SAND: An AND-RX Feistel Lightweight Block Cipher Supporting S-box-based Security Evaluations

Shiyao Chen^{1,2}, Yanhong Fan^{1,2}, Ling Sun^{1,2}, Yong Fu^{3}, Haibo Zhou^{1,2}, Yongqing Li^{1,2}, Meiqin Wang^{1,2}, Weijia Wang^{1,2}, and Chun Guo^{1,2}

^{1} School of Cyber Science and Technology, Shandong University, Qingdao, China
^{2} Key Laboratory of Cryptologic Technology and Information Security of Ministry of Education, Shandong University, Qingdao, China
{sychen,fanyh,haibozhou,yqli}@mail.sdu.edu.cn
{lingsun,mqwang,wjwang,chun.guo}@sdu.edu.cn
^{3} Qilu University of Technology, Jinan, China
yongfu0976@gmail.com

Abstract. We revisit designing AND-RX block ciphers, that is, the designs assembled with the most fundamental binary operations—AND, Rotation and XOR operations and do not rely on existing units. Likely, the most popular representative is the NSA cipher SIMON, which remains one of the most efficient designs, but suffers from difficulty in security evaluation.

As our main contribution, we propose SAND, a new family of lightweight AND-RX block ciphers. To overcome the difficulty regarding security evaluation, SAND follows a novel design approach, the core idea of which is to restrain the AND-RX operations to be within nibbles. By this, SAND admits an equivalent representation based on a 4×8 *synthetic S-box* (*SSb*). This enables the use of classical S-box-based security evaluation approaches. Consequently, for all versions of SAND, (a) we evaluated security bounds with respect to differential and linear attacks, and in both single-key and related-key scenarios; (b) we also evaluated security against impossible differential and zero-correlation linear attacks.

This better understanding of the security enables the use of a relatively simple key schedule, which makes the ASIC round-based hardware implementation of SAND to be one of the state-of-art Feistel lightweight ciphers. As to software performance, due to the natural bitslice structure, SAND reaches the same level of performance as SIMON and is among the most software-efficient block ciphers.

Keywords: Lightweight Cryptography · Feistel Structure · AND-Rotation-XOR · Synthetic S-box · Related-key Security

1 Introduction

With the rapid development of pervasive computing, particularly the imminent 5th generation (5G) networking technology for seamless communication, the Internet will be characterized by more and more application scenarios, like smart home, smart cities and industry 4.0. Under the fusion of 5G, Artificial Intelligence (AI) and Internet of Things (IoT), the power of data will also be unleashed, gradually. However, this fast-growing number of connected devices and ubiquitous communications in the network also poses a potential threat to the security of data and privacy. In particular, the presence of plenty of constrained devices, which can only devote a small fraction of resources to security, has motivated seeking for new security/efficiency trade-offs. Consequently, the design and analysis of lightweight symmetric schemes have been one of the most productive lines of research in recent years.

Actually, the term *lightweight* is not strictly defined due to a variety of usage scenarios, from low power consumption (passive RFID tags) to low energy consumption (battery-powered devices) or low latency application (disk encryption). However, area cost is usually a major criterion for lightweight block ciphers, but when taking power, energy and throughput into consideration, it is rather a complex task to compare different lightweight primitives. In fact, these metrics also show different requirements, faced with various deployments in the real-world, many lightweight block ciphers have been proposed for different applications in recent years. For instance, `Midori` [6] is designed for energy and power efficiency of hardware implementation. `PRINCE` [24], `MANTIS` [13] and `Qarma` [4] all aim to achieve a low latency. The block ciphers `SIMON` (AND-RX) and `SPECK` (ARX) from the NSA [12] are two quite elegant and competitive algorithms, where the former is hardware-oriented and the latter is software-oriented. `RECTANGLE` [75] is designed to facilitate bitslice implementations and is suitable for multiple platforms. `SKINNY` [13] is then proposed as a competitor to `SIMON` in terms of performance and provides much stronger security guarantees with regard to differential/linear attacks. `GIFT` [9] pursues even smaller area consumption but still with sound security. Most recently, the cipher `CRAFT` [15] takes the implementations of efficient protection against DFA attacks as its main design criteria. Some lightweight proposals also have been standardized as ISO/IEC standards, like `PRESENT` [22] and `CLEFIA` [64]. To summarize, significant advances have been made regarding the performance as well as approaches to design and security analysis of lightweight block ciphers.[4]

To further elaborate, `SKINNY` and `GIFT` are considered as two prominent examples. Both are among the best SPN ciphers, and have been chosen by a number of submissions to the NIST Lightweight Standardization Process as the underlying building blocks. In detail, the designs `TGIF` [41], `SUNDAE-GIFT` [7], `GIFT-COFB` [8], `HYENA` [26], and `LOTUS-AEAD/LOCUS-AEAD` [25] are built upon `GIFT`, while the designs `ForkAE` [1], `SKINNY-AEAD/SKINNY-HASH` [14], `Remus` [39], and `Romulus` [40] are based on `SKINNY`.

In contrast, less progress has been made regarding the counterpart Feistel block ciphers. A Feistel cipher is involution-like, i.e., encryption and decryption are the same up to different round key assignments. When hardware implementation of the decryption functionality is required, this significantly reduces the

[4] For more details, we refer to the main page `https://www.cryptolux.org/index.php/Lightweight_Cryptography`, maintained by the CryptoLUX research group.

cost, which constitutes the main advantage of the Feistel scheme compared to the SPN approach. Efficient Feistel lightweight ciphers were mostly designed years before, with LBlock [72], Piccolo [63], TWINE [68], RoadRunneR [11], SIMON and SIMECK [74] as notable examples.

Among the above, SIMON achieves the best performance due to its extremely simple AND-RX structure. Yet, as a side effect, it turns out rather cumbersome to establish security lower bounds against differential and linear cryptanalysis, in both single-key and related-key scenarios. The difficulties are two-fold. First, the bit-oriented nature of SIMON effectively prohibits the use of the well-understood active S-box proof approaches [31,55,67]. Moreover, the dependency problem in the round function raises additional difficulty in understanding its exact differential and linear properties. The original design paper [12] has eluded any security analysis and bounds as well as the design rationale. Then, the characterizations of the exact differential and linear properties of SIMON were found by Kölbl *et al.* [47], which adopted Boolean satisfiability problem (SAT) and satisfiability modulo theories (SMT) based search methods. However, security bounds remain intractable for large versions such as SIMON-128 due to the large search space. This was eventually solved by Liu *et al.* [49], who identified the provably optimal differential trails of SIMON-64, SIMON-96 and SIMON-128 in the single-key scenario, using a delicate variant of Matsui's branch-and-bound search algorithm.

Despite the above exciting progress regarding their security analysis, it remains an open problem to obtain AND-RX designs with easier provable bounds and utilize their hardware efficiency. The usual approach is to start from ad hoc designs and apply the above search tools for the security evaluation. Note that, for the similar ARX designs, the landscape already turns different: the recently proposed *long trail design strategy* (LTS) and the result SPARX [33] exhibited that *ARX ciphers can be designed with both provable bounds and efficient software performance*. Motivated by these, we ask if it is possible to design Feistel lightweight block ciphers with AND-RX operations (more precisely, based on SIMON) with better efficiency and stronger security bounds, even in the related-key setting.

1.1 Our Contributions

In this paper, we answer the above question positively. In detail, we propose a new family of AND-RX block ciphers SAND. Due to the novel design approach, SAND admits an equivalent S-box-based representation (which we will elaborate later) and thus supporting S-box-based security evaluation approaches. This allows for both strong security bounds and competitive performance:

- **High security.** Benefiting from the well-developed active S-box-based security evaluation approaches, SAND ensures strong security in *both single-key and related-key scenario*. We remark that until now, only a few lightweight designs are designed to resist related-key attacks, with LED [37], PICCOLO, TWINE and SKINNY to name a few.

– **Efficient hardware and software performances.** As shown in Table 1, for ASIC round-based implementations, all versions of SAND have competitive area cost for both the encryption-only and the unification of encryption/decryption among the current most efficient lightweight block ciphers. Regarding software implementations, the natural bitslice structure and AND-RX operations push SAND to be among the best—in particular, comparable with SIMON.

SAND's design idea, and New approach to AND-RX We now elaborate on why our new design SAND admits dual representations (S-box-based and AND-RX based structures). A fundamental unit function in SIMON is $f(x_1, x_2, x_3) := x_1 \odot x_2 \oplus x_3$, which maps three input bits to a single output bit. We still rely on this (non-linear) AND-RX function. The crucial trick is that we first divide the round function input into several nibbles and then apply two bijective "unit" AND-RX functions on every nibble. These two "unit" AND-RX functions are "2-round" (multi-branch) Feistel-like transformations using the aforementioned f as "round functions". In contrast, in the SIMON round function, every input bit is tripled with two different other input bits and taken as the inputs of f. Consequently, the SIMON round function is somewhat "undividable", while our design admits a dual representation.

The size of the nibbles in SAND is 4 bits (whereas larger nibbles are of course supported by this approach). By this, every nibble gives rise to an 8-bit output, and thus the two "unit" AND-RX functions could be viewed as a single 4×8-bit *synthetic S-box* (SSb). In this vein, the SAND round function could be written as parallel applications of such synthetic S-boxes. Such a substitution layer clearly doubles the size of the intermediate state. To compress the state back to fit into the other Feistel branch, we XOR a half of the state to the other and then apply a bit permutation for diffusion. It is this SSb-based representation that enables SAND to enjoy the traditional active S-box-based security evaluations, which gives rise to security lower bounds of SAND in both single-key and related-key scenarios.

We stress again that the 4×8-bit SSb is built upon (multi-branch) Feistel-style functions rather than a smaller SIMON-like non-linear function. Our approach of first applying SSb and then compressing increases the algebraic degree of the round function: the maximal degree of SSb is 3, while that of smaller SIMON-like functions is 2. Also, it achieves better diffusion. In all, our 4×8-bit SSb approach turns out crucial for the security and efficiency of SAND. We defer detailed discussion and comparison with similar design approaches (e.g., NOEKEON [30], KECCAK [18] and ASCON [34]) to Section 3.1. Besides the instance SAND, we believe that this new design approach is another important methodological contribution.

1.2 Outline

This paper is organized as follows. In Section 2, we first give the specifications of the new block cipher SAND. Then, the design rationales of SAND are given in

Ciphers	Type	Area (GE)	Delay (ns)	Cycles	$\text{TP}_{\text{100KHz}}$ (Kbps)	TP_{Max} (Mbps)	Related-key Security
SAND-64/128	Enc	1287	0.86	48	133.3	1550.4	✓
	Enc&Dec	1563	0.91	48	133.3	1465.2	
SKINNY-64/128	Enc	1306	1.63	36	177.8	1090.7	✓
	Enc&Dec	1916	2.28	36	177.8	779.7	
TWINE-64/128	Enc	1389	1.51	36	177.8	1177.3	✓[††]
	Enc&Dec	1687	1.58	36	177.8	1125.2	
SIMECK-64/128	Enc	1300	0.69	44	145.5	2108.0	Unknown
	Enc&Dec	1714	0.83	44	145.5	1752.5	
SIMON-64/128	Enc	1329	0.75	44	145.5	1939.4	Unknown
	Enc&Dec	1779	0.84	44	145.5	1731.6	
CRAFT-64/128	Enc	1316	0.70	32	200.0	2857.1	No[†]
	Enc&Dec	1618	1.00	32	200.0	2000.0	
ANU-64/128	Enc	1460	1.66	25	256.0	1542.2	No[‡]
	Enc&Dec	1835	2.44	25	256.0	1049.2	
SAND-128/128	Enc	1874	0.88	54	237.0	2693.6	✓
	Enc&Dec	2264	0.88	54	237.0	2693.6	
SKINNY-128/128	Enc	1849	2.12	40	320.0	1509.4	✓
	Enc&Dec	2973	2.31	40	320.0	1385.3	
WARP-128/128	Enc	1632	0.75	41	312.2	4162.6	Unknown[‡‡]
	Enc&Dec	1775	0.75	41	312.2	4162.6	
ANT-128/128	Enc	1898	0.85	46	278.3	3273.7	Unknown
	Enc&Dec	2284	0.88	46	278.3	3162.1	
SIMON-128/128	Enc	1930	0.78	68	188.2	2413.3	Unknown
	Enc&Dec	2535	0.84	68	188.2	2240.9	

[†] Although the authors of CRAFT do not claim any security in the related-key model, the practical related-key attacks on full round CRAFT are presented in [35].
[††] Based on TWINE, the designers of Tweakable TWINE [59] claim the related-key security for T-TWINE-128, which can be reduced to that of TWINE.
[‡] The related-key distinguisher and attacks on full round ANU are given in [60].
[‡‡] We also consider a recently proposed block cipher WARP [5], which is an efficient design with the single-key security claim.

Table 1: Comparison of lightweight block ciphers implemented in this paper, synthesized with TSMC 90nm standard cell library (All implementations are round-based and synthesized under area optimization).

Section 3. In Section 4, the security analysis of `SAND` is provided. Later, the hardware and software evaluations are given in Section 5 and Section 6 respectively. Finally, we conclude the paper in Section 7.

2 Specifications of SAND

`SAND` is a family of two AND-RX block ciphers with Feistel structure: `SAND`-64 and `SAND`-128. Both of them accept 128-bit keys and have different block sizes $2n$, where n stands for the length of the branch ($n = 32$ for `SAND`-64 and $n = 64$ for `SAND`-128). The basic parameters of `SAND`-64 and `SAND`-128 are listed in Table 2.

Cipher	**Block size $2n$**	**Branch size n**	**Key size m**	**Rounds R**
`SAND`-64	64	32	128	48
`SAND`-128	128	64	128	54

Table 2: Parameters for `SAND`-64 and `SAND`-128.

For the detailed specification, the following notations will be employed.

– $x = (x_{n-1}, x_{n-2}, \cdots, x_0)$: the n-bit variable with $(n \mod 4) \equiv 0$, where x_{n-1} is the most significant bit (MSB) and x_0 represents the least significant bit (LSB). A two-dimensional representation is also used for the variable x, which can be viewed as a $4 \times \frac{n}{4}$ array of bits

$$x = \begin{bmatrix} x_{n-1} \cdots x_7 \; x_3 \\ x_{n-2} \cdots x_6 \; x_2 \\ x_{n-3} \cdots x_5 \; x_1 \\ x_{n-4} \cdots x_4 \; x_0 \end{bmatrix}.$$

– $x \parallel y$: concatenation of variables x and y.
– $x \ll s$: shift x by s bits to the left.
– $x \lll t$: rotate x by t bits to the left.
– $x \lll_{\frac{n}{4}} t$: the n-bit variable x with $(n \mod 4) \equiv 0$, which can be divided into four $\frac{n}{4}$-bit words $x = (x_{n-1}, x_{n-2}, \cdots, x_0) = x\{3\}\|x\{2\}\|x\{1\}\|x\{0\}$, and each word $x\{i\}$ is rotated by t bits to the left within the word, that is $x \lll_{\frac{n}{4}} t = (x\{3\} \lll t)\|(x\{2\} \lll t)\|(x\{1\} \lll t)\|(x\{0\} \lll t)$.
– $x \odot y$: bitwise AND operation of x and y.
– $x \oplus y$: bitwise exclusive-or (XOR) operation of x and y.
– $x[i]$: the i-th nibble of variable x, e.g., for $x = (x_{n-1}, x_{n-2}, \cdots, x_0)$ with $(n \mod 4) \equiv 0$, $x[\frac{n}{4} - 1] = (x_{n-1}, x_{n-2}, x_{n-3}, x_{n-4})$, ..., $x[1] = (x_7, x_6, x_5, x_4)$, $x[0] = (x_3, x_2, x_1, x_0)$.

– $\mathrm{Act}(x[i])$: the characteristic function of $x[i]$, i.e.,

$$\mathrm{Act}(x[i]) = \begin{cases} 0, & \text{if } x[i] = \mathtt{0x0}, \\ 1, & \text{otherwise.} \end{cases}$$

For $x = x[\frac{n}{4} - 1] \| \cdots \| x[1] \| x[0]$,

$$\mathrm{Act}(x) = \mathrm{Act}\left(x[\frac{n}{4} - 1]\right) \| \cdots \| \mathrm{Act}(x[1]) \| \mathrm{Act}(x[0]).$$

For example, $\mathrm{Act}(x) = 0111$ if $x = \mathtt{0x0123}$.

We also give some definitions that will be used throughout the paper. It should be noted that the S-box SSb of `SAND` allows non-trivial linear transitions with bias $\frac{1}{2}$, which should not be counted as active S-boxes. Thus, we exclude the input/output masks that lead to a bias of $\frac{1}{2}$ in Definition 2 and Definition 3, instead of only the output mask 0 for the usual S-box.

Definition 1 (MDP - Maximal Differential Probability). *Let S be a vectorial Boolean function $S : \mathbb{F}_2^a \to \mathbb{F}_2^b$, the maximal differential probability of S is defined as:*

$$\max_{\Delta_{in} \in \mathbb{F}_2^a \setminus \{0\}, \Delta_{out} \in \mathbb{F}_2^b} \frac{\#\{x \in \mathbb{F}_2^a | S(x) \oplus S(x \oplus \Delta_{in}) = \Delta_{out}\}}{2^a}.$$

Definition 2 (MALB - Maximal Absolute Linear Bias). *Let S be a vectorial Boolean function $S : \mathbb{F}_2^a \to \mathbb{F}_2^b$, the maximal absolute linear bias of S is defined as:*

$$\max_{\substack{\Gamma_{in} \in \mathbb{F}_2^a, \Gamma_{out} \in \mathbb{F}_2^b \\ 0 < \#\{x \in \mathbb{F}_2^a | \Gamma_{in} \odot x = \Gamma_{out} \odot S(x)\} < 2^a}} \left| \frac{\#\{x \in \mathbb{F}_2^a | \Gamma_{in} \odot x = \Gamma_{out} \odot S(x)\}}{2^a} - \frac{1}{2} \right|,$$

where $\odot$ denotes the inner product operation.

Definition 3 (Active S-box and Non-active S-box). *Let S be a vectorial Boolean function $S : \mathbb{F}_2^a \to \mathbb{F}_2^b$, for any given differential transition $\Delta_{in} \to \Delta_{out}(\Delta_{in} \in \mathbb{F}_2^a, \Delta_{out} \in \mathbb{F}_2^b)$, whose probability is $prob_{\Delta_{in} \to \Delta_{out}}$. It will result in an active S-box when $0 < prob_{\Delta_{in} \to \Delta_{out}} < 1$ and a non-active S-box when $prob_{\Delta_{in} \to \Delta_{out}} = 1$. Similarly, for any given linear transition $\Gamma_{in} \to \Gamma_{out}(\Gamma_{in} \in \mathbb{F}_2^a, \Gamma_{out} \in \mathbb{F}_2^b)$, whose linear bias is $bias_{\Gamma_{in} \to \Gamma_{out}}$. It will result in an active S-box when $0 < |bias_{\Gamma_{in} \to \Gamma_{out}}| < \frac{1}{2}$ and a non-active S-box when $|bias_{\Gamma_{in} \to \Gamma_{out}}| = \frac{1}{2}$.*

2.1 Round Function

(1) State Loading

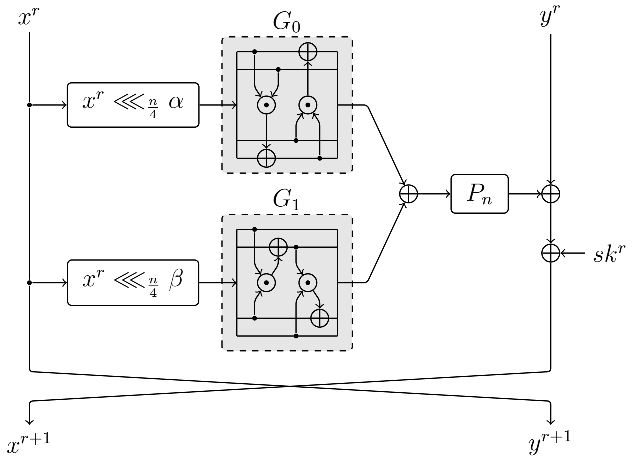

x^r
G_0
y^r
$x^r \lll_{\frac{n}{4}} \alpha$
G_1
P_n
sk^r
$x^r \lll_{\frac{n}{4}} \beta$
x^{r+1}
y^{r+1}

Figure 1: Round function of `SAND`.

Denote by $P = (Pl, Pr)$ the plaintext, then the left n-bit half $Pl = (Pl_{n-1}, \cdots, Pl_1, Pl_0)$ and the right n-bit half $Pr = (Pr_{n-1}, \cdots, Pr_1, Pr_0)$ can be viewed as two $4 \times \frac{n}{4}$ arrays of bits

$$Pl = \begin{bmatrix} Pl_{n-1} \cdots Pl_7\ Pl_3 \\ Pl_{n-2} \cdots Pl_6\ Pl_2 \\ Pl_{n-3} \cdots Pl_5\ Pl_1 \\ Pl_{n-4} \cdots Pl_4\ Pl_0 \end{bmatrix} = \begin{bmatrix} x^0\{3\} \\ x^0\{2\} \\ x^0\{1\} \\ x^0\{0\} \end{bmatrix}, Pr = \begin{bmatrix} Pr_{n-1} \cdots Pr_7\ Pr_3 \\ Pr_{n-2} \cdots Pr_6\ Pr_2 \\ Pr_{n-3} \cdots Pr_5\ Pr_1 \\ Pr_{n-4} \cdots Pr_4\ Pr_0 \end{bmatrix} = \begin{bmatrix} y^0\{3\} \\ y^0\{2\} \\ y^0\{1\} \\ y^0\{0\} \end{bmatrix}.$$

The input state (x^0, y^0) of the encryption is loaded from P in a row-by-row manner.

$$\begin{aligned} x^0 &= Pl_{n-1} \cdots Pl_7 Pl_3 \| Pl_{n-2} \cdots Pl_6 Pl_2 \| Pl_{n-3} \cdots Pl_5 Pl_1 \| Pl_{n-4} \cdots Pl_4 Pl_0 \\ &= x^0\{3\} \| x^0\{2\} \| x^0\{1\} \| x^0\{0\}, \\ y^0 &= Pr_{n-1} \cdots Pr_7 Pr_3 \| Pr_{n-2} \cdots Pr_6 Pr_2 \| Pr_{n-3} \cdots Pr_5 Pr_1 \| Pr_{n-4} \cdots Pr_4 Pr_0 \\ &= y^0\{3\} \| y^0\{2\} \| y^0\{1\} \| y^0\{0\}. \end{aligned}$$

Denote $C = (Cl, Cr)$ the ciphertext, the left n-bit half is $Cl = (Cl_{n-1}, \cdots, Cl_1, Cl_0)$ and the right n-bit half is $Cr = (Cr_{n-1}, \cdots, Cr_1, Cr_0)$, then C can be loaded from the output state of the encryption (x^R, y^R) as below

$$Cl = \begin{bmatrix} Cl_{n-1} \cdots Cl_7\ Cl_3 \\ Cl_{n-2} \cdots Cl_6\ Cl_2 \\ Cl_{n-3} \cdots Cl_5\ Cl_1 \\ Cl_{n-4} \cdots Cl_4\ Cl_0 \end{bmatrix} = \begin{bmatrix} x^R\{3\} \\ x^R\{2\} \\ x^R\{1\} \\ x^R\{0\} \end{bmatrix}, Cr = \begin{bmatrix} Cr_{n-1} \cdots Cr_7\ Cr_3 \\ Cr_{n-2} \cdots Cr_6\ Cr_2 \\ Cr_{n-3} \cdots Cr_5\ Cr_1 \\ Cr_{n-4} \cdots Cr_4\ Cr_0 \end{bmatrix} = \begin{bmatrix} y^R\{3\} \\ y^R\{2\} \\ y^R\{1\} \\ y^R\{0\} \end{bmatrix}.$$

(2) Iterative Round Function F

An illustration of F can be found in Figure 1, where G_0 and G_1 are non-linear functions, and P_n is a permutation providing diffusion. For the input state

(x^r, y^r) and the subkey sk^r of the r-th round (for $0 \leq r < R$), the output state (x^{r+1}, y^{r+1}) is computed with

$$(x^{r+1}, y^{r+1}) = F_{sk^r}(x^r, y^r) = (P_n(G_0(x^r \lll_{\frac{n}{4}} \alpha) \oplus G_1(x^r \lll_{\frac{n}{4}} \beta)) \oplus y^r \oplus sk^r, x^r),$$

where the tuple of the rotation constants (α, β) is fixed to $(0, 1)$ for all versions of SAND. To maintain the consistency in the encryption and decryption, the swap between the two branches is *omitted* in the last round.

(3) Non-linear Functions G_0 and G_1

Let the n-bit variable x be the input value of G_0 and G_1, which is regarded as the concatenation of four $\frac{n}{4}$-bit words $x\{3\}\|x\{2\}\|x\{1\}\|x\{0\}$. Let $y = y\{3\}\|y\{2\}\|y\{1\}\|y\{0\}$ denote the output value. For G_0, we have

$$\begin{aligned} y\{0\} &= x\{3\} \odot x\{2\} \oplus x\{0\}, \\ y\{3\} &= y\{0\} \odot x\{1\} \oplus x\{3\}, \\ y\{2\} &= x\{2\}, \\ y\{1\} &= x\{1\}. \end{aligned}$$

As to the function G_1, the output is calculated as

$$\begin{aligned} y\{2\} &= x\{3\} \odot x\{1\} \oplus x\{2\}, \\ y\{1\} &= y\{2\} \odot x\{0\} \oplus x\{1\}, \\ y\{3\} &= x\{3\}, \\ y\{0\} &= x\{0\}. \end{aligned}$$

(4) Bit Permutation P_n

Let variable x be the input value of the n-bit permutation P_n, which can be seen as the parallel application of a $\frac{n}{4}$-bit permutation $p_{\frac{n}{4}}$ on four $\frac{n}{4}$-bit words. For the i-th input word $x\{i\} = (x_{\frac{n}{4}\cdot i+\frac{n}{4}-1}, \ldots, x_{\frac{n}{4}\cdot i+1}, x_{\frac{n}{4}\cdot i})$, the element of the i-th output word $y\{i\}$ is defined as

$$y_{\frac{n}{4}\cdot i+p_{\frac{n}{4}}(j)} = x_{\frac{n}{4}\cdot i+j}, \text{ for } 0 \leq j < \frac{n}{4} \text{ and } 0 \leq i < 4.$$

The bit permutations p_8 and p_{16} used in SAND-64 (P_{32}) and SAND-128 (P_{64}) are given in Table 3 and Table 4, respectively. To further facilitate understanding, an illustration of p_8 is given in Figure 2.

Table 3: p_8 for SAND-64.

j	0	1	2	3	4	5	6	7
$p_8(j)$	7	4	1	6	3	0	5	2

Table 4: p_{16} for SAND-128.

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$p_{16}(j)$	14	15	8	9	2	3	12	13	6	7	0	1	10	11	4	5

2.2 Key Schedule

Key schedule of SAND-64. The 128-bit master key K is viewed as four 32-bit words, i.e., $K = K^3\|K^2\|K^1\|K^0$, which are the initial state of the LFSR shown

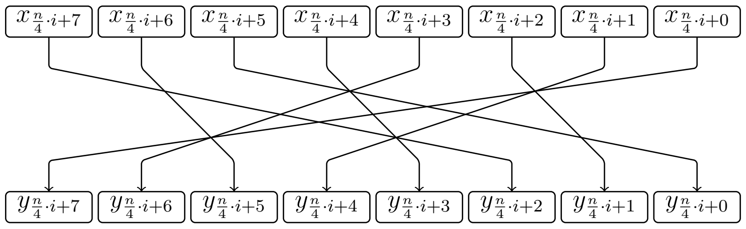

$x_{\frac{n}{4}\cdot i+7}$
$x_{\frac{n}{4}\cdot i+6}$
$x_{\frac{n}{4}\cdot i+5}$
$x_{\frac{n}{4}\cdot i+4}$
$x_{\frac{n}{4}\cdot i+3}$
$x_{\frac{n}{4}\cdot i+2}$
$x_{\frac{n}{4}\cdot i+1}$
$x_{\frac{n}{4}\cdot i+0}$
$y_{\frac{n}{4}\cdot i+7}$
$y_{\frac{n}{4}\cdot i+6}$
$y_{\frac{n}{4}\cdot i+5}$
$y_{\frac{n}{4}\cdot i+4}$
$y_{\frac{n}{4}\cdot i+3}$
$y_{\frac{n}{4}\cdot i+2}$
$y_{\frac{n}{4}\cdot i+1}$
$y_{\frac{n}{4}\cdot i+0}$

Figure 2: An illustration of p_8 operating on the i-th input word $x\{i\}$.

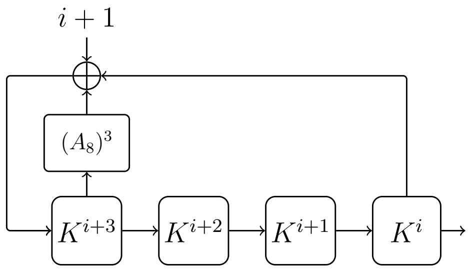


Figure 3: Key schedule of `SAND`-64.

in Figure 3. The update function of the LFSR is

$$K^{i+4} \leftarrow (A_8)^3 \left(K^{i+3}\right) \oplus K^i \oplus (i+1),$$

where $(i+1)$ is the round constant and with $0 \le i < R-4$. The operation A_8 is applied to K^{i+3} for three times, iteratively. Note that the r-th round subkey sk^r $(0 \le r < R)$ is loaded from K^r. The 32-bit state in K^r can be represented two-dimensional as a 4×8 array

$$K^r = \begin{bmatrix} K^r_{31} \cdots K^r_7 \; K^r_3 \\ K^r_{30} \cdots K^r_6 \; K^r_2 \\ K^r_{29} \cdots K^r_5 \; K^r_1 \\ K^r_{28} \cdots K^r_4 \; K^r_0 \end{bmatrix},$$

which will be loaded into current round subkey sk^r in the same manner as done for the input state of the encryption, then we have

$$sk^r = K^r_{31} \cdots K^r_3 \| K^r_{30} \cdots K^r_2 \| K^r_{29} \cdots K^r_1 \| K^r_{28} \cdots K^r_0.$$

As illustrated in Figure 4, A_8 is a nibble-oriented function, and its input is divided into eight nibbles, say $X[7]\| \cdots \|X[1]\|X[0]$. The output of A_8 is

$$(X[7] \lll t_1) \oplus X[0] \| X[7] \oplus (X[7] \ll t_0) \| X[6] \| X[5] \| X[4] \| X[3] \| X[2] \| X[1],$$

where t_0 and t_1 are set to 3 and 1, respectively.

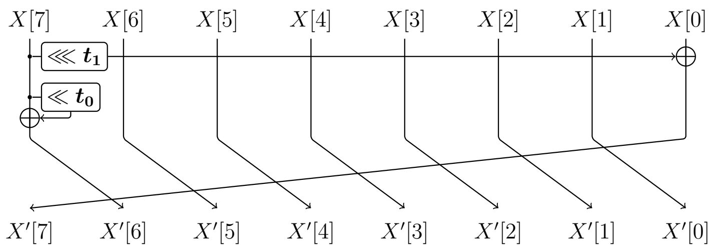


Figure 4: Operation A_8 for SAND-64.

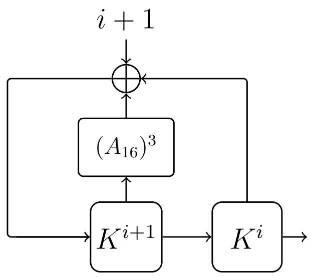


Figure 5: Key schedule of SAND-128.

Key schedule of SAND-128. The 128-bit master key K is viewed as two 64-bit words, i.e., $K = K^1 \| K^0$, which are used to initialize the state of the LFSR in Figure 5. The update function of the LFSR is

$$K^{i+2} \leftarrow (A_{16})^3(K^{i+1}) \oplus K^i \oplus (i+1),$$

where $(i+1)$ is also introduced as the round constant and with $0 \leq i < R-2$. The operation A_{16} is applied to K^{i+1} for three times, iteratively. Also, the subkey $sk^r (0 \leq r < R)$ is loaded from K^r. Similarly, the 64-bit state in K^r can be represented as a 4×16 array

$$K^r = \begin{bmatrix} K^r_{63} \cdots K^r_7 \; K^r_3 \\ K^r_{62} \cdots K^r_6 \; K^r_2 \\ K^r_{61} \cdots K^r_5 \; K^r_1 \\ K^r_{60} \cdots K^r_4 \; K^r_0 \end{bmatrix},$$

which will be loaded into the subkey

$$sk^r = K^r_{63} \cdots K^r_3 \| K^r_{62} \cdots K^r_2 \| K^r_{61} \cdots K^r_1 \| K^r_{60} \cdots K^r_0.$$

As shown in Figure 6, A_{16} is also nibble-oriented, with the 64-bit input X split into 16 nibbles $X[15] \| \cdots \| X[1] \| X[0]$. Then, the output is

$$(X[15] \lll t_1) \oplus X[0] \| X[15] \oplus (X[15] \ll t_0) \| X[14] \| X[13] \| \cdots \| X[2] \| X[1],$$

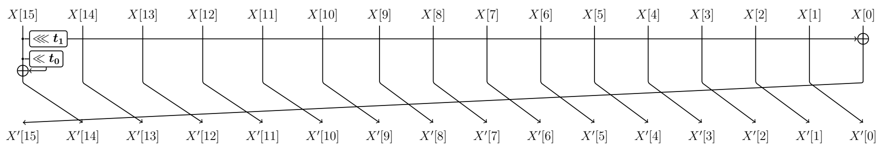

$X[15]$ $X[14]$ $X[13]$ $X[12]$ $X[11]$ $X[10]$ $X[9]$ $X[8]$ $X[7]$ $X[6]$ $X[5]$ $X[4]$ $X[3]$ $X[2]$ $X[1]$ $X[0]$
$\lll t_1$
$\lll t_0$
$X'[15]$ $X'[14]$ $X'[13]$ $X'[12]$ $X'[11]$ $X'[10]$ $X'[9]$ $X'[8]$ $X'[7]$ $X'[6]$ $X'[5]$ $X'[4]$ $X'[3]$ $X'[2]$ $X'[1]$ $X'[0]$

Figure 6: Operation A_{16} for SAND-128.

where t_0 and t_1 are set to 3 and 1, respectively.

To further help understanding, we present simple implementations of SAND from the bitslice view in Appendix B.

3 Design Rationales

3.1 The Designing of SAND

Before we detail the design rationale of SAND, we would like to share the motivations and design history of this new block cipher. We also make comparison to similar designs and highlight our novelties.

Motivations. During the past decade, NSA's SIMON family of block ciphers has been witnessed to achieve a huge leap regarding both software and hardware performances. Subsequently, Yang *et al.* [74] combined some features of SIMON and SPECK and designed SIMECK as a new security/efficiency tradeoff. The round function of SIMECK bears a strong resemblance with SIMON. As such, they were later referred to as SIMON-like ciphers.

Despite the performance breakthrough, a major drawback of SIMON-like ciphers is the (relative) difficulty in establishing security bounds against differential and linear attacks. A related design approach known as *ARX based ciphers* used to suffer from similar difficulties, which was however largely remedied by Dinu *et al.*'s *long trail design strategy* (LTS) [33]. This enabled them to present an efficient software-oriented lightweight block cipher SPARX with provable differential/linear security bounds. On the downside, the LTS advocates using large (ARX-based) S-boxes, which is imperfect for hardware. In view of these, it is natural to ask if similar methodological breakthroughs are feasible regarding AND-RX hardware-efficient designs.

Since Feistel networks are compatible with non-invertible round functions, we decide to leverage the (non-invertible) expand-then-compress design principle, which has been adopted by many designs, including SIMON and PICARO [58]. Compared to Feistel ciphers in which the state is not expanded, the expanding phase enlarges the space for evaluating the operations in parallel, which reduces latency and improves throughput. However, this expanding may also result in the *dependency problem*, which is caused by the same input for multiple non-linear operations. The situation is further complicated by the aforementioned security evaluation difficulty due to the use of AND-RX structure. To solve this problem, our idea is to *identify an approach to design AND-RX round functions, such that the round function has some equivalent (classical) S-box representations.* This will enable the use of the well-developed active S-box-based security evaluation methods and tools.

Besides easing security evaluations, two additional issues have to be carefully addressed. First, the impact of the dependency problem on the differential and linear properties shall be controllable. Second, the AND and XOR bit operations shall be carefully utilized for a competitive performance. With all the above motivations, we come up with the `SAND` family of block ciphers.

Comparison with `SIMON`-like ciphers. As mentioned before, both `SAND` and `SIMON`-like ciphers are combinations of AND-RX functions and Feistel network, and this allows for extremely efficient implementations in both hardware and software. However, significant differences emerge regarding security evaluations. In fact, the designers of `SIMON` did not provide any security analysis nor design rationale, and various sophisticated tools have been proposed by third-parties, which managed to bridge the gap several years later [47,49,50]. On the other hand, the novel design of `SAND` admits an equivalent representation using 4×8 S-boxes, and thus lends itself to *easy security evaluation* via the classical S-box-based tools. This enables establishing security lower bounds even against related-key attacks, which remains an open problem for `SIMON`. The deepened understanding also opens the way to using a *simpler* key schedule with *smaller* hardware area, compared to `SIMON`.

Compared to `SIMON`, the added XOR operations in each round of `SAND` indeed ensure a faster diffusion. In detail, `SAND`-64 and `SAND`-128 achieve full (bit level) diffusion after 7 and 9 rounds respectively,[5] while the corresponding number of rounds is 9 in `SIMON`-64 and 13 in `SIMON`-128 respectively. Again, we owe all these advantages to the carefully constructed round function.

Comparison with `ANT` and `ANU`. The block cipher `ANT` [28] is another AND-RX Feistel cipher. The round function of `ANT` bears strong similarity with `SAND`. However, in `ANT`, the rotations are *not* limited to be "within nibbles", which has an advantage in speed of diffusion. Therefore, `ANT` does not admit S-box-based representations, and the designers had to resort to (less efficient) bit-level automatic search and more conservative parameters. For example, for `ANT`-128/128, the designers were only able to ensure the absence of useful differential characteristics with more than 28 rounds, while we could draw similar conclusions with 23 rounds for `SAND`-128/128. The fine-grained security evaluation also allows using a simplified key schedule in `SAND` with reduced hardware area, compared to `ANT`.

`ANU` is another block cipher utilizing expand-then-compress style round functions [10]. However, `ANU` is not an AND-RX cipher: the round function of `ANU` appears more like a parallel application of two GOST-like round functions, that

[5] The diffusion test codes are available at https://github.com/sand-bar/SAND-Diffusion-Test.

is, the composition of a classical S-box layer and a rotation, and each nibble becomes the input to two parallel S-boxes. Unfortunately, the designers of `ANU` overlooked the dependency problem: they consider the differential properties of the two parallel S-boxes as *independent*, whereas they are actually *dependent* as they share the same input. Consequently, their security evaluation turns out inaccurate, and concrete full-round related-key attacks have been exhibited [60]. We remark that we overcome this via viewing the "two parallel S-boxes" as a single small-to-big synthetic S-box rather than two independent ones. As such, our theoretical evaluations are matched by experiments on reduced-scale round functions.

Comparison with other designs. One may notice that any cryptographic design can be represented by a composed of AND,Rotation and XOR operations, like the S-box-based designs `Piccolo`, `Midori`, `SKINNY`, `WARP` and etc, also the bitslice oriented designs `NOEKEON`, `KECCAK`, `ASCON`, `Rectangle` and etc. But, a representative feature that distinguishes our algorithm from these existing ciphers is the design approach. For a traditional design, its design flow is usually from top to bottom, briefly speaking, that is determining the overall structure of the round function first, then finding a non-linear layer and a linear layer with desired properties respectively. However, in our new design, the design flow is from bottom to top, that is we start with the expand-then-compress principle and exploit pure AND-RX bit operations, then construct the parallel non-linear layers by AND-RX bit operations and finally combine the rotation constants and permutations to pursue the equivalent transformation between AND-RX and S-box based structures. Indeed, any S-boxes can be developed by AND-RX operations. However, in most cases, the design of the S-box is a selection task among all or a part of permutations. After fixing an S-box with desired properties, the designers may represent the S-box with Boolean expressions for efficient software and hardware implementations. While in the design phase of `SAND`, we do not have any ready-made units for the non-linear layer and face with the target to tactfully organize various binary operations. Note that the linear and non-linear layers are closely related to each other. The wise selection of parameters in the algorithm enables us to transform the AND-RX cipher into an S-box based one.

As for the security analysis, our design possesses a native AND-RX structure, and it is commonly acknowledged that performing security evaluations against differential and linear attacks for this kind of ciphers is difficult. To overcome this difficulty, based on the parallel G_0 and G_1, we carefully select the rotation constants (α, β) and bit permutation P_n, which leads to a 4×8 S-box (SSb) under the equivalent transformation. Then we do some researches about this unusual 4×8 S-box and reveal some good properties (MDP and MALB) like the traditional 4×4 S-boxes, which makes it possible to take advantage of the traditional S-box-based cryptanalytic methods.

3.2 The Designing of the Round Function

As mentioned before, the round function of SAND is assembled with AND, Rotation and XOR operations. According to the principle of expand-then-compress, the left input branch is expanded into two branches so that the round function supports parallel confusion and diffusion. The two branches originating from the same input branch also bring us the dependency problem. Thus, how to arrange the simple operations so that we can explore the dependency caused by the expanding phase directly determines the level of resistance against differential and linear attacks.

As discussed in Section 2, the most important components of the round function are the non-linear functions G_0 and G_1. In this subsection, we first argue the structure of G_0 and G_1, for which the arrangements of AND and XOR operations are deliberately selected to ensure that the dependency can be easily controlled. After that, we illustrate how to integrate G_0,G_1 and thus reduce the AND-RX structure to an algorithm with several 4×8 S-boxes. Under this transformation, the SAND family of block ciphers exhibits excellent resistance against differential and linear attacks under a traditional active S-box-based method. At last, we discuss the selections of the rotation constants (α, β) and the parallel bit permutation $p_{\frac{n}{4}}$.

The structure of G_0 and G_1 composed of AND and XOR operations. The overall structures of G_0 and G_1 are demonstrated in Figure 7. Our design consideration starts by determining the number of operations in G_0 and G_1. Recall that the function operating on the left branch of SIMON-64/128 consists of 32 AND and 32 XOR operations. To minimize the cost and pursue a competitive performance, we consider employing the same number of AND operations in the round function of SAND, one half in G_0 and the other half in G_1 in order to maintain a balance. On the other hand, to achieve a better diffusion, we decide to increase the number of the XOR operations. Since the compression operation after G_0 and G_1 already takes 32 XOR operations, we limit the number of the XOR operations in G_0 and G_1 to be no more than 32. We then carefully consider

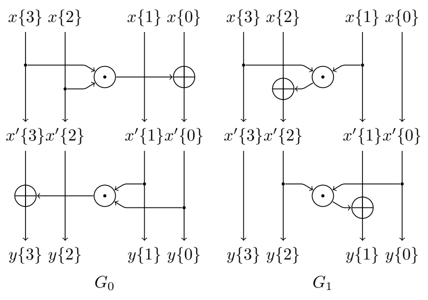


Figure 7: Overall structures of G_0 and G_1.

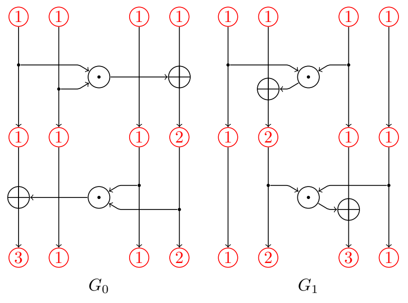


Figure 8: Algebraic degrees of G_0 and G_1.

organizing these operations and take the following issues into consideration:

- to address the dependency brought by the same input of G_0 and G_1, and
- to maximize the algebraic degree of the output branch after the XOR compression.

To ensure that none of the output bits of the compression operation has an algebraic degree less than 2, we employ two-layer structures for G_0 and G_1. In fact, the above constraints has significantly restricted the choices.

Now, we explain how to fix the operations one by one according to the above constraints. Denote the n-bit input of G_0 (resp., G_1) as $x = x\{3\}\|x\{2\}\|x\{1\}\|x\{0\}$. As shown in Figure 7, in the first layer of G_0, we may as well combine the information from $x\{3\}$ and $x\{2\}$ with AND operation. To make sure that each input bit is involved in one non-linear operation after G_0, we pass $x\{1\}$ and $x\{0\}$ to another AND operation in the second layer. Then, we study how to set the positions of the AND operations in G_1. Owing to the dependency between the inputs of G_0 and G_1, this should be accomplished with great care. As depicted in Figure 7, to avoid directly mixing $x\{3\}$ and $x\{2\}$ with the AND operation in G_1, we let $x\{3\}$ and $x\{1\}$ be the inputs of the AND operation in the first layer. Again, to ensure that each input bit is involved in a non-linear operation after G_1, we pass $x\{2\}$ and $x\{0\}$ to the AND operation in the second layer. After fixing the positions of the AND operations, we need to consider the arrangements of the XOR operations with respect to the algebraic degree, which will lead to the degree after the XOR compression of these two branches later and we want it to be equal or greater than 2. Please see Figure 8 for an illustration of the algebraic degrees (marked with red circles) of the outputs for G_0 and G_1.

We have been aware of some other choices of G_0 and G_1, which are however *equivalent* to our choice.

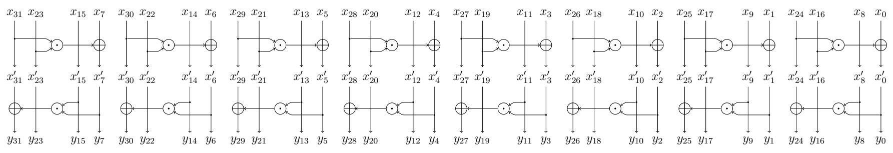


Figure 9: Equivalent representation of G_0 in SAND-64.

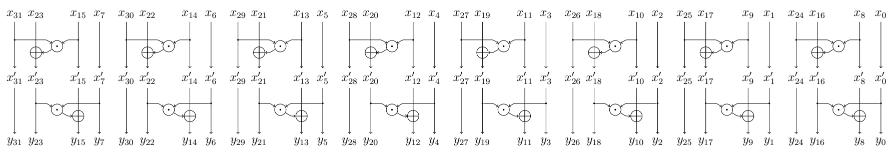


Figure 10: Equivalent representation of G_1 in SAND-64.

Combining G_0 and G_1 into multiple 4×8 S-boxes. The key issue is to divide G_0 and G_1 into different nibbles, and then combine the pair of nibbles in G_0 and G_1 with the same input 4-bit values, which is the origin of the dependency of the round function.

In detail, as the first step, we decompose G_0 (resp., G_1). Let $x = (x_{n-1}, \ldots, x_1, x_0)$ be the n-bit input of G_0 (resp., G_1). The (equivalent) decomposed representations of G_0 and G_1 in `SAND`-64 are shown in Figure 9 and Figure 10, respectively. It can be seen that G_0 and G_1 operate on $\frac{n}{4}$ nibbles in parallel. When the operations in G_0 and G_1 are considered, separately, the low costs of G_0 and G_1 are bound to be weak regarding differential and linear attacks. Naturally, the parallel computations on different nibbles can be formulated into the same 4-bit S-box. Denote the S-box abstracted from G_0 (resp., G_1) as N_0 (resp., N_1), and if we consider the same 4-bit input value, it will lead to the 4×8 *synthetic S-box*. All of them are listed in Table 5. In addition, to show the flexible structure of `SAND` block ciphers, we also provide the implementations from *SSb* view in Appendix B.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$N_0(x)$	0	1	2	b	4	5	6	f	8	9	a	3	d	c	7	e
$N_1(x)$	0	1	2	3	4	7	6	5	8	9	e	d	c	f	a	b
$SSb(x)$	00	11	22	b3	44	57	66	f5	88	99	ae	3d	dc	cf	7a	eb

Table 5: The S-boxes N_0, N_1 and SSb.

Naturally, we generate the differential distribution tables (DDT) and linear approximation tables (LAT) for N_0 and N_1, which can be found in Figure 11 and Figure 12, respectively, where Δ_{in} and Δ_{out} stand for the input and output differences, Γ_{in} and Γ_{out} represent input and output masks. Note that these DDTs and LATs present significant regularities on the one hand. On the other hand, the maximal differential probabilities of N_0 and N_1 achieve 2^{-1}, and the maximal absolute linear biases equal 2^{-1}. These facts reflect the weakness of N_0 and N_1. Nevertheless, as we mentioned before, even though the individual component in the round function of `SAND` is simple and weak at first glance, the composition of these components will result in enhanced level of security and exhibit excellent properties.

To this end, for each nibble in G_0 and G_1, we backtrace the same 4-bit input value, then we combine their outputs. By this way, we can obtain $\frac{n}{4}$ *synthetic S-boxes* with 4-bit input and 8-bit output. Then, we find that the MDP of *SSb* is 2^{-2}, and the MALB of *SSb* is also 2^{-2}. The detailed differential and linear properties of *SSb* will be respectively summarised in Lemma 1 and Lemma 2 later. Thus, the problem of dependency can be handled by using these *synthetic S-boxes*, whose input values are independent of each other. The realization of the round function of `SAND`-64 regarding these *SSb*s is demonstrated in Figure 13.

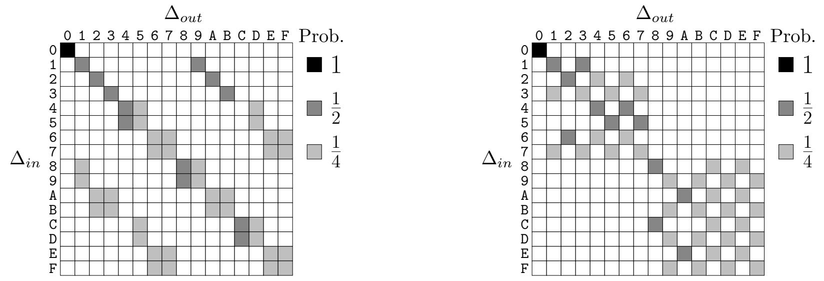


Figure 11: DDTs of N_0 (left) and N_1 (right)

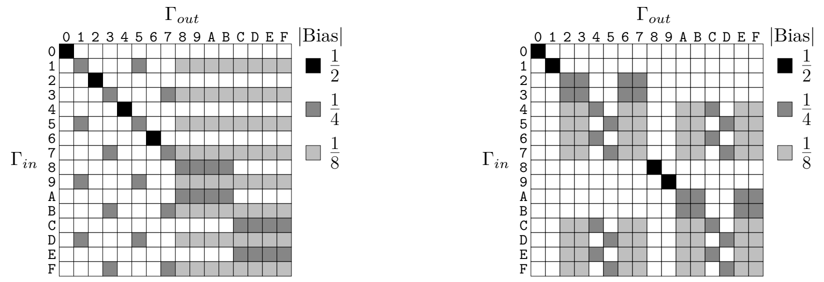


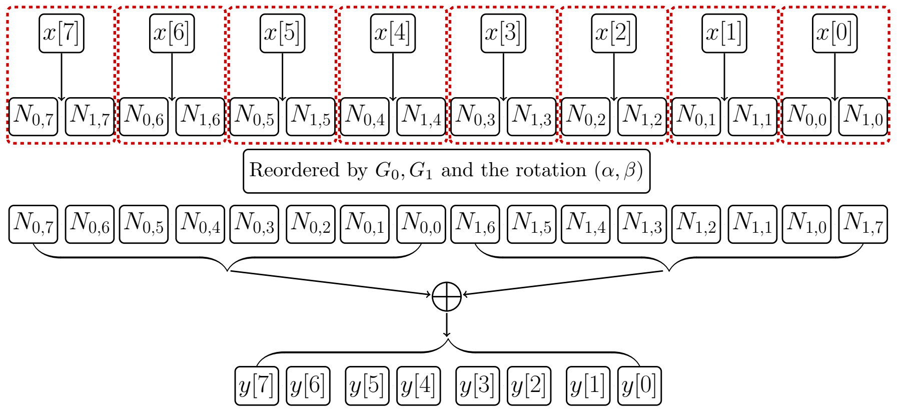
Figure 12: LATs of N_0 (left) and N_1 (right)

Note that here we consider the S-box structure, x represents the 32 bits from the input state of left branch, which can be divided into 8 nibble to pass N_0 and N_1 described above. The output y is also reordered under this S-box transformation and denotes the 32-bit value after the XOR compression. For the sake of simplicity, $N_{(0,j)}$ stands for the output of N_0 corresponding to $x[j]$ and $N_{(1,j)}$ stands for the output of N_1 regarding to $x[j]$ $(0 \leqslant j \leqslant 7)$. Let DDT_{SSb} denote the differential distribution table of SSb, which is provided in Appendix C, Table 14. Based on the observation from this DDT_{SSb}, we formulate the following Lemma 1.

Lemma 1. *For any valid*[6] *differential propagation pair $(\Delta_{in}, \Delta_{out})$ of SSb, if the 4-bit input difference Δ_{in} is non-zero, then it will result in an active S-box with MDP being 2^{-2} and leads to an 8-bit output difference Δ_{out}. Moreover, the high 4-bit value $\Delta_{out}[1]$ (most-significant nibble) and the low 4-bit value $\Delta_{out}[0]$ (least-significant nibble) of Δ_{out} are both non-zero.*

Similarly, let LAT_{SSb} denote the linear approximation table of SSb, we also generated LAT_{SSb} and summarized its features in Property 1-3 and Lemma 2.

[6] The valid means that the entry of corresponding differential propagation pair is non-zero in DDT. Similarly, for the linear introduced later, it means that the entry of corresponding linear propagation mask pair is non-zero in LAT.



$x[7]$
$x[6]$
$x[5]$
$x[4]$
$x[3]$
$x[2]$
$x[1]$
$x[0]$
$N_{0,7}$ $N_{1,7}$ $N_{0,6}$ $N_{1,6}$ $N_{0,5}$ $N_{1,5}$ $N_{0,4}$ $N_{1,4}$ $N_{0,3}$ $N_{1,3}$ $N_{0,2}$ $N_{1,2}$ $N_{0,1}$ $N_{1,1}$ $N_{0,0}$ $N_{1,0}$
Reordered by G_0, G_1 and the rotation (α, β)
$N_{0,7}$ $N_{0,6}$ $N_{0,5}$ $N_{0,4}$ $N_{0,3}$ $N_{0,2}$ $N_{0,1}$ $N_{0,0}$ $N_{1,6}$ $N_{1,5}$ $N_{1,4}$ $N_{1,3}$ $N_{1,2}$ $N_{1,1}$ $N_{1,0}$ $N_{1,7}$
$y[7]$ $y[6]$ $y[5]$ $y[4]$ $y[3]$ $y[2]$ $y[1]$ $y[0]$

Figure 13: Representation of the round function of `SAND`-64 regarding SSb (circled with red rectangles) .

Act($\mathit{\Gamma}_{in}$)	Act($\mathit{\Gamma}_{out}$)
0	00
1	01
1	10
0 or 1	11

Table 6: All possible patterns of input and output masks for SSb.

Besides, the active status of all possible propagations of LAT_{SSb} are listed in Table 6.

Property 1. For any given valid mask pair $(\Gamma_{in}, \Gamma_{out})$, if the 8-bit output mask Γ_{out} is `0x00`, then the input mask Γ_{in} should be `0x0`. This case corresponds to a non-active S-box (with the linear bias being 2^{-1}).

Property 2. For any given valid mask pair $(\Gamma_{in}, \Gamma_{out})$, if the 4-bit input mask Γ_{in} is non-zero, then the 8-bit output mask Γ_{out} must be non-zero.

Property 3. For any given valid mask pair $(\Gamma_{in}, \Gamma_{out})$, if the 4-bit input mask Γ_{in} is `0x0`, then the output mask Γ_{out} satisfies $\mathrm{Act}(\Gamma_{out}) = 00/11$.

Lemma 2. *For any valid linear propagation mask pair $(\Gamma_{in}, \Gamma_{out})$ of SSb, the corresponding non-active transitions are listed in Table 7. Otherwise, it must be an active S-box with MALB being 2^{-2}.*

Γ_{in}	Γ_{out}
0x0	0x00
0x1	0x01
0x8	0x08
0x9	0x09
0x2	0x20
0x3	0x21
0xA	0x28
0xB	0x29
0x4	0x40
0x5	0x41
0xC	0x48
0xD	0x49
0x6	0x60
0x7	0x61
0xE	0x68
0xF	0x69

Table 7: All non-active linear transitions of SSb.

Big S-box method vs small S-box method. Now, the round function is transformed into several small S-boxes SSb, and we can directly treat the round function of `SAND` as a black-box (precisely, we just consider the function operating on the left branch of the Feistel round without permutation P_n, please see Figure 13), which is called big S-box here, its differential distribution table $\mathrm{DDT}_{\mathrm{big}}$ (resp., linear approximation table $\mathrm{LAT}_{\mathrm{big}}$) contains all the information about differential (resp., linear) transitions. Simultaneously, we can also use DDT_{SSb} (resp., LAT_{SSb}) of these small S-boxes to derive $\mathrm{DDT}_{\mathrm{big}}$ (resp., $\mathrm{LAT}_{\mathrm{big}}$). As

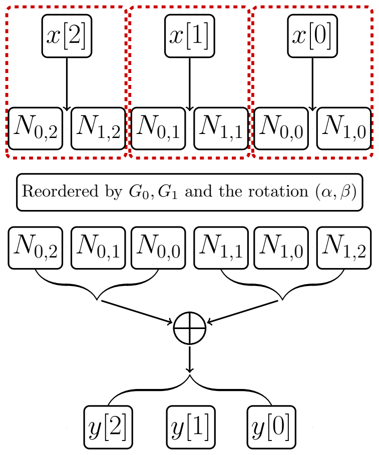


Figure 14: 12-bit small variant of the round function of `SAND`.

the input values for these parallel SSbs are divided into independent nibbles, their outputs will XOR together due to the compression, and the propagations of these nibbles follow the rule in DDT_{SSb} (resp., LAT_{SSb}). Thus, the following two properties can be got,

Property 4. Let (Δ_x, Δ_y) be the n-bit input and output differences of the big S-box, then the number of SSbs is $\frac{n}{4}$ and we have,

$$\mathrm{DDT}_{big}[\Delta_x][\Delta_y] = \sum_{\Delta_z^{high} \oplus (\Delta_z^{low} \lll 4) = \Delta_y} \prod_{0 \le i < \frac{n}{4}} \mathrm{DDT}_{SSb}[\Delta_x[i]][\Delta_z^i],$$

where $\Delta_x[i]$ and Δ_z^i are the i-th SSb's 4-bit input and 8-bit output difference respectively, $\Delta_z^{high} = \Delta_z^{\frac{n}{4}-1}[1] || \cdots || \Delta_z^0[1]$ and $\Delta_z^{low} = \Delta_z^{\frac{n}{4}-1}[0] || \cdots || \Delta_z^0[0]$.

Property 5. Let (Γ_x, Γ_y) be the n-bit input and output masks of the big S-box, then the number of SSbs is $\frac{n}{4}$ and we have,

$$\mathrm{LAT}_{big}[\Gamma_x][\Gamma_y] = 2^{\frac{n}{4}-1} \cdot \prod_{0 \le i < \frac{n}{4}}^{\Gamma_y = \Gamma_z^{high} = (\Gamma_z^{low} \lll 4)} \mathrm{LAT}_{SSb}[\Gamma_x[i]][\Gamma_z^i],$$

where $\Gamma_x[i]$ and Γ_z^i are the i-th SSb's 4-bit input and 8-bit output masks respectively, $\Gamma_z^{high} = \Gamma_z^{\frac{n}{4}-1}[1] || \cdots || \Gamma_z^0[1]$ and $\Gamma_z^{low} = \Gamma_z^{\frac{n}{4}-1}[0] || \cdots || \Gamma_z^0[0]$.

For these two methods, the latter one allows us to use these small S-boxes to describe the differential (resp. linear) behavior of the big S-box, which is critical for the efficiency of the search model.

Also, as verifications for the Property 4 and Property 5, we perform the tests on the small variants of the round function, including three SSbs with 12-bit input and 12-bit output (depicted in Figure 14) and four SSbs with 16-bit input and 16-bit output. It should be noted that for the round function with 16-bit scale, we perform the tests of differential and linear, partially. The results of these two variants show that DDTs and LATs generated by the two methods are completely the same, which exhibits the accuracy of our model. The verifications of these small variants are provided at `https://github.com/sand-bar/SAND-Synthetic-Sbox`.

The selections of the rotation constants (α, β) **and bit permutation** $p_{\frac{n}{4}}$**.** The rotation constants (α, β) are used to mix the input bits within a word, and the parallel bit permutation $p_{\frac{n}{4}}$ aims at breaking the order of output bits within a word. Due to the symmetry, we may as well let $\beta > \alpha$.

For SAND-64, benefiting from the efficient S-box-based estimation of differential pattern model (at nibble level), which does not need to consider the detailed DDT of SSb, we can perform exhaustive search for all pairs of rotation constants ($C_8^2 = 28$) and all 8-bit permutations ($8! = 40320$), which in total have $C_8^2 \times 8! \approx 2^{20.11}$ combinations. Under each combination, we evaluate the number of rounds activating at least 32 differential active S-boxes to get some candidates. Then, we consider the linear property of these candidates by evaluating active S-boxes under the bit level model, which has to describe the detailed LAT of SSb. Finally, we get $(\alpha, \beta) = (0, 1)$ and $p_8 = \{7, 4, 1, 6, 3, 0, 5, 2\}$.

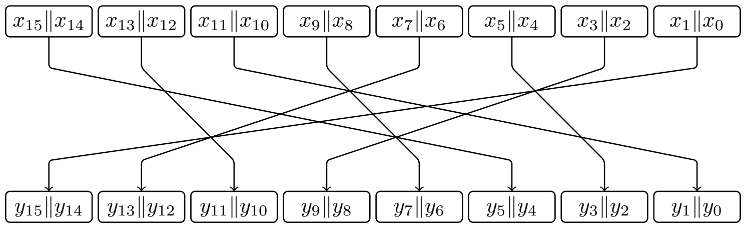


Figure 15: An illustration of p_{16} operating on 16-bit value x.

For SAND-128, the number of options for p_{16} is $16! \approx 2^{44.25}$, which is far beyond our computing power. Thus, we still utilize the permutation p_8 in SAND-64 and extend it into a 16-bit permutation. To be more precise, we regard two adjacent bits in a word as one element and apply p_8 on the word, which is depicted in Figure 15. In this way, we obtain p_{16} for SAND-128 (see Table 4). Even though this p_{16} may not be the optimal choice, the analysis results show that this selection guarantees good security bounds for SAND-128.

3.3 The Designing of the Key Schedule

Since we aim at achieving related-key security within a hardware-efficient design, the key schedule should meet the following three basic requirements:

– Consume the hardware area as small as possible;
– Spread the master key material to each subkey as much as possible;
– Keep the delay of the key schedule as short as possible.

It is necessary to strike a balance between the three aspects above. So, the primary idea is to use a linear key schedule, which is more and more popular in recent designs, like `GIFT` and `SKINNY`. Within the linear key schedule, its update function can make use of round-based Feistel structure since this kind of design is able to save the hardware implementation cost by sacrificing some delay but still guarantees a good diffusion for the entire key. This design idea for the key schedule is first proposed in `CUBE` [17], an SPN based cipher. In this paper, we generalize this design principle so that it fits the two versions of `SAND` and supplies priority to hardware performance—area and delay. In this direction, we first set $t_0 = 3$ (for the shift operation in a nibble, this parameter will just leave one bit for the subsequent XOR operation). Then, we let $t_1 = 1$ and the XOR operation after the left rotation is positioned at the rightmost nibble, which are suitable for diffusion. Several positions of the XOR after t_0 shift operation are tested, considering the resistance against related-key attack, we just let the XOR go to the leftmost nibble ($X[7]$ for `SAND`-64 and $X[15]$ for `SAND`-128). We also perform the propagation analysis of the master key material by doing algebraic calculations, the detailed results are provided in `https://github.com/sand-bar/SAND-Diffusion-Test`. Compared to the linear key schedules of `SIMON`, the ones adopted by `SAND` consume less hardware area cost but still achieve good diffusion effect. To avoid possible circular shift symmetries and slide attacks, we introduce a 6-bit counter as the round constant. All these efforts lead to the final key expanding strategies.

4 Security Analysis

In this section, we present our security analysis of the `SAND` family. It should be noted that our security evaluation leverages the S-box-based method. Thus, as elaborated in Section 3.2, the bit in the state variables of the trails are represented in a 4-bit S-box order thereinafter, rather than the bitslice order described in Section 2.

4.1 Security Claim

For the claimed security, we do not claim any security in the chosen-key or known-key settings, but we do claim security in the single-key and related-key settings.

Considering the provided analyses in this section, the related-key attacks seems be the most promising cryptanalysis on `SAND`, as the effective related-key differential characteristics can be upper bounded by 28 rounds (`SAND`-64/128) and 33 rounds (`SAND`-128/128), respectively. For the single-key setting, we present the security comparisons of `SAND` and some other block ciphers, which are listed in Table 8.

Cipher	Rounds for Full Diffusion	Best Distinguisher for Single-key Setting	Best Attack for Single-key Setting[†]	Total Rounds
SAND-64/128	7	15[‡]	-	48
SIMON-64/128	9	23 [49,51]	31 [27,71]	44
SIMECK-64/128	11	30 [48]	42 [48]	44
CRAFT-64/128	7	15 [36]	19 [36]	32
SAND-128/128	9	23[‡]	-	54
SIMON-128/128	13	42 [48]	53 [48]	68

† We list the corresponding best attacks of each cipher in terms of the covering attack rounds until now, which may be extended in the future.

‡ The upper bound of the rounds of the effective differential or linear characteristic.

- Although, the concrete attack is not mounted for SAND in this paper, its best attack rounds can be roughly estimated by the summation of the rounds for the best distinguisher and full diffusion.

Table 8: Security comparisons under the single-key setting.

Hence, considering the total rounds, we claim 128-bit security for SAND in both the single-key and related-key models.

4.2 Differential and Linear Attacks

Differential cryptanalysis [21] (DC) and linear cryptanalysis [54] (LC) are the most classic cryptanalytic methods. The resistance of a given cipher against differential and linear attacks can be reflected by the lower bounds for the number of active S-boxes to a certain degree.

By using Lemma 1, we can perform the differential active S-box estimation for SAND under the efficient differential pattern model. Regarding linear cryptanalysis, due to the features of SSb according to Table 6, Table 7 and Lemma 2, we count the number of active S-boxes by describing the detailed LAT of SSb into the search model. The computation of the minimum number of active S-boxes utilized the automatic methods in [55,67]. These are commonly used tools in the cryptanalysis of block ciphers, and there are several openly available tools (e.g., CryptoSMT [66]) facilitating the application. In the search around SAND, we employ different methods in different settings for efficiency. For the differential pattern search model, we use Mouha *et al.*'s [55] framework for word-oriented distinguishers. We utilize the SAT/SMT based automatic search method [67] to accomplish the search of (related-key) differential characteristics and linear characteristics.

With automatic search methods relying on SMT/SAT,[7] for two versions of SAND, we get the minimum number of active S-boxes in differential and linear

[7] We provide our source codes in https://github.com/sand-bar/SAND-Trail-Search to serve more details of these searching models, which is based on [66,2,38].

trails for different numbers of rounds, which are summarised in Table 9 and Table 10.

Rounds	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DC	1	2	4	6	9	12	16	18	20	22	24	26	29	32	34
LC	1	2	4	6	9	12	15	18	20	22	24	26	29	32	34

Table 9: Minimum number of active S-boxes in `SAND`-64 under single-key setting.

Rounds	2	3	4	5	6	7	8	9	10	11	12
DC	1	2	4	6	9	12	16	20	25	30	34
LC	1	2	4	6	9	12	16	18	25	30	34

Table 10: Minimum number of active S-boxes in `SAND`-128 under single-key setting.

It can be observed in Table 9 that 15-round differential and linear characteristics of `SAND`-64 have at least 32 active S-boxes. 30-round differential/linear characteristics will activate 64 S-boxes at least, and these trails will not threaten the security of `SAND`-64, essentially. Thus, we believe that 48-round `SAND`-64 is enough to withstand DC and LC under the single-key attack scenario. Additionally, some example trails and experiments of differential and linear properties (like clustering for differential and linear hull effect) for `SAND`-64 are given in Appendix D and Appendix E, respectively.

Regarding the security of `SAND`-128, it can be observed in Table 10 that 23-round differential trail will have at least 64 active S-boxes since the optimal 11-round (resp., 12-round) trail activates at least 30 (resp., 34) S-boxes. A similar analysis reveals that 23-round linear characteristic has at least 64 active S-boxes. Therefore, we believe that 54-round `SAND`-128 is sufficient to provide resistance against DC and LC under the single-key attack scenario.

Remark. The bound of 2^{-64} for `SAND`-64 is reached after 15 rounds, and the 2^{-128} bound for `SAND`-128 is reached after 23 rounds. As for `SIMON`, the single trail of `SIMON`-64 attains the bound 2^{-64} with 19 rounds [47], and the trail of `SIMON`-128 touches the bound 2^{-128} with 37 rounds [49,50].

4.3 Related-Key Differential Attacks

Related-key attacks [44,19] usually outperform the conventional differential attacks due to attackers' additional power to manipulate the keys. To ensure security against related-key differential attacks, we use the SMT/SAT-based automatic method to determine the lower bound for the number of active S-boxes under related-keys, with results in Table 11.

Rounds	4	5	6	7	8	9	10	11	12	13	14
SAND-64	0	0	1	2	3	4	6	8	11	14	≥ 16
SAND-128	1	2	3	4	9	12	17	22	-	-	-

Table 11: Minimum number of active S-boxes in SAND under related-key setting.

For SAND-64, since the first four subkeys are extracted from the master key, directly, it is trivial to see that it is possible to have no active S-boxes from 1-round to 4-round. From Table 11, we know that 14-round encryption will activate at least 16 active S-boxes, and thus the probability of the optimal 14-round related-key differential characteristic is upper bounded by 2^{-32}. Therefore, there is no effective related-key differential characteristic exceeding 27 rounds and 48-round SAND-64/128 is enough to resist related-key differential attack.

For SAND-128, the result in Table 11 indicates that there is no 11-round related-key differential characteristic with probability higher than 2^{-44}. Thus, we expect that the probability of the optimal 33-round characteristic is upper bounded by 2^{-132} and 54-round SAND-128/128 is sufficient to provide resistance against related-key differential attack.

4.4 Impossible Differential and Zero-Correlation Attacks

Impossible differential cryptanalysis [20,45] (IDC) utilizes a pair of differences $(\Delta_{in}, \Delta_{out})$, and Δ_{in} never propagates to Δ_{out} under the given algorithm. The similar idea adapted in linear cryptanalysis develops into zero-correlation cryptanalysis [23] (ZC). The resistance against these two attacks is generally explored by the longest distinguisher we can discover. Several automatic methods targeting the search of impossible differential and zero-correlation distinguishers are available [61,29].

In this paper, we exploit the method in [29] and fix the number of active S-boxes in the input and output differences/masks as one, then perform an exhaustive search at nibble level, i.e., all the 4-bit variables of the states are replaced with the value of its characteristic function (Note this is also done under the S-box view). For SAND-64, the longest impossible differential and zero-correlation distinguishers identified with the model both achieve 10 rounds, and we just select two of them listed as follows

$$(\texttt{0x00}, \texttt{0x80}) \overset{\text{10R-IDC}}{\nrightarrow} (\texttt{0x80}, \texttt{0x00}),$$
$$(\texttt{0x80}, \texttt{0x00}) \overset{\text{10R-ZC}}{\nrightarrow} (\texttt{0x00}, \texttt{0x80}).$$

For SAND-128, the longest impossible differential and zero-correlation distinguishers both achieve 14 rounds, which are appended in the following as an example

$$(\mathtt{0x0000}, \mathtt{0x8000}) \overset{\text{13R-IDC}}{\nrightarrow} (\mathtt{0x0040}, \mathtt{0x0000}),$$
$$(\mathtt{0x8000}, \mathtt{0x0000}) \overset{\text{13R-ZC}}{\nrightarrow} (\mathtt{0x0000}, \mathtt{0x2000}).$$

Although longer distinguishers may be detected with a more accurate model, e.g., a more precise model at the bit level, we think that according to the current results, both versions of SAND can withstand impossible differential and zero-correlation attacks.

4.5 Integral Attacks

A popular method to estimate the resistence against integral cryptanalysis [46] is to evaluate the division property [69] of a given cipher. Thus, we use the bit-based division property[70] to evaluate the algebraic degree. Several automatic tools for the search of division property are available. In this paper, we apply the model in [73] to trace the propagation of bit-based division property of SAND. From the test result, we find that the longest integral distinguisher for SAND-64 achieves 12 rounds, which is

$$(C^1 A^{31}, A^{32}) \xrightarrow{\text{12R}} (U^{32}, U^1 B^1 U^{30}),$$

where 'A^i', 'B^i', 'C^i', 'U^i' respectively stand for i consecutive active, zero-sum, constant and unknown bits. The longest integral distinguisher for SAND-128 reaches 16 rounds,

$$(C^1 A^{63}, A^{64}) \xrightarrow{16R} (U^{64}, U^5 B^1 U^{58}).$$

Considering the total rounds for SAND, our design is secure against integral attacks.

4.6 Rotational-XOR Attacks

Rotational-XOR (RX) cryptanalysis [3] is a related-key attack targeting ARX ciphers, which is a generalization of rotational cryptanalysis [43]. Recently, RX cryptanalysis is extended to AND-RX ciphers [53]. So, we analyze the propagations of RX-differences through an AND-RX round (not including the permutation P_n) and the key schedule of SAND. As a result, the propagation probability through an RX-difference for the round function is the same as that of a normal XOR-difference. In other words, the DDTs generated by these two methods are same, which is verified by performing tests on 12-bit small variant mentioned before. As for the key schedule, due to the iterative Feistel linear operation (A_8 or A_{16}), it will produce probability on the key space for each round (at least 2^{-8} for the different rotation choice). Thus, RX attacks will not threaten the security of SAND.

4.7 Meet-in-the-Middle Attacks

Meet-in-the-Middle (MITM) attacks [32] have been applied to the security analysis of many block ciphers. As has been discussed in Section 3.3, the key schedule of SAND is designed to spread master key bits to each subkey as much as possible. Considering the rounds for full diffusion, many master key bits will be processed into the round function quickly, we think that all versions of SAND have a good resistance against MITM attacks.

5 Hardware Implementations

In order to compare the hardware performance of SAND with that of other lightweight block ciphers fairly, we implemented all these lightweight designs in Verilog HDL using an ASIC round-based architecture, which is a frequently used hardware implementation evaluation and can exhibit the top performance criterion of a target cipher in general.

With regard to the synthesis, we used the Synopsys Design Compiler (version J-2014.09-SP3) with the standard cell library of TSMC 90nm logic process. Note that all hardware realizations were synthesized with the compiler being specifically instructed to optimize the circuit of area. For a better and fairer comparison, the throughput at a maximally achievable frequency and at a frequency of 100 KHz are both provided. The power consumption is simulated at a frequency of 10 MHz.

The results of hardware performance of SAND and other lightweight block ciphers are summarized in Table 12. For SAND-64/128, it has the smallest area requirement compared to other lightweight designs. For SAND-128/128, when considering the related-key security, only SKINNY-128/128 can compete with our design in terms of area for the encryption-only implementation. But SAND-128/128 has a smaller critical path, which means it can be operated at higher frequencies and provides better throughput. For more details of threshold implementation [56,57], in Appendix F, we discuss the adaptability of SAND to a popular (hardware-oriented) side-channel countermeasure.

6 Software Implementations

In the process of designing, we also tried to equip SAND with some features suitable for software implementations. For the AND-RX operations and inherent bitslice structure of SAND, we naturally consider bitslice implementations in this section, which is the most efficient software implementation of SAND. From the six device/server use cases for lightweight encryption summarized in [16], bitslice implementations are very common and can be used for any parallel mode.

To ensure the fairness of the comparison, we consider the costs of packing and unpacking the data. When doing the benchmarks, the encryption is given with the prepared subkeys and using the CTR mode. We realized the implementations

Ciphers	Type	Area (GE)	Delay (ns)	Cycles	TP_{100KHz} (Kbps)	TP_{Max} (Mbps)	$Power_{10Mhz}$ (μW)	Energy (pJ)
SAND-64/128	Enc	1287	0.86	48	133.3	1550.4	32.97	158.26
	Enc&Dec	1563	0.91	48	133.3	1465.2	37.21	178.61
SKINNY-64/128[†]	Enc	1306	1.63	36	177.8	1090.7	33.36	120.10
	Enc&Dec	1916	2.28	36	177.8	779.7	42.62	153.43
TWINE-64/128[*]	Enc	1389	1.51	36	177.8	1177.3	34.22	123.19
	Enc&Dec	1687	1.58	36	177.8	1125.2	39.98	143.93
SIMECK-64/128	Enc	1300	0.69	44	145.5	2108.0	33.03	145.33
	Enc&Dec	1714	0.83	44	145.5	1752.5	39.26	172.74
SIMON-64/128	Enc	1329	0.75	44	145.5	1939.4	33.59	147.80
	Enc&Dec	1779	0.84	44	145.5	1731.6	39.49	173.76
CRAFT-64/128[⋆]	Enc	1316	0.70	32	200.0	2857.1	30.46	97.47
	Enc&Dec	1618	1.00	32	200.0	2000.0	39.68	126.98
ANU-64/128[*]	Enc	1460	1.66	25	256.0	1542.2	34.17	85.43
	Enc&Dec	1835	2.44	25	256.0	1049.2	39.75	99.38
SAND-128/128	Enc	1874	0.88	54	237.0	2693.6	45.33	244.78
	Enc&Dec	2264	0.88	54	237.0	2693.6	47.76	257.90
SKINNY-128/128	Enc	1849	2.12	40	320.0	1509.4	46.18	184.72
	Enc&Dec	2973	2.31	40	320.0	1385.3	60.92	243.68
WARP-128/128	Enc	1632	0.75	41	312.2	4162.6	39.47	161.83
	Enc&Dec	1775	0.75	41	312.2	4162.6	42.92	175.97
ANT-128/128	Enc	1898	0.85	46	278.3	3273.7	45.58	209.67
	Enc&Dec	2284	0.88	46	278.3	3162.1	48.50	223.10
SIMON-128/128	Enc	1930	0.78	68	188.2	2413.3	44.82	304.78
	Enc&Dec	2535	0.84	68	188.2	2240.9	52.65	358.02

[†] SKINNY-64/128 is implemented with the tweakey schedule.
[*] The optimized S-box hardware implementation is adopted with the help of the tool LIGHTER proposed in [42].
[⋆] CRAFT is implemented without tweak part and in dynamic key mode, that is considering the key storage.

Table 12: Comparison of hardware performance for round-based implementations, synthesized with TSMC 90nm standard cell library under area optimization.

of SIMON and SAND with AVX2 registers and also compared to SIMON-SPECK-SUPERCOP [52]. All benchmarking results can be found in Table 13 and all the tests are accomplished on a PC with Intel Skylake processor (i7 6700) and 8GB 2400 MHz DDR4. It can be observed that the software performance of SAND is very efficient and reaches the same level of SIMON. Since both implementations may still be improved in the future, we provide our codes in https://github.com/sand-bar/SAND-Software.

Block size	**Key size**	**Parallel n-way**	**Speed (cycle per byte)**		
			SAND	SIMON	SIMON [52]
64	128	64	1.93	1.79	2.27
128	128	32	2.34	2.49	3.45

Table 13: Comparisons of software performance under bitslice implementations with 2MB of messages.

7 Conclusion

In this paper, we presented a new AND-RX based Feistel lightweight block cipher—SAND but supporting an S-box-based cryptanalysis. Benefiting from this novel and flexible S-box transformation in the security evaluation, SAND reaches strong security level under both single-key and related-key scenarios.

Actually, the S-box transformation is motivated by pursing an easier active S-box-based security analysis and covering the dependency of the round function, which is caused by the expanding process. When we bridge the components in the round function together, it will lead us to the *synthetic S-box* with good differential and linear properties. Thus, the AND-RX based design can be regarded as the cipher with several 4×8 S-boxes. This design idea, we believe, provides a new way for the designing of AND-RX based cipher in the future, that also seeks for an easier S-box-based cryptanalysis.

Compared to the current efficient lightweight block ciphers, SAND also achieves a competitive hardware performance for ASIC round-based implementations. For software performance, the natural bitslice structure and AND-RX operations allow SAND to reach the similar performance as SIMON, which is one of the most efficient lightweight primitives on software.

Acknowledgements

The authors would like to thank the anonymous reviewers and François-Xavier Standaert for their helpful comments.

References

1. Andreeva, E., Lallemand, V., Purnal, A., Reyhanitabar, R., Roy, A., Vizár, D.: ForkAE v.1. In: Submission to Round 2 of the NIST Lightweight Cryptography Standardization process (2020)
2. Ankele, R., Kölbl, S.: Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In: Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. pp. 163–190 (2018), https://doi.org/10.1007/978-3-030-10970-7_8
3. Ashur, T., Liu, Y.: Rotational cryptanalysis in the presence of constants. IACR Trans. Symmetric Cryptol. 2016(1), 57–70 (2016), https://doi.org/10.13154/tosc.v2016.i1.57-70
4. Avanzi, R.: The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. IACR Trans. Symmetric Cryptol. 2017(1), 4–44 (2017), https://doi.org/10.13154/tosc.v2017.i1.4-44
5. Banik, S., Bao, Z., Isobe, T., Kubo, H., Liu, F., Minematsu, K., Sakamoto, K., Shibata, N., Shigeri, M.: WARP : Revisiting GFN for lightweight 128-bit block cipher. IACR Cryptol. ePrint Arch. 2020, 1320 (2020), https://eprint.iacr.org/2020/1320
6. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II. pp. 411–436 (2015), https://doi.org/10.1007/978-3-662-48800-3_17
7. Banik, S., Bogdanov, A., Peyrin, T., Sasaki, Y., Sim, S.M., Tischhauser, E., Todo, Y.: SUNDAE-GIFT v1.0. In: Submission to Round 2 of the NIST Lightweight Cryptography Standardization process (2020)
8. Banik, S., Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT-COFB v1.0. In: Finalists of the NIST Lightweight Cryptography Standardization process (2021)
9. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. pp. 321–345 (2017), https://doi.org/10.1007/978-3-319-66787-4_16
10. Bansod, G., Patil, A., Sutar, S., Pisharoty, N.: ANU: an ultra lightweight cipher design for security in IoT. Security and Communication Networks 9(18), 5238–5251 (2016)
11. Baysal, A., Sahin, S.: RoadRunneR: A small and fast bitslice block cipher for low cost 8-bit processors. In: Lightweight Cryptography for Security and Privacy - 4th International Workshop, LightSec 2015, Bochum, Germany, September 10-11, 2015, Revised Selected Papers. pp. 58–76 (2015), https://doi.org/10.1007/978-3-319-29078-2_4
12. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptology ePrint Archive 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., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. pp. 123–153 (2016), https://doi.org/10.1007/978-3-662-53008-5_5
14. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: SKINNY-AEAD and SKINNