sLiSCP: Simeck-based Permutations for Lightweight Sponge Cryptographic Primitives

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 propose a family of lightweight cryptographic permutations called sLiSCP, with the sole aim to provide a realistic *minimal design* that suits a variety of lightweight device applications. More precisely, we argue that for such devices the chip area dedicated for security purposes should, not only be consumed by an encryption or hashing algorithm, but also provide as many cryptographic functionalities as possible. Our main contribution is the design of a lightweight permutation employing a 4-subblock Type-2 Generalized-like Structure (GFS) and round-reduced unkeyed Simeck with either 48 or 64-bit block length as the two round functions, thus resulting in two lightweight instances of the permutation, sLiSCP-192 and sLiSCP-256. We leverage the extensive security analysis on both Simeck (Simon-like functions) and Type-2 GFSs and present bounds against differential and linear cryptanalysis. In particular, we provide an estimation on the maximum differential probability of the round-reduced Simeck and use it for bounding the maximum expected differential/linear characteristic probability for our permutation. Due to the iterated nature of the Simeck round function and the simple XOR and cyclic shift mixing layer of the GFS that fosters the propagation of long trails, the *long trail strategy* is adopted to provide tighter bounds on both characteristics. Moreover, we analyze sLiSCP against a wide range of distinguishing attacks, and accordingly, claim that there exists no structural distinguishers for sLiSCP with a complexity below $2^{b/2}$ where b is the state size. We demonstrate how sLiSCP can be used as a unified round function in the duplex sponge construction to build (authenticated) encryption and hashing functionalities. The parallel hardware implementation area of the unified duplex mode of sLiSCP-192 (resp. sLiSCP-256) in CMOS 65 nm ASIC is 2289 (resp. 3039) GEs with a throughput of 29.62 (resp. 44.44) kbps, and their areas in CMOS 130 nm are 2498 (resp. 3319) GEs.

Keywords: Lightweight cryptography, Cryptographic permutation, Simeck block cipher, Generalized Feistel Structure, Sponge duplexing, Authenticated encryption, Hash function.

1 Introduction

The area of lightweight cryptography has been investigated in the literature for over a decade, however, only recently NIST [57] has recognized that there is a lack of standards that suit the bursting variety of constrained applications. In fact, long before NIST's lightweight cryptography project [57], the cryptographic community has, in an ad-hoc manner, tried to establish some common criteria on how to define a lightweight cryptographic design (e.g., 2000 GE for hardware area) [6, 47, 29]. Nevertheless, such criteria are rather generic, and specifically the established bound on hardware area represents an upper bound for a passive RFID tag which may contain between 1000 and 10000 GEs, out of which, a maximum of 20% is to be used for all security functionalities [47]. Certainly, it is not reasonable to compare a vehicular embedded system with an Electronic Product Code (EPC) tag or an implantable medical device where the critical operation environment imposes strict constraints on the area and power consumption. While the latency maybe considered of a paramount importance for some applications such as automotive embedded

This is the full version of a paper due to appear in the Springer-Verlag proceedings of SAC 2017.

systems that require a fast response time, it can be relaxed so that smaller area is realized in highly hardware constrained applications (e.g., EPC tags). What remains the most important aspect in an acceptable realistic secure lightweight cryptographic design is its hardware footprint given that it offers acceptable metrics for throughput and latency.

Over the last decade, numerous symmetric-key primitives such as block ciphers, stream ciphers and hash functions have been proposed to secure resource-constrained applications. Examples of block ciphers include TEA [67], KATAN and KTANTAN [37], LED [44], PRESENT [29], HIGHT [46], EPCBC [73], KLEIN [41], LBlock [69], TWINE [63], Piccolo [61], PRINCE [32], SIMON and SPECK [12], SIMECK [72], PRIDE [4] and SKINNY [13], lightweight hash function examples include PHOTON [43], QUARK [7], SPONGENT [28], and LHash [68], and lightweight stream cipher examples encompass Grain-128 [45], Trivium [36], MICKY [11], and WG [58]. These proposals aim to achieve hardware efficiency by adopting efficient round or feedback functions so that the targeted cryptographic functionality is provided while guaranteeing its security. However, none of theses proposals have considered providing multiple cryptographic functions with low overhead, which might be a determining factor for its realistic adoption in many constrained devices. In other words, it is reasonable to assume that the available chip area dedicated for security purposes should be used to provide encryption, authentication, hash computation, and possibly pseudo-random bit generation, which are the basic functionalities required by a security protocol. Similar to the advantage of having an encryption algorithm where both encryption and decryption use the same round function, the concept of *cryptographic minimal design* aims to unify one design for as many cryptographic functionalities as possible. As a trade-off for having a minimal design, some redundancy may be introduced and thus, latency and throughput of individual functionalities may not be optimized.

In recent years, various authenticated encryption (AE) schemes have been developed (e.g., CAESAR competition [34]). Of particular interest are NORX-16 [9] and Ketje-JR [14] as they have state sizes of 256 (2880 GEs) and 200 (1270 LUTs), respectively, and also the lightweight AE scheme Grain-128a (est. 2243 GE) [3]. However, all the latter lightweight AE schemes are optimized (e.g., MonkeyDuplexing [20]) for authenticated encryption and not to be used as a hash function. One can achieve a minimal design using the Keccak permutation family [21]. However, the smallest instance of the Keccak family is Keccak-200 whose implementation cost in the duplex mode is 4900 GE for 130 nm ASIC technology [48]. Consequently, we believe that there is a need to explore the design space of secure lightweight cryptographic permutations which are suitable for unifying a cryptographic design with a minimal overhead of multiple cryptographic functionalities.

Our Contributions. In this work, we present sLiSCP which is a family of lightweight cryptographic permutations with the sole aim to provide a realistic lightweight cryptographic minimal design. Thus, our contributions in this paper are as follows.

- We design the sLiSCP family of permutations, which adopts two of the most efficient and extensively analyzed cryptographic structures, namely a 4-subblock Type-2 Generalized Feistel-like Structure (GFS) [59, 31], and a round-reduced unkeyed version of the Simeck encryption algorithm [72]. Specifically, the round function of Simeck is an independently parameterized version of the Simon round function [12] and has set a new record in terms of hardware efficiency and performance on almost all platforms. Moreover, Simeck, Simon and Simon-like variants have been extensively cryptanalyzed by the public cryptographic community [65, 2, 24, 51, 52, 54]. Simeck utilizes shift parameters that are more hardware efficient than those of Simon, and also in terms of bit diffusion, Simeck-48 is better than other efficient shift parameters that are investigated in [51]. As for the overall GFS construction, it is proven that Type-2 GFSs offer better diffusion than Type-1 GFSs, and having 4 subblocks bounds the number of round functions to only two which satisfies the constrained area requirement [62]. Additionally, 4-subblock Type-2 GFSs reach full subblock diffusion in 4 rounds which enables sLiSCP to provide good security margins after an acceptable number of rounds. Also, we leverage the benefits of using Linear Feedback Shift Register (LFSR) based constants in the round function in order to further optimize the hardware footprint.
- We investigate the security of the sLiSCP permutation against a wide variety of distinguishing attacks. As the design of sLiSCP is based on Type-2 GFS and Simeck's round function, our analysis makes use of the extensive analysis on both Type-2 GFSs, Simeck, and Simon-like round functions. Unlike primitives such as PRESENT [29] and LED [44] block ciphers that offer strong security bounds which

are relatively easy to cryptanalyze because of the use of small Sboxes and/or MDS diffusion layers, the Simeck round function adopts a rather efficient and simple bitwise operations (AND, XOR and Rotation), which makes their cryptanalysis not as easy. We use the SMT/SAT tool developed in [51] to compute the bounds on the maximum expected probabilities for differential and linear characteristics for Simeck. Using these bounds and encoding sLiSCP in a Mixed Integer Linear Programming (MILP) model, we calculate the bounds for the probabilities of the differential and linear distinguishers based on the minimum number of active round-reduced Simeck functions. The bounds on the minimum number of active Simeck functions trail are further tightened by adopting the long trail strategy, which shows that the analysis based on the MILP and MEDCP (Maximum Expected Differential Characteristic Probability) provides loose bounds on the actual security of sLiSCP against differential and linear cryptanalysis. The security of sLiSCP against the known attacks exploiting the low bit diffusion is ensured by choosing the number of rounds to be three times the number of rounds required for achieving full bit diffusion, as proposed by Gueron and Mouha in Simpira V2 [42]. We also investigate the security of sLiSCP against integral, impossible differential, zero correlation, zero-sum, and self symmetry-based distinguishers and claim that it has no structural distinguishers with complexity less than $2^{b/2}$ where b is the state size. This kind of claim has been used in the settings of the security claims of Keccak permutation [16] and Simpira V.2 [42] in response to mainly zero-sum distinguishers which can cover the full round permutation with very high complexity.

- We demonstrate how to use the sLiSCP permutation to construct authenticated encryption (AE) schemes and hash functions in a duplex unified sponge construction. The parameter choices and the security levels offered by the proposed instances are reported.
- We implement the two instances of sLiSCP in the duplex unified sponge mode, and our parallel ASIC implementation results in CMOS 65 nm show that the areas of sLiSCP-192 and sLiSCP-256 are 2289 GEs and 3039 GEs with a throughput of 29.62 and 44.44 kbps, respectively, and their areas in CMOS 130 nm are 2498 GEs (resp. 3319).

The rest of the paper is organized as follows. Section 2 presents the rationales behind our design and choice of parameters. In Section 3, we define the notations used throughout the paper and review the concepts and definitions related to our construction. We present the general construction of the permutation sLiSCP and its two instances, the Simeck^u-m box and its cryptographic properties in Section 4. In Section 5, we analyze sLiSCP with respect to a variety of distinguishing attacks. In Section 6, we show how we use sLiSCP in a sponge framework to construct authenticated encryption, stream cipher, MAC, and hash functionalities. The hardware implementation and a comparison with other lightweight cryptographic primitives are given in Section 7. Finally, the paper is concluded in Section 8.

2 Design Rationale

Our main objective for a minimal design is to provide as many cryptographic primitives as possible. Using sLiSCP in the duplex sponge construction is an excellent choice as it offers a number of key features that enable the design of multiple cryptographic functionalities using the same hardware circuitry. In other words, both keyed primitives such as (authenticated) encryption and MAC computation, and unkeyed primitives such as hash function and pseudo randome bit generator are easily realized with minimum overhead. The sponge construction ensures provable security [15] when the underlying permutation is indistinguishable from a random function. Accordingly, the main challenge is to design a secure and hardware efficient permutation for resource constrained applications. For an unkeyed sponge-based primitive with state size $b = r + c$, where r and c denote the rate and capacity, respectively, a bound of $2^{c/2}$ against generic attacks is proven [15], which sets a lower bound on the state size of the permutation. If the permutation is used to construct a hash function with output of size t, the permutation state size should be at least $(2t + r)$ bits. For lightweight applications, a hash digest of 160 bits restricts the state size b to a minimum of 192 bits for a rate of 32 bits, which means that around 1000 GEs are reserved only for the state. While a substitution permutation network (SPN) based design requires a relatively small number of rounds to achieve the desired security, offers good throughput, and is simpler to cryptanalyze, the hardware implementation cost becomes expensive for a larger state size due to the large number of substitution boxes and their cost. To design a lightweight permutation, we opted for a non-SPN based design consisting of Type-2 GFS and a

round-reduced Simeck-m (i.e. Simeck with m-bit state) as a round function, which are based on the Feistel network. Our design choices are motivated by the following design goals:

- **Hardware efficiency:** Both the implementations of Type-2 GFS and the Simeck-m round functions are extremely efficient in terms of the hardware footprint. In fact, even though Simon has one of the minimum hardware footprints in the literature, more GE savings are achieved by selecting more hardware friendly shift parameters which is how Simeck is designed [72]. To provide an average estimation on the GE count, we assume an ASIC $65\,nm$ technology that requires 2.5 GE for an XOR, 2 GE for an AND. Given the latter estimates, a 4-subblock Type-2 GFS using Simeck-48 mixing requires around $2 \times 48 \times 2.5 = 240$ GE and each Simeck-48 round function consumes around $24 \times (2.5 + 2 + 2.5) = 168$ GE, which sums to around 576 GEs for the parallel implementation of the permutation round function logic with a state size of $4 \times 48 = 192$ bits.
- **Security:** Our goal is to design a permutation that is indistinguishable from a random one. The security of sLiSCP is based on the Type-2 GFS and the Simeck round function, both of which have been extensively cryptanalyzed in literature. Our design criteria that offer bounds against differential and linear cryptanalysis, and the choice of the number of steps to be three times the number of steps required for the full bit diffusion offer the security against known attacks. Nevertheless, we investigate sLiSCP against integral, impossible differential, zero correlation and zero-sum distinguishers. According to our analysis, we claim that there are no structural distinguishers for the full round permutation with complexity less than $2^{b/2}$ where b denotes the state size of the permutation. Since, we plan to use sLiSCP in the sponge-based designs, our security claim follows that of the Keccak permutation [16] which is also used for the Simpira v2 permutation [42].

The sponge construction is well-investigated and has been cryptanalyzed and proven secure for different keyed and unkeyed applications including the SHA-3 winner Keccak [16]. Moreover, its security when instantiated with a specific permutation F relies on the resistance of F against distinguishing attacks and accordingly, we focus our cryptanalysis efforts to investigating sLiSCP against a wide variety of such attacks.

Why Simeck is our design choice? The Simeck round function is an independently parameterized version of the generalized Simon counterpart. More precisely, the shift parameters (a, b, c) used in Simon and Simeck are $(1, 8, 2)$ and $(5, 0, 1)$, respectively, which make Simeck more hardware efficient, and based on the analysis presented in [51, 52, 70] there is only slight difference in the security against differential, linear, integral, and impossible differential distinguishers. In particular, due to the fact that Simeck has one bit less diffusion, such distinguishers can usually cover one or two more rounds compared to using the Simon round function. However, among the hardware efficient shift parameters which have been investigated in [51] and have been shown to provide slightly better behavior with respect to differential and linear cryptanalysis, we found that Simeck-48 parameters provide a slightly better bit diffusion. Specifically, the shift parameters (7,0,2) have been shown to provide full bit diffusion after nine rounds of applying the Simon-48 round function. While this is the same case for Simeck-48, when investigating the round-reduced functions, for example, for six rounds, we find that $(7, 0, 2)$ provide 40 and 32 bits of diffusion in the two halves of the Feistel state and Simeck with shift parameters (5,0,1) provides 41 and 34 bits of diffusion. Additionally, Simeck has been independently cryptanalysed in the literature with respect to a wide variety of attack sources for over more than three years [52, 70, 54].

3 Preliminaries

In this section, we define our notations for expected linear and differential probability and give a brief description of the Type-2 Generalized Feistel Structure.

3.1 Expected Linear and Differential Probability

We denote by $\mathbb{F}_2$ the field with two elements and by $\mathbb{F}_2^m$ the m-dimensional vector space over $\mathbb{F}_2$. Moreover, $(\mathbb{F}_2^m)^n$ denotes the n-dimensional vector space over $\mathbb{F}_2^m$. By $+$ and $\odot$ we mean bitwise XOR and bitwise

AND in $\mathbb{F}_2^m$, respectively. For a vectorial boolean function $f : \mathbb{F}_2^m \to \mathbb{F}_2^m$ with input (resp. output) difference δ_{in} (resp. δ_{out}), the differential probability, denoted by $\Pr(\delta_{in} \to \delta_{out})$, is defined as

$$\Pr(\delta_{in} \to \delta_{out}) = \frac{|\{x | f(x) + f(x + \delta_{in}) = \delta_{out}\}|}{2^m}.$$

The squared correlation of f with input mask Γ_{in} and output mask Γ_{out} is defined by

$$C^2(\Gamma_{in} \to \Gamma_{out}) = \left(\frac{\tilde{f}(\Gamma_{in}, \Gamma_{out})}{2^m}\right)^2,$$

where $\tilde{f}(\Gamma_{in}, \Gamma_{out}) = \sum_{x \in \mathbb{F}_2^m} (-1)^{(\langle x, \Gamma_{in} \rangle \oplus \langle f, \Gamma_{out} \rangle)}$, $\langle x, y \rangle$ denotes the inner product between vectors x and y, and $\oplus$ is the XOR operation over $\mathbb{F}_2$.

Let f^u denote the u-fold iteration of f. We say that a differential characteristic $\delta_0 \to \delta_1 \to \ldots \to \delta_u$ is the maximum differential characteristic with probability p ($p \neq 0$) if no other u round differential characteristic exists with probability greater than p. Let Δ^u be the set of all differential characteristics of f^u with probability $p > 0$. For f^u, the maximum expected differential characteristic probability (MEDCP) is given by

$$MEDCP(f^u) = \max_{(\delta_0 \to \cdots \to \delta_u) \in \Delta^u} \prod_{j=0}^{u-1} \Pr(\delta_j \to \delta_{j+1}) \tag{1}$$

where $\Pr(\delta_j \to \delta_{j+1})$ is the expected differential probability of $\delta_j \to \delta_{j+1}$ over one round of the keyed function f, when the key is picked uniformly and independently at random [39, 49]. The maximum expected linear characteristic correlation (MELCC) can be defined analogously.

