018), 81:1–81:26.

[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] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., and Wingers, L. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.

[7] 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, 2017.

[8] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. Caesar submission: Ketje v2, 2014. http://ketje.noekeon.org/Ketjev2-doc2.0.pdf.

[9] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Sponge functions. In *ECRYPT hash workshop* (2007), vol. 2007.

[10] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. On the security of the keyed sponge construction. In *Symmetric Key Encryption Workshop* (2011).

[11] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *SAC* (2012), A. Miri and S. Vaudenay, Eds., Springer, pp. 320–337.

[12] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Permutation-based encryption, authentication and authenticated encryption. *DIAC* (2012).

[13] Biryukov, A., and Wagner, D. Slide attacks. In *FSE* (1999), L. Knudsen, Ed., Springer, pp. 245–259.

[14] Chakraborti, A., Datta, N., Nandi, M., and Yasuda, K. Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems 2018*, 2 (2018), 218–241.

[15] Jovanovic, P., Luykx, A., and Mennink, B. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In *ASIACRYPT* (2014), P. Sarkar and T. Iwata, Eds., Springe, pp. 85–104.

[16] 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.

[17] 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.

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

[19] 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.

[20] Yi, Y., and Gong, G. Implementation of three LWC Schemes in the WiFi 4-Way Handshake with Software Defined Radio. To appear in NIST LWC Workshop 2019. Also available at https://arxiv.org/abs/1909.11707.

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. 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. 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. In Table A.1, the submissions are classified based on their functionalities, mode of operation parameters and ASIC CMOS 65 *nm* hardware area.

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	sLiSCP-light-256	AEAD	256	64	128	Monkey Duplex	2611
SpoC-64 [2]	sLiSCP-light-192	AEAD	192	64	128	SpoC	2329
SpoC-128 [2]	sLiSCP-light-256	AEAD	256	128	128	SpoC	3054

Appendix B

Test Vectors

B.1 Simeck Sbox

Test vector for Simeck sbox with input = 0000000000000000 and $rc = 0x07$.

Round	State
0	0000000000000000
1	FFFFFFFF00000000
2	FFFFFFFFFFFFFFFF
3	00000000FFFFFFFF
4	0000000100000000
5	FFFFFFFC00000001
6	FFFFFF9AFFFFFFFC
7	00000C2DFFFFFF9A
8	00001C1E00000C2D

B.2 sLiSCP-light Permutation

Table B.1: Test vector for sLiSCP-light 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 129530024935 1E2E
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.3 Spix

Key	00111122335588DD 00111122335588DD
Nonce	111122335588DD00 111122335588DD00
Associated data	1122335588DD0011 1122335588DD00
Plaintext	335588DD00111122 335588DD001111
Ciphertext	4FEF0A8A5681A6D8EEC67E0B450F95
Tag	58B18A5FA8A59353D8F160B0A2019A23