sLiSCP-light: Towards Lighter Sponge-specific Cryptographic Permutations

Riham AlTawy, Raghvendra Rohit, Morgan He, Kalikinkar Mandal, Gangqiang Yang and Guang Gong

Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, N2L 3G1, CANADA.
{raltawy, rsrohit, myhe, kmandal, g37yang, ggong}@uwaterloo.ca

Abstract. In this paper, we revisit the design approach of the sLiSCP family of lightweight cryptographic permutations which was proposed in SAC 2017. sLiSCP is designed to be used in a unified duplex sponge construction to provide minimal overhead for multiple cryptographic functionalities within one design. The design of sLiSCP follows a 4-subblock Type-2 Generalized Feistel-like Structure (GFS) with unkeyed round-reduced Simeck as the round function which are extremely efficient building blocks in terms of their hardware area requirements. By tweaking the GFS design, we turn it into an elegant Partial Substitution-Permutation Network construction to present sLiSCP-light, which further reduces the hardware areas of the sLiSCP permutations by around 16% of their original values. The new design also enhances the bit diffusion and algebraic properties of the permutations, and enables us to reduce the number of steps, thus achieving a better throughput in both the hashing and authentication modes. We perform a thorough security analysis of the new design with respect to its diffusion, differential and linear, and algebraic properties. For sLiSCP-light-192, we report parallel implementation hardware areas of 1820 (resp. 1892) GE in CMOS 65 nm (resp. 130 nm) ASIC. The areas for sLiSCP-light-256 are 2397 and 2500 GE in CMOS 65 nm and 130 nm ASIC, respectively. Overall, the unified duplex sponge mode of sLiSCP-light-192 which provides (authenticated) encryption and hashing functionalities, satisfies the area (1958 GE), power (3.97 μW), and throughput (44.4 kbps) requirements of passive RFID tags.

Keywords: Lightweight cryptography · Cryptographic permutations · Simeck block cipher · Sponge duplexing · Partial Substitution and Permutation Network (PSPN).

1 Introduction

Ever since the introduction of the Sponge-based permutation dependent functions [BDPVA07] and the ability of such a construction to provide almost all of the major cryptographic functionalities, there has been a natural inclination towards designing cryptographic permutations. Starting from the Keccak family of permutations [BDPVA09], and later Ascon [DEMS16], Norx [AJN14], Simpira [GM16], and Gimli [BKL+17], all such proposals have the conventional aspect that designing a cryptographic permutation is better than a specific cryptographic primitive. While Ascon and Norx are used to instantiate a MonkeyDuplex sponge construction [BDPVA12] that is optimized for authenticated encryption (AE) only, other proposals such as Simpira and Gimli focus mainly on the permutation design and only suggest application modes. What is clearly common among most of the permutation designs is that they either have sufficiently large state sizes (≥ 320 bits) which directly translate to large hardware areas, or are optimized for software to make use of processor specific instructions (e.g., `Gimli` and `Simpira`). Lightweight AE schemes `Norx`-8 and `Norx`-16 with internal state sizes of 128 and 256 bits have lower bounded

estimated areas of 1368 and 2880 GE, respectively. However, both these `Norx` instances are specifically optimized for authenticated encryption that offer 80-bit and 96-bit security, respectively, and their security and instantiation for unkeyed modes are not investigated in the literature.

In light of NIST's lightweight cryptography project [MBSTM17] that recognizes the apparent lack of cryptographic standards suitable for the whole spectrum of lightweight applications, several proposals have emerged. However, all of such proposals offer a single cryptographic functionality within the constrained hardware area (around 2000 GE [JW05]) dedicated for all security purposes. Examples of these algorithms are either block ciphers such as `Led` [GPPR11], `Present` [BKL+07], `Simon` and `Speck` [BSS+13], `Simeck` [YZS+15], `Skinny` [BJK+16], and `Gift` [BPP+17] or lightweight hash functions such as `Photon` [GPP11], `Quark` [AHMNP13], and `Spongent` [BKL+11].

In SAC 2017, the sLiSCP family of lightweight cryptographic permutations [ARH+17] was proposed specifically to address the limited hardware area which is dedicated for all security purposes in resource constrained devices. More precisely, the authors of sLiSCP stressed that for such devices, it is desirable (if not only realistic) that a cryptographic design should provide low overhead for multiple cryptographic functionalities including (authenticated) encryption, hashing, and pseudorandom bit generation. Hence, sLiSCP is proposed to be used in the unified sLiSCP duplex sponge construction to provide (authenticated) encryption and hashing functionalities. sLiSCP aims to provide an efficient and secure design for a sponge-specific permutation taking into perspective the relation between the state size and security parameters of both keyed and unkeyed modes. In doing so, sLiSCP adopts two of the most efficient and extensively cryptanalyzed constructions; a 4-subblock Type-2 GFS and unkeyed round-reduced Simeck as the two GFS round functions. sLiSCP offers two instances of the permutation with block sizes 192 and 256 bits with fully parallelized hardware areas of 2153 and 2833 GE in CMOS 65 *nm* ASIC, and 2318 and 3040 GE in CMOS 130 *nm* ASIC, respectively.

1.1 Our Contributions

In this work, we revisit the design approach of the sLiSCP family of permutations with the main aim of further reducing its hardware area. We propose the sLiSCP-light family of permutations by adopting a new strategy that not only reduces the hardware area of sLiSCP but also enhances both the bit diffusion and algebraic properties of the original sLiSCP permutations. As a result, the number of steps required for sufficiently good security margins is reduced by 30%, which increases the throughput by the same percentile. Our new design reaches the limit where around 80% of the permutation parallel implementation GE area is attributed to the area consumed by the state storage, which is lower bounded by the sponge security constraints [BDPV14] and the Simeck-based Sboxes which are extremely hardware efficient for their sizes. Our goal is to design a hardware efficient sponge-specific cryptographic permutation. The contributions of this paper are summarized as follows.

Hardware optimized sponge-specific permutation. We propose a new design, the sLiSCP-light family of permutations, by tweaking the original Type-2 GFS design of sLiSCP. Our tweak gives up some of the desired features of the generic Feistel constructions which we do not make use of when the permutation is used in a sponge construction. Our adopted approach turns the Type-2 GFS into an elegant Partial SPN (PSPN) construction where the substitution layer updates half the state only and the permutation layer mixes the whole state resulting in a fully nonlinearly updated state after one step only (vs. half state as in sLiSCP). Iterated version of round-reduced unkeyed version of the Simeck encryption algorithm [YZS+15] are used as large Sboxes in the substitution layer.

Thorough security analysis of sLiSCP-light**.** We assess the security of sLiSCP-light in terms of its diffusion, differential and linear, and algebraic properties. We evaluate the

number of steps required for full bit diffusion and heuristically estimate the avalanche effect corresponding to individual bit flips. Accordingly, we choose the number of steps to be equal to three times the number of steps required for full bit diffusion which offers safe margin against meet/miss-in-the-middle distinguishers. We further use the SMT/SAT tool proposed in [KLT15], develop a Mixed Integer Linear Programming (MILP) model of our design, and run a parallelized exhaustive search to get exact differential probabilities for the constant-based Simeck Sboxes which further tightens the expected maximum differential and linear characteristics probabilities. Despite the cryptanalytic results on the differential behavior of PSPN cipher designs [BODD+15, RASA14], our cryptanalysis shows that sLiSCP-light is adequate for the use in duplex sponge constructions where access to the input state is limited to the absorbing part of the state (i.e., rate bits). Finally, we run a MILP-based division property [Tod15, BKL+17] test to get an exact evaluation of the algebraic degree of the component functions after two steps, and give upper bounds for the remaining steps.

Competitive implementation results. We implement a fully parallelized architecture for the sLiSCP-light-192 (resp. sLiSCP-light-256) permutation and report a hardware area in CMOS 65 nm of 1820 (resp. 2397) GE, and in CMOS 130 nm of 1892 (resp. 2500) GE. Moreover, we implement the unified sLiSCP duplex sponge mode using sLiSCP-light-192 and sLiSCP-light-256 permutations in CMOS 65 nm to provide (authenticated) encryption and hashing functionalities with hardware areas of 1958 and 2603 GE, respectively. The respective areas in CMOS 130 nm are 2053 and 2731 GE. The unified design using sLiSCP-light-192 and sLiSCP-light-256 permutations has a throughput of 44.44 and 66.67 kbps, respectively. We compare our results to existing permutations with nearly comparable state sizes as shown in Table 1 which lists our smallest implementations of the two instances of sLiSCP-light before the place and route in CMOS 65nm and 130nm ASICs. In addition, the corresponding throughput and power consumption after the place and route are also provided. Furthermore, we implement the sLiSCP-light-192 and 256 permutations using SSE2 and AVX2 instruction sets and measure their performances on Intel Skylake and Haswell processors. The best speed achieved by the AVX2 implementation on Skylake is 4.32 cycles/byte for sLiSCP-light-192 and 4.97 cycles/byte for sLiSCP-light-256.

2 Tweak Approach

In this section, we first identify an important aspect in the sLiSCP design where an extra hardware overhead is unjustified. Next, we explore and contrast new design options by which we can avoid the identified overhead and finally detail the tweaking approach which we have adopted in the new design of sLiSCP-light.

2.1 Extra Hardware Overhead of the sLiSCP Design

sLiSCP adopts a 4-subblock Type-2 GFS construction (see left side of Figure 1), which is like other generic Feistel constructions offers the following features: 1) no constraints on the bijectivity of F; and 2) low overhead for the inverse round function implementation. By investigating the parallel round-based hardware implementation of sLiSCP (cf. Section 7 in [ARH+17]), we noticed that, for a b-bit state, there are two registers of size $b/4$ bits each, that are used as temporary storage. This half state of temporary storage is required because of the iterative nature of the two Simeck boxes (F in Figure 1). More precisely, the application of the two Simeck boxes requires the use of two extra $b/4$-bit registers to perform iterated updates on the input, and keep the intermediate and initial values of the registers at the same time. Initial values of such two registers are required to update the state of the following step through the linear cyclic shift permutation of Type-2 GFS. Since sLiSCP utilizes a u-round iterated unkeyed Feistel-based Simeck-m, where m is the block size, round function as the GFS F which is extremely hardware efficient (est. area of around 168 (resp.224) GE for Simeck-48 (resp. Simeck-64) boxes), an addition of

Table 1: Comparison of the parallel hardware implementation results of sLiSCP-light permutations with other existing designs. Throughput and power are given at 100 kHz. Gray colored cells denote the estimated area obtained after subtracting the area of the module for the sponge process.

Permutation	**State size** (bits)	**Technology** (nm)	**Area** (GE)	**Cycles**	**Throughput**† (kbps)	**Power** (μW)
sLiSCP-light	192	65	1820	72	266.66	3.97
		130	1892			5.05
	256	65	2397	96		4.77
		130	2500			7.27
sLiSCP [ARH+17]	192	65	2153	108	177.77	4.62 ‡
		130	2318			7.44 ‡
	256	65	2833	144		5.88 ‡
		130	3040			8.75 ‡
`Photon` [GPP11]	196	180	1949	180	108.88	4.35
	256		2637	204	125.49	6.5
`Spongent` [BKL+11]	176	130	2110	90	195.55	4.47
	240		2739	120	200.00	6.8
`Quark` [AHMNP13]	176	180	2739	88	200.00	4.76
	256		4480	64	400.00	8.39
`Keccak` [KY10]	200	130	4540	18	1111.11	27.6
`Norx-16` [AJN15]	256	-	2496 ⋆	-	-	-
`Ascon` [GWDE15]	320	90	7080	12	26.66 ††	43
`Gimli` [BKL+17]	384	28	8097	24	1600.00	-
		180	5314		1600.00	-

† Given by $\frac{\text{State size}}{\#\text{Cycles}} \times 100$
‡ Our implementation results
†† Unit of throughput is Mbps as frequency is 1 MHz
⋆ Designers gave the estimated area only, no implementation results are available

around 400 (resp. 500) GE for temporary storage is not justified, especially for resource constrained applications.

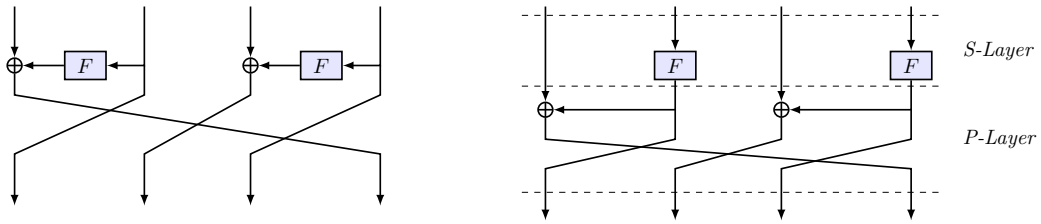
2.2 Solution Space Exploration

The new design of sLiSCP-light was triggered by the above observation and the need for finding an answer to the following question: *"how can we get rid of these two extra registers?"*. We primarily wanted to keep the structure of the iterated Simeck-m box as the nonlinear component because it is extremely efficient in hardware as a large Sbox that encompasses both the nonlinear and linear/permutation mixing. Moreover, we can leverage the available extensive cryptanalysis on Simeck and Simon-like functions to derive the cryptographic properties of such a large Sbox. Accordingly, we found that the following two solutions enable us to remedy the sLiSCP temporary storage problem.

- **Unrolled implementation.** For a u-round iterated Simeck-m box, a trivial solution would be to implement u sequential Simeck round function blocks so that no intermediate storage is required. More precisely, the output of one Simeck round is directly fed to the following one without storing it and by the end of the clock cycle, both the output of the u-rounds and initial state values are available for linear mixing and updating the state for the following step. However, we found out that although each round of Simeck-m costs around 168 (resp. 224) GE for Simeck-48 (resp. Simeck-64), the total hardware

area for implementing u blocks of each round surpasses the hardware areas of the two registers which we are trying to save. Note that $u = 6$ (resp. $u = 8$) is the number of Simeck-48 (resp. Simeck-64) rounds used by the designers of sLiSCP to provide the claimed security arguments.

- **Tweaking the original design.** To discard the two temporary registers, we eliminate the need for storing the initial values of the odd indexed subblocks. Since the final values are the output of the u-round iterated Simeck-m box that is the only nonlinear component in the step function of the permutation, we cannot remove it. Accordingly, as depicted in Figure 1, we opted for tweaking the original 4-subblock Type-2 GFS design so that the initial values of the odd indexed subblocks are not used in updating the even indexed subblocks in the next step. Our tweak is as follows. We use the nonlinearly updated values, by the Simeck-box, to update the even indexed subblocks in the next step. The new tweaked design can be interpreted as a Partial Substitution and Permutation Network (PSPN), which can also be viewed as a mix between Skipjack Rule A [BBW14] and a Type-2 GFS function. In particular, we have a substitution layer that operates only on odd indexed subblocks (starting from index 0) followed by a permutation layer that mixes all the four subblocks resulting in a fully nonlinear-updated state. Below we describe some advantages of our tweaked design.



(a) sLiSCP (b) sLiSCP-light

Figure 1: Block diagrams of (a) the original 4-subblock Type-2 GFS used in sLiSCP, and (b) the new partial SPN design of sLiSCP-light.

2.3 Advantages of Partial SPN

A partial SPN is a special instance of the generic SPN where the S-layer consists of Sboxes that are applied to a part of the state, followed by a P-layer that linearly mixes the whole state resulting in a full state that is updated nonlinearly. An AES-like partial substitution layer is utilized in Zorro [GGNPS13] to enable efficient masking, and a conditional one is adopted in the LowMC cipher [ARS+15] to reduce the multiplicative complexity in the round function. By adopting our new design tweak, we give up some advantageous features of the Feistel constructions, for instance, efficient invertability using a non-bijective function. Since the u-round unkeyed Simeck-m box is bijective, it can be easily shown that the round function of the tweaked design is also invertible. For our tweaked design, we do not need to consider the efficiency of inverting the sLiSCP-light permutation as the permutation is designed to be used, specifically, in the sLiSCP unified duplex sponge mode [ARH+17] in which during both encryption and decryption operations, the permutation is always evaluated in the forward direction. The advantages of adopting the new partial SPN design are twofold:

- **Hardware efficient sponge-specific permutations.** The hardware implementation results provided in Section 5 shows that compared to sLiSCP, sLiSCP-light has around 400 and 500 GE savings in the parallel hardware round-based implementation of both sLiSCP-light-192 and sLiSCP-light-256 instances, respectively. In fact, the new design

makes the unified duplex sponge mode of sLiSCP-light-192 satisfy the area (1958 GE), power (3.97 μW), and throughput (44.4 kbps) requirements of passive RFID tags, and provide (authenticated) encryption and hashing functionalities all at the same time.

- **Enhanced security.** According to our analysis (see Section 4), higher diffusion is achieved by the new design as full bit diffusion is reached in 4 steps (vs. 6 steps in sLiSCP) and accordingly, the total number of steps is reduced to 12 (vs. 18 in sLiSCP) which results in 30% increase in the throughput of all instances of sLiSCP-light. Moreover, the step degree grows more uniformly as all the subblocks are updated nonlinearly, which inhibits integral (division property) and zero-sum distinguishers from covering as many steps as in sLiSCP (1 or 2 steps less).

3 Specification of sLiSCP-light

The sLiSCP-light family of permutations is a family of iterated permutations based on the partial SPN construction illustrated in Figure 1. In this section, we describe the design of the sLiSCP-light family of permutations.

3.1 Step Function of the Permutation

An s-step sLiSCP-light permutation takes an input of b bits from $\mathbb{F}_2^b$ and produces an output of b bits after applying the step function s times sequentially where $b = 4 \times m$ and m is an even positive integer. 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. The state of the permutation is divided into 4 m-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.

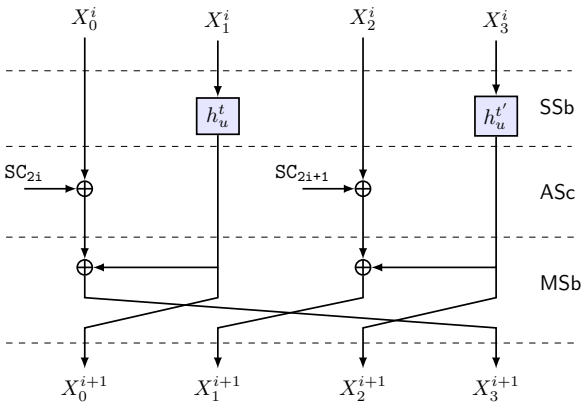


Figure 2: sLiSCP-light permutation step.

3.1.1 `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 box (henceforth referred to as Simeck^u-m or h_u^t) to the odd indexed subblocks only. The SSb transformation is defined as

$$\mathsf{SSb}(X_0^i, X_1^i, X_2^i, X_3^i) = (X_0^i, h_u^t(X_1^i), X_2^i, h_u^{t'}(X_3^i))$$

where h_u^t is the Simeck box applied on $m = \frac{b}{4}$ bits and t is a u-bit constant. Below we provide some details about the Simeck box h_u^t.

Definition 1 (Simeck^u-m box [ARH+17])**.** *A Simeck^u-m box is a permutation of m-bit input constructed by iterating the Simeck-m cipher round function for u rounds with round constant addition in place of key addition. The nonlinear operation of such an Sbox is provided by iterating a simple AND operation followed by bitwise shifts and XORs for u rounds.*

An illustrated description of the Simeck^u-m box is shown in Figure 3 and is given by:

$$(x_{2u}, x_{2u+1}) \leftarrow h_u^t(x_0, x_1) = h_{u-1}^{t_{u-1}} \circ \ldots \circ h_1^{t_1} \circ h_0^{t_0}(x_0, x_1),$$

where $h_i^{t_i} : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^m$ is given by $h_i^{t_i}(x_{2i}, x_{2i+1}) = (f(x_{2i}) \oplus x_{2i+1} \oplus (\mathbf{1}^{m/2-1}||t_i), x_{2i})$, and $f(x) = ((x \lll 5) \odot x) \oplus (x \lll 1)$ and $t_i \in \mathbb{F}_2$ and $0 \leq i < u$. We denote the bitwise AND by $\odot$ and an $m/2$ dimensional vector of the form $(1, 1, \ldots, 1, t_i)$ by $\mathbf{1}^{m/2-1}||t_i$. If t is the integer representation of the u-tuple $(t_0, t_1, \cdots, t_{u-1})$, then we get a set of Simeck^u-m boxes parameterized by the added round constants t, hence t in h_u^t. We use the same round constants as used in sLiSCP. Round constants are added to the Simeck rounds to mitigate the preservation of rotational properties between the Sbox inputs and outputs.

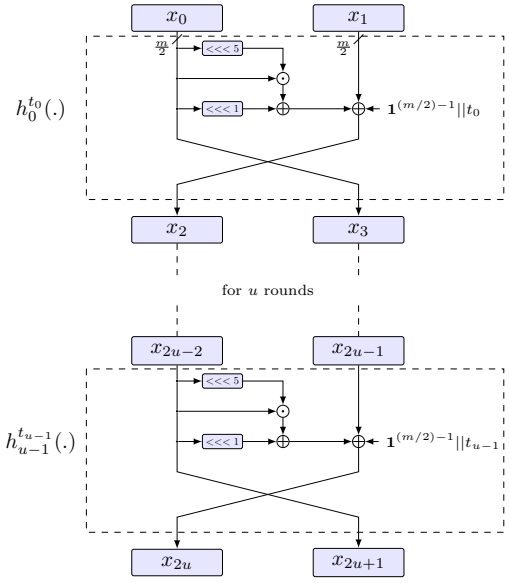


Figure 3: A block diagram of Simeck^u-m box structure.

3.1.2 `AddStepconstants` (ASc)

In this layer, the step constants $\mathtt{SC_{2i}}$ and $\mathtt{SC_{2i+1}}$ are XORed with the two even indexed subblocks X_0 and X_2, respectively, $i = 0, 1, \ldots s-1$. Each $\mathtt{SC_j}$ is an m-bit constant of the form $\mathbf{1}^{m-6}||0^2||sc_j$ (resp. $\mathbf{1}^{m-8}||sc_j$) for Simeck^u-48 (resp. Simeck^u-64), where sc_j is 6 (resp. 8)-bit constant generated by an LFSR. The ASc transformation is given by

$$\mathsf{ASc}(X_0^i, h_u^t(X_1^i), X_2^i, h_u^{t'}(X_3^i)) = (X_0^i \oplus \mathtt{SC_{2i}}, h_u^t(X_1^i), X_2^i \oplus \mathtt{SC_{2i+1}}, h_u^{t'}(X_3^i)).$$

Considering the pair of 6 or 8-bit constants (sc) and the round constants that are used in the Simeck^u-m box, all $2s$ constant pairs (t, sc) are unique due to the periodicity of the m-sequence generated by the LFSR [ARH+17]. The addition of these step constants is essential to thwart slide [BW99] and invariant subspace [LAAZ11] distinguishers.

3.1.3 `MixSubblocks` (MSb)

This layer applies the linear transformation that is used in the Type-2 GFS 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

$$(X_0^{i+1}, X_1^{i+1}, X_2^{i+1}, X_3^{i+1}) \leftarrow \mathsf{MSb}(X_0^i \oplus \mathtt{SC_{2i}}, h_u^t(X_1^i), X_2^i \oplus \mathtt{SC_{2i+1}}, h_u^{t'}(X_3^i)),$$

where

$$\begin{aligned} X_0^{i+1} &= h_u^t(X_1^i), & X_1^{i+1} &= X_2^i \oplus h_u^{t'}(X_3^i) \oplus \mathtt{SC_{2i+1}}, \\ X_2^{i+1} &= h_u^{t'}(X_3^i), & X_3^{i+1} &= X_0^i \oplus h_u^t(X_1^i) \oplus \mathtt{SC_{2i}}. \end{aligned}$$

The output of MSb has more bit diffusion than that of the original Type-2 GFS and more uniform degree distribution. In particular, after one step, all the components functions of all the subblocks have a degree equal to that of the Simeck^u-m box. Whereas in the Type-2 GFS, after one step, two even indexed subblocks have degree equals one as they are directly copied from the odd indexed subblocks. Moreover, if the output of the Simeck^u-m box has x bit diffusion, then two subblocks have x bit diffusion and the other two subblocks have $(x+1)$ bit diffusion. In the case of Type-2 GFS, two subblocks have one bit diffusion and the remaining subblocks have $(x+1)$ bit diffusion. Accordingly, the sLiSCP-light enhances the security of the whole permutation and hinders the extension of most of the distinguishers to more than 8 steps (vs. 9 steps in sLiSCP) as shown in Section 4.

3.2 sLiSCP-light Permutation Instances

sLiSCP-light offers two lightweight instances, named sLiSCP-light-192 and sLiSCP-light-265, with state sizes 192 and 256 bits, respectively. Both instances adopt a PSPN step function that is iterated for $s = 12$ times. Simeck^u-48 and Simeck^u-64 boxes, where $u = 6$ and 8, are used as Sboxes in the SSb layer of sLiSCP-light-192 and sLiSCP-light-256, respectively. We keep the number of rounds u equal to 6 (resp. 8) for Simeck^u-48 (resp. Simeck^u-64) because it has been shown in [ARH+17] that these parameters provide a good balance between the permutation throughput and differential and algebraic properties. We refer to one PSPN (resp. Simeck box) iteration by one step (resp. one round). Table 2 presents the recommended parameters for two lightweight instances of the sLiSCP-light permutation.

Table 2: Recommended parameter set for sLiSCP-light-192 and sLiSCP-light-256 permutations.

Permutation (b-bit)	Sbox size m	Rounds u	Steps s	Total # rounds ($u \cdot s$)
sLiSCP-light-192	48	6	12	72
sLiSCP-light-256	64	8	12	96

4 Security Analysis

In this section, we analyze the security of the sLiSCP-light permutation by assessing its behavior 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.

4.1 Diffusion

We assess the diffusion behavior of sLiSCP-light by carrying out two experiments on full bit diffusion and avalanche effect properties.

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 (similar to the optimum Type-2 GFS full subblock diffusion [SM10]). These results are better than the ones for sLiSCP using the original GFS structure which allows us to comfortably reduce the number of steps for the permutation.

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. We follow a similar approach as in [BKL+17], 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. Tables 8 (resp. 9) in Appendix A depict 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), where the average number of changes varies between 95.13 and 96.56 (resp. 126.98 and 128.90).

It can be seen that the new design of sLiSCP-light offers a better bit diffusion than the original sLiSCP which is attributed to the additional m-bit mixing of the Simeck^u-m box that affects two extra subblocks. Given 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 backward and forward directions.

4.2 Differential and Linear Cryptanalysis

In this section, we first analyze the differential and linear properties of the Simeck^u-m boxes and then use a MILP model for the sLiSCP-light permutation to bound the minimum number of differentially and linearly active Simeck^u-m boxes an present evaluation for the expected maximum probabilities of differential and linear characteristics.

4.2.1 Differential properties of Simeck^u-m

In the original sLiSCP, the authors used the SAT/SMT tools proposed in [KLT15] to derive estimates for the Maximum Differential Probabilities (MDP) of Simeck^6-48 and Simeck^8-64 boxes. However, their method adopts the conventional Markov assumption, thus ignoring the effect of the constants similar to keyed ciphers. In our analysis, we derive tighter estimates for the MDP of the constant-based Simeck^u-m boxes by running an exhaustive parallel search on specific differentials that are associated with characteristics with optimal probabilities. More precisely, if $\Pr(\delta_{in} \rightarrow \delta_{out})$ denotes the differential probability given by $\Pr(\delta_{in} \rightarrow \delta_{out}) = \frac{|\{x|f(x)+f(x+\delta_{in})=\delta_{out}\}|}{2^m}$, where δ_{in} (resp. δ_{out}) denotes an m-bit input (resp. output) difference and $f : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^m$, then the optimal probability of a differential characteristic is defined as follows.

Definition 2 (Optimal differential characteristic probability [KLT15, ARH+17])**.** *Let f^u denote the u-fold iteration of a keyed round function f where the key is picked uniformly and independently at random. Let Δ^u be the set of all differential characteristics of f^u with probability $p > 0$. For f^u, the optimal differential characteristic probability is given by:* $\max_{(\delta_0 \rightarrow \cdots \rightarrow \delta_u) \in \Delta^u} \Pr(\delta_0 \rightarrow \delta_u)$.

MDP analysis for Simeck^u-m boxes. Our analysis is based on two important observations on the differential properties of different block sizes of the Simeck round function.

First, the probabilities of optimal differential characteristics are exactly equal for reduced-round Simeck with block sizes 32, 48, and 64 bits [LLW17]. Second, some differentials which are associated with optimal characteristics in Simeck-32 are also associated with optimal characteristics in Simeck-48 and Simeck-64 (we only add zeros at specific positions to match the block sizes). While, the authors in [ARH+17] estimated a tighter value for the MDP of Simeck^u-m boxes by considering the differential effect, we adopt the following procedure to further tighten it and get exact differential probabilities of the *constant dependent* Simeck^u-m boxes.

- For Simeck^6-32 (resp. Simeck^8-32), we use the SAT/SMT tool [KLT15] and find all differential characteristics with optimal probabilities, then we extract all the differentials which are associated with these characteristics. We obtain 3072 (resp. 2560) differentials that are expected to have the maximum probability since they are extracted from optimal characteristics [GM16, DPU+16].

- For each constant-dependent Simeck^6-32 (resp. Simeck^8-32) box, we calculate the exact differential probability of all the 3072 (resp. 2560) extracted differentials. We run a parallel exhaustive search for the number of solutions (out of the 2^{32} possible inputs) that satisfy each differential, then set the MDP for each investigated Simeck^6-32 (resp. Simeck^8-32) to the maximum probability among all the 3072 (resp. 2560) tested differentials. Table 3 gives the number of differentials out of 3072 (resp. 2560) for each round constant parametrized Simeck^6-32 (resp. Simeck^8-32) box that have maximum probability and their associated MDP (given in $\log_2(.)$ scale).

Table 3: Expected Maximum differential probabilities (MDPs) of Simeck^u-32 box parametrized by t, where $u = 6$ or 8, and the round constants are the same as used in sLiSCP permutations.

Rounds u	Round constants (t)	#differentials with MDP	MDP ($\log_2(.)$)
6	7, 27, 4, 6, 25, 26, 24, 34 17, 35, 15, 19, 8, 38, 3b, a	4	-10.348948
	12, 20	4	-10.354342
	2e, 1c, 1f, c, 2f, 3f	64	-10.376119
8	b	1	-15.401482
	f, 47, 4, 96, 7 b2, a1, f1, 73	1	-15.535678
	78	1	-15.544898
	44, 4c	1	-15.575210
	43, 82	2	-15.586901
	e5	2	-15.689352
	b5, 37, f5, ee	1	-15.721551

We notice that we obtain six (resp. eleven) sets of round-constants parametrized Simeck^6-32 (resp. Simeck^8-32) boxes where each set has the exact number and value of the differentials with maximum probabilities (See Tables 10 and 11 in Appendix B). Given that the optimal differential characteristic probability of a u-round Simeck-m box is equal when the block sizes are 32, 48, and 64 bits, we pick one differential candidate[1] from each set of Simeck^6-32 (resp. Simeck^8-32) and check if it is one of the differentials associated with the extracted optimal characteristics that we extracted for Simeck^6-48 (resp. Simeck^8-48). For these selected differentials, we run a parallel exhaustive search to get their exact probability on the round-constant parametrized Simeck^6-48 and Simeck^8-48 boxes. All the experiments are conducted on a server with the following specifications: 8

[1]Differentials marked with blue color in Appendix B

cores per node, 16 GB RAM per node, Intel Xeon E5540 @2.53 GHz, 64 bit Linux Centos 6.4 OS. Table 4 shows the probabilities of the differentials that we have selected from the sets in Tables 10 and 11 in Appendix B. Note that we do not utilize Simeck^8-48 in sLiSCP-light but we use its results to conjecture the MDP of Simeck^8-64.

Table 4: Differential probabilities of selected differentials on specific t-parametrized Simeck^u-48, where $u = 6$ or 8.

Rounds u	Round constants (t)	Selected differential ($\Delta_{in}, \Delta_{out}$)	MDP ($\log_2(.)$)
6	27	(0400000a0000, 1a00000a0000)	-10.655704
	15	(000400000a00, 011a00000a00)	-10.655704
	8	(0a00001a0000, 0a0000040000)	-10.655704
	3b	(0a00001a0001, 0a0000040000)	-10.655704
	12	(400001400023, 400001800000)	-10.655704
	2e	(000200000500, 008900000500)	-10.655704
8	b	(0a0000120001, 3a0000100000)	-15.86146
	f	(00a00011a000, 02a000010000)	-15.92975
	b2	(01000002a000, 01800000a000)	-15.95349
	f1	(01000002a000, 11a00000a000)	-15.92975
	78	(1000002a0000, 5a00010a0000)	-15.94942
	4c	(00a000118000, 02a000010000)	-15.93096
	43	(00a00011a000, 02a000010000)	-15.92936
	82	(00a00011a000, 02a000010000)	-15.92936
	e5	(800000500001, d00008500000)	-15.92698
	b5	(800000500001, d00008500000)	-15.91479
	ee	(500000d00008, 500001800000	-15.91479

The constant-parameterized Simeck^u-m probabilities are important in the sense that they give the expected maximum probability of a differential path that activates known Simeck^u-m boxes. However, for the expected upper bound of the probability of a differential characteristic over the 12 steps of permutation, we use $2^{-10.655704}$ as the MDP for all Simeck^6-48 because all of them have equal MDPs. We notice that the expected MDP for u-round iterated Simeck boxes with 48-bit block size is slightly lower than that of 32-bit block size. Since it is computationally infeasible for us to run the exhaustive search on 64-bit blocks, we conjecture that the MDP of Simeck^8-64 box is also slightly lower than that of Simeck^8-48 and, so we set it to be equal to the one we extracted for Simeck^8-48 which is equal to $2^{-15.86146}$. Accordingly, we can get an expected MDP bound (not strong) for Simeck^8-64.

4.2.2 Linear properties of Simeck^u-m

There is a duality between optimal differential and squared correlation characteristics probabilities of Simeck^u-m boxes [KLT15]. More precisely, the squared correlation of f with input mask τ_{in} and output mask τ_{out} is defined by $C^2(\tau_{in} \to \tau_{out}) = \left(\frac{\tilde{f}(\tau_{in},\tau_{out})}{2^m}\right)^2$, where $\tilde{f}(\tau_{in}, \tau_{out}) = \sum_{x \in \mathbb{F}_2^m} (-1)^{(\langle x,\tau_{in}\rangle \oplus \langle f,\tau_{out}\rangle)}$ and $\langle x, y\rangle$ denotes the inner product between vectors x and y. The optimal linear characteristic squared correlation is defined analogous to the optimal differential characteristic probability. Accordingly, we run the exact above procedure to find the expected maximum linear squared correlation (MLSC) for both Simeck^6-48 and Simeck^8-64 boxes. We set the expected MLSC of Simeck^6-48 and Simeck^8-64 to $2^{-10.83007}$ and $2^{-15.64046}$, respectively.

4.2.3 Expected maximum probabilities of differential and linear characteristics for sLiSCP-light

We use a MILP model to find the minimum number of differentially and linearly active Simeck^u-m boxes in 12 steps. Our MILP model bounds such a number to 12 and consequently, the expected maximum differential probability/linear squared correlation characteristics are given by:

$$\begin{aligned}
\text{sLiSCP-light-192:} \quad & (\text{MDP}(\text{Simeck}^6\text{-}48))^{12} = (2^{-10.655704})^{12} = 2^{-127.868} \\
& (\text{MLSC}(\text{Simeck}^6\text{-}48))^{12} = (2^{-10.83007})^{12} = 2^{-129.961} \\
\text{sLiSCP-light-256:} \quad & (\text{MDP}(\text{Simeck}^8\text{-}64))^{12} = (2^{-15.86146})^{12} = 2^{-190.337} \\
& (\text{MLSC}(\text{Simeck}^8\text{-}64))^{12} = (2^{-15.64046})^{12} = 2^{-187.686}.
\end{aligned}$$

We expect that the better maximum probabilities for sLiSCP-light exist because given the iterated nature of the employed Simeck^u-m boxes followed by a simple subblock XOR and cyclic shift permutation, long trails are preserved. Accordingly, the long trail strategy (LTS) [DPU+16] offers a better security argument than simply counting the minimum number of active Simeck^u-m boxes. In particular, when an uninterrupted differential path activates r consecutive Simeck^u-m boxes, it makes sense that one considers the optimal probability of Simeck^{ru}-m box rather than that of (Simeck^u-m)^r where such a former probability is usually much less than the latter one. An example for bounding the probability of an optimum differential trail using LTS is provided in Appendix C.

4.3 Algebraic Properties

In this section, we evaluate the algebraic degree of sLiSCP-light and assess its security against integral and zero-sum distinguishers. We find that the algebraic degree of Simeck^6-48 (resp. Simeck^8-64) is 19 (resp. 36). Furthermore, we use a tweaked version of the division property to find the degree of an s-step sLiSCP-light permutation [TM16, BKL+17]. Table 5 provides an upper bound on the algebraic degree for each component function of the sLiSCP-light instances.

Table 5: Upper bounds on the algebraic degree of sLiSCP-light.

		Component function							
sLiSCP-light-192	steps (s)	0-23	24-47	48-71	72-95	96-119	120-143	144-167	168-191
	1	19	13	19	13	19	13	19	13
	2	57	51	57	51	57	51	57	51
	3	129	125	129	125	129	125	129	125
	4	178	177	178	177	178	177	178	177
	5	189	189	189	189	189	189	189	189
sLiSCP-light-256	steps (s)	0-31	32-63	64-95	96-127	128-159	160-191	192-223	224-255
	1	36	27	36	27	36	27	36	27
	2	92	83	92	83	92	83	92	83
	3	183	182	183	182	183	182	183	182
	4	247	247	247	247	247	247	247	247

Integral distinguishers. To search for the longest length integral distinguisher, we set the 0-th bit of the input state as constant (C) and rest as active (A). We then evaluate the algebraic degree at the s-th step of each component function in terms of the involved active bits. If the algebraic degree equals the number of active bits then the bit is unknown (U) (i.e., XOR sum of the component function is unpredictable), otherwise, it is balanced (B) in which case the xor sum is always zero. Accordingly, we find that for sLiSCP-light-192, after the 8-th step, the component functions 0-59, 70-107, 118-191 have degree less than 191, and hence bits 0-59, 70-107, 118-191 are balanced. As for sLiSCP-light-256, bits 0-63, 192-255 are balanced. Thus, 8-step integral distinguishers exist for both sLiSCP-light-192 and sLiSCP-light-256.

Zero-sum distinguishers. This is a type of integral distinguisher where all the bits of the state are balanced. We note that the maximum number of steps covered by zero-sum distinguishers in one direction is at most 7. This is because integral distinguisher can cover up to 8 steps. For example $CA^{191} \xrightarrow{7 \text{ steps}} B^{192}$, $CA^{255} \xrightarrow{7 \text{ steps}} B^{256}$. Hence, $7 + 7$ steps zero-sum distinguisher exists for sLiSCP-light-192 (resp. sLiSCP-light-256).

4.4 Security Claims

We reiterate that we design sLiSCP-light to be specifically used in the sLiSCP duplex sponge mode [ARH+17] and thus, provide a low overhead for multiple cryptographic functionalities in resource constrained applications. Accordingly, we do not push for ideal properties for both instances of the sLiSCP-light permutations because theoretical nonrandom properties such as the ideal bounds on the probabilities of differential and linear characteristics and zero-sum distinguishers have negligible effect on the claimed security of the application modes which are provided in [ARH+17]. Particularly, in keyed modes, sLiSCP-light-192 (resp. sLiSCP-light-256) offers 80- and 112-bit (resp. 128 bit) security depending on the allowed data usage. In the hashing mode, a 160-bit digest with 128-bit of preimage security is offered by sLiSCP-light-192. On the other hand, sLiSCP-light-256 offers a 192-bit digest with either 128-bit or 160-bit of preimage security depending upon the squeezing rate.

If the ideal properties are required, we recommend having at least 18 steps of the permutation where a minimum of 18 Simeck^u-m boxes are differential/linearly activated. Hence, given the bounds on the maximum probabilities (resp. squared correlation) of differential (resp. linear) optimal 6-step characteristic based on the LTS provided in Appendix C, such bounds over 18 steps of sLiSCP-light-b are lower than 2^{-b}. In this case, the throughput of sLiSCP-light-b is equal to that of sLiSCP-b.

5 Implementations and Benchmarking

In this section, we provide the details of our ASIC hardware and bitsliced software implementations of both instances of the sLiSCP-light permutation. Moreover, we implement the hashing and authenticated encryption modes of sLiSCP-light in ASIC CMOS 65 nm and 130 nm technologies and provide a comparison with existing proposals in Table 12 in Appendix D.

5.1 Hardware Implementation

sLiSCP-light is highly hardware optimized and has very efficient ASIC implementations particularly because of its partial layers. More precisely, the Simeck^u-m boxes, step constant addition, and linear mixing are all applied on half of the state. Additionally, each Simeck^u-m box is itself a very efficient unkeyed Feistel round function. The datapath of the round-based ASIC parallel architecture implementation is depicted in Figure 4.

The implementations of both instances of sLiSCP-light in ASIC are carried out using STMicroelectronics CMOS 65 nm CORE65LPLVT library and IBM CMOS 130 nm library. Our parallel implementations in CMOS 65 nm show that the area of sLiSCP-light-192 (resp. sLiSCP-light-256) is 1820 (resp. 2397) GE. Their areas in CMOS 130 nm are 1892 GE and 2500 GE, respectively. A breakdown of the area of the different components of the sLiSCP-light-b permutations in both CMOS technologies is provided in Figure 5. Note that using the same CMOS technology, both instances of sLiSCP-light component breakdowns have the same ratio.

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

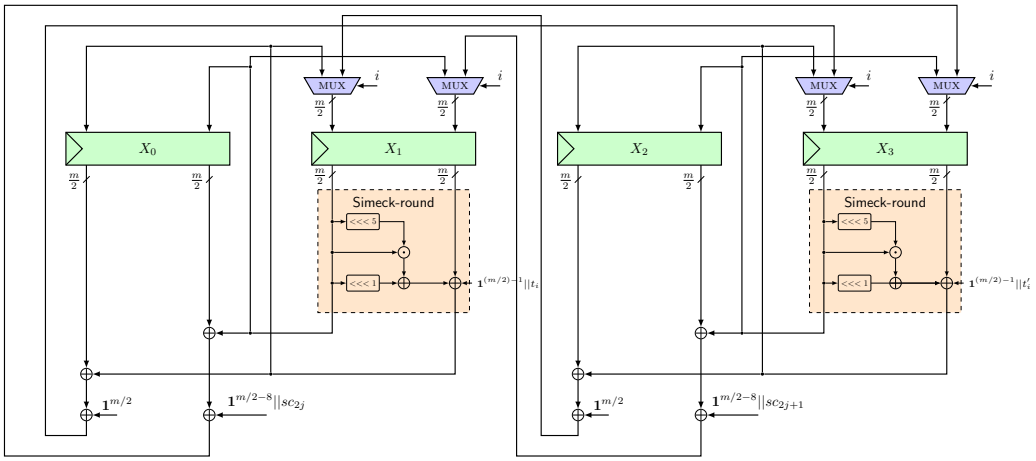


Figure 4: Parallel datapath of the sLiSCP-light permutation step function.

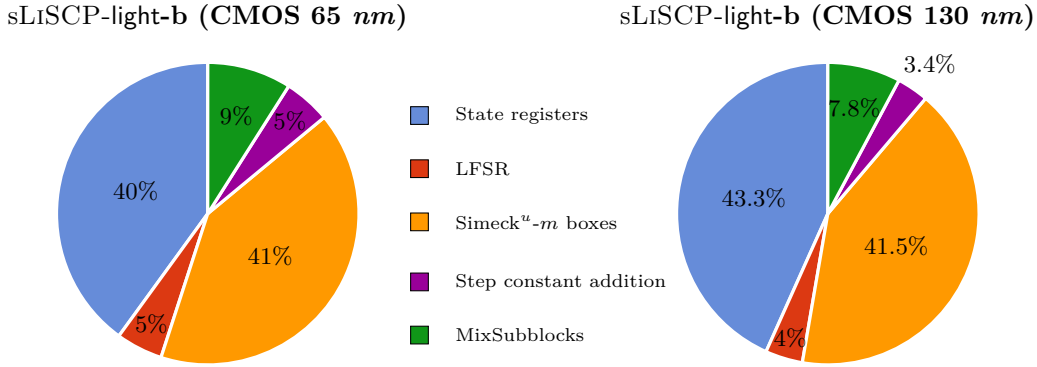


Figure 5: Breakdown of the area requirements of the two instances of sLiSCP-light components.

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 after the logic synthesis for comparison with existing proposals in Table 1. Due to the structural similarities between sLiSCP-light and sLiSCP, we have generated the power consumption results for sLiSCP for comparison purposes.

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.

5.1.1 Description of the round-based implementation

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. As depicted in Figure 4, all four m-bit registers are divided into two parts to accommodate the Feistel execution of the Simeck^u-m boxes. Two counters i and j of 3 and 4 bits, respectively are utilized, where i $(0 \leq i \leq u-1)$ controls the round function of Simeck and j $(0 \leq j \leq s-1)$ controls the

permutation step function.

During each clock cycle when $0 \leq i < u-1$, we first XOR the right half of registers X_1 (resp. X_3) with $\mathbf{1}^{m/2-1}||t_i$ (resp. $\mathbf{1}^{m/2-1}||t'_i$) where t_i, t'_i are LFSR generated bits. 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 i 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 $\mathbf{1}^{m/2}$. 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 $\mathbf{1}^{m/2-8}||sc_{2j}$.

In particular, for sLiSCP-light-192, the $(m/2-8)$ bits are first padded with two 0's followed by padding the 6-bit constant sc_{2j}. 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 boxes when $i = u-1$ and the cyclically shifted registers. Finally, a new permutation step begins where j is incremented by 1 and i is reset to 0.

Table 6: 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
Simeck^u-m boxes	AND	2×24	2×32
	XOR	2×49	2×65
Add step constants	XOR	2×6	2×8
Mix Subblocks	XOR	2×48	2×64
LFSR	Registers	6	7
	XOR	6	9

5.1.2 Where does sLiSCP-light stand?

sLiSCP-light is specifically optimized for resource constrained applications. Table 1 shows our smallest ASIC implementations of both sLiSCP-light instances in CMOS 65 *nm* and 130 *nm* technologies as well as those of other existing permutations. sLiSCP-light has the lowest area of 1820 (resp. 2397) GE for a 192- (resp. 256-) bit state in CMOS 65 *nm*. Although the state sizes of sLiSCP-light instances are close to the state size of `Keccak-`f[200], the areas of both instances of sLiSCP-light are significantly lower than that of `Keccak-`f[200] (est. 4540 GE). When compared with `Norx`-16 (est. area of 2496 GE), sLiSCP-light-256 has a slightly lower area in CMOS 65 *nm* and comparable area in CMOS 130 *nm*. For larger state sizes, the areas of the 256-bit state sLiSCP-light in CMOS 65 *nm* and 130 *nm* technologies are several magnitude smaller than that of `Ascon` and `Gimli`, which are mainly optimized for software platforms and/or specific processors.

Regarding the throughput, both 192 and 256-bit instances of sLiSCP-light have a throughput of 266.7 kbps, which is higher than that of `Photon`, `Spongent`, and `Quark` with a state of 176 bits. When sLiSCP-light instances are compared with `Gimli` and `Ascon`, the throughput of the sLiSCP-light instances is smaller as they have a larger number of rounds compared to `Gimli` and `Ascon`. However, such permutations are not considered suitable for lightweight applications due to their larger GE areas.

Application modes. We have also implemented the authenticated encryption and hashing modes using both instances of sLiSCP-light. We report our implementation

results in CMOS 65 *nm* and 130 *nm* technologies, and contrast the areas and throughput with other sponge-based primitives with similar or close state sizes and security parameters in Table 12 in Appendix D. One can clearly see that both instances of sLiSCP-light offer a competitive advantage over existing proposals in terms of both area and throughput. Specifically, our smallest implementation of the unified duplex sponge mode using sLiSCP-light-192 offers 80-bit and 112-bit security in the keyed modes depending on the data usage exponent [BDPVA11], and 160-bit digest in the hashing mode. On top of that, such an implementation is realized with only 1958 GE and offers a throughput of 44.44 kbps which makes it suitable for providing almost all cryptographic functionalities of the most resource constrained devices such as passive RFID tags. When used in the sLiSCP duplex mode, a simpler initialization phase is adopted to offer competitive throughputs for the authenticated encryption of short messages (cf. Section 6.3 in [ARH+17]).

5.1.3 Estimates for 1-bit serialized implementations

Various serialization degrees has been demonstrated by the designers of the Simeck block cipher [YZS+15]. Accordingly, using the same methods adopted in [YZS+15], in what follows, we provide estimates for the areas of both sLiSCP-light instances when the Simeck^u-m boxes are serialized by degree 1.

The parallel Simeck^u-m implementation utilizes $m/2$ ANDs and $(2m+1)$ XORs which can be serialized using 1 AND, 3 XORs and 4 2-1 MUXs. Two of the MUXs are used to select the cyclic shift inputs, one is used to select the input of the registers, and another MUX is used to XOR the round constant.

When sLiSCP-light-192 is implemented in CMOS 65 *nm* technology that costs 2.25 GE per XOR, 2 GE per MUX and 1.25 GE per AND, then the serialized implementation of degree 1 of the Simeck^u-48 saves $(24\times1.25+49\times2.25)-(1\times1.25+3\times2.25+4\times2)=124.25$ GE. Since there are two Simeck Sboxes, this results in 248 GE savings for sLiSCP-light-192. The respective saving for sLiSCP-light-256 is given by $(32\times1.25+65\times2.25-(1\times1.25+3\times2.25+4\times2))\times2\approx340$ GE. Thus, without considering other savings, the estimated areas for the 1-bit serialized implementations of sLiSCP-light-192 and sLiSCP-light-256 in CMOS 65nm ASIC can be obtained with at least 1572 and 2057 GE, respectively. For the CMOS 130 *nm* technology, the number of discrete components that we can save are the same. This technology requires 2 GE per XOR, 2.25 GE per MUX and 1.25 GE per AND. For sLiSCP-light-192, the serialized implementation saves around $(24\times1.25+49\times2-(1\times1.25+3\times2+4\times2.25))\times2=223.5$ GE. Similarly, we save around $(32\times1.25+65\times2-(1\times1.25+3\times2+4\times2.25))\times2\approx307.5$ GE for sLiSCP-light-256. Hence, an estimated area for sLiSCP-light-192 (resp. sLiSCP-light-256) can be at least achieved with 1669 (resp. 2193) GE in CMOS 130 *nm* ASIC.

5.2 Software Implementation

The sLiSCP-light family of permutations is designed to be hardware efficient in order to suit lightweight applications. However, even for lightweight applications, a server communicating with such devices needs to perform the encryption/decryption, and hashing operations at high speed. In this section, we consider the server-side software implementation scenario and present a bit-sliced implementation of sLiSCP-light permutations using SIMD instruction sets. We provide details about the software implementations of sLiSCP-light permutations and compare it with existing permutations.

5.2.1 Bit-sliced Implementation of sLiSCP-light

We use different instructions in the SSE2 and AVX2 instruction sets to implement the sLiSCP-light permutations 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 cycle 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-box 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-256 are evaluated using YMM registers, we pack data for the Simeck-box 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. We explain the packing and unpacking and the round function implementation while taking sLiSCP-light-256 as an example. The implementation details for sLiSCP-light-192 are omitted because they are very similar to that of sLiSCP-light-256.

Packing and Unpacking for sLiSCP-light**-256.** 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-256 state by $S_i = s_0^i s_1^i s_2^i s_3^i s_4^i s_5^i s_6^i s_7^i$ where each s_j^i 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-256 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-64 box 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-box, we regroup the data in A and B for the homogeneity of operations in the round function of the Simeck-box. For this, we need the second pair of packing and unpacking operation for the SSb layer, which are given by

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

Translating 4-Parallel sLiSCP-light**-256 Instances.** In sLiSCP-light's design, the construction of Simeck-box 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-box, 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-256 is simplified to the steps in Algorithm 1. We have provided the complete C code of AVX2 implementation in Appendix E.

Algorithm 1 Four parallel instances of sLiSCP-light-256 computation

Input: (R_0, R_1, R_2, R_3)
Output: (R_0, R_1, R_2, R_3)
$(R_0, R_1, R_2, R_3) \leftarrow \text{PACK}(R_0, R_1, R_2, R_3)$;
for j from 0 to $s-1$ **do**
`//SSb layer`
  $R_2, R_3 \leftarrow \text{PACK\_SSb}(R_2, R_3)$;
  for i from 0 to $u-1$ **do**
    $R_2, R_3 \leftarrow \text{ROAX}(R_2, R_3, RC_1[i], RC_2[i])$;
  end for
  $R_2, R_3 \leftarrow \text{UNPACK\_SSb}(R_2, R_3)$;
`//ASc layer`
  $C \leftarrow$ `0xffffff00`; $D \leftarrow$ `0xffffffff`;
  $R_0 \leftarrow R_0 \oplus (D, C \oplus SC_1[j], D, C \oplus SC_2[j], D, C \oplus SC_1[j], D, C \oplus SC_2[j])$;
  $R_1 \leftarrow R_1 \oplus (D, C \oplus SC_1[j], D, C \oplus SC_2[j], D, C \oplus SC_1[j], D, C \oplus SC_2[j])$;
`//MSb layer`
  `tmp0` $\leftarrow R_0$; `tmp1` $\leftarrow R_1$;
  $R_0 \leftarrow R_2$;
  $R_1 \leftarrow R_3$;
  $R_2 \leftarrow \text{SWAPBLK}(R_2 \oplus$ `tmp0`$)$;
  $R_3 \leftarrow \text{SWAPBLK}(R_3 \oplus$ `tmp1`$)$;
end for
return $(R_0, R_1, R_2, R_3) \leftarrow \text{UNPACK}(R_0, R_1, R_2, R_3)$;

In SSE2 implementation, we use four 128-bit XMM registers to evaluate two parallel instances of the sLiSCP-light-256 permutation. Since the SSE2 implementation of the two sLiSCP-light instances is very similar to the AVX2 implementation, we omit the details. All implementation codes will be provided upon request and made publicly available later on.

5.2.2 Benchmarking

We implement the sLiSCP-light-192 and 256 permutations in C using SSE2 and AVX2 instruction sets and measure their performances on two different Intel processors Skylake and Haswell. The codes were compiled using `gcc 5.4.0` on 64-bit machines with the compiler flags `-O2 -funroll-all-loops -march=native`. For both SSE2 and AVX2 implementations, we evaluate 64 parallel instances of the sLiSCP-light permutations and compute the throughput of the permutations. Note that 64 parallel instances are grouped into 16 groups and each with four instances. Table 7 presents the performance results in cycles per byte for both implementations. In our implementation, we include the costs for all packing and unpacking operations. The best speed achieved by the AVX2 implementation on Skylake is 4.32 cycles/byte for sLiSCP-light-192 and 4.97 cycles/byte for sLiSCP-light-256. When we compare the speed for sLiSCP-light with `Simpira v2`, `NORX-32-4-1`, `Gimli` (2 and 4 blocks), sLiSCP-light is little slower as it has a larger number of steps compared to other SPN-based permutations and is designed to be significantly efficient in hardware. The sLiSCP-light instances' hardware implementation cost is much less than that of other permutations as shown in Table 1.

Table 7: Benchmarking the results for the sLiSCP-light permutations on Intel processors. 64 instances of sLiSCP-light-256 were evaluated in parallel. The throughput of the permutations are measured in cycles per byte (c/B). A performance comparison with other permutations is made.

Primitive	Speed c/B	Instruction Set	CPU Name Spec.
sLiSCP-light-192	7.38	SSE2	Skylake
	4.32	AVX2	`Intel i7-6700 CPU@3.40GHz`
	8.66	SSE2	Haswell
	4.83	AVX2	`Intel i7-4790 CPU@3.60GHz`
sLiSCP-light-256	8.56	SSE2	Skylake
	4.97	AVX2	`Intel i7-6700 CPU@3.40GHz`
	10.20	SSE2	Haswell
	5.85	AVX2	`Intel i7-4790 CPU@3.60GHz`
`Simpira v2` [GM16]	1.44	`AESENC`	Haswell⋆
`Norx-32-4-1` [BL17]	2.84	–	Haswell⋆
`Ascon`† [DEMS16]	7.1	–	Haswell⋆
`Gimli` 1-block [BKL+17]	4.46	SSE2	Haswell⋆
`Gimli` 2-block [BKL+17]	2.33	AVX2	”
`Gimli` 4-block [BKL+17]	1.77	AVX2	”

†This includes the encryption timing as well.
⋆CPU specification not known.
– Instruction set info not available.

6 Conclusions

The wide spectrum of lightweight devices that need to be secured utilizing a tight GE budget for this specific purpose, requires a realistic cryptographic design that realizes as many cryptographic functionalities as possible. sLiSCP-light is a hardware optimized family of lightweight cryptographic permutations that is designed with such an application concept in mind. Specifically, sLiSCP-light is built to be used in the sLiSCP duplex sponge mode to offer (authenticated) encryption and hashing functionalities in the same cryptographic design. What mainly distinguish sLiSCP-light from existing permutations are the following twofold.

sLiSCP-light is a sponge-specific lightweight permutation. In other words, sLiSCP-light fills the gap in the available cryptographic permutations with low GE area and offering acceptable security parameters for both keyed and unkeyed modes of sponge constructions. More precisely, the state size of the underlying permutation is bounded below by the sponge security parameters (i.e., capacity and rate), especially in hashing mode, a 160-bit digest with 32-bit hashing rate requires a permutations state of at least 192-bits. Both instances of the sLiSCP-light permutation are extremely hardware efficient to the point (e.g., 2000 GE), where the 192-bit sLiSCP-light costs only 1820 (resp. 1892) GE in CMOS 65 nm (resp. 130 nm) ASIC. Moreover, our smallest implementation of the fully parallelized unified sLiSCP duplex sponge mode using sLiSCP-light-192 has a cost of 1958 GE with a throughput of 44.44 kbps, and offers 80-bit and 112-bit security in the keyed modes depending on the data usage, and 160-bit digest in the hashing mode.

The security of sLiSCP-light is set with respect to application modes. Unlike most permutations, sLiSCP-light does not aim for provable ideal properties such as the ideal 2^b

and $2^{b/2}$ bounds against differential and linear distinguishers, respectively. We believe that such trade-offs between ideal properties, the GE area, and the throughput are acceptable, especially, when such non-ideal properties does not directly lead to any attacks when the permutation is used in the duplex sponge mode. Thus, the claimed security of both keyed and unkeyed modes remains intact. That being stated, we have provided a thorough security analysis of sLiSCP-light with respect to its diffusion, differential, linear and algebraic properties. We have calculated the exact probabilities of differentials and linear masks that are associated with optimal characteristics over the constant-based Simeck^u-m box, and consequently, provided evaluation of the expected maximum probabilities of differential and linear characteristics. Finally, the upper bounds of the algebraic degrees of the permutations and security against division property and zero-sum distingushers are provided.

7 Acknowledgment

This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the National Institute of Standards and Technology (NIST).

References

[AHMNP13] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. *Journal of Cryptology*, 26(2):313–339, 2013.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX: Parallel and scalable AEAD. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *19th European Symposium on Research in Computer Security, Part II*, pages 19–36. Springer, 2014.

[AJN15] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Norx8 and Norx16: Authenticated encryption for low-end systems. Cryptology ePrint Archive, Report 2015/1154, 2015. http://eprint.iacr.org/2015/1154.

[ARH+17] Riham AlTawy, Raghvendra Rohit, Morgan He, Kalikinkar Mandal, Gangqiang Yang, and Guang Gong. sLiSCP: Simeck-based Permutations for Lightweight Sponge Cryptographic Primitives. Cryptology ePrint Archive, Report 2017/747, 2017. http://eprint.iacr.org/2017/747, (also available at: http://cacr.uwaterloo.ca/techreports/2017/cacr2017-04.pdf), (to appear in SAC 2017).

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT*, pages 430–454. Springer, 2015.

[BBW14] Céline Blondeau, Andrey Bogdanov, and Meiqin Wang. On the (in)equivalence of impossible differential and zero-correlation distinguishers for feistel- and skipjack-type ciphers. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *ACNS*, pages 271–288. Springer, 2014.

[BDPA14] G Bertoni, J Daemen, M Peeters, and GV Assche. Caesar submission: Ketje v2, 2014. http://ketje.noekeon.org/Ketjev2-doc2.0.pdf.

[BDPV14] G Bertoni, J Daemen, M Peeters, and G Van Assche. Cryptographic sponge functions, 2014. http://sponge.noekeon.org/CSF-0.1.pdf.

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007, 2007.

[BDPVA09] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Keccak specifications. submission to NIST (Round 2), 2009.

[BDPVA11] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the security of the keyed sponge construction. In *Symmetric Key Encryption Workshop*, 2011.

[BDPVA12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. *DIAC*, 2012.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO*, pages 123–153. Springer, 2016.

[BKL+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, pages 450–466. Springer, 2007.

[BKL+11] Andrey Bogdanov, Miroslav Knežević, Gregor Leander, Deniz Toz, Kerem Varıcı, and Ingrid Verbauwhede. spongent: A lightweight hash function. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, pages 312–325. Springer, 2011.

[BKL+17] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. Gimli: a cross-platform permutation. Cryptology ePrint Archive, Report 2017/630, 2017. http://eprint.iacr.org/2017/630 (to appear in CHES 2017).

[BL17] Daniel J. Bernstein and Tanja Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems. https://bench.cr.yp.to, accessed 11 October 2017.

[BODD+15] Achiya Bar-On, Itai Dinur, Orr Dunkelman, Virginie Lallemand, Nathan Keller, and Boaz Tsaban. Cryptanalysis of sp networks with partial non-linear layers, 2015.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Siang Meng Sim, Yosuke Todo, and Yu Sasaki. Gift: A small present. Cryptology ePrint Archive, Report 2017/622, 2017. http://eprint.iacr.org/2017/622 (*to appear* in CHES 2017).

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.

[BW99] Alex Biryukov and David Wagner. Slide attacks. In Lars Knudsen, editor, *FSE*, pages 245–259. Springer, 1999.

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Submission to the CAESAR competition: http://competitions.cr.yp.to/round3/asconv12.pdf (http://ascon.iaik.tugraz.at/implementation.html), 2016.

[DPU+16] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design strategies for arx with provable bounds: Sparx and lax. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT*, pages 484–513. Springer, 2016.

[GGNPS13] B. Gérard, Vincent Grosso, M. Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES*, pages 383–399. Springer, 2013.

[GM16] Shay Gueron and Nicky Mouha. Simpira v2: A family of efficient permutations using the aes round function. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT*, pages 95–125. Springer, 2016.

[GPP11] Jian Guo, Thomas Peyrin, and Axel Poschmann. The photon family of lightweight hash functions. In Phillip Rogaway, editor, *CRYPTO*, pages 222–239. Springer, 2011.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, pages 326–341. Springer, 2011.

[GWDE15] Hannes Groß, Erich Wenger, Christoph Dobraunig, and Christoph Ehrenhöfer. Suit up!–made-to-measure hardware implementations of ascon. In *Digital System Design (DSD), 2015 Euromicro Conference on*, pages 645–652. IEEE, 2015.

[JW05] Ari Juels and Stephen A. Weis. Authenticating pervasive devices with human protocols. In Victor Shoup, editor, *CRYPTO*, pages 293–308. Springer, 2005.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the Simon block cipher family. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO*, pages 161–185. Springer, 2015.

[KY10] Elif Bilge Kavun and Tolga Yalcin. A lightweight implementation of keccak hash function for radio-frequency identification applications. In Siddika Berna Ors Yalcin, editor, *RFIDSec 2010, Revised Selected Papers*, pages 258–269. Springer, 2010.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of printcipher: The invariant subspace attack. In Phillip Rogaway, editor, *CRYPTO*, pages 206–221. Springer, 2011.

[LLW17] Zhengbin Liu, Yongqiang Li, and Mingsheng Wang. Optimal differential trails in Simon-like ciphers. *IACR TOSC*, pages 358–379, 2017.

[MBSTM17] Kerry McKay, Larry Bassham, Meltem Sönmez Turan, and Nicky Mouha. Report on lightweight cryptography (NISTIR8114), 2017.

[RASA14] Shahram Rasoolzadeh, Zahra Ahmadian, Mahmoud Salmasizadeh, and Mohammad Reza Aref. Total break of zorro using linear and differential attacks. *The ISC Int'l Journal of Information Security*, 6 (1):23–34, 2014.

[SM10] Tomoyasu Suzaki and Kazuhiko Minematsu. Improving the generalized feistel. In Seokhie Hong and Tetsu Iwata, editors, *F*, pages 19–39. Springer, 2010.

[TM16] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In *International Conference on Fast Software Encryption*, pages 357–377. Springer, 2016.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT*, pages 287–314. Springer, 2015.

[YZS+15] Gangqiang Yang, Bo Zhu, Valentin Suder, Mark D. Aagaard, and Guang Gong. The simeck family of lightweight block ciphers. In Tim Güneysu and Helena Handschuh, editors, *CHES*, pages 307–329. Springer, 2015.

A Avalanche Effect

The average numbers of flipped bits after 4 steps corresponding to flipping the individual 192 (resp. 256) bit positions are depicted in the following tables. Sampling has been done over 1024 independent random inputs.

Table 8: Average number of flipped bits after 4 steps of sLiSCP-light-192. Bit# denotes the bit position of input flipped bit.

Bit#	Avg.	Bit#	Avg.	Bit#	Avg.	Bit#	Avg.	Bit#	Avg.	Bit#	Avg.
0	95.77	32	95.69	64	96.22	96	96.00	128	96.05	160	95.93
1	95.73	33	95.96	65	96.34	97	95.95	129	95.73	161	96.15
2	95.81	34	95.68	66	95.99	98	95.59	130	96.03	162	96.16
3	95.71	35	95.92	67	96.11	99	96.15	131	95.86	163	96.29
4	96.45	36	95.87	68	95.85	100	96.09	132	96.24	164	96.09
5	95.87	37	95.54	69	96.08	101	96.12	133	95.13	165	96.15
6	96.02	38	95.96	70	95.95	102	96.05	134	96.32	166	96.01
7	96.18	39	95.69	71	96.21	103	96.12	135	95.55	167	95.79
8	95.26	40	95.90	72	96.05	104	96.38	136	96.05	168	96.13
9	95.80	41	95.99	73	95.65	105	95.86	137	96.29	169	96.32
10	95.59	42	95.93	74	95.96	106	96.21	138	95.40	170	96.16
11	96.37	43	96.09	75	95.90	107	96.22	139	96.16	171	95.90
12	95.67	44	95.95	76	95.97	108	96.12	140	96.17	172	95.95
13	95.84	45	96.36	77	96.22	109	96.27	141	95.94	173	96.09
14	96.43	46	95.62	78	96.36	110	95.87	142	95.71	174	95.58
15	96.29	47	95.92	79	95.51	111	96.13	143	96.00	175	96.00
16	95.74	48	96.30	80	95.98	112	95.96	144	95.97	176	96.00
17	96.40	49	96.35	81	96.18	113	96.38	145	96.05	177	95.88
18	95.85	50	95.47	82	96.05	114	95.71	146	95.77	178	95.86
19	96.10	51	96.35	83	96.03	115	95.13	147	95.89	179	95.84
20	95.84	52	95.90	84	95.57	116	95.95	148	96.14	180	95.70
21	96.12	53	96.56	85	96.07	117	96.13	149	95.89	181	96.19
22	95.87	54	96.01	86	96.00	118	95.68	150	96.21	182	95.68
23	96.21	55	95.96	87	95.60	119	95.50	151	96.18	183	95.76
24	96.12	56	95.76	88	95.61	120	95.96	152	96.02	184	96.18
25	96.03	57	95.89	89	96.22	121	96.03	153	95.54	185	95.72
26	96.20	58	95.66	90	96.07	122	96.16	154	95.68	186	96.06
27	95.98	59	95.84	91	96.00	123	95.87	155	95.99	187	96.49
28	95.59	60	96.10	92	95.83	124	95.87	156	96.28	188	96.10
29	95.81	61	95.99	93	96.01	125	95.90	157	96.25	189	96.09
30	95.97	62	95.92	94	96.28	126	96.11	158	95.91	190	95.82
31	95.74	63	95.93	95	96.05	127	96.24	159	96.08	191	95.71

B Differentials with Maximum Probabilities for Constant-dependent Simeck^6-32 and Simeck^8-32 boxes

Tables 10 and 11 depict the six and eleven sets of maximum probability differentials parametrized by the round constants for Simeck^6-32 and Simeck^8-32, respectively. The blue colored differentials are the selected representatives for each set which we have chosen to get their exact probabilities for Simeck^6-48 and Simeck^8-48.

Table 9: Average number of flipped bits after 4 steps of sLiSCP-light-256. Bit# denotes the bit position of input flipped bit.

Bit#	Avg.	Bit#	Avg.	Bit#	Avg.	Bit#	Avg.	Bit#	Avg.	Bit#	Avg.
0	127.91	48	127.38	96	128.14	144	127.89	192	127.85	240	127.97
1	127.45	49	128.23	97	127.52	145	128.35	193	127.92	241	127.95
2	127.53	50	127.90	98	127.84	146	127.81	194	128.16	242	128.29
3	128.28	51	128.47	99	127.77	147	127.85	195	127.79	243	127.93
4	127.71	52	127.73	100	127.77	148	127.83	196	128.35	244	128.09
5	128.07	53	127.85	101	127.94	149	127.55	197	128.01	245	128.10
6	127.76	54	127.72	102	128.02	150	128.26	198	127.92	246	127.96
7	128.30	55	127.43	103	128.15	151	127.98	199	127.72	247	127.78
8	127.97	56	128.31	104	127.90	152	128.41	200	128.23	248	128.25
9	128.90	57	128.12	105	128.25	153	128.31	201	128.18	249	128.00
10	127.84	58	127.53	106	128.13	154	127.89	202	127.84	250	128.35
11	127.96	59	128.22	107	128.14	155	126.98	203	127.65	251	128.44
12	127.60	60	127.66	108	128.02	156	127.28	204	127.93	252	127.95
13	128.10	61	127.93	109	127.95	157	127.66	205	127.78	253	127.69
14	127.97	62	128.13	110	127.84	158	127.93	206	128.07	254	127.61
15	128.75	63	127.50	111	127.77	159	127.97	207	128.34	255	128.09
16	127.48	64	127.44	112	127.91	160	128.30	208	128.00		
17	128.13	65	128.10	113	128.45	161	128.04	209	128.16		
18	127.10	66	128.24	114	127.87	162	127.97	210	127.86		
19	128.23	67	128.35	115	127.86	163	127.97	211	127.64		
20	128.16	68	128.16	116	128.16	164	128.09	212	128.06		
21	128.08	69	127.90	117	127.94	165	127.97	213	128.31		
22	127.99	70	127.86	118	128.02	166	127.99	214	127.72		
23	128.21	71	127.77	119	127.63	167	128.01	215	128.20		
24	127.74	72	127.57	120	127.92	168	128.06	216	128.19		
25	127.98	73	128.58	121	128.13	169	128.05	217	127.98		
26	128.15	74	128.18	122	128.27	170	127.90	218	128.66		
27	128.21	75	127.99	123	127.75	171	128.24	219	128.28		
28	127.59	76	127.76	124	128.09	172	127.66	220	127.82		
29	128.22	77	128.04	125	127.66	173	127.72	221	127.86		
30	127.99	78	127.69	126	127.80	174	127.73	222	127.99		
31	128.12	79	127.88	127	128.30	175	127.57	223	128.01		
32	127.61	80	127.89	128	127.52	176	127.81	224	128.04		
33	127.71	81	127.71	129	128.35	177	128.19	225	127.80		
34	128.14	82	128.10	130	127.89	178	127.89	226	127.90		
35	127.91	83	127.79	131	127.69	179	128.29	227	127.79		
36	128.13	84	127.95	132	127.88	180	127.83	228	128.31		
37	128.19	85	128.02	133	128.13	181	128.14	229	127.96		
38	128.08	86	127.44	134	128.13	182	127.91	230	128.17		
39	127.87	87	127.96	135	128.30	183	127.92	231	127.99		
40	127.71	88	127.67	136	128.39	184	127.48	232	127.76		
41	128.18	89	127.91	137	128.11	185	127.64	233	127.80		
42	127.63	90	127.38	138	127.95	186	128.11	234	127.83		
43	127.93	91	128.21	139	128.10	187	128.31	235	128.16		
44	128.13	92	127.77	140	127.92	188	128.24	236	128.53		
45	127.98	93	127.95	141	128.34	189	127.68	237	128.28		
46	127.59	94	128.10	142	127.77	190	128.09	238	128.30		
47	128.19	95	127.80	143	128.01	191	128.14	239	128.23		

Table 10: Differentials with maximum probability for Simeck^6-32 boxes. Here time denotes the average time to find all solutions of one differential.

Round constants (t)	($\Delta_{in}, \Delta_{out}$)	# solutions	MDP ($log_2(.)$)	Time(s)
7, 27, 4, 6, 25, 26, 24	(04000a00, 1a000a00), (04000a00, 5a000a00), (04000a00, 52000a00), (04000a00, 12000a00)	3293184	-10.348948	324.75
34, 17, 35, 15	(04000a00, 1a010a00), (04000a00, 5a010a00), (04000a00, 52010a00), (04000a00, 12010a00)	3293184	-10.348948	323.62
19, 8, 38	(0a001a00, 0a000400), (0a005a00, 0a000400), (0a005200, 0a000400), (0a001200, 0a000400)	3293184	-10.348948	324.26
3b, a	(0a001a01, 0a000400), (0a005a01, 0a000400), (0a005201, 0a000400), (0a001201, 0a000400)	3293184	-10.348948	323.92
12, 20	(40014023, 40018000), (4001402b, 40018000), (80004001, 402b4001), (80004001, 40234001)	3280896	-10.354342	324.03
2e, 1c, 1f, c, 2f, 3f	(00020005, 00890005), (00018002, 80548002), (80004001, 402a4001), (20005000, 900a5000), (00018002, 80448002), (00020005, 00a90005), (0004000a, 0152000a), (4000a000, 2011a000), (80004001, 40224001), (0004000a, 0112000a), (20005000, 90085000), (00800140, 22400140), (10002800, 48052800), (10002800, 48042800), (05008900, 05000200), (000500a9, 00050002), (00200050, 08900050), (4001402a, 40018000), (4000a000, 2015a000), (0a001201, 0a000400), (00800140, 2a400140), (00080014, 02a40014), (00080014, 02240014), (5000900a, 50002000), (08001400, 24021400), (08001400, a4021400), (28004805, 28001000), (80028054, 80020001), (0500a900, 05000200), (00280548, 00280010), (80028044, 80020001), (50009008, 50002000), (28004804, 28001000), (40014022, 40018000), (a0002011, a0004000), (00050089, 00050002), (02804480, 02800100), (004000a0, 150000a0), (a0002015, a0004000), (000a0112, 000a0004), (004000a0, 110000a0), (00100028, 04680028), (04000a00, 12000a00), (02805480, 02800100), (000a0152, 000a0004), (00100028, 05680028), (1400a402, 14010800), (00200050, 0a900050), (14002402, 14010800), (0a005201, 0a000400), (04000a00, 52000a00), (01402240, 01400080), (001402a4, 00140008), (00140224, 00140008), (01000280, 54800280), (01000280, 44800280), (00a01520, 00a00040), (01402a40, 01400080), (00a01120, 00a00040), (00500890, 00700020), (00280448, 00280010), (02000500, a9000500), (00500a90, 00700020), (02000500, 89000500)	3231744	-10.376119	281.97

Table 11: Differentials with maximum probability for Simeck^8-32 boxes.

Round constants (t)	($\Delta_{in}, \Delta_{out}$)	# solutions	MDP ($log_2(.)$)	Time(s)
b	(0a001201, 3a001000)	99232	-15.401482	405.02
f, 47, 4, 96, 7	(00a011a0, 02a00100)	90418	-15.535678	404.84
b2, a1	(010002a0, 118000a0)	90418	-15.535678	404.88
f1, 73	(010002a0, 11a000a0)	90418	-15.535678	404.86
78	(10002a00, 5a010a00)	89842	-15.544898	406.29
44, 4c	(00a01180, 02a00100)	87974	-15.575210	396.84
43	(00a011a0, 02a00100), (010002a0, 11a000a0)	87264	-15.586901	405.95
82	(00a011a0, 02a00100), (010002a0, 118000a0)	87264	-15.586901	398.71
e5	(80005001, d0085000), (5000d008, 50018000)	81282	-15.689352	405.10
b5, 37, f5,	(80005001, d0085000)	79488	-15.721551	405.02
ee	(5000d008, 50018000)	79488	-15.721551	404.39

C Bounding the Probability of an Optimal Differential Characteristic using the Long Trail Strategy (LTS)

In this section, we provide a justification that the bounds presented in Section 4.2.1 for the differential and linear characteristics based on the minimum number of active Simeck^u-m boxes are not strong and better bounds are given by the LTS [DPU+16]. The partial SPN structure adopted in sLiSCP-light's design allows the propagation of uninterrupted long trails through several u-round iterated Simeck boxes. More precisely, in sLiSCP-light a long trail of length r is a differential path that passes through r Simeck^u-m boxes without other differential paths branching in through XORs. In such a case the optimum differential probability of this path is evaluated by the MDP(Simeck^{ur}-m) which is $\ll$ (MDP(Simeck^u-m))^r. However, in this case, we cannot get the exact MDP of the Simeck^{ur}-m as we did for Simeck^u-m because between each u rounds there are difference-free unknown values (and not the known constants) influencing the input solutions. Alternatively, we provide a close estimate for such an MDP by considering the differential effect. We assume the Markov assumption and ignore the effect of constants and the difference-free values. More precisely, for a ur-rounds iterated Simeck, we extract the differential associated with an optimal characteristic, and calculate its probability as follows

$$\Pr(\delta_0 \rightarrow \delta_{ur}) = \sum_{i=w_{min}}^{w_{max}} s_i \cdot 2^{-i}$$

where w_{min} and w_{max} denote the minimum and maximum $\log_2$ characteristic probabilities, respectively, and s_i indicates the number of characteristics having probability equal to 2^{-i}. We find that $\Pr(010000230000 \xrightarrow{18\ rounds} 010000000000) \approx 2^{-39.73960}$ for 48-bit Simeck and $\Pr(00000001A0000022 \xrightarrow{24\ rounds} 00000011A0000000) \approx 2^{-54.90473}$ for 64-bit Simeck. Hence, we approximate the MDP(Simeck^{18}-48) (resp. MDP(Simeck^{24}-64)) by $2^{-39.73960}$ (resp. $2^{-54.90473}$).

Figure 6 depicts a 6-step differential path with the minimum number of active Simeck^u-m boxes, where on the left, according to the previously presented bounds, the optimum probability of such a path is given by: $(\text{MDP(Simeck}^6\text{-48)})^6 = (2^{-10.655704})^6 = 2^{-63.934}$. However, on the right, the same differential path can be decomposed into four long trails (colored trails), three out of which are of length one and the remaining one is of length 3 (passing the red active F's). Thus, for sLiSCP-light-192 (6 steps), the optimal probability of such a path is then given by: $(\text{MDP(Simeck}^6\text{-48)})^3 \times \text{MDP(Simeck}^{18}\text{-48)} \approx (2^{-10.655704})^3 \times 2^{-39.73960} = 2^{-71.7067}$. Similarly, such a probability is around $2^{-102.48911}$ $((2^{-15.86146})^3 \times 2^{-54.90473})$ for sLiSCP-light-256. The optimal squared correlation of a linear characteristic is evaluated analogously.

D Hardware ASIC implementation of Hashing and Authenticated Encryption modes

Our parallel implementation results in both CMOS 65nm and CMOS 130nm ASICs for the hash and authenticated encryption modes of sLiSCP-light are presented in Table 12, along with a comparison with other lightweight hash functions and AE algorithms. If a unified mode is used for both functionalities, then the consumed GE area is equal to that of the AE circuit.

Hashing mode. In our CMOS 65 nm implementation, we report areas of 1938 (resp. 2584) GE with a throughput of 44.44 (resp. 66.67 kbps or 33.33 kbps depending on a 64 or 32 bit squeezing rate) kbps for the hashing mode of sLiSCP-light-192 (resp. sLiSCP-light-256). Their CMOS 130 nm respective areas are 2051 and 2714 GE. As depicted in Table 12, the area of sLiSCP-light-192 is the lowest among other sponge-based hash

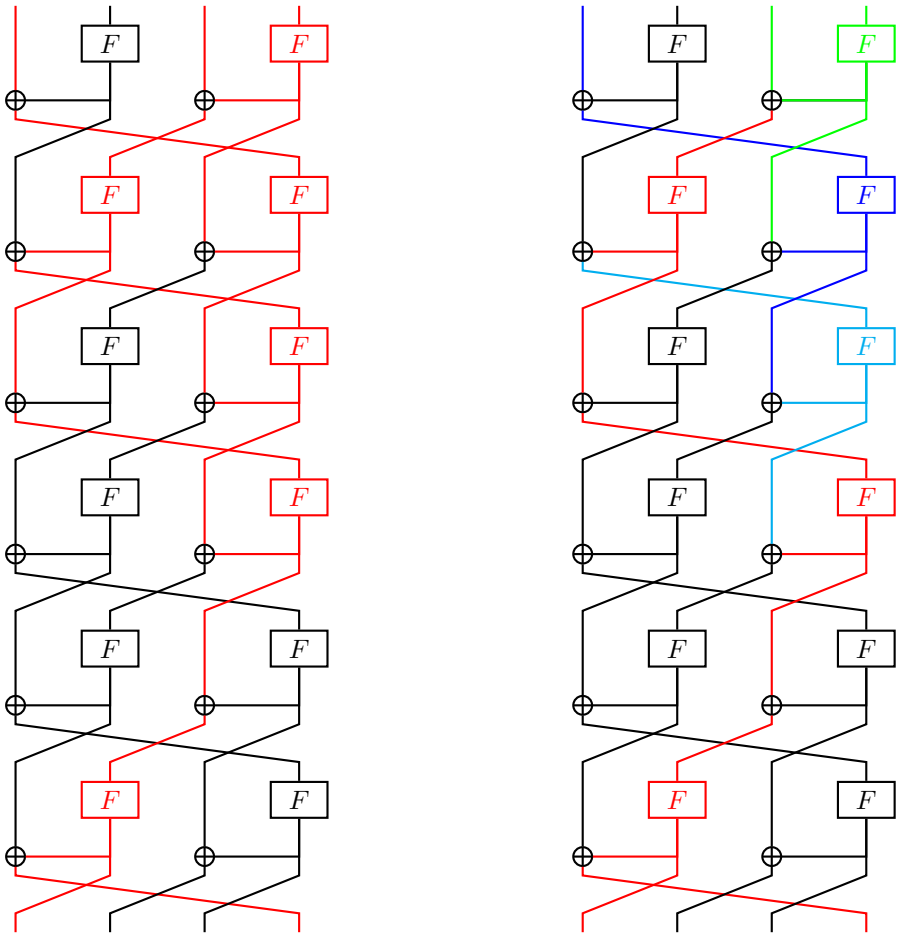


Figure 6: (Left) A differential path with a minimum number of active Simeck^u-m boxes over six steps. (Right) the same differential path decomposed into four long differential trails.

function and even lower than that of the serialized implementation of `Photon-160/36/36`. In terms of throughput, sLiSCP-light-192 is better than `Photon-160/36/36`, sLiSCP-192, `D-Quark`, and `Spongent-160/160/16`. The areas of the hashing modes of sLiSCP-light-256 is smaller than most of the hash functions with equal state sizes. The relevant throughput of sLiSCP-light-256 is also comparable to that of `Spongent-160/160/80` and `S-Quark`, and quite higher than that of `Photon-224/32/32`, `Spongent-224/224/16`, and `Spongent-256/256/16`.

AE mode. The areas in CMOS 65 nm of sLiSCP-light-192 (resp. sLiSCP-light-256) is 1958 (resp. 2603) GE with a throughput of 44.44 (resp. 66.67) kbps. Their respective areas in CMOS 130 nm are 2053 and 2731 GE. sLiSCP-light-256 has a GE area that is less than the estimated area of `NORX-16`, while sLiSCP-light-192 is quite smaller than `NORX-16`. Both areas of sLiSCP-light-192 and sLiSCP-light-256 are much smaller than that of `Ketje-Jr` and `Ascon`.

To this end, we note that both the hash and authenticated encryption modes of sLiSCP-light are extremely competitive with other proposals in terms of area and throughput, as well as the offered application mode parameters [ARH+17].

Table 12: Parallel hardware implementation results of sLiSCP-light modes and comparison with other competitors. Throughput and power are given at a frequency of 100 kHz.

Hash function	Parameters ‡				Technology	Latency	Area	Throughput	Power
	r	c	r'	h	(nm)	(Cycles)	(GE)	(kbps)	(μW)
sLiSCP-light-192	32	160	32	160	65	72	**1938**	44.44	3.97
sLiSCP-light-192	32	160	32	160	130	72	**2051**	44.44	5.05
Photon-160/36/36 [GPP11]	36	160	36	160	180	180	2117	20.00	4.35
Spongent-160/160/16 [BKL+11]	16	160	16	160	130	90	2190	17.78	4.47
sLiSCP-192 [ARH+17]	32	160	32	160	65	108	2271	29.62	4.62 †
sLiSCP-192 [ARH+17]	32	160	32	160	130	108	2492	29.62	7.44 †
D-Quark [AHMNP13]	16	160	16	176	180	88	2819	18.18	4.76
Keccak-f[72,128] [KY10]	72	128	72	200	130	18	4900	400.00	27.6
sLiSCP-light-256	64	192	64	192	65	96	**2584**	66.67	4.77
sLiSCP-light-256	64	192	32	192	65	96	**2584**	33.33	4.77
sLiSCP-light-256	64	192	64	192	130	96	**2714**	66.67	7.27
sLiSCP-light-256	64	192	32	192	130	96	**2714**	33.33	7.27
Photon-224/32/32 [GPP11]	32	224	32	224	180	204	2786	15.69	6.50
Spongent-224/224/16 [BKL+11]	16	224	16	224	130	120	2903	13.33	5.97
sLiSCP-256 [ARH+17]	64	192	64	192	65	144	3019	44.44	5.88 †
sLiSCP-256 [ARH+17]	64	192	32	192	65	144	3019	22.22	5.88 †
Spongent-160/160/80 [BKL+11]	80	160	80	160	130	120	3139	66.67	6.8
Spongent-256/256/16 [BKL+11]	16	256	16	256	130	140	3281	11.43	6.62
sLiSCP-256 [ARH+17]	64	192	64	192	130	144	3305	44.44	8.75 †
sLiSCP-256 [ARH+17]	64	192	32	192	130	144	3305	22.22	8.75 †
S-Quark [AHMNP13]	32	224	32	256	180	64	4640	50	8.39
AE algorithm				t					
sLiSCP-light-192/80	32	160	32	80	65	72	**1958**	44.44	3.97
sLiSCP-light-192/112	32	160	32	112	65	72	**1958**	44.44	3.97
sLiSCP-light-192/80	32	160	32	80	130	72	**2053**	44.44	5.05
sLiSCP-light-192/112	32	160	32	112	130	72	**2053**	44.44	5.05
sLiSCP-192/80 [ARH+17]	32	160	32	80	65	108	2289	29.62	4.62 †
sLiSCP-192/112 [ARH+17]	32	160	32	112	65	108	2289	29.62	4.62 †
sLiSCP-192/80 [ARH+17]	32	160	32	80	130	108	2498	29.62	7.44 †
sLiSCP-192/112 [ARH+17]	32	160	32	112	130	108	2498	29.62	7.44 †
sLiSCP-light-256/128	64	192	64	128	65	96	**2603**	66.67	4.77
sLiSCP-light-256/128	64	192	64	128	130	96	**2731**	66.67	7.27
Norx-16 [AJN15]	128	128	128	96	-	-	2880	-	-
sLiSCP-256/128 [ARH+17]	64	192	64	128	65	144	3039	44.44	5.88 †
sLiSCP-256/128 [ARH+17]	64	192	64	128	130	144	3319	44.44	8.75 †
Ketje-Jr⋆[BDPA14]	16	184	16	96	-	-	4540	-	-
Ascon [GWDE15]	64	256	64	128	90	6	7950	10.66††	43

‡ r, r', c, h and t denote the input bitrate, output bitrate, capacity, digest length and tag size, respectively.
† Our implementation results
⋆ Uses Keccak-200 as its underlying permutation, so its area is at least 4540 GE in 130 nm technology.
†† Unit of throughput is Mbps and power is given at 1MHz.

E AVX2 C Code of sLiSCP-light-256 permutation

```
#include<stdint.h>
#include<x86intrin.h>

#define STATE_SIZE      8
#define SIMECK_ROUND    8
#define STEP_ROUND      12

typedef unsigned long long int u64;
typedef unsigned int u32;
typedef unsigned int u8;
```

```
static const uint8_t RC1_256[12]={0x8, 0x86, 0xe2, 0x89, 0xe6,
0xca, 0x17, 0x8e, 0x64, 0x6b, 0x6f, 0x2c};
// Step constants (sc_{2i})
static const uint8_t RC2_256[12]={0x64, 0x6b, 0x6f, 0x2c, 0xdd,
0x99, 0xea, 0xf, 0x4, 0x43, 0xf1, 0x44};
// Step constants (sc_{2i+1})
static const uint8_t rc1_256[12]={0xf, 0x4, 0x43, 0xf1, 0x44,
0x73, 0xe5, 0xb, 0x47, 0xb2, 0xb5, 0x37};
// Round constants (t)
static const uint8_t rc2_256[12]={0x47, 0xb2, 0xb5, 0x37, 0x96,
0xee, 0x4c, 0xf5, 0x7, 0x82, 0xa1, 0x78};
// Round constants (t')

static inline __m256i _mm256_loadu2_m128i(__m128i *low,
__m128i *high){
        return _mm256_inserti128_si256(_mm256_castsi128_si256(
               _mm_loadu_si128(high)),_mm_loadu_si128(low),1);
}

#define ROT5(x)(_mm256_slli_epi32(x, 5)|_mm256_srli_epi32(x, 27))
#define ROT1(x)(_mm256_slli_epi32(x, 1)|_mm256_srli_epi32(x, 31))
#define SWAPREG1(x)(
        _mm256_permutevar8x32_epi32(x,
        _mm256_set_epi32(7, 5, 3, 1, 6, 4, 2, 0))
)
#define SWAPREG2(x)(
        _mm256_permutevar8x32_epi32(x,
        _mm256_set_epi32(7, 6, 3, 2, 5, 4, 1, 0))
)
#define SWAPBLK(x)(
        _mm256_permute4x64_epi64(x, _MM_SHUFFLE(2,3,0,1))
)
#define RC(t1, t2)(
        _mm256_set_epi32(0xfffffffe^t2, 0xfffffffe^t1,
         0xfffffffe^t2, 0xfffffffe^t1,  0xfffffffe^t2,
         0xfffffffe^t1, 0xfffffffe^t2, 0xfffffffe^t1)
)
#define SC(t1, t2)(
        _mm256_set_epi32(0xffffff00^t2, 0xffffffff,
         0xffffff00^t1, 0xffffffff, 0xffffff00^t2,
         0xffffffff, 0xffffff00^t1, 0xffffffff)
)

#define ROAX(x, y, t1, t2)\
{\
        __m256i x2tmp;\
        x2tmp = x;\
        x = (ROT5(x)&x)^ROT1(x)^RC(t1, t2)^y;\
        y = x2tmp;\
}

#define PACK_SSb(x, y)\
{\
        __m256i xtmp, ytmp;\
        xtmp = SWAPREG1(x);\
        ytmp = SWAPREG1(y);\
        x = _mm256_permute2x128_si256(xtmp,ytmp,0x20);\
```

```
        y = _mm256_permute2x128_si256(xtmp,ytmp, 0x31);\
}

#define UNPACK_SSb(x, y)\
{\
        __m256i xtmp, ytmp;\
        xtmp = _mm256_unpacklo_epi32(x, y);\
        ytmp = _mm256_unpackhi_epi32(x, y);\
        x = _mm256_permute2x128_si256(xtmp, ytmp,0x20);\
        y = _mm256_permute2x128_si256(xtmp, ytmp, 0x31);\
}

#define PACK(x, y, z, w)\
{\
        __m256i x2tmp, x3tmp;\
        x2tmp = SWAPREG2(x);\
        x3tmp = SWAPREG2(z);\
        x = _mm256_permute2x128_si256(x2tmp,x3tmp,0x20);\
        z = _mm256_permute2x128_si256(x2tmp,x3tmp, 0x31);\
        x2tmp = SWAPREG2(y);\
        x3tmp = SWAPREG2(w);\
        y = _mm256_permute2x128_si256(x2tmp,x3tmp,0x20);\
        w = _mm256_permute2x128_si256(x2tmp,x3tmp, 0x31);\
}

#define UNPACK(x,y,z,w)\
{\
        __m256i x2tmp, x3tmp;\
        x2tmp = _mm256_unpacklo_epi64(x, z);\
        x3tmp = _mm256_unpackhi_epi64(x, z);\
        x = _mm256_permute2x128_si256(x2tmp,x3tmp,0x20);\
        z = _mm256_permute2x128_si256(x2tmp,x3tmp, 0x31);\
        x2tmp = _mm256_unpacklo_epi64(y, w);\
        x3tmp = _mm256_unpackhi_epi64(y, w);\
        y = _mm256_permute2x128_si256(x2tmp,x3tmp,0x20);\
        w = _mm256_permute2x128_si256(x2tmp,x3tmp, 0x31);\
}

void sliscp_light256(u32 *state)
{
        u8 i, j;
        u32 t1, t2;
        __m256i x2tmp, x3tmp;
        __m256i x[4];

//Packing state
x[0]=_mm256_loadu2_m128i((void *)(state+4),(void *)(state));
x[2]=_mm256_loadu2_m128i((void *)(state+12),(void *)(state+8));
x[1]=_mm256_loadu2_m128i((void *)(state+20),(void *)(state+16));
x[3]=_mm256_loadu2_m128i((void *)(state+28),(void *)(state+24));
PACK(x[0], x[1], x[2], x[3]);

for(i = 0; i < STEP_ROUND; i++ ){
        //SSb
        PACK_SSb(x[2],x[3]);
        for ( j = 0; j < SIMECK_ROUND; j++ ){
                t1 = (u32)((rc1_256[i] >> j)&1);
```

```
                t2 = (u32)((rc2_256[i] >> j)&1);
                ROAX(x[2], x[3], t1, t2);
        }
        UNPACK_SSb(x[2],x[3]);

        //ASc
        t1 = (u32)RC1_256[i];
        t2 = (u32)RC2_256[i];
        x2tmp = x[0]^SC(t1,t2);
        x3tmp = x[1]^SC(t1,t2);

        //MSb
        x[0] = x[2];
        x[1] = x[3];
        x2tmp^=x[2];
        x3tmp^=x[3];
        x[2] = SWAPBLK(x2tmp);
        x[3] = SWAPBLK(x3tmp);
}

// Unpacking state
UNPACK(x[0], x[1], x[2], x[3]);
_mm_storeu_si128(
        (void *)(state),_mm256_extracti128_si256(x[0],0) );
_mm_storeu_si128(
        (void *)(state+4),_mm256_extracti128_si256(x[0],1) );
_mm_storeu_si128(
        (void *)(state+8),_mm256_extracti128_si256(x[2],0) );
_mm_storeu_si128(
        (void *)(state+12),_mm256_extracti128_si256(x[2],1) );
_mm_storeu_si128(
        (void *)(state+16),_mm256_extracti128_si256(x[1],0) );
_mm_storeu_si128(
        (void *)(state+20),_mm256_extracti128_si256(x[1],1) );
_mm_storeu_si128(
        (void *)(state+24),_mm256_extracti128_si256(x[3],0) );
_mm_storeu_si128(
        (void *)(state+28),_mm256_extracti128_si256(x[3],1) );

return;
}
```

F Test Vectors for sLiSCP-light Permutations

F.0.1 sLiSCP-light-192.

Input and output of the sLiSCP-light-192 permutation:
Input:000000000000000000000000000000000000000000000000
Output:B667F4B427DB32B6F61B8396808CBFD644FB94305A1A1B09

F.0.2 sLiSCP-light-256.

Input and output of the sLiSCP-light-256 permutation:
Input:0000000000000000000000000000000000000000000000000000000000000000
Output:2B858D69E03F180C96536EBDE32B14371B3E1E8EAD09B3725B6D84811668EACE