3.2 Type-2 Generalized Feistel Structure

A Type-2 GFS [59, 31] is a Feistel network with m (even) branches, $\frac{m}{2}$ functions $g_0, g_1, \ldots, g_{\frac{m}{2}-1}$ and a function $G : (\mathbb{F}_2^n)^m \to (\mathbb{F}_2^n)^m$ given by $G(X_0, X_1, \ldots, X_{m-1}) = (Y_0, Y_1, \ldots, Y_{m-1})$, where

$$Y_i = \begin{cases} X_{(i+1)}, & i \equiv 0 \ mod\ 2 \\ g_{\frac{i+1}{2} mod (\frac{m}{2})}(X_{(i+2) mod (m)}) + X_{(i+1) mod (m)}, & i \not\equiv 0 \ mod\ 2 \end{cases}$$

and the functions $g_i : \{0,1\}^n \to \{0,1\}^n$ are called round functions. Figure 1 depicts the iterated construction of a generic Type-2 GFS.

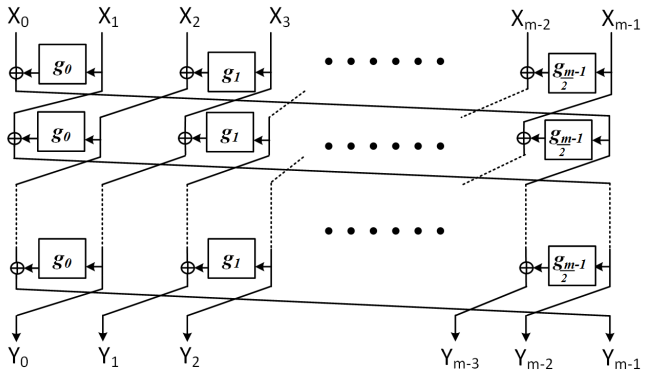

Fig. 1: A block diagram of the Type-2 Generalized Feistel Structure.

4 Specification of sLiSCP

In this section, we formally describe our sLiSCP permutation, which has two instances. The core algorithm of the sLiSCP permutation is built upon the Simeck cipher's round function and a 4-subblock Type-2 GFS construction. We present two lightweight instances of the sLiSCP permutation.

4.1 Description of Simeck^u-m

We use Simeck^u-m as a round function in the sLiSCP permutation. Simeck^u-m is derived from the Simeck cipher whose block length equal to m and round function is given by:

$$h_i(x) = R_i(x_0, x_1) = (((x_0 \lll 5) \odot (x_0 \lll 0)) + (x_0 \lll 1) + x_1 + rk_i, x_0), x = x_0 \| x_1$$

where $h_i : \mathbb{F}_2^m \to \mathbb{F}_2^m$, $\lll$ is a left cyclic shift operator, x_0 and x_1 are $\frac{m}{2}$-bit words and rk_i is a $\frac{m}{2}$-bit round key added at the i-th round. We modify the round function as follows. Instead of adding a round key in $h_i, 0 \leq i \leq u-1$, we add a round constant rc_i in h_i where $rc_i = (C|t_i), C = (2^{\frac{m}{2}} - 2), t_i \in \mathbb{F}_2$ and $C|t_i$ denotes the bitwise OR between C and t_i. Let t be the integer representation of the u-tuple $(t_0, t_1, \cdots, t_{u-1})$. Simeck^u-$m$ is defined as

$$\text{Simeck}^u\text{-}m(x) = h_{u-1} \circ h_{u-2} \circ \cdots \circ h_0(x), x = x_0 \| x_1,$$

where the round constant rc_i is used in h_i at the i-th round. The round constants are generated using an LFSR described in Section 4.4. We, henceforth, refer to Simeck^u-m as h_t^u. We use the subscript t to uniquely instantiate h_t^u as $h_{t_1}^u$ and $h_{t_2}^u$, for $t_1 \neq t_2$, which are parametrized by different round constants, t_1 and t_2. In sLiSCP, we choose all the round functions of GFS to be Simeck^u-m and we consider it as an Sbox. We study the cryptographic properties of Simeck^u-m and present the bounds against differential and linear cryptanalysis for sLiSCP based on the minimum number of active Sboxes.

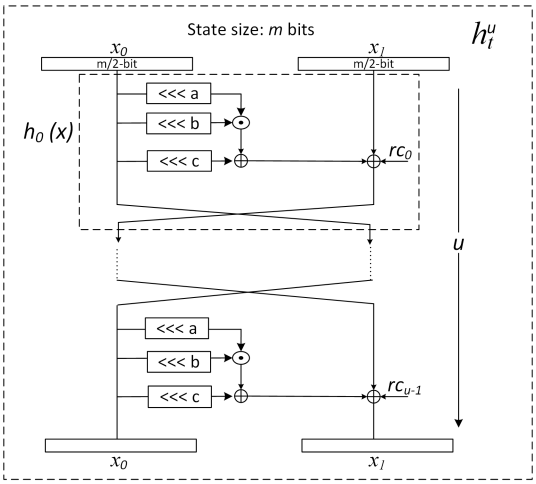

Fig. 2: Simeck round function where $(a, b, c) = (5, 0, 1)$.

Definition 1 (Simeck^u-m box) *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.*

4.2 Cryptographic properties of Simeck^u-m

The cryptographic properties of an Sbox include differential uniformity (a.k.a maximum differential probability (MDP)), algebraic degree, and branching number. For an Sbox with large input size, it is infeasible

to compute the exact value of the MDP. We use the maximum estimated differential probability (MEDP) to provide better estimates for differential characteristics probabilities for the sLiSCP permutation. In what follows, we analyze the above properties for Simeck^u-m.

Estimating the MEDP of Simeck^u-m. In our permutations, we employ a modified Simeck-48 and Simeck-64 round functions for Simeck^u-48 and Simeck^u-64, respectively. Due to their large state sizes, it is infeasible to build their differential distribution tables and compute the exact MDP values. Alternatively, we estimate the MEDP of Simeck^u-m as tight as possible using the results in [51]. Table 1 presents the log_2 probabilities of maximum expected differential and linear squared correlation characteristics for Simeck-48 and Simeck-64.

Table 1: Probabilities of the maximum differential and linear characteristics for Simeck^u-m, where $m = 48$ and 64. The probabilities are given in the $log_2(\cdot)$ scale. Squared correlation is used for linear characteristic, thus the duality between both probabilities.

Rounds u	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	$\cdots$	24
Simeck^u-m	0	-2	-4	-6	-8	-12	-14	-18	-20	-24	-26	-30	-32	-36	-38	-44	-44	-46	$\cdots$	-62

For Simeck^u-m, we run the tool in [51] based on the SAT/SMT solvers CryptoMiniSat and STP to obtain the results in Table 1. The maximum differential probability is usually approximated by the maximum probability of a differential characteristic where such a probability is commonly assumed to be associated with differentials from characteristics with the optimal probability [42, 39]. However, this probability may change if such a differential is satisfied with a large number of characteristics (*differential effect*). Accordingly, we extract these differentials from all the optimal characteristics and get their corresponding probabilities, then we approximate the MDP of the Simeck^u-m box by the maximum probability among these differentials. For that, we get tighter (but not strong) bounds for the probabilities of differential and linear characteristics of the whole permutation than by using the Simeck's box maximum differential characteristic probability. For Simeck^u-m, the probability of a differential (δ_0, δ_u) is defined as

$$\Pr(\delta_0 \xrightarrow{h_t^u} \delta_u) = \sum_{i=w_{min}}^{w_{max}} s_i \cdot 2^{-i}$$

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 probabilities equal to 2^{-i}. Accordingly, we are able to evaluate the probability of such differentials, and thus use the maximum probability among them as the MEDP of the Simeck^u-m box (shown in Table 2) which is then calculated using Eq. (2) and is given by

$$\text{MEDP}(\text{Simeck}^u\text{-}m) = \max_{(\delta_0,\delta_u)\in\Delta^u} (\Pr(\delta_0 \xrightarrow{h_t^u} \delta_u)), \tag{2}$$

where Δ^u denotes the set of differentials associated with characteristics with maximum probabilities.

Table 2: Estimation for the MDP for Simeck$^u - m$ for $m = 48$ and 64

Rounds (u)	1	2	3	4	5	6	7	8	9
MEDP (Simeck^u-48)	0	-2	-4	-6	-8	-11.299	-13.298	-16.597	-18.595

Instantiating u in Simeck^u-m. Based on our analysis, we decided to choose $u = 6$ for Simeck^u-48 and $u = 8$ for Simeck^u-64 because we found that if we opted for nine (resp. 11) rounds so that the full bit

diffusion is achieved at one sub-block after one step, then we need five steps for the full bit diffusion at the state level (i.e., every state bit depends on all input bits), and thus based on our design criteria, fifteen steps are required for the permutation. This results in $15 \times 9 = 135$ (resp. $15 \times 11 = 165$) Simeck rounds for the permutation. On the other hand, by having six (resp. 8) rounds for Simeck^u-m, full bit diffusion is achieved at the state level in six steps, as a result, we only need eighteen steps for the permutation. Hence, sLiSCP employing Simeck-48 requires $18 \times 6 = 108$ rounds and sLiSCP employing Simeck-64 requires $18 \times 8 = 144$ rounds. It can be seen in Table 1 that for Simeck-48 (resp. Simeck-64), the optimal characteristic probability decreases by a factor of 16 between 5 rounds and 6 rounds (resp. between 7 and 8 rounds) and by a factor of only 4 between 6 and 7 rounds (resp. between 8 and 9 rounds), which further enhances the resistance of Simeck^u-m parametrized by our round choices against differential and linear attacks. To achieve, a better throughput and good resistance to differential and linear cryptanalysis, we opted for $u = 6$ rounds and $u = 8$ rounds for Simeck^u-48 and Simeck^u-64, respectively.

For both Simeck^6-48 and Simeck^8-64, we consider all $(\delta_{in}, \delta_{out})$ differential pairs with $wt(\delta_{in}) = wt(\delta_{out}) = 1$ and we found that the probability of such differentials is zero, which means that both Sboxes have a branching number larger than 2[1]. Using a tweaked method based on the propagation of the division property [64], we have computed the algebraic degree of Simeck^6-48 (resp. Simeck^8-64) component functions and our experimental results show that half of the components has algebraic degree of 13 (resp. 27) and the other half has a degree of 19 (resp. 36).

4.3 The Permutation F

The sLiSCP permutation is an iterative permutation, denoted by sLiSCP-b, over $\mathbb{F}_2^b$ where $b = 4 \times m$ and m is even. The construction of sLiSCP is based on Simeck^u-m (denoted as h_t^u) and a Type-2 GFS construction. An architecture of the sLiSCP permutation is shown in Fig. 3. Let $(X_0^0, X_1^0, X_2^0, X_3^0)$ and $(X_0^s, X_1^s, X_2^s, X_3^s)$ be the input and output to the s-step permutation F, respectively where $X_i^0, X_i^s \in \mathbb{F}_2^m, 0 \leq i \leq 3$. Let $f : \mathbb{F}_2^{4m} \rightarrow \mathbb{F}_2^{4m}$ denote the step function. Then the permutation F is defined in terms of f as

$$F(X_0^0, X_1^0, X_2^0, X_3^0) = f^s(X_0^0, X_1^0, X_2^0, X_3^0) = (X_0^s, X_1^s, X_2^s, X_3^s),$$

where $(X_0^{j+1}, X_1^{j+1}, X_2^{j+1}, X_3^{j+1}) = f(X_0^j, X_1^j, X_2^j, X_3^j)$, for all $j = 0, 1, \ldots, s-1$. The step function f is built on a 4 sub-block Type-2 GFS with the second and fourth blocks at step j being processed by Simeck^u-m, which is defined as

$$f(X_0^j, X_1^j, X_2^j, X_3^j) = (X_1^j, h_{t'}^u(X_3^j) + X_2^j + (C'|SC_{2j+1}), X_3^j, h_t^u(X_1^j) + X_0^j + (C'|SC_{2j})), \tag{3}$$

where $t = (RC_{12j}, \cdots, RC_{12j+2(u-1)})$, $t' = (RC_{12j+1}, \cdots, RC_{12j+2u+1})$, $C' = 2^m - 256$, $P|Q$ denotes the bitwise OR between P and Q, and h_i is given by

$$h_i(x_0, x_1) = (((x_0 \lll 5) \odot (x_0 \lll 0)) + (x_0 \lll 1) + x_1 + (C|\gamma_i), x_0), 0 \leq i \leq u-1,$$

where $C = 2^{\frac{m}{2}} - 2$ and

$$\gamma_i = \begin{cases} t_i, & \text{for } h_t^u \\ t_i', & \text{for } h_{t'}^u. \end{cases}$$

For the round constants RC_i and SC_i generation, the reader is referred to Section 4.4. The number of rounds for sLiSCP-b is $(u \cdot s)$. Table 3 presents the parameters for two lightweight instances of the sLiSCP permutation, which are denoted by sLiSCP-192 and sLiSCP-256.

4.4 Round and Step Constants for sLiSCP-192

To mitigate self-similarity attacks, we add constants in the Simeck function rounds and GFS permutation steps. The constants added to Simeck^u-m are called round constants and the constants added to the GFS are called step constants. We use only one LSFR to generate both the round and step constants. For

[1] Finding the exact branch number is infeasible due to large size of Simeck^u-m

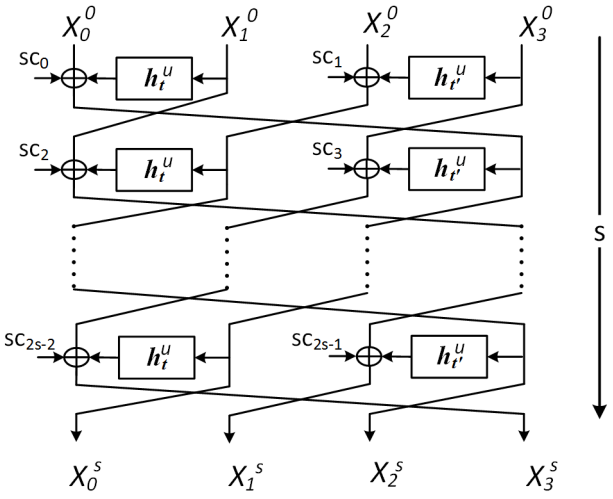

Fig. 3: sLiSCP permutation following 4-subblock Type-2 GFS using Simeck^u-m as h_t^u.

Table 3: Parameter set for the 4 subblock GFS permutation F (sLiSCP-192 and sLiSCP-256).

Permutation (b-bit)	Subblock width m	Rounds u	Steps s	State width ($b = 4m$)	Total # rounds ($u \cdot s$)
sLiSCP-192	48	6	18	192	108
sLiSCP-256	64	8	18	256	144

sLiSCP-192, we use a 6-stage LFSR to generate a one bit constant for each round of h_t^6 and $h_{t'}^6$. Each single bit is further appended to $C = 2^{\frac{m}{2}} - 2$ and added to the right half of a given h_t^6 ($h_{t'}^6$) as shown in Fig. 2. Note that we let the LFSR run continuously so the 6-bit constant sequence for h_t^6 at different steps are also different. Moreover, we add two external step 6-bit constants SC_{2j}, SC_{2j+1} to the 48-bit output of h_t^6 and $h_{t'}^6$ in the j-th step of F in Eq. (3) so that all steps are distinctive and every h_t^6 out of the $18 \times 2 = 36$ functions used in sLiSCP-192 permutation is made unique by the 12-bit tuple (t, SC_i). We assume the LFSR is generated by a primitive polynomial of degree 6: $x^6 + x + 1$ and denote the generated sequence by $(s_0, s_1, s_2, s_3, \cdots)$ and its 2-decimated sequence is given by $(s_0, s_2, s_4, s_6, \cdots)$.

Round constants (RC). We use the LFSR depicted in Fig. 4 to generate the round constants. The j-th step round constants for h_t^6 and $h_{t'}^6$ are given by $(C|RC_{12j+2i})$ and $(C|RC_{12j+2i+1}), 0 \leq i \leq 5$, respectively. Note that, $\{s_{2k}\}$ and $\{s_{2k+1}\}$ are both 2-decimated sequences with same period $(2^6 - 1)$ of the original one. In addition, the period of the 6-tuple $(s_{12k}, s_{12k+2}, s_{12k+4}, s_{12k+6}, s_{12k+8}, s_{12k+10})$ is $63/gcd(6, 63) = 21$. Hence, the pair of round constants (t, t') starts repeating after 21 iterations.

Step constants (SC). At the 6-th round of the internal function, a 6-bit constant will be generated from the same parallel LFSR as the round constants with four extra XORs. At the 6-th clock cycle in each 6-th round at j-th step, the states for the LFSR are $(s_{12j+10}, s_{12j+11}, s_{12j+12}, s_{12j+13}, s_{12j+14}, s_{12j+15})$, we assign $(s_{12j+10}, s_{12j+12}, s_{12j+14})$ to SC_{2j} and $(s_{12j+11}, s_{12j+13}, s_{12j+15})$ to SC_{2j+1}. For the other three values of SC_{2j} $(s_{12j+16}, s_{12j+18}, s_{12j+20})$ and SC_{2j+1} $(s_{12j+17}, s_{12j+19}, s_{12j+21}$), we use the following equations:

$$s_{12j+16} = s_{12j+10} \oplus s_{12j+11},\ s_{12j+17} = s_{12j+11} \oplus s_{12j+12},\ s_{12j+18} = s_{12j+12} \oplus s_{12j+13}$$
$$s_{12j+19} = s_{12j+13} \oplus s_{12j+14},\ s_{12j+20} = s_{12j+14} \oplus s_{12j+15},\ s_{12j+21} = s_{12j+15} \oplus s_{12j+16}.$$

Therefore, we use four more XORs to generate the remaining three bits of SC_{2j} and SC_{2j+1} based on the parallel LFSR in Fig. 4. Thus

$$SC_{2j} = s_{12j+10}||s_{12j+12}||s_{12j+14}||s_{12j+16}||s_{12j+18}||s_{12j+20}$$
$$SC_{2j+1} = s_{12j+11}||s_{12j+13}||s_{12j+15}||s_{12j+17}||s_{12j+19}||s_{12j+21}.$$

The step constants at the j-th step of F are then given by $(C'|SC_{2j})$ and $(C'|SC_{2j+1})$ where $C' = 2^m - 256$. The entire architecture of the parallel LFSR with feedback function $x^6 + x + 1$ to generate the j-th step round and step constants is shown in Fig. 4.

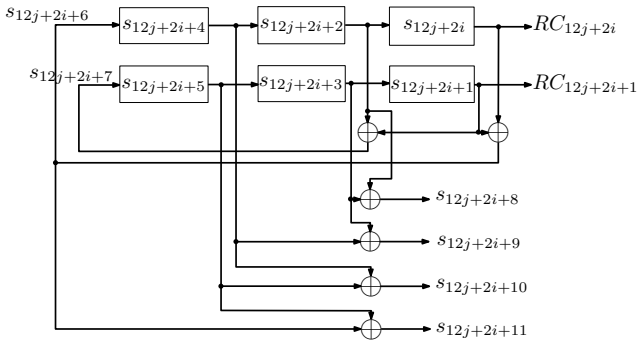

Fig. 4: Round and step constants generated by parallel LFSR at i-th round of j-th step.

Both the round and step constants for sLiSCP-192 are given in Table 4 in hexadecimal. At the j-th step of F, there is a pair of 6-bit round constants (t, t') and another pair of 6-bit step constants (SC_{2j}, SC_{2j+1}). For example, at step 0, the round constants $(t, t') = (7, 27)$ are assigned to $(h_t^6, h_{t'}^6)$ and similarly, the step constants $(SC_0, SC_1) = (8, 29)$ are used.

Table 4: Round and Step Constants for sLiSCP-192

Step j	**Round Constants** (t, t')	**Step Constants** (SC_{2j}, SC_{2j+1})
0 - 5	(7, 27), (4, 34), (6, 2e), (25, 19), (17, 35), (1c, f)	(8, 29), (c, 1d), (a, 33), (2f, 2a), (38, 1f), (24, 10)
6 - 11	(12, 8), (3b, c), (26, a), (15, 2f), (3f, 38), (20, 24)	(36, 18), (d, 14), (2b, 1e), (3e, 31), (1, 9), (21, 2d)
12 - 17	(30, 36), (28, d), (3c, 2b), (22, 3e), (13, 1), (1a, 21)	(11, 1b), (39, 16), (5, 3d), (27, 3), (34, 2), (2e, 23)

4.5 Round and Step Constants for sLiSCP-256

For sLiSCP-256, we use a 7-stage LFSR to generate a one bit constant for each round of h_t^8 and $h_{t'}^8$. Each single bit is appended to the constant $C = (2^{\frac{m}{2}} - 2)$ which is then added to the right half of a given h_t^8 $(h_{t'}^8)$ as shown in Fig. 2. Unlike the sLiSCP-192 case, we use a primitive polynomial of degree 7: $x^7 + x + 1$ with initial states of all "1"s to generate the LFSR constants. We let the LFSR run continuously so that the 8-bit constant sequence for h_t^8 at different steps are also different. Moreover, we add two external step 8-bit constants SC_{2j}, SC_{2j+1} to the 64-bit output of h_t^8 and $h_{t'}^8$ in the j-th step of F in Eq. (3) so that all steps are distinct and every h_t^8 out of the $18 \times 2 = 36$ functions used in the sLiSCP-256 permutation is made unique by the 16-bit tuple (t, SC_i).

Round constants (RC). We use the LFSR depicted in Fig. 5 to generate the round constants. The j-th step round constants for h_t^8 and $h_{t'}^8$ are given by $(C|RC_{16j+2i})$ and $(C|RC_{16j+2i+1}), 0 \leq i \leq 7$, respectively. Note that, both $\{s_{2k}\}$ and $\{s_{2k+1}\}$ are 2-decimated sequences with same period $(2^7 - 1)$.

Step constants (SC). At the 8-th round of the internal function, a 8-bit constant will be generated from the same parallel LFSR as the round constants with seven extra XORs. At the 8-th clock cycle in each 8-th round at j-th step, the states for the LFSR are $(s_{16j+14}, s_{16j+15}, s_{16j+16}, s_{16j+17}, s_{16j+18}, s_{16j+19}, s_{16j+20})$, we assign $(s_{16j+14}, s_{16j+16}, s_{16j+18}, s_{16j+20})$ to SC_{2j} and $(s_{16j+15}, s_{16j+17}, s_{16j+19})$ to SC_{2j+1}. The other four values of SC_{2j} $(s_{16j+22}, s_{16j+24}, s_{16j+26}, s_{16j+28})$ and five values of SC_{2j+1} $(s_{16j+21}, s_{16j+23}, s_{16j+25}, s_{16j+27}, s_{16j+29})$ are generated using the following equations:

$$s_{16j+21} = s_{16j+14} \oplus s_{16j+15},\ s_{16j+22} = s_{16j+15} \oplus s_{16j+16},\ s_{16j+23} = s_{16j+16} \oplus s_{16j+17}$$
$$s_{16j+24} = s_{16j+17} \oplus s_{16j+18},\ s_{16j+25} = s_{16j+18} \oplus s_{16j+19},\ s_{16j+26} = s_{16j+19} \oplus s_{16j+20}$$
$$s_{16j+27} = s_{16j+20} \oplus s_{16j+21},\ s_{16j+28} = s_{16j+21} \oplus s_{16j+22},\ s_{16j+29} = s_{16j+22} \oplus s_{16j+23}.$$

Therefore, seven more XORs are needed to generate the remaining nine bits of SC_{2j} and SC_{2j+1} based on the parallel LFSR in Fig. 5. Thus

$$SC_{2j} = s_{16j+14}||s_{16j+16}||s_{16j+18}||s_{16j+20}||s_{16j+22}||s_{16j+24}||s_{16j+26}||s_{16j+28}$$
$$SC_{2j+1} = s_{16j+15}||s_{16j+17}||s_{16j+19}||s_{16j+21}||s_{16j+23}||s_{16j+25}||s_{16j+27}||s_{16j+29}.$$

The step constants at the j-th step of F are then given by $(C'|SC_{2j})$ and $(C'|SC_{2j+1})$ where $C' = 2^m - 256$. The entire architecture of the parallel LFSR with feedback function $x^7 + x + 1$ to generate the j-th step round and step constants is shown in Fig. 5.

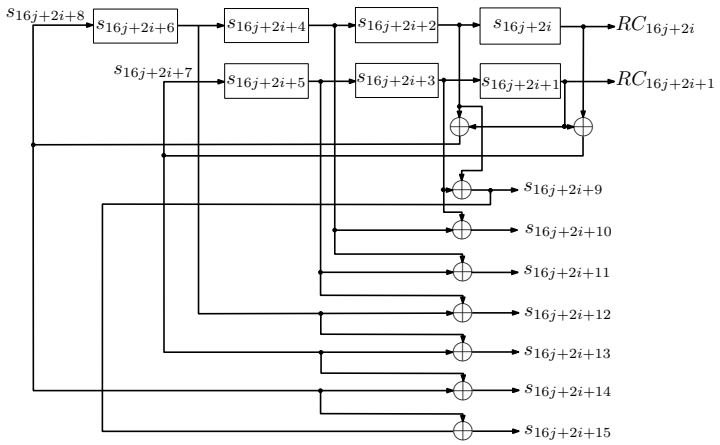

Fig. 5: Round and step constants generated by parallel LFSR at i-th round of j-th step.

Both the round and step constants for sLiSCP-256 are given in Table 5 in hexadecimal. At the j-th step of F, there is a pair of 8-bit round constants (t, t') and another pair of 8-bit step constants (SC_{2j}, SC_{2j+1}). For example, at step 0, the round constants $(t, t') = (f, 47)$ are assigned to $(h_t^8, h_{t'}^8)$ and similarly, the step constants $(SC_0, SC_1) = (8, 64)$ are used.

Table 5: Round and Step Constants for sLiSCP-256

Step j	**Round Constants** (t, t')	**Step Constants** (SC_{2j}, SC_{2j+1})
0 - 5	(f, 47), (4, b2), (43, b5), (f1, 37), (44, 96), (73, ee)	(8, 64), (86, 6b), (e2, 6f), (89, 2c), (e6, dd), (ca, 99)
6 - 11	(e5, 4c), (b, f5), (47, 7), (b2, 82), (b5, a1), (37, 78)	(17, ea), (8e, 0f), (64, 04), (6b, 43), (6f, f1), (2c, 44)
12 - 17	(96, a2), (ee, b9), (4c, f2), (f5, 85), (7, 23), (82, d9)	(dd, 73), (99, e5), (ea, 0b), (0f, 47), (04, b2), (43, b5)

5 Security Analysis

In this section, we analyze the security of the sLiSCP permutation. Since, the security of our design mainly relies on the indistinguishability of the underlying permutation, we assess its behavior against various

distinguishing attacks. As the design of the sLiSCP permutation is based on the Simeck^u-m box and the Type-2 GFS, the permutation is analyzed against distinguishing attacks targeting Type-2 GFS designs when the employed round function is a large Sbox with specific differential and linear properties.

5.1 Differential and Linear Cryptanalysis

Differential [23] and linear [56] cryptanalysis are very powerful cryptanalytic techniques against symmetric key primitives. We assess the security of the sLiSCP permutation (denoted by F) against these two attacks by evaluating lower bounds on the probabilities of the differential and linear characteristics[2].

Bounding trails using the Wide Trail Strategy (WTS). To evaluate the lower bounds on differential and linear characteristics of sLiSCP, we first follow the WTS and compute the minimum number of active Simeck^u-m boxes (h_t^u). In such a context, a Simeck^u-m box is referred to as active if a non zero difference or a linear mask is presented at its input. Table 6 presents a lower bound on the minimum number of active h_t^u for up to 18 steps. The bounds presented in Table 6 are generated by running a Mixed Integer Linear Programming (MILP) optimizer on a model describing our design.

Table 6: Lower bounds on the number of active h_t^u with respect to the number of steps.

Step	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Min. # of active h_t^u	0	1	2	3	4	6	6	7	8	9	10	12	12	13	14	15	16	18

One can see that the number of active h_t^u cycles every six steps. Given the most tight bounds on the differential and linear correlation probabilities which we derived in Table 2, we can now evaluate $MEDCP(f^s)$ and $MELCC(f^s)$. When the number of steps $s = 18$ in both sLiSCP-192 and sLiSCP-256,

$$\begin{aligned} &\text{sLiSCP-192: } MEDCP(f^{18}) = (MEDP(\text{Simeck}^6\text{-}48))^{18} = (2^{-11.299})^{18} = 2^{-203.382} \\ &\text{sLiSCP-256: } MEDCP(f^{18}) = (MEDP(\text{Simeck}^8\text{-}64))^{18} = (2^{-16.597})^{18} = 2^{-298.746} \end{aligned} \tag{4}$$

The duality between linear and differential cryptanalysis enables us to similarly apply the same approach to compute a bound on the MELCC. Over 18 steps, the maximum expected linear characteristic (squared) correlation is upper bounded by $2^{-203.382}$ and $2^{-295.2}$ for sLiSCP-192 and sLiSCP-256, respectively.

Bounding trails using the Long Trail Strategy (LTS). While the above bounds are for a single optimistically found characteristic, we can argue that such a characteristic might not be valid as it is pointed out in [39, 49], having all the 18 active Simeck^u-m boxes exhibiting the maximum differential probability transitions is not always valid. Additionally, Dinu *et al.* [39] presented the tong trail strategy which proposes an interesting security argument with respect to designs that utilize rather weak but large Sboxes (i.e., iterating a weak nonlinear function for some rounds), followed by a slow diffusion linear layer. LTS states that better bounds on differential and linear characteristics probabilities may be given if the design allows the propagation of long uninterrupted trails instead of the wide ones used in WTS. This definition suits our design which uses two Simeck^u-m boxes where the Simeck round function is iterated for u rounds and only two subblock XORs and subblock cyclic shift as the linear permutation. More precisely, a long trail in our case is defined as follows:

Definition 2 *(Long Trail for n Simeck^u-m boxes) A differential path that includes a single path passing though n successive Simeck^u-m boxes with no other paths branching in. Such a trail can be static where an output subblock of the linear layer is equal to one of its inputs, or a dynamic trail where an output subblock of the linear layer is the result of XORing one of its input non active subblocks with the output of an active Simeck^u-m box.*

[2] We use characteristic and trail interchangeably throughout this section.

Based on the above definition, the $MEDCP$ of a given differential trail is bounded by the product of the $MEDCPs$ of its component long trails, both dynamic or static ones. Given a differential trail that is decomposed into i trails, where $\{\kappa_i\}_{i\geq 1}$ denotes the set of their corresponding lengths, then, the probability (pd) of such trail is upper bounded by

$$pd \leq \prod_{i\geq 1} MEDCP(\text{Simeck}^{u\kappa_i}\text{-}m).$$

When using the LTS in bounding the differential and linear probabilities of characteristics, tighter bounds are derived. More precisely, if $ds(r)$ and $ls(r)$ denote the MEDCP and MELCC of Simeck^u-m box, then it can be observed from Table 1:

$$ds(nr) \ll ds(r)^n, \text{ and } ls(nr) \ll ls(r)^n.$$

In our design, only dynamic trails would be beneficial because static trails can only include one active Simeck^u-m box and thus have a maximum length equal to one. Such an observation is attributed to the fact that all applications of round functions in Type-2 GFS designs are directly followed by XORs. Accordingly, if a given differential trail is only decomposed into static trails then the $MEDCP$ of such trail using the LTS is equivalent to that of the WTS as we only independently count the minimum number of active Simeck^u-m boxes.

Decomposition of an optimum trail. We apply the trail decomposition process on a 6-step trail with only six active Simeck^u-m box which is the trail returned by the MILP solver corresponding to the minimum number of active Sboxes. As depicted in Figure 6, the whole differential trail covers all the colored Sboxes. In such a trail, the XORs that receive two active input subblocks (indicated by the dashed colored lines) are assumed to cancel them resulting in zero-difference subblocks which are marked by black colored lines. Using the LTS, we can decompose this trail into five long trails, four out of them are static with length one (green, blue, purple, and orange colored trails) and the remaining one is dynamic (red trail) with length two. Accordingly, the $MEDCP$ of this differential trail is evaluated by $MEDCP(\text{Simeck}^u\text{-}m)^4 \times MEDCP(\text{Simeck}^{2u}\text{-}m)$. Given the exact number of rounds that we used in the probability calculation using the minimum number of active Sboxs method (see Equation 4), for sLiSCP-192: $MEDCP = (2^{-12})^4 \times 2^{-30} = 2^{-78}$ which is considerably lower than the previously anticipated 6-step bound calculated using WTS in Equation 4, which is $MEDCP \approx (2^{-11.299})^6 \approx 2^{-67.794}$. Similarly, for sLiSCP-256: $MEDCP = (2^{-18})^4 \times 2^{-44} = 2^{-116} \ll MEDCP \approx (2^{-16.597})^6 = 2^{-99.582}$.

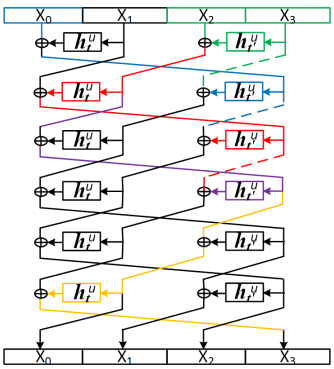

Fig. 6: A minimum number of active Simeck^u-m boxes trail decomposed into four long trails.

To this end, we claim that our analysis based on the minimum number of active Sboxes underestimates the security of sLiSCP with respect to differential and linear cryptanalysis, and that although the above analysis provide a provable bound against these types of attacks, they are not tight and as shown, tighter bounds can be provided using the LTS.

5.2 Meet/Miss in the Middle Distinguishers

A meet-in-the-middle attack [38] divides the permutation into two or three parts such that some intermediate state variables can be computed by evaluating the permutation and its inverse using only incomplete sets of the input and output bits, respectively. Based on our design criteria, we require that the total number of steps is equal to three times the number of steps required for full bit diffusion. Consequently, splitting sLiSCP into two or even three parts always results in an intermediate state that achieves full bit-diffusion. More precisely, the evaluation of each bit in an intermediate state i requires the knowledge of the whole b bits of state $i-6$. For that, the evaluation of an intermediate partial state has no advantage over brute force guessing, thus we believe that sLiSCP is secure against these types of attacks. On the other hand, miss-in-the-middle distinguishers are a special kind of meet-in-the-middle distinguisher that exploit contradicting conditions at an intermediate state which makes matching such an intermediate state attributes impossible [22].

Impossible differential (ID) distinguisher. These distinguishers enable a miss-in-the-middle scenario where two contradicting differential paths are combined such that one of them holds with probability one in the forward direction and the other also holds with certainty in the backward direction [22]. Accordingly, all input pairs and corresponding output pairs satisfying the input differential pattern of the forward trail and the corresponding input difference of the backward one can be used to distinguish the permutation from a random one, where such an impossible differential pair is expected to be possible.

Zero-correlation (ZC) distinguishers. A miss-in-the-middle distinguisher that exploits a situation where there exist a zero-correlation linear approximations over a part of the primitive. Such linear approximations hold with probability p that is exactly equal to $1/2$. More precisely, the distinguishing property in zero-correlations rely on rigidly unbiased approximations with correlation $c = 2p - 1 = 0$ [30].

Type-2 GFS constructions have been extensively analyzed with respect to such miss in the middle distinguishers. Specifically, the work of Blondeau *et al.* [26] has given a proof for the equivalence between ID and ZC distinguishers. In their analysis, they define three types for both distinguishers: type 1, type 2, and type 3 where the input and output differences (respectively linear masks) are independent, different, and equal, respectively. In Figure 7, we present a type 2 nine-step ID distinguisher and its dual ZC on sLiSCP-b.

It can be seen that although both input and output differences $((\Delta,0,0,0),(0,0,0,\Delta))$ (resp. linear masks $((0,0,0,a),(0,0,a,0))$) are not equal, they are not independent too which is a bit restrictive when the permutation is operating in a keyed setting.

ID and ZC distinguishers for Simeck^u-m. Considering that the Simeck^u-m box is a u-round iteration of the Simeck round function, the work of Kondo *et al.* investigates different shift parameters with respect to their impossible differential behavior [52]. For Simeck-48, we have used the search algorithm provided in [52] to search for impossible differential distinguishers and we are able to find a 13-round impossible differential distinguisher as depicted in Table 7 where a "?" denotes an unknown difference, and thus cannot be used to indicate an impossible differential. For Simeck-64, a 15-round impossible differential distinguisher is shown in Table 8.
Zero-correlation attacks on the Simon family of block ciphers have been investigated by Wang *et al.* [65]. Given the analogy between impossible differential and zero correlation distinguishers, we have modified the algorithm in [52] to search for zero-correlation distinguishers and we found a 13-round distinguisher for Simeck-48 as depicted in Table 9, and a 15-round distinguisher for Simeck-64 is shown in Table 10.

Given the above analysis, it is then reasonable to assume that exploiting the structure of our Type-2 GFS design is more powerful with respect to ID and ZC propagations. However, if one considers combining both the overall GFS structure of sLiSCP and that of Simeck similar to the analysis presented by Abdelkhalek *et al.* on SPARX [1], then at most ten steps may be analyzed. Accordingly, we believe that having eighteen steps of sLiSCP-b is sufficient to mitigate distinguishers based on meet and miss in the middle approaches. It also should be noted that although both ID and ZC distinguisher can be verified trivially, however, using them in a distingushing attack for the permutation requires the full codebook.

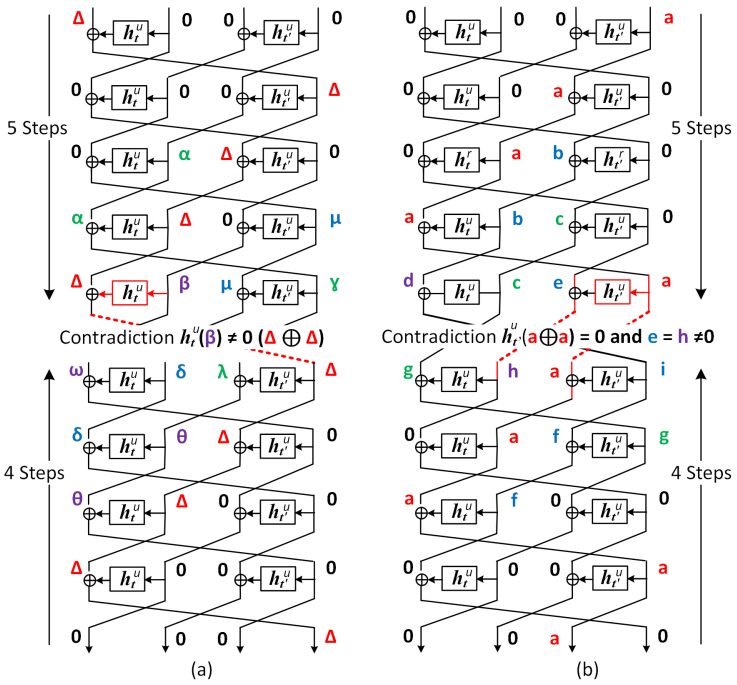

Fig. 7: (a) Impossible differential distinguisher for nine steps of sLiSCP-b, (b) Nine-step zero-correlation distinguisher for sLiSCP-b. Unlike differences and linear approximations are individually colored/marked.

5.3 Integral Distinguishers

In an integral attack [50], also known as square and saturation attacks [35, 55], one has to first look for an integral distinguisher which starts with a set of chosen inputs that have constant equal values in some parts whereas the other parts take all possible values. Moreover, an integral distinguisher ends with the output set which sum to zero at some particular locations. While most of the results of integral attacks are on SPN structures because of their bijective byte/word oriented nature, the behavior of GFSs have been also studied with respect to integral cryptanalysis. The results in [27] implies the existence of 8-round integral distinguisher for sLiSCP. However, traditional integral attacks are less effective on bit-oriented ciphers which lead to the development of the new generalized division property [64].

Division property. The division property proposed by Todo [64] is a generalization of the integral attacks. The main idea is to find the parity of $\bigoplus_{x \in X} \pi_u(x)$ for a multiset $X \subset (\mathbb{F}_2^m)^n$, where $\pi_u : (\mathbb{F}_2^m)^n \rightarrow \mathbb{F}_2$ defined by $\pi_u(x) = \prod_{i=0}^{n-1} \prod_{j=0}^{m-1} x_i[j]^{u_i[j]}$ and $u, x \in (\mathbb{F}_2^m)^n$. In other words, if X satisfies the division property $D_k^{m,n}$, then $\bigoplus_{x \in X} \pi_u(x) = 0$ for all $u \in (\mathbb{F}_2^m)^n$ satisfying $wt(u_i) < k_i$ for $i = 0, 1, \ldots, n-1$, where $k = (k_0, k_1, \ldots, k_{n-1})$ and $k_i \in \{0, 1, \ldots, m\}$. The authors in [74] used this property and showed that $2n + 1$ rounds integral distinguisher exists for Type-2 GFS with n subblocks and bijective round functions. Their results also imply that the length of the distinguisher can be increased further if the degree of the round function is less than $m - 1$, where m is the subblock size. For sLiSCP-192, h_t^6 is bijective with degree 19, so to find how many steps such a distinguisher can cover, we start with a multiset having division property of $D_k^{48,4}$, where $k = \{(48, 46, 48, 48)\}$ and trace its propagation over iterations, see Table 11. We found that the terminal condition {(0,0,0,1), (1,0,0,0), (0,1,0,0), (0,0,1,0)} is reached at step 10. Thus, there exists a 9-step distinguisher for sLiSCP-192 with data complexity 2^{190}. Additionally, using a MILP-based division property propagation [70], a 9-round division property distinguisher with data complexity of 2^{188} is generated. For sLiSCP-256, the algebraic degree of h_t^8 is 36, hence, starting with the division property of $D_{(64,63,64,64)}^{64,4}$, we obtain a 9-step distinguisher with data complexity of 2^{255}. Both distinguishers can be extended by one round using the method in [65]. As for Simeck-48 and Simeck-64, the MILP model in [70] leads to 17-round and 20-round integral distinguisher with data complexity of

Table 7: Impossible differential distinguisher for thirteen rounds of Simeck-48 where red bold faced bits correspond to contradicting bit-difference propagations.

Round	Δx_0	Δx_1
0:	0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000
1:	1000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000
2:	?000 0000 0000 0000 000? 1000	1000 0000 0000 0000 0000 0000
3:	?000 0000 0000 00?? 100? ?000	?000 0000 0000 0000 000? 1000
4:	?000 0000 0??? 10?? ?00? ?000	?000 0000 0000 00?? 100? ?000
5:	?000 ???? 1??? ?0?? ?00? ?000	?000 0000 0??? 10?? ?00? ?000
6:	???? ???? ???? ?0?? ?00? ?00?	?000 ???? 1??? ?0?? ?00? ?000
7:	???? ???? ???? ?0?? ?0?? ????	???? ???? ???? ?0?? ?00? ?00?
8:	???? ???? ???? ???? ???? ????	???? ???? ???? ?**0**?? ?0?? ????
5:	??00 0?00 0000 00?? ?10? ??00	??00 0?00 0??? ?**1**?? ??0? ??00
4:	??00 0?00 0000 0000 000? ?100	??00 0?00 0000 00?? ?10? ??00
3:	?100 0?00 0000 0000 0000 0000	??00 0?00 0000 0000 000? ?100
2:	0000 0100 0000 0000 0000 0000	?100 0?00 0000 0000 0000 0000
1:	0000 0000 0000 0000 0000 0000	0000 0100 0000 0000 0000 0000
0:	0000 0100 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000

2^{47} and 2^{63}, respectively, where both distinguishers have 5 balanced bits in the output sum as depicted in Table 12. Both distinguishers can be further extended by at most one round using the method presented in [65].

5.4 Zero-sum Distinguishers

A zero-sum distinguisher [10] for a vector boolean function $g : \mathbb{F}_2^n \to \mathbb{F}_2^n$ finds a set of inputs $X = \{x_1, x_2, \ldots, x_k\}$ such that $\bigoplus_{i=1}^{k} x_i = \bigoplus_{i=1}^{k} g(x_i) = 0$. Let d_{r_1} and d_{r_2} ($d_{r_1} < n$ and $d_{r_2} < n$) be the degree of g and g^{-1} iterated in r_1 forward and r_2 backward rounds, respectively. Then, for $d = \max(d_{r_1}, d_{r_2})$ one can choose 2^{d+1} intermediate states with $(n - d - 1)$ bits fixed to constant and obtain 2^{d+1} states in r_1 forward and r_2 backward rounds. Since, the degree of g is less than or equal to d in both the directions, the sum of these state values is zero [10, 33]. Hence, for a given d, there exist 2^{n-d-1} sets of input size 2^{d+1} that satisfies the zero-sum property.

To apply this technique to sLiSCP, we first need to find the bounds for the algebraic degree of f^s. We found that the algebraic degree of h_t^6 is 19 and the algebraic degree of h_t^8 is 36. Using these values, we calculate the bounds on the algebraic degree of f^s with $s = 1, .., 5$ for sLiSCP-192 and sLiSCP-256 and present the bounds in Table 13. We present the complexity of the zero-sum distinguisher below.

Basic zero-sum distinguisher for $2s$ steps. To create a zero-sum distinguisher for $2s$ steps of sLiSCP, we target the state after the s-step as the intermediate state. Since, the bounds of f^s and its inverse are same, we can only move to s steps in either direction. Thus, there exist $2, 4, 6$ and 8-step distinguishers with data complexities $2^{20}, 2^{58}, 2^{111}$ and 2^{158} for sLiSCP-192. The same number of steps can also be covered for sLiSCP-256, but with data complexities $2^{37}, 2^{93}, 2^{176}$ and 2^{241}.

Table 8: Impossible differential distinguisher for fifteen rounds of Simeck-64 where red bold faced bits correspond to contradicting bit-difference propagations

Round	Δx_0	Δx_1
0:	0000 0000 0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000 0000 0000
1:	1000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
2:	?000 0000 0000 0000 0000 0000 000? 0001	1000 0000 0000 0000 0000 0000 0000 0000
3:	?000 0000 0000 0000 0000 00?0 00?? 001?	?000 0000 0000 0000 0000 0000 000? 0001
4:	?000 0000 0000 0000 0?00 0??0 0??? 01??	?000 0000 0000 0000 0000 00?0 00?? 001?
5:	?000 0000 0000 ?000 ??00 ???0 ???? 1???	?000 0000 0000 0000 0?00 0??0 0??? 01??
6:	?000 000? 000? ?00? ??0? ???? ???? ????	?000 0000 0000 ?000 ??00 ???0 ???? 1???
7:	?0?0 00?? 00?? ?0?? ???? ???? ???? ????	?000 000? 000? ?00? ??0? ???? ???? ????
8:	???0 0??? 0??? ???? ???? ???? ???? ????	?0?0 00?? 00?? ?0?? ???? ???? ???? ????
9:	???0 ???? ???? ???? ???? ???? ???? ????	???0 0??? 0??? ???? ???? ???? ???? ????
10:	???? ???? ???? ???? ???? ???? ???? ????	???**0** ???? ???? ???? ???? ???? ???? ????
5:	???0 1??? 0000 0000 0000 0000 ?000 ??00	???**1** ???? 0000 0000 000? 000? ?00? ??0?
4:	0??0 01?? 0000 0000 0000 0000 0000 0?00	???0 1??? 0000 0000 0000 0000 ?000 ??00
3:	00?0 001? 0000 0000 0000 0000 0000 0000	0??0 01?? 0000 0000 0000 0000 0000 0?00
2:	0000 0001 0000 0000 0000 0000 0000 0000	00?0 001? 0000 0000 0000 0000 0000 0000
1:	0000 0000 0000 0000 0000 0000 0000 0000	0000 0001 0000 0000 0000 0000 0000 0000
0:	0000 0001 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000

Adding an extra step. As the design of sLiSCP is based on Type-2 GFS, by carefully choosing the position of $d+1$ bits, one free step can be added. For example, if we choose 20 bits in the first block, then in forward direction these bits are processed by h_t^6 at the second step. Hence, the degree reaches 19 after step 2. But in backward direction, the degree has already reached 19 after step 1 as h_t^6 is applied to the same bits at step 1 only. Thus, we can construct $(2+1)$, $(4+1)$, $(6+1)$ and $(8+1)$-step distinguishers with data complexities $2^{20}, 2^{58}, 2^{111}$ and 2^{158} for sLiSCP-192. Similarly, the number of steps for sLiSCP-256 can be increased by one with the same data complexities like the basic zero-sum distinguisher.

The complexity of such distinguishers can be improved further by finding better bounds for the algebraic degree of f^s. Boura et al. [33] proposed that, by calculating the Walsh spectrum of an Sbox, a better bound can be achieved. However, such a technique cannot be applied to sLiSCP due to the large size of Simeck^u-m Sbox. Moreover, one can exploit the division property of f^s to construct a 17-step (9 forward + 8 backward) distinguisher with data complexity 2^{190} (resp. 2^{255}) for sLiSCP-192 (resp. sLiSCP-256).

5.5 Self Symmetry-based Distinguishers

A cryptographic permutation can be obtained by setting a keyed construction subround keys to publicly-known constant values. However, an attempt to set these subround keys to equal values or zero all of them makes the permutation vulnerable to a wide range of distinguishers based on symmetry between rounds or steps.

Slide distinguishers. To destroy symmetry between the steps in the GFS layer, our design employs two different 6-bit constants (SC_{2j}, SC_{2j+1}) at the j-th step so that all the round functions determined by

Table 9: Zero-correlation distinguisher for thirteen rounds of Simeck-48 where red bold faced bits correspond to contradicting bit-masks.

Round	Γx_0	Γx_1
0:	1000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000
1:	0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000
2:	1000 0000 0000 0000 0000 0000	?100 0?00 0000 0000 0000 0000
3:	?100 0?00 0000 0000 0000 0000	??10 0??0 00?0 0000 0000 0000
4:	??10 0??0 00?0 0000 0000 0000	???1 0??? 00?? 000? 0000 0000
5:	???1 0??? 00?? 000? 0000 0000	???? 1??? ?0?? ?00? ?000 ?000
6:	???? **1**??? ?0?? ?00? ?000 ?000	???? ???? ???? ??0? ??00 ??00
7:	0??? **0**??? ???? ???? ???? ????	0??0 0??? ???? ???? ???0 ???0
6:	0??0 0??? ???? ???? ???0 ???0	0?00 0??? ?1?? ??0? ??00 ??00
5:	0?00 0??? ?1?? ??0? ??00 ??00	0000 0??? 10?? ?00? ?000 ?000
4:	0000 0??? 10?? ?00? ?000 ?000	0000 0??1 00?? 000? 0000 0000
3:	0000 0??1 00?? 000? 0000 0000	0000 0?10 00?0 0000 0000 0000
2:	0000 0?10 00?0 0000 0000 0000	0000 0100 0000 0000 0000 0000
1:	0000 0100 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000
0:	0000 0000 0000 0000 0000 0000	0000 0100 0000 0000 0000 0000

(SC_{2j}, SC_{2j+1}) are different. Accordingly, slide distinguishers [25] where an attacker can distinguish the permutation F by evaluating two inputs X and Y, where $Y = f^j(X)$, and checking if $F(Y) = f^j(F(X))$ are mitigated.

Invariant subspace distinguishers. We require that the step constants are different for both $(h_t^u, h_{t'}^u)$, because we need to mitigate invariant subspace distinguishers based on the symmetry between the even-odd shuffle of some GFS designs as has been shown in [42]. Specifically, an invariant subspace attack as introduced by Leander *et al.* implies that there exists a coset of the input vector space which after being iterated for any number of rounds of the cryptographic round function maps to cosets of the same subspace [53]. It has been shown by the attack of Rønjom [60] on Simpira v1 which employs the Yanagihara and Iwata Type-1.x (4,2) GFS [71] that because the employed round constants in each subblock function only differ in one byte, three out of the four subblocks remain the same. In our case, even though one constant in each step is enough to avoid slide attacks, if the same step constant SC is XORed with the output of the two round functions h_t^u and $h_{t'}^u$, (assuming $h_t^u = h_{t'}^u$), an invariant subspace distinguisher can be produced for sLiSCP as depicted in Fig. 8. Accordingly, we require that all functions in one step have different step constants.

Rotational distinguishers. Our design uses Simeck^u-m as a round function where the nonlinear operation in Simeck is the bitwise AND operation. Without the round constant addition between interleaving internal rounds, the rotational properties are preserved with certainty for any number of rounds. If two inputs x and y are processed by the Simeck^u-m box, where $y = (x_0 \lll l) | (x_1 \lll l)$, where $x = x_0 \| x_1$, then $h^u(y) = (h^u(x) \lll l)$ where h^u is the u-fold operation of the Simeck round function without round constant. Accordingly, we require the addition of round constants between interleaving rounds of the Simeck^u-m box. Although only one new bit is added to each internal round which is generated by our efficient 6-bit m-sequence LFSR, we in fact add a $\frac{m}{2}$-bit constant which contains at least $(\frac{m}{2} - 1)$ '1'bits.

Table 10: Zero-correlation distinguisher for fifteen rounds of Simeck-64 where red bold faced bits correspond to contradicting bit-masks.

Round	Γx_0	Γx_1
0:	1000 0000 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
1:	0000 0000 0000 0000 0000 0000 0000 0000	1000 0000 0000 0000 0000 0000 0000 0000
2:	1000 0000 0000 0000 0000 0000 0000 0000	?100 0?00 0000 0000 0000 0000 0000 0000
3:	?100 0?00 0000 0000 0000 0000 0000 0000	??10 0??0 00?0 0000 0000 0000 0000 0000
4:	??10 0??0 00?0 0000 0000 0000 0000 0000	???1 0??? 00?? 000? 0000 0000 0000 0000
5:	???1 0??? 00?? 000? 0000 0000 0000 0000	???? 1??? ?0?? ?00? ?000 ?000 0000 0000
6:	???? **1**??? ?0?? ?00? ?000 ?000 0000 0000	???? ???? ???? ??0? ??00 ??00 0?00 0000
9:	???? **0**??? ???? ???? ???? ???? ???? ????	???0 0??? ???? ???? ???? ???? ???? ???0
8:	???0 0??? ???? ???? ???? ???? ???? ???0	??00 0?0? ???? ???? ???? ???? ??0? ??00
7:	??00 0?0? ???? ???? ???? ???? ??0? ??00	?000 000? ???? ???? ???? ?0?? ?00? ?000
6:	?000 000? ???? ???? ???? ?0?? ?00? ?000	0000 000? ???1 ???? 0??? 00?? 000? 0000
5:	0000 000? ???1 ???? 0??? 00?? 000? 0000	0000 000? ??10 ???0 0??0 00?0 0000 0000
4:	0000 000? ??10 ???0 0??0 00?0 0000 0000	0000 000? ?100 ??00 0?00 0000 0000 0000
3:	0000 000? ?100 ??00 0?00 0000 0000 0000	0000 000? 1000 ?000 0000 0000 0000 0000
2:	0000 000? 1000 ?000 0000 0000 0000 0000	0000 0001 0000 0000 0000 0000 0000 0000
1:	0000 0001 0000 0000 0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
0:	0000 0000 0000 0000 0000 0000 0000 0000	0000 0001 0000 0000 0000 0000 0000 0000

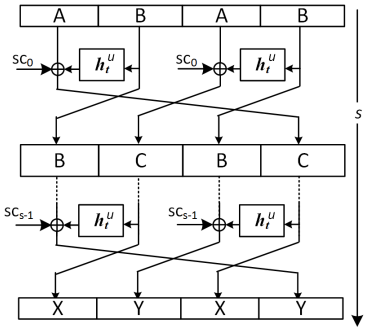

Fig. 8: Type-2 GFS: Assuming that the two step constants SC are identical, and if the odd-numbered input subblocks are equal, and the even-numbered input subblocks are equal, then the invariant subspace property is preserved for any number of rounds.

Having all these '1's result in a lot of inversions which breaks the propagation of the rotational property in one step.

To sum up, we add a 12-bit constant including step and round constants in each h_t^u and our choice of LFSR ensures that each pair of such constants does not repeat due to the periodicity of the 6-tuple sequence constructed from the decimated m-sequence of period 63, which enables our design to avoid self symmetry-based distinguishers.

Table 11: 9-step division property based distinguisher for sLiSCP-192.

Round	Division property
0	{(48, 46, 48, 48)}
1	{(46, 48, 48, 48)}
2	{(10, 48, 48, 48), (29, 48, 48, 47), (48, 48, 48, 46)}
3	{(1, 48, 48, 13), (10, 48, 48, 12), (29, 48, 48, 11), (48, 48, 48, 10), (1, 48, 47, 32), (10, 48, 47, 31), (29, 48, 47, 30), (48, 48, 47, 29), (48, 48, 46, 48)}
4	{(1, 48, 13, 4), (10, 48, 13, 3), (29, 48, 13, 2), (48, 48, 13, 1), (1, 48, 12, 13), (10, 48, 12, 12), (29, 48, 12, 11), (48, 48, 12, 10), (1, 48, 11, 32), (10, 48, 11, 31), (29, 48, 11, 30), (48, 48, 11, 29), (48, 48, 10, 48), (1, 47, 32, 4), (10, 47, 32, 3), (29, 47, 32, 2), (48, 47, 32, 1), (1, 47, 31, 13), (10, 47, 31, 12), (29, 47, 31, 11), (48, 47, 31, 10), (1, 47, 30, 32), (10, 47, 30, 31), (29, 47, 30, 30), (48, 47, 30, 29), (48, 47, 29, 48), (48, 46, 48, 48)}
5	{(1, 13, 0, 4), (1, 12, 13, 4), (10, 13, 0, 3), (10, 12, 13, 3), (29, 13, 0, 2), (29, 12, 13, 2), (48, 13, 0, 1), (48, 12, 13, 1), (1, 12, 12, 13), (10, 12, 12, 12), (29, 12, 12, 11), (48, 12, 12, 10), (1, 12, 11, 32), (10, 12, 11, 31), (29, 12, 11, 30), (48, 12, 11, 29), (48, 12, 10, 48), (1, 11, 32, 4), (10, 11, 32, 3), (29, 11, 32, 2), (48, 11, 32, 1), (1, 11, 31, 13), (10, 11, 31, 12), (29, 11, 31, 11), (48, 11, 31, 10), (1, 11, 30, 32), (10, 11, 30, 31), (29, 11, 30, 30), (48, 11, 30, 29), (48, 11, 29, 48), (48, 10, 48, 48), (0, 32, 0, 4), (0, 31, 13, 4), (9, 32, 0, 3), (9, 31, 13, 3), (28, 32, 0, 2), (28, 31, 13, 2), (47, 32, 0, 1), (47, 31, 13, 1), (0, 31, 12, 13), (0, 31, 11, 32), (0, 30, 32, 4), (9, 30, 32, 3), (28, 30, 32, 2), (47, 30, 32, 1), (0, 30, 31, 13), (0, 30, 30, 32)}
6	{(0, 1, 0, 2), (0, 0, 4, 2), (13, 1, 0, 1), (13, 0, 4, 1), (0, 0, 3, 11), (13, 0, 3, 10), (0, 0, 2, 30), (13, 0, 2, 29), (13, 0, 1, 48), (12, 13, 0, 1), (12, 12, 13, 1), (12, 11, 32, 1), (11, 32, 0, 1), (11, 31, 13, 1), (11, 30, 32, 1), (32, 1, 0, 0), (32, 0, 4, 0), (32, 0, 3, 9), (32, 0, 2, 28), (32, 0, 1, 47), (31, 13, 0, 0), (31, 12, 13, 0), (31, 11, 32, 0), (30, 32, 0, 0), (30, 31, 13, 0), (30, 30, 32, 0)}
7	{(0, 1, 0, 1), (0, 0, 2, 1), (1, 1, 0, 0), (1, 0, 2, 0), (0, 4, 0, 0), (0, 3, 11, 0), (0, 2, 30, 0), (0, 0, 1, 13), (13, 0, 1, 12), (32, 0, 1, 11), (0, 0, 0, 32), (13, 0, 0, 31), (32, 0, 0, 30)}
8	{(1, 1, 0, 0), (1, 0, 1, 0), (0, 0, 0, 1), (4, 0, 0, 0), (0, 2, 0, 0), (0, 1, 13, 0), (0, 0, 32, 0)}
9	{(0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), (2, 0, 0, 0)}
10	{(0,0,0,1), (1,0,0,0), (0,1,0,0), (0,0,1,0)}

6 Applications of sLiSCP

The sLiSCP permutation is designed to be used in lightweight applications to provide as many cryptographic functionalities as possible. We use sLiSCP in the sponge framework to construct authenticated encryption, stream cipher, MAC and hash function.

Table 12: Integral distinguishers for Simeck-48 and Simeck-64 where `'C'` denotes a fixed value in the same position in the set of states, `'A'` denotes all possible values appearing equal number of times, `'B'` denotes the values at a given position in the set of states sum to zero, and `'?'` indicates that no property can be determined.

Simeck-48	Input:	[CAAAAAAAAAAAAAAAAAAAAAA,AAAAAAAAAAAAAAAAAAAAAA]
	Output:	[??????????????????????????,B???BB?????????????BB???]
Simeck-64	Input:	[CAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA,AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA]
	Output:	[????????????????????????????????,BB???B?????????????????????B???B]

Table 13: Bounds for the algebraic degree of f^s.

s	Degree of f^s (sLiSCP-192)	Degree of f^s (sLiSCP-256)
1	19	36
2	57	92
3	110	175
4	157	240
5	191	255

6.1 Why the Sponge Framework?

Sponge constructions are very diversified in terms of the offered security level, particularly, it is proven that the sponge and its single pass duplex mode offer a $2^{c/2}$ bound against generic attacks [15, 19] which attains a lower bound on the width of the underlying permutation. However, for keyed modes such as MAC, stream encryption and authenticated encryption a security level of 2^{c-a} is proven when the number of queries is upper bounded by 2^a [20]. When restricting the data complexity to a maximum of 2^a queries with $a < c$, one can reduce the capacity and increase the rate for a better throughput with the same security level. It has also been shown that, in keyed modes, the restrictions on the underlying permutation can be relaxed, especially the required number of rounds as has been shown in MonkeyDuplexing which accepts calls to variants of the permutation with a different number of rounds [20]. Furthermore, in sponge keyed encryption modes, nonce reuse enables the encryption of two different messages with the same key stream which undermines the privacy of the primitive. More precisely, the sponge duplex authenticated encryption mode requires the uniqueness of a nonce when encrypting different messages with the same key because the ability of the attacker to acquire multiple combinations of input and output differences leaks information about the inner state bits which may lead to the reconstruction of the full state [19, 14]. Nonce reuse in duplex constructions reveals the XOR difference between the first two plaintexts by XORing their corresponding ciphertexts. On the other hand, a nonce reuse differential attack may be exploited if the attacker is able to inject a difference in the plaintext and cancel it out by another difference after the permutation application. However, such an attack depends on the probability of the best differential characteristic and the number of rounds of the underlying permutation, accordingly, if such a permutation offers enough resistance to differential cryptanalysis, the feasibility of nonce reuse differential attacks is minimal. The condition on the differential behavior of the underlying permutation is also important when considering resynchronization attacks where related nonces are to be used, and for that reason, even if nonce reuse is not permitted, the underlying permutation used in the initialization stage should be strong enough to mitigate differential attacks.

Given the above results, we find that a combining version of the duplex sponge mode used in Ascon [40, 20] and NORX [8] realizes the objectives we aimed for our sLiSCP permutation. We call this mode the sLiSCP mode. Such objectives are attributed to the properties, which are derived directly from sponge-based primitives, below

- The flexibility to adapt the same circuitry to provide both keyed and unkeyed functionalities (e.g., hashing, MAC computation, and pseudo random bit generation) as we adopt a unified round function for all functionalities.

- High key agility which fits the lightweight requirements because all keyed modes require no key scheduling.
- Simplicity as there is no need to implement decryption algorithm because the same encryption algorithm is used for decryption.
- Both plaintext and ciphertext blocks are generated online without the need to process the whole input message and encrypted material first.

6.2 The sLiSCP Mode

The utilized sponge mode modifies the keyed initialization and keyed finalization stages of the Ascon [40] and NORX [8] modes which make key recovery hard even if the internal state is recovered and also renders universal forgery with the knowledge of the internal state unattainable. The adopted modification makes the initialization and finalization stages more hardware efficient and adaptable to different primitive modes. In particular, instead of initializing the state with the key, K, and then again XORing it with the permutation output that requires an extra $|K|$ XORs, we initialize the state with the key and then again absorb the key in the rate part during the initialization and finalization phases. We also use the domain separation technique as used in NORX because it runs for all rounds of all stages, and thus reduces the chances of side channel analysis and offers uniformity across different stages. The separation between the processing of different types of inputs is important to distinguish between the roles of the processed data. To this end, we only have one round function (See Fig. 9) that incorporates absorption, squeezing, and domain separation, and according to the fed inputs, we decide which stage and functionality to implement. The sLiSCP mode used in the authenticated encryption is further depicted in Fig. 10.

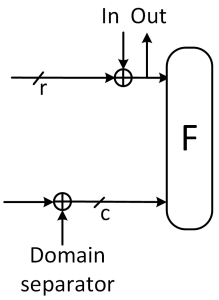

Fig. 9: Unified round function which can be used in all stages of both keyed and unkeyed modes.

Initialization, absorbing, and squeezing. Our sLiSCP permutation is based on Type-2 GFS where, apart of the permutation size, each subblock is either 48 or 64 bits. Since we use it in sponge-based modes, we need to specify exactly from where the r-bit input is absorbed and the r-bit output is squeezed. For sLiSCP permutations, we consider the b-bit state as a series of four m-bit subblocks, X_0, X_1, X_2, X_3 (see Figure 3), where m is equal to 48 and 64 for sLiSCP-192 and sLiSCP-256, respectively. We divide the state S of sLiSCP as bytes, $S = (B_0, B_1, \cdots, B_{l-1})$ where $l = 24$ and 32, for sLiSCP-192 and sLiSCP-256, respectively. Moreover, each subblock X_i can be viewed as a series of $j = \frac{m}{8}$ bytes, $B_{ij+0}, B_{ij+1}, \cdots, B_{ij+(j-1)}$ arranged from left to right. In nonce-based keyed modes, initially, the state is loaded with the nonce and key bytes, denoted by NB_w, $w = 0, 1, \cdots, n$, and KB_z, $z = 0, 1, \cdots, k$, respectively and remaining bytes are set to zero. Also, when the hashing mode is employed, we load the state by a 3-byte IV and the remaining bytes are set to zero. Specifically, the first two IV bytes are assigned to the first two bytes of X_0, and the remaining IV byte is loaded in the first byte of X_2. All initialization public variables either nonce or IV are ceiling divided and loaded in the even indexed subblocks, X_0 and X_2, in an ascending byte order. The key is loaded in the odd indexed subblocks, X_1 and X_3 in the same manner, and if the key size is larger than half the state size, then remaining key bytes populate the remaining bytes in X_0 and X_2 equally and in an ascending order. For example, for sLiSCP-192/80, sLiSCP-192/112, and sLiSCP-256/128, the state is

initialized as follows:

sLiSCP-192/80: $B_0 \leftarrow NB_0, B_1 \leftarrow NB_1, B_2 \leftarrow NB_2, B_3 \leftarrow NB_3, B_4 \leftarrow NB_4, B_{12} \leftarrow NB_5, B_{13} \leftarrow NB_6,$
$B_{14} \leftarrow NB_7, B_{15} \leftarrow NB_8, B_{16} \leftarrow NB_9, B_6 \leftarrow KB_0, B_7 \leftarrow KB_1, B_8 \leftarrow KB_2, B_9 \leftarrow KB_3,$
$B_{10} \leftarrow KB_4, B_{18} \leftarrow KB_5, B_{19} \leftarrow KB_6, B_{20} \leftarrow KB_7, B_{21} \leftarrow KB_8, B_{22} \leftarrow KB_9.$

sLiSCP-192/112: $B_0 \leftarrow NB_0, B_1 \leftarrow NB_1, B_2 \leftarrow NB_2, B_3 \leftarrow NB_3, B_4 \leftarrow NB_4, B_{12} \leftarrow NB_5, B_{13} \leftarrow NB_6,$
$B_{14} \leftarrow NB_7, B_{15} \leftarrow NB_8, B_{16} \leftarrow NB_9, B_6 \leftarrow KB_0, B_7 \leftarrow KB_1, B_8 \leftarrow KB_2, B_9 \leftarrow KB_3,$
$B_{10} \leftarrow KB_4, B_{11} \leftarrow KB_5, B_{18} \leftarrow KB_7, B_{19} \leftarrow KB_8, B_{20} \leftarrow KB_9, B_{21} \leftarrow KB_{10}, B_{22} \leftarrow KB_{11},$
$B_{23} \leftarrow KB_{12}, B_5 \leftarrow KB_6, B_{17} \leftarrow KB_{13}$

sLiSCP-256/128: $B_0 \leftarrow NB_0, B_1 \leftarrow NB_1, B_2 \leftarrow NB_2, B_3 \leftarrow NB_3, B_4 \leftarrow NB_4, B_5 \leftarrow NB_5, B_6 \leftarrow NB_6, B_7 \leftarrow NB_7,$
$B_{16} \leftarrow NB_8, B_{17} \leftarrow NB_9, B_{18} \leftarrow NB_{10}, B_{19} \leftarrow NB_{11}, B_{20} \leftarrow NB_{12}, B_{21} \leftarrow NB_{13}, B_{22} \leftarrow NB_{14},$
$B_{23} \leftarrow NB_{15}, B_8 \leftarrow KB_0, B_9 \leftarrow KB_1, B_{10} \leftarrow KB_2, B_{11} \leftarrow KB_3, B_{12} \leftarrow KB_4, B_{13} \leftarrow KB_5,$
$B_{14} \leftarrow KB_6, B_{15} \leftarrow KB_7, B_{24} \leftarrow KB_8, B_{25} \leftarrow KB_9, B_{26} \leftarrow KB_{10}, B_{27} \leftarrow KB_{11}, B_{28} \leftarrow KB_{12},$
$B_{29} \leftarrow KB_{13}, B_{30} \leftarrow KB_{14}, B_{31} \leftarrow KB_{15}$

In the sLiSCP modes, we use *initialize(x)* to denote the process of loading the state with x in the positions described above. As for absorbing and squeezing, we want the input bits to be processed by the Simeck^u-m box as soon as possible so we achieve better diffusion. Accordingly, choosing the right place for absorbing data determines how fast it is processed by the round function which is important since not all the subblocks in GFS constructions receive the same amount of processing at first. The same $r/8$ bytes are used for absorbing and squeezing and they are denoted by the following state bytes:

sLiSCP-192: B_6, B_7, B_{18}, B_{19}
sLiSCP-256: $B_8, B_9, B_{10}, B_{11},$
$B_{24}, B_{25}, B_{26}, B_{27}.$

In the AE mode, the tag is extracted from the same byte positions which are used in the key initialization stage. Hence, the process of tag extraction from state S is denoted by $tagextract(S)$. We denote the rate and capacity parts of the state S by S_r and S_c, respectively, thus $S = (S_r, S_c)$. In what follows, we show how we use the unified round function depicted in Figure 9 to implement several functionalities using sLiSCP permutation in the sLiSCP mode.

6.3 Authenticated Encryption

An authenticated encryption algorithm takes as input a secret key K, a nonce N, a block header A (a.k.a, associated data) and a message M and outputs a ciphertext C with $|C| = |M|$, and an authentication tag T. Mathematically, an authenticated encryption $\mathcal{AE}$ mode is defined as

$$\mathcal{AE} : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^t$$

with

$$\mathcal{AE}(K, N, A, M) = (C, T)$$

where k is the bit length of K, n is the bit length of N. We denote an instance of sLiSCP in a keyed mode by sLiSCP-b/k, where b and k denote the state size and the key length, respectively. In such a mode, we limit the number of processed bits per key to 2^a, which is known as the data usage limit [18]. Specifically, 2^a denotes the value that an implementation restricts the maximum message size (data queries) that can be processed per a given key such that one can attain bit security equal to 2^k when $c \geq k + a + 1$. Recommended parameters for sLiSCP when used in AE mode are listed in Table 14.

The depiction of the encryption and decryption processes using the sLiSCP sponge mode is shown in Figure 10. We describe the padding rule and the algorithms of the AE below.

Padding: Padding is necessary when the length of the processed data is not a multiple of the rate r value and also to act as a delimiter between data of unknown lengths. Since the keys are of fixed length, we need

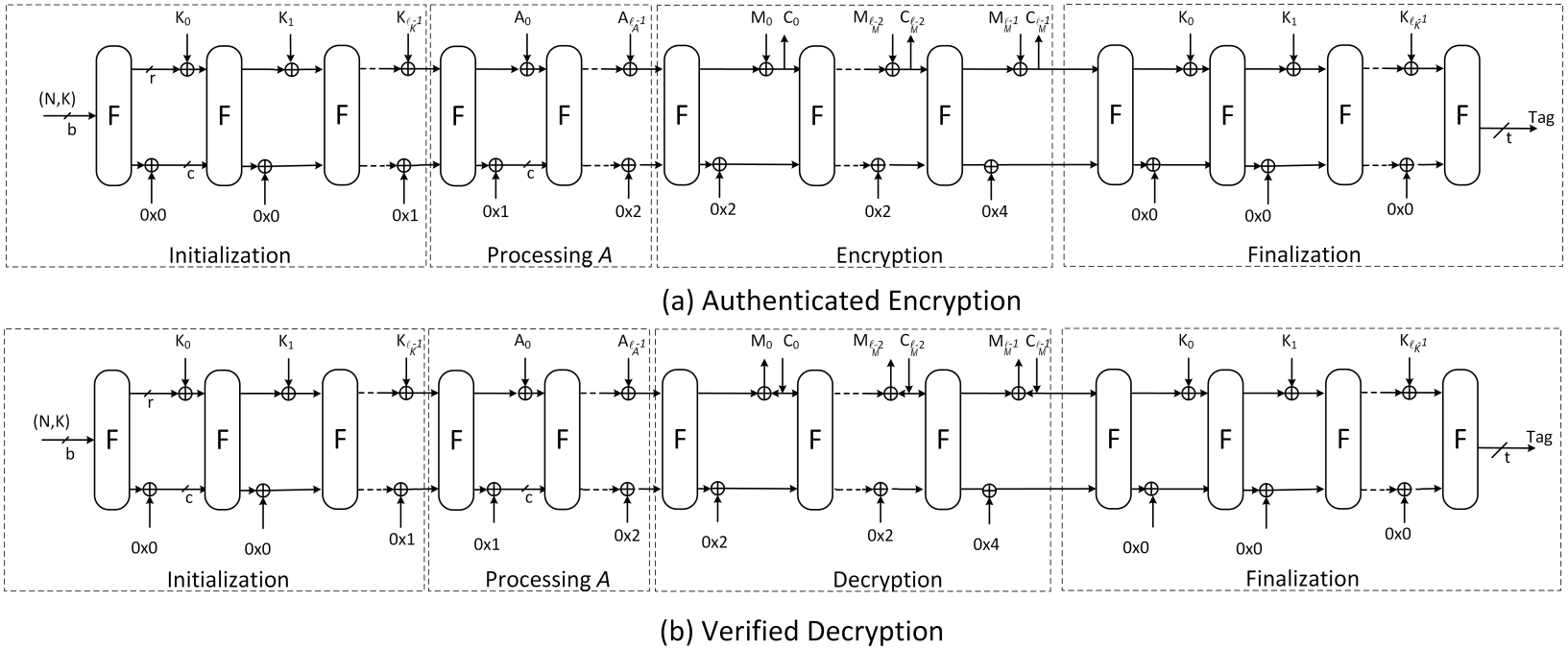

Fig. 10: The sLiSCP sponge mode of operation (authenticated encryption and decryption) using sLiSCP as its underlying permutation.

Table 14: Recommended parameter set for sLiSCP-b/k when used in authenticated encryption mode.

Algorithm	Key	Nonce	Tag	Block size r	Capacity c	Usage exponent a
sLiSCP-192/80	80	80	80	32	160	72
sLiSCP-192/112	112	80	112	32	160	40
sLiSCP-256/128	128	128	128	64	192	56

to pad it by appending zeros only if its length is not a multiple of r bits such that the padded K is divided into ℓ_K r-bit blocks $K_0\|K_1\|\cdots\|K_{\ell_K-1}$. Afterwards, the padding rule (10^*) denoting a single 1 followed by required 0's is applied to the message M such that its length after padding is a multiple of r. Then the resulting padded message is divided into ℓ_M r-bit blocks $M_0\|M_1\|\cdots\|M_{\ell_M-1}$. A similar procedure is carried out on the associated data A which results in ℓ_A r-bit blocks $A_0\|A_1\|\cdots\|A_{\ell_A-1}$. In the case where no associated data is present, no processing is necessary. We summarize the padding rules for key, message and associated data below.

$$
\begin{aligned}
&pad_r(K) \rightarrow K\|0^{r-(|K| \bmod r)}, \text{ if } |K| \bmod r \neq 0\\
&pad_r(M) \rightarrow M\|1\|0^{r-1-(|M| \bmod r)}\\
&pad_r(A) \rightarrow \begin{Bmatrix} A\|1\|0^{r-1-(|A| \bmod r)} & if|A|>0 \\ \phi & if|A|=0 \end{Bmatrix}
\end{aligned}
$$

Initialization: The initial state S is loaded with the nonce N and key K as described in Section 6.2. Each r-bit key block K_i is absorbed by XORing it to the S_r part of the state and a one bit domain separator is XORed to the most significant bit in byte B_{23} and B_{31} for sLiSCP-192 and sLiSCP-256 with the absorption of the last key block K_{ℓ_K-1}, respectively. Afterwards, the sLiSCP permutation is applied to the whole state. The initialization steps are described below.

$$
\begin{aligned}
&S \leftarrow F(initialize(N,K))\\
&S \leftarrow F((S_r \oplus K_i), S_c),\ i=0,\cdots,\ell_K-2\\
&S \leftarrow F((S_r \oplus K_i), (S_c \oplus 0^{c-1}\|1)),\ i=\ell_K-1
\end{aligned}
$$

Processing A: If there is associated data, each r-bit block A_i, $i=0,\cdots,\ell_A-1$ is XORed to the first S_r part of the internal state S and one-bit domain separator is XORed to the last byte of the states. Then, sLiSCP permutation is applied on the whole state.

$$
\begin{aligned}
&S \leftarrow F((S_r \oplus A_i), (S_c \oplus 0^{c-1}\|1)),\ i=0,\cdots,\ell_A-2\\
&S \leftarrow F((S_r \oplus A_i), (S_c \oplus 0^{c-2}\|2)),\ i=\ell_A-1
\end{aligned}
$$

Encryption: Similar to the processing of A, however, with a different domain separator, each message r-bit block M_i, $i=0,\cdots,\ell_M-1$ is XORed to the S_r part of the internal state S resulting in the corresponding ciphertext C_i which is then extracted from the S_r part of the state. After the computation of each C_i, the

whole internal state S is permuted by F.

$$\begin{aligned} C_i &\leftarrow S_r \oplus M_i, \\ S &\leftarrow F(C_i, (S_c \oplus 0^{c-2} \| 2)) \text{ if } 0 \leq i < \ell_M - 2 \\ S &\leftarrow F(C_i, (S_c \oplus 0^{c-3} \| 4)) \text{ if } i = \ell_M - 1 \end{aligned}$$

To keep a minimal overhead, the last ciphertext block $C_{\ell_M - 1}$ is truncated so that its length is equal to that of the last unpadded message block $M_{\ell_M - 1}$ (i.e., $C_{\ell_M - 1} = \lfloor C_{\ell_M - 1} \rfloor_{(|M| \bmod r)}$).

Decryption: Each ciphertext r-bit block C_i, $i = 0, \cdots, \ell_M - 1$ is XORed to the S_r part of the internal state S to calculate the corresponding message block M_i, then the same C_i replaces the r-bit block S_r in the internal state, then the whole internal state S is transformed by the permutation F

$$\begin{aligned} M_i &\leftarrow S_r \oplus C_i \\ S &\leftarrow F(C_i, (S_c \oplus 0^{c-2} \| 2)), \ 0 \leq i < \ell_M - 2 \end{aligned}$$

The last message block $M_{\ell_M - 1}$ is calculated by XORing the ciphertext block $C_{\ell_M - 1}$ to the truncated S_r part of the state, then replacing the S_r part by $C_{\ell_M - 1} \| (\lceil S_r \rceil^{(r - |M| \bmod r)} \oplus (1 \| 0^{(r-1-|M| \bmod r)}))$.

$$\begin{aligned} M_{\ell_M - 1} &\leftarrow \lfloor S_r \rfloor_{(|M| \bmod r)} \oplus C_{\ell_M - 1} \\ S &\leftarrow F(C_{\ell_M - 1} \| (\lceil S_r \rceil^{(r - |M| \bmod r)} \oplus (1 \| 0^{(r-1-|M| \bmod r)})), (S_c \oplus 0^{c-3} \| 4)). \end{aligned}$$

Finalization: Finally, the ℓ_K key blocks are absorbed and the tag is extracted from the chosen bytes of the state as described earlier.

$$\begin{aligned} S &\leftarrow F((S_r \oplus K_i), S_c), \ i = 0, \cdots, \ell_K - 1 \\ T &\leftarrow tagextract(S). \end{aligned}$$

The decryption procedure returns the message blocks M_i ,$i = 0, 2, \cdots \ell_M - 1$, only if the calculated tag value is equal to the received tag value. The AE mode assumes nonce respecting adversary and we do not claim security in the event of nonce reuse, although, the initialization and finalization stages combined by the number of rounds used in the sLiSCP permutation tremendously reduces the effect of such attacks. We claim no security for reduced-round versions of the sLiSCP permutation operating in the sLiSCP mode. In summary, our security claims are given in Table 15

Table 15: Security claims for sLiSCP operating in the sLiSCP AE mode where sLiSCP-b/k denotes sLiSCP with state size b and key size k.

Security property	sLiSCP-192/80	sLiSCP-192/112	sLiSCP-256/128
Data confidentiality	80	112	128
Data integrity	80	112	128
Associated data integrity	80	112	128
Nonce data integrity	80	112	128

An authenticated encryption algorithm can be easily used to provide either encryption or authentication only. More precisely, when using sLiSCP for encryption only, we run the algorithm as usual and stop after

the last message block is encrypted and since we do not care about tag forgery, we can omit/ignore the finalization stage and the tag extraction stage. We also set to zero all the domain separation as we are only processing one domain of messages. For the MAC generation, we can ignore the initialization phase and directly load both the key and nonce in the state and start absorbing the message blocks directly after applying F once to the initialized state. This design decision is attributed to the fact that during the MAC generation, there is no leaked part of the state and the attacker has little control (only probabilistic) over the state which makes state recovery attacks harder than that in the case of authenticated encryption or encryption only. However, since we care about tag forgery, we need to maintain a strong keyed finalization stage, also, in this mode, we may zero all domain separator XORs because we are authenticating data apart of its role. Also, the adopted initialization and finalization stages are not efficient throughput wise when the processed message is short. However, in such a case, the ability of attacker to recover the internal state is reduced too, so when processing short messages we can directly initialize the state with the key and nonce, and skip the initialization phase.

6.4 Hash Computation

A hash function takes as input a message M, and a standard initialization vector IV, and then returns a fixed size output H, called hash or message digest. Formally, the hash mode is specified by

$$\mathcal{H} : \{0,1\}^* \times \{0,1\}^{iv} \rightarrow \{0,1\}^h$$

with $H = \mathcal{H}(M, IV)$ where iv is the length of the IV and h is the length of the hash. The depiction of the hashing process of the sLiSCP mode is shown in Fig. 11.

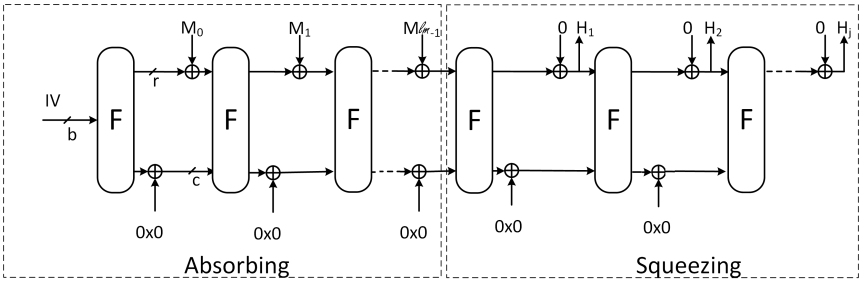

Fig. 11: Hash computation of the sLiSCP mode of operation.

We adapt the sLiSCP mode such that it can be used to initially absorb the message blocks and then squeeze hash blocks to output the desired hash value. This is an unkeyed mode where we do not need an initialization or finalization stage. It has been shown that inverting the squeezing phase falls in the category of "multiblock constrained-input constrained-output problem" which requires $2^{\min(h,b)-r}$ computations to recover the state before the squeezing phase. Once such an internal state is recovered one can launch a meet-in-the-middle attack with around $2^{c/2}$ computations to get a preimage of a given hash of length h [16, 43]. The latter condition reduces the generic preimage attack on the sponge-based hash functions from 2^h to $\min(2^{\min(h,b)}, \max(2^{\min(h,b)-r}, 2^{c/2}))$ where such a preimage security is usually dominated by $2^{\min(h,b)-r}$ and accordingly highly dependent on the squeezed bit rate. In [43], Guo *et al.* suggested using a flexible squeezing bit rate $r' < r$ that offers a trade-off between speeding up the hash computation and preimage security (similar solution has been suggested by Andreeva *et al.* [5]). More precisely, a smaller r' would make the time complexity of a preimage attack equal to $2^{h-r'}$ (assuming that the hash length is less than the state length) which is close to that of the expected generic one 2^h. On the other hand, if small inputs are hashed (e.g., electronic product code (EPC) data, which is a 96-bit string), small squeezing bit rate may make the computation too slow as one needs $\lceil h/r' \rceil - 1$ calls to the underlying permutation. Another solution to reach the expected generic preimage security is to run one more squeezing round after one extracts the desired hash length h [43, 7], thus increasing the acquired output to $h + r'$ and in this case the complexity of the generic preimage attack is equal to :

$$\min(2^{\min(h+r',b)}, \max(2^{\min(h,b-r')}, 2^{c/2})) \geq 2^h \text{ when } c + r - r' \geq h.$$

We adopt a standard initialization vector that combines the parameters of a given sLiSCP instance. In particular, we use the same format used by Guo *et al.* in PHOTON such that for any instance the state is first initialized by $IV = h/2\|r\|r'$, where 8 bits are used to encode each of the used $h/2$, r, and r' sizes. The claimed security levels for the recommended parameters for sLiSCP in the hashing mode are given in Table 16.

Table 16: Recommended parameter set for sLiSCP-b when used in hashing mode and the associated bit securities.

Algorithm	IV	h	r	r'	c	collision	Sec. preimage	Primage
sLiSCP-192	$0x502020$	160	32	32	160	80	80	128
sLiSCP-256	$0x604040$	192	64	64	192	96	96	128
sLiSCP-256	$0x604020$	192	64	32	192	96	96	160

Initialization and Message Padding The state is first initialized with the IV and the padding rule (10^*) is applied to the input message M where a single 1 followed by enough $0s$ is appended to it such that its length after padding is a multiple of r bits. Then the resulted padded message is divided into ℓ_M r-bit blocks $M_0\|M_1\|\cdots\|M_{\ell_M-1}$. Accordingly, the message padding procedure is given by: $pad_r(M) \rightarrow M\|1\|0^{r-1-(|M| \bmod r)}$

Absorbing and Squeezing: Initially each message block is absorbed by XORing it to the S_r part of the state, then sLiSCP permutation is applied afterwards. After absorbing all the message blocks, the h-bit output is extracted from the S_r part of the state r' bits at a time followed by the application of sLiSCP permutation until a total of $\lceil h/r' \rceil$ extractions are completed, then if the resulting extracted bits are more than the desired hash length, truncation is performed. Note that if $r' < r$, then its byte size is extracted from the same subblocks used in squeezing, X_1 and X_3, such that the first and second halves of the r' bytes are extracted from X_1 and X_3, respectively, in an ascending byte order.

$$Absorbing : S \leftarrow F(S_r \oplus M_i, S_c) \text{ for } 0 \leq i \leq \ell_M - 1$$

$$\begin{aligned} Squeezing : H_i &\leftarrow S'_r \\ S &\leftarrow F(S),\ 1 \leq i \leq j,\ \text{for } j = \lceil h/r' \rceil - 1 \\ H &\leftarrow \lfloor H_1\|H_2\|\cdots\|H_j \rfloor_h \end{aligned}$$

Reseedable Pseudo Random Bit Generator (PRBG). The hash construction can be used as a reseedable pseudo random bit generator [17] where initially the state is loaded by an all zero vector, and then, the initial seed is fed through a series of absorbing rounds. After the last absorbed rate part of the seed, the output stream is squeezed in r bits as needed. Also, because we are using a sponge duplex construction a new seed can be fed to the state while squeezing output at the same time, thus allowing the construction of a reseedable PRBG.

7 Hardware Implementation and Results

We implement our sLiSCPpermutation using the parallel hardware architecture as shown in Figure 12. Each of the four m-bit subblocks of the registers are divided into two parts. In order to control the internal rounds and the steps, two counters (i and j) are adopted, where i ($0 \leq i \leq u$) controls the round function of Simeck and j ($0 \leq j \leq s$) controls the steps of the permutation. The output of Simeck round function

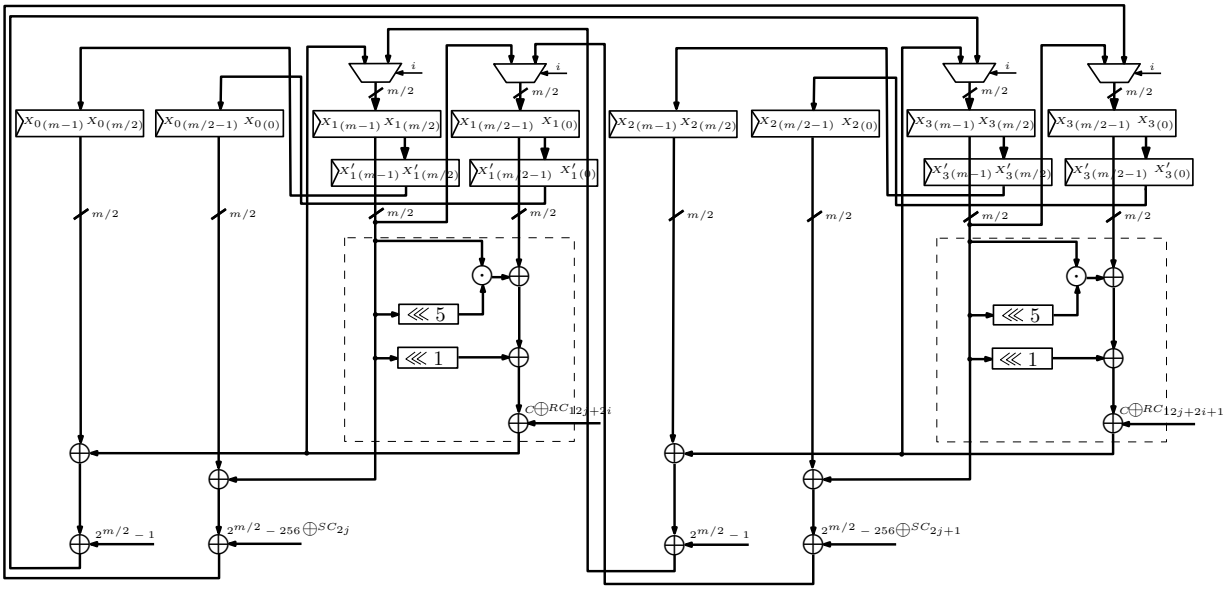

Fig. 12: Hardware architecture of the sLiSCP permutation

(dashed box) on registers X_1 and X_3 is feedback to the left half of the registers during each clock cycle when $0 \leq i < u$, and at the same time the left half of the registers is shifted to the right half. The 1-bit round constant RC_{12j+2i} and $RC_{12j+2i+1}$ are first padded with $m/2-1$ bits '1's and then are added to the Simeck round function in each clock cycle and they are generated using the parallel LFSR as described in Section 4.4. The two extra registers X_1' and X_3' are used to store the initial values of registers X_1 and X_3 when i equals 0. At the last clock cycle, *i.e.*, i equals u, one step of the permutation begins. During this clock cycle, the output of the Simeck round function based on register X_1 is first XORed with the left half of register X_0, and then is XORed with a constant of $m/2$ bits '1's. This new value is sent to the left half of the register X_3. Due to the two different inputs for register X_3, a $m/2$ bits multiplexer is needed. Meanwhile, the left half of the register X_1 is XORed with the right half of the register X_0, and then is XORed with $m/2-8$ bits '1's padded with a step constant SC_{2j} (In particular, for sLiSCP-192, the $m/2-8$ bits '1's is padded with two 0's and then padded with the step constant SC_{2j}). The generated new value is shifted to the right half of the register X_3. A multiplexer in this case is needed as well. The similar case for the output of the Simeck round function based on register X_3 XORed with register X_2, where the new results are sent to the register 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. At the end of this clock cycle, all the registers are updated with a new value, and one step of permutation is finished, the counter j is increased by 1, and the counter i returns to 0. After s steps, one permutation is finished. Table 17 shows the number of discrete components in the sLiSCP permutation F, where XOR is 1-bit xor operation and MUX is 1-bit multiplexer.

The authenticated encryption and hash modes of sLiSCP involve running the permutation multiple times in the sponge structure, where r-bit input is absorbed using r XORs and r'-bit output is squeezed using r' XORs. The input for the authenticated encryption mode are key, nonce, associated data, and message, whereas there is only message for hash mode. In addition, a three bit domain separator is taken in the authenticated encryption mode, hence three more XORs are required.

We use the same ASIC design flow and metrics as described in Simeck [72]. Our implementation results are based on STMicroelectronics CMOS 65 nm CORE65LPLVT library and IBM CMOS 130 nm library and the areas are obtained before the place and route phase in order to compare fairly with other lightweight candidates. To keep the consistency with other sponge based primitives, the throughput is computed at a frequency of 100 kHz using the following formula: $Throughput = \frac{r'}{(u*s)} * 100$ kbps. Our implementation areas, as shown in Table 18, for sLiSCP-192 permutation are 2153 GEs and 2318 GEs in 65 nm and 130 nm ASICs respectively, and that for sLiSCP-256 permutation are 2833 GEs and 3040 GEs respectively.

Table 17: The number of discrete components in sLiSCP permutation

Permutation F	**Components**	**Numbers**	
		sLiSCP-192	sLiSCP-256
Step Function f	Registers	6×48	6×64
	XOR	108	140
	MUX	96	128
Simeck Round Function	AND	24	32
	XOR	49	65
LFSR Constants	Registers	6	7
	XOR	6	9

Table 18: Parallel hardware implementation results of sLiSCP-192/256 permutations

Permutation F	**Process** (nm)	**Areas** (GEs)
sLiSCP-192	65	2153
sLiSCP-192	130	2318
sLiSCP-256	65	2833
sLiSCP-256	130	3040

By applying the sLiSCP permutations repeatedly, we carry two implementations for the hash and AE modes in order to contrast with other dedicated designs. The implementations of these two modes involve the input of key, nonce, associated data, message, and the bit domain separator; and also involve the output of the ciphertext, hash value and tag. Therefore, the resulting areas are increased from the permutations by using extra XORs to deal with the inputs and bit domain separator, and using multiplexers to select different inputs to the permutation. More specifically, for the AE mode of sLiSCP-192, 32 XORs are used for the inputs, 3 XORs are used for the bit domain separator, 35 multiplexers are used for selecting from the initial value or the XORs' results. While for the AE mode of sLiSCP-256, the corresponding numbers of them are 64 XORs, 3 XORs, 67 multiplexers. The 3 XORs for the bit domain separator are deleted for the hash mode. It is worth noting that the number of multiplexers depends on the specific architecture undertaken, and there is a tradeoff between area and throughput. Our results in both CMOS 65nm and CMOS 130nm ASICs for the hash and authenticated encryption modes of sLiSCP are presented in Table 19, as well as a comparison with other lightweight hash functions and AE algorithms. If a unified mode is used for both functionalities, then the consumed GE area will be dominated by that of the AE mode. The hash mode can be achieved from the circuit of the AE mode by inputting 0's to the bit domain separator's XORs, which means the hardware of AE mode can be used for both functionalities.

Our implementation in CMOS $65\,nm$ shows that the area for the hash mode of sLiSCP-192 (resp. sLiSCP-256) is 2271 (resp. 3019) GEs with a throughput of 29.62 (resp. 44.44 kbps or 22.22 kbps depending on r') kbps. Their area results in CMOS $130\,nm$ are 2492 and 3305 GEs respectively. When compared with other primitives with similar internal states, the area of sLiSCP-192 is slightly larger than that of the serialized implementation of Photon-160/36/36 and is comparable with that of Spongent-160/160/16. However, the area of sLiSCP-192 is quite smaller than that of D-Quark, Keccak-f[40,160], Keccak-f[72,128], where the areas of Keccak-f[40,160] and Keccak-f[72,128] are permutations only. In terms of throughput, sLiSCP-192 is better than Photon-160/36/36, D-Quark, and Spongent-160/160/16. The area of sLiSCP-256 is larger than that of the serialized result of Photon-224/32/32 and is comparable with that of Spongent-160/160/80, Spongent-224/224/16, Sponogent-256/256/16, and is smaller than that of S-Quark. The relevant throughput is only smaller than that of Spongent-160/160/80 and S-Quark.

For the authenticated encryption mode, the area in CMOS $65\,nm$ of sLiSCP-192 (resp. sLiSCP-256) is 2289 (resp. 3039) GEs with a throughput of 29.62 (resp. 44.44) kbps. Their respective areas in CMOS $130\,nm$ are 2498 and 3319 GEs. sLiSCP-256 has a GE area that is comparable with the estimated area

Table 19: Parallel hardware implementation of sLiSCP modes and comparison with other lightweight hash and AE primitives. Throughput is given for a frequency of 100 kHz.

Hash function	**Parameters** [a]				**Security(bits)**			**Process**	**Latency**	**Area**	**Throughput**
	r	c	r'	h	Pre	2nd Pre.	Coll.	(nm)	(Cycles)	(GEs)	(kbps)
sLiSCP-192	32	160	32	160	128	80	80	65	108	**2271**	29.62
sLiSCP-192	32	160	32	160	128	80	80	130	108	**2492**	29.62
Photon-160/36/36 [43]	36	160	36	160	124	80	80	180	180	2117	20.00
D-Quark [7]	16	160	16	176	160	80	80	180	88	2819	18.18
Spongent-160/160/16 [28]	16	160	16	160	144	80	80	130	90	2190	17.78
Keccak-f[40,160] [48]	40	160	40	200	160	160	80	130	18	4900	222.22
Keccak-f[72,128] [48]	72	128	72	200	128	128	64	130	18	4900	400.00
sLiSCP-256	64	192	64	192	128	96	96	65	144	**3019**	44.44
sLiSCP-256	64	192	64	192	128	96	96	130	144	**3305**	44.44
sLiSCP-256	64	192	32	192	160	96	96	65	144	**3019**	22.22
sLiSCP-256	64	192	32	192	160	96	96	130	144	**3305**	22.22
Photon-224/32/32 [43]	32	224	32	224	192	112	112	180	204	2786	15.69
Spongent-160/160/80 [28]	80	160	80	160	80	80	80	130	120	3139	66.67
Spongent-224/224/16 [28]	16	224	16	224	208	112	112	130	120	2903	13.33
Spongent-256/256/16 [28]	16	256	16	256	240	128	128	130	140	3281	11.43
S-Quark [7]	32	224	32	256	224	112	112	180	64	4640	50
AE algorithm				t	Con. [b]	Int.[c]					
sLiSCP-192/80	32	160	32	80	80	80	-	65	108	**2289**	29.62
sLiSCP-192/80	32	160	32	80	80	80	-	130	108	**2498**	29.62
sLiSCP-192/112	32	160	32	112	112	112	-	65	108	**2289**	29.62
sLiSCP-192/112	32	160	32	112	112	112	-	130	108	**2498**	29.62
sLiSCP-256/128	64	192	64	128	128	128	-	65	144	**3039**	44.44
sLiSCP-256/128	64	192	64	128	128	128	-	130	144	**3319**	44.44
Ketje-Jr [14]	16	184	16	96	96	96	-	-	-	4900[d]	-
NORX-16 [9]	128	128	128	96	96	96	-	-	-	2880	-

[a] r, c, r', h and t denote the input bitrate, capacity, output bitrate, digest length and tag size, respectively.
[b] Confidentiality of plaintext.
[c] Integrity of plaintext, associated data and nonce.
[d] Considering it uses Keccak-200 as its underlying permutation, its area is at least 4900 GEs.

of NORX-16, while sLiSCP-192 is quite smaller than NORX-16. Both areas of sLiSCP-192 and sLiSCP-256 are much smaller than that of Ketje-Jr. We note that serialized implementations of sLiSCP modes result in more savings in GE area and thus enable its adoption in highly constrained devices such as EPC tags. Overall, both the hash and authenticated encryption modes of sLiSCP are competitive with others in terms of area and throughput.

8 Concluding Remarks

In this section, we conclude the paper by highlighting some of the design choices that we have made for the construction of the sLiSCP permutations. Most of the contents of this section have been stated in a scattered way in earlier sections, so we aim by addressing these points again to reiterate some important conclusions that may have been missed by a reader.

-*"Another sponge-based primitive:"* We design sLiSCP in response to the noticeable shortage of lightweight secure cryptographic permutations which can be used in the sponge framework to provide a unified secure

design which offers as many cryptographic functionalities as possible. In fact, most of the lightweight symmetric key primitives that exist in the literature are dedicated to offer a specific cryptographic functionality and accordingly are optimized as such. Other than Keccak-200 permutation which has a parallel hardware implementation cost of around 4900 GE [48], we cannot find a lightweight cryptographic permutation. On the other hand, sLiSCP-192 has a parallel implementation cost of 2289 GEs on a 65 nm ASIC technology which enables its realistic adoption in constrained lightweight applications to provide a minimal cryptographic design.

- *"Simeck is based on the generalized round function of NSA's Simon:"* The justifications by the NSA of the parameters and design choices for Simon remain unclear. However, Simeck is an independently parameterized unkeyed version of the generalized Simon round function. In addition to being fully analyzed by its designers [72] where all the parameter choices have been justified, Simeck has been publicly cryptanalyzed for over three years. Finally, Simeck offers one of the lowest hardware footprints which is even lower than Simon's.

- *"The* sLiSCP *permutation is based on a GFS like the MD/SHA family of hash functions"* The MD/SHA family is a special instantiation of the Feistel construction which is vulnerable to the Wang *et al.* differential attacks [66]. However, such attacks are successful on this family of hash functions due to the ability of the attacker to manipulate the propagation of differences in the internal state through message modification techniques, which are effective because the algorithm allows a user to feed the state with independent message blocks for a substantial number of rounds. Nevertheless, without the message feeding algorithm, the Wang *et al.* attacks are ineffective and sLiSCP is an unkeyed permutation where the attacker has no means to manipulate the value of the internal state amid execution.

- *"Simeck operations are bit-based and so it scales linearly with state size thus, can be used directly on a large state for* sLiSCP*:"* Large Simeck states such as Simeck-128 are hard to analyze, even the probabilities of their differential and linear characteristics are harder to bound for the extended number of rounds [51, 54]. Thus, providing security guarantees for Simeck with an even larger state is almost unpractical. Consequently, the adoption of Simeck-48 and Simeck-64 in a Type-2 GFS construction enables us to leverage their existing cryptanalysis and further provide bounds on the probabilities of differential and linear characteristics for our whole permutation.

References

[1] Abdelkhalek, A., Tolba, M., and Youssef, A.: Impossible differential attack on reduced round sparx-64/128. In: Joye M., Nitaj A. (Eds.), AFRICACRYPT 2017. LNCS, vol. 10239, pp. 135-146. Springer, Cham (2017)

[2] Abed, F., List, E., Lucks, S., and Wenzel, J.: Differential cryptanalysis of round-reduced simon and speck. In: Cid, C. Rechberger, C. (Eds.), FSE 2014. LNCS, vol. 8540, pp. 525-545. Springer, Heidelberg (2015)

[3] Agren, M., Hell, M., Johansson, T., and Meier, W.: Grain-128a: A new version of grain-128 with optional authentication. Int. J. Wire. Mob. Comput. 5(1), 48–59. (2011)

[4] Albrecht, M. R., Driessen, B., Kavun, E. B., Leander, G., Paar, C., and Yalçin, T.: Block ciphers – focus on the linear layer (feat. pride). In: J. A. Garay and R. Gennaro (Eds.), CRYPTO 2014. LNCS, vol. 8616, pp. 57–76. Springer, Heidelberg (2014)

[5] Andreeva, E., Mennink, B., and Preneel, B.: The parazoa family: Generalizing the sponge hash functions. International Journal of Information Security 11(3), 149–165. (2012)

[6] Armknecht, F., Hamann, M., and Mikhalev, V.: Lightweight authentication protocols on ultra-constrained rfids - myths and facts. In: Saxena, N. Sadeghi, A.-R. (Eds.), RFIDSec 2014, Revised Selected Papers. LNCS, vol. 8651, pp. 1-18. Springer, Cham (2014)

[7] Aumasson, J.-P., Henzen, L., Meier, W., and Naya-Plasencia, M.: Quark: A lightweight hash. Journal of Cryptology 26(2), 313–339. (2013)

[8] Aumasson, J.-P., Jovanovic, P., and Neves, S.: NORX: Parallel and scalable aead. In: M. Kutyłowski and J. Vaidya (Eds.), ESORICS 2014. LNCS, vol. 8713, pp. 19-36. Springer, Cham (2014)

[9] Aumasson, J.-P., Jovanovic, P., and Neves, S. Norx8 and norx16: Authenticated encryption for low-end systems. Cryptology ePrint Archive, Report 2015/1154, 2015. http://eprint.iacr.org/2015/1154.

[10] Aumasson, J.-P., and Meier, W.: Zero-sum distinguishers for reduced keccak-f and for the core functions of luffa and hamsi. Rump session of CHES 2009, 67.
[11] Babbage, S., and Dodd, M.: The MICKEY stream ciphers. In: New stream cipher designs. LNCS, vol. 4986, pp. 191-209. Springer, Heidelberg (2008)
[12] Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., and Wingers, L.: The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.
[13] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., and Sim, S. M.: The skinny family of block ciphers and its low-latency variant MANTIS. In: M. Robshaw and J. Katz (Eds.), CRYPTO 2016. LNCS, vol. 9815, pp. 123–153. Springer, Heidelberg (2016)
[14] Bertoni, G., Daemen, J., Peeters, M., and Assche, G.: Caesar submission: Ketje v2, 2014. http://ketje.noekeon.org/Ketjev2-doc2.0.pdf.
[15] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G.: On the indifferentiability of the sponge construction. In: N. Smart (Ed.), EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008)
[16] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G.: Keccak specifications. Submission to nist (round 2). (2009)
[17] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Sponge-based pseudo-random number generators. In: Mangard S., Standaert FX. (Eds.), CHES 2010. LNCS, vol. 6225, pp. pp. 33–47. Springer, Heidelberg (2010)
[18] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G.: On the security of the keyed sponge construction. In: Symmetric Key Encryption Workshop 2011.
[19] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: A. Miri and S. Vaudenay (Eds), SAC 2012. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012)
[20] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G.: Permutation-based encryption, authentication and authenticated encryption. In: DIAC 2012.
[21] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G.: Cryptographic sponge functions, 2014. http://sponge.noekeon.org/CSF-0.1.pdf.
[22] Biham, E., Biryukov, A., and Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: J. Stern, (Ed.), EUROCRYPT 1999. LNCS, vol. 1592, pp. 12–23. Springer, Heidelberg (1999)
[23] Biham, E., and Shamir, A.: Differential cryptanalysis of des-like cryptosystems. Journal of CRYPTOLOGY 4(1), 3–72. (1991)
[24] Biryukov, A., Roy, A., and Velichkov, V.: Differential analysis of block ciphers simon and speck. In: C. Cid and C. Rechberger (Eds.), FSE 2014. LNCS, vol. 8540, pp. 546-570. Springer, Heidelberg (2014)
[25] Biryukov, A., and Wagner, D.: Slide attacks. In: L. Knudsen (Ed.), FSE 1999. LNCS, vol. 1636, pp. 245–259. Springer, Heidelberg (1999)
[26] Blondeau, C., Bogdanov, A., and Wang, M.: On the (in)equivalence of impossible differential and zero-correlation distinguishers for feistel- and skipjack-type ciphers. In: I. Boureanu, P. Owesarski, and S. Vaudenay (Eds.), ACNS 2014. LNCS, vol. 8479, pp. 271-288. Springer, Cham (2014)
[27] Blondeau, C., and Minier, M.: Analysis of impossible, integral and zero-correlation attacks on type-ii generalized feistel networks using the matrix method. newblock In: G. Leander (Ed.), FSE 2015. LNCS, vol. 9054, pp. 92–113. Springer, Heidelberg (2015)
[28] Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varici, K., and Verbauwhede, I.: Spongent: A lightweight hash function. In: B. Preneel and T. Takagi (Eds.), CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
[29] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: P. Paillier and I. Verbauwhede (Eds.), CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
[30] Bogdanov, A., Leander, G., Nyberg, K., and Wang, M.: Integral and multidimensional linear distinguishers with correlation zero. In: X. Wang and K. Sako, (Eds.), ASIACRYPT 2012. LNCS, vol. 7658, pp. 244–261. Springer, Heidelberg (2012)
[31] Bogdanov, A., and Shibutani, K.: Generalized feistel networks revisited. Designs, Codes and Cryptography 66(1), 75–97. (2013)
[32] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E. B., Knezevic, M., Knudsen, L. R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S. S., and Yalçin, T.: PRINCE – A low-latency block cipher for pervasive computing applications. In: Wang X., Sako K. (Eds), ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)

[33] Boura, C., and Canteaut, A.: Zero-sum distinguishers for iterated permutations and application to keccak-f and hamsi-256. In: Biryukov A., Gong G., Stinson D.R. (Eds.), SAC 2010. LNCS, vol. 6544, pp. 1–17. Springer, Heidelberg (2010)
[34] CAESAR: Competition for authenticated encryption: security, applicability, and robustness. https://competitions.cr.yp.to/caesar.html.
[35] Daemen, J., Knudsen, L., and Rijmen, V.: The block cipher square. In: E. Biham (Ed.), FSE 1997. LNCS, vol. 1267, pp. 149–165. Springer, Heidelberg (1997)
[36] De Cannière, C.: Trivium: A stream cipher construction inspired by block cipher Design Principles. In: Katsikas S.K., Lpez J., Backes M., Gritzalis S., Preneel B.(Eds.), Information Security 2006. LNCS, vol. 4176, pp. 171–186. Springer, Heidelberg (2006)
[37] De Cannière, C., Dunkelman, O., and Knežević, M.: KATAN and KTANTAN — A family of small and efficient hardware-oriented block ciphers. In: Clavier C., Gaj K. (Eds.), CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
[38] Diffie, W., and Hellman, M. E.: Special feature exhaustive cryptanalysis of the nbs data encryption standard. Computer 10(6), 74–84. (1977)
[39] Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Grossschädl, J., and Biryukov, A.: Design strategies for arx with provable bounds: Sparx and lax. In: J. H. Cheon and T. Takagi (Eds.), ASIACRYPT 2016. LNCS, vol. 10031, pp. 484–513. Springer, Heidelberg (2016)
[40] Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M.: Ascon v1.2. Submission to the caesar competition: http://competitions.cr.yp.to/round3/asconv12.pdf. (2016)
[41] Gong, Z., Nikova, S., and Law, Y. W.: KLEIN: A new family of lightweight block ciphers. In: Juels A., Paar C. (Eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 1–18. Springer, Heidelberg (2012)
[42] Gueron, S., and Mouha, N.: Simpira v2: A family of efficient permutations using the aes round function. In: J. H. Cheon and T. Takagi (Eds.), ASIACRYPT 2016. LNCS, vol. 10031, pp. 95–125. Springer, Heidelberg (2016)
[43] Guo, J., Peyrin, T., and Poschmann, A.: The photon family of lightweight hash functions. In: P. Rogaway (Ed.), CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
[44] Guo, J., Peyrin, T., Poschmann, A., and Robshaw, M.: The led block cipher. In: B. Preneel and T. Takagi (Eds.), CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)
[45] Hell, M., Johansson, T., Maximov, A., and Meier, W.: A stream cipher proposal: Grain-128. In: IEEE International Symposium on Information Theory, 1614–1618. (2006)
[46] Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.-S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., and Chee, S.: HIGHT: A new block cipher suitable for low-resource device. In: Goubin L., Matsui M (Eds.), CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
[47] Juels, A., and Weis, S. A.: Authenticating pervasive devices with human protocols. In: Shoup V. (Ed.), CRYPTO 2005. LNCS, vol. 3621, pp. 293–308. Springer, Heidelberg (2005)
[48] Kavun, E. B., and Yalcin, T.: A lightweight implementation of keccak hash function for radio-frequency identification applications. In: Ors Yalcin S.B (Ed.), RFIDSec 2010. LNCS, vol. 6370, pp. 258–269. Springer, Heidelberg (2010)
[49] Keliher, L.: Exact maximum expected differential and linear probability for two-round advanced encryption standard. IET Information Security 1, 53–57(4). (2007)
[50] Knudsen, L., and Wagner, D.: Integral cryptanalysis. In: J. Daemen and V. Rijmen (Eds.) FSE 2002. LNCS, vol. 2365, pp. 112–127. Springer, Heidelberg (2002)
[51] Kölbl, S., Leander, G., and Tiessen, T.: Observations on the simon block cipher family. In: R. Gennaro and M. Robshaw (Eds.), CRYPTO 2015. LNCS, vol. 9215, pp. 258–269. Springer, Heidelberg (2015)
[52] Kondo, K., Sasaki, Y., and Iwata, T.: On the design rationale of simon block cipher: Integral attacks and impossible differential attacks against simon variants. In: M. Manulis, A.-R. Sadeghi, and S. Schneider (Eds.), ACNS 2016. LNCS, vol. 9696, pp. 518–536. Springer, CHAM (2016)
[53] Leander, G., Abdelraheem, M. A., AlKhzaimi, H., and Zenner, E.: A cryptanalysis of printcipher: The invariant subspace attack. In: P. Rogaway (Ed.), CRYPTO 2011. LNCS, vol. 6841, pp. 206–221. Springer, Heidelberg (2011)
[54] Liu, Z., Li, Y., and Wang, M.: Optimal differential trails in simon-like ciphers. IACR TOSC 2017, 358–379. (2017)
[55] Lucks, S.: The saturation attack a bait for twofish. In: Matsui M. (Ed.) FSE 2001. LNCS, vol. 2355, pp. 1-15. Springer, Heidelberg (2002)
[56] Matsui, M., and Yamagishi, A.: A new method for known plaintext attack of feal cipher. In: Rueppel R.A. (Eds.) EUROCRYPT 1992. LNCS, vol. 658, pp. 81-91. Springer, Heidelberg (1993)
[57] McKay, K., Bassham, L., Sönmez Turan, M., and Mouha, N.: Report on lightweight cryptography (NISTIR8114). (2017)

[58] Nawaz, Y., and Gong, G.: Wg: A family of stream ciphers with designed randomness properties. Inf. Sci. 178(7), 1903–1916. (2008)
[59] Nyberg, K.: Generalized feistel networks. In: K. Kim and T. Matsumoto (Eds.), ASIACRYPT 1996. LNCS, vol. 1163, pp. 91–104. Springer, Heidelberg (2011)
[60] Rønjom, S.: Invariant subspaces in Simpira. Cryptology ePrint Archive, Report 2016/248, 2016. http://eprint.iacr.org/2016/248.
[61] Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., and Shirai, T.: Piccolo: An ultra-lightweight blockcipher. In: Preneel B., Takagi T. (Eds.), CHES 2011. LNCS, vol. 6917, pp. 342–357. Springer, Heidelberg (2011)
[62] Suzaki, T., and Minematsu, K.: Improving the generalized feistel. In: Hong S., Iwata T. (Eds), FSE 2010. LNCS, vol. 6147, pp. 19–39. Springer, Heidelberg (2010)
[63] Suzaki, T., Minematsu, K., Morioka, S., and Kobayashi, E.: Twine: A lightweight block cipher for multiple platforms. In: Knudsen L.R., Wu H. (Eds.), SAC 2012. LNCS, vol. 7707, pp. 339–354. Springer, Heidelberg (2012)
[64] Todo, Y.: Structural evaluation by generalized integral property. In: E. Oswald and M. Fischlin (Eds.), EUROCRYPT 2015. LNCS, vol. 9056, pp. 287–314. Springer, Heidelberg (2015)
[65] Wang, Q., Liu, Z., Varici, K., Sasaki, Y., Rijmen, V., and Todo, Y.: Cryptanalysis of reduced-round simon32 and simon48. In: W. Meier and D. Mukhopadhyay (Eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 143–160. Springer, Cham (2014)
[66] Wang, X., and Yu, H.: How to break md5 and other hash functions. In: R. Cramer (Ed.), EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
[67] Wheeler, D., and Needham, R.: Tea, a tiny encryption algorithm. In: Preneel B. (Ed.), FSE 1994. LNCS, vol. 1008, pp. 97–110. Springer, Heidelberg (2005)
[68] Wu, W., Wu, S., Zhang, L., Zou, J., and Dong, L.: LHash: A lightweight hash function. In: Lin D., Xu S., Yung M. (Eds), Inscrypt 2013. LNCS, vol. 8567, pp. 291–308
[69] Wu, W., and Zhang, L.: LBlock: A lightweight block cipher. In: Lopez J., Tsudik G. (Eds), ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer, Heidelberg (2011)
[70] Xiang, Z., Zhang, W., Bao, Z., and Lin, D.: Applying milp method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: J. H. Cheon and T. Takagi (Eds.), ASIACRYPT 2016. LNCS, vol. 10031, pp. 648–678. Springer, Heidelberg (2016)
[71] YANAGIHARA, S., and IWATA, T.: Type 1.x generalized feistel structures. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 97(4), 952-963. (2014)
[72] Yang, G., Zhu, B., Suder, V., Aagaard, M. D., and Gong, G.: The simeck family of lightweight block ciphers. In: T. Güneysu and H. Handschuh (Eds.), CHES 2015. LNCS, vol. 9293, pp. 307–329. Springer, Heidelberg (2015)
[73] Yap, H., Khoo, K., Poschmann, A., and Henricksen, M.: EPCBC - A block cipher suitable for electronic product code encryption. In: Lin D., Tsudik G., Wang X.(Eds.), CANS 2011. LNCS, vol. 7092, pp. 76–97. Springer, Heidelberg (2011)
[74] Zhang, H., and Wu, W.: Structural evaluation for generalized feistel structures and applications to lblock and twine. In: A. Biryukov and V. Goyal (Eds.), INDOCRYPT 2015. LNCS, vol. 9462, pp. 218–237. Springer, Cham (2015)

A Test Vectors for sLiSCP AEAD and Hash Instances

In this section we provide the test vectors for each instance of the sLiSCP AEAD and hash. The key, initial vector (IV), associated data (AD), message, ciphertext and tag are represented in hexadecimal.

A.1 sLiSCP Permutations

sLiSCP-192. The input and output of the sLiSCP-192 permutation are given below.
Input: `00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
Output: `c2 e9 68 54 41 38 70 d0 c4 23 bb 2a bd d2 4e 65 6e c7 f9 50 41 56 dd c1`

sLiSCP-256. The input and output of the sLiSCP-256 permutation are given below.
Input: `00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00`
Output: `09 15 02 b1 4d 0d c8 45 91 86 66 88 e8 df 52 5f f6 92 c3 07 a5 f5 59 ba 72 b6 f8 0d c9 f0 5d 53`

A.2 Authenticated Encryption Mode of sLiSCP

sLiSCP-192/80.

Key:	10 10 10 10 10 10 10 10 10 10
Nonce:	44 44 44 44 44 44 44 44 44 44
AD:	66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66
Message:	88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88
Ciphertext:	89 a8 37 38 62 05 d9 a6 2f f2 05 be c7 e1 ef 91
Tag:	67 f3 d8 4b 54 a0 83 d5 2e ad

sLiSCP-192/112.

Key:	10 10 10 10 10 10 10 10 10 10 10 10 10 10
Nonce:	44 44 44 44 44 44 44 44 44 44
AD:	66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66
Message:	88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88
Ciphertext:	93 26 5d 94 50 9b bd 1a 3c 26 1d ee 9e d5 7a fe
Tag:	1e d4 e4 9b 46 87 87 a5 8f 06 be 8b b5 68

sLiSCP-256/128.

Key:	10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
Nonce:	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44
AD:	66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66
Message:	88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88
Ciphertext:	77 3f c8 0b d0 6c 21 28 f6 cc 4c be d8 0b 15 0e
Tag:	77 68 fc a3 b5 7e 8e 2c c0 a7 7b 1e 07 55 1e 80

A.3 Hash Mode of sLiSCP

sLiSCP-192. Rate $r' = 32$

IV:	50 20 20
Message (M):	88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88
Hash value $H(M)$:	aa 5c 69 05 0c 39 26 58 5e 97 45 f0 74 38 7c ec cc 67 26 f3

sLiSCP-256. Rate $r' = 32$

IV:	60 40 20
Message (M):	88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88
Hash value $H(M)$:	52 2a d6 ba e2 3c 06 a8 84 7a d2 ad ec 88 90 ca 96 7a f5 62 28 50 46 a8

sLiSCP-256. Rate $r' = 64$

IV:	60 40 40
Message (M):	88 88 88 88 88 88 88 88 88 88 88 88 88 88 88 88
Hash value $H(M)$:	c5 14 1f 85 71 d1 36 66 b9 a9 61 ef 0f c9 09 04 df be 10 64 f9 9b 9d 7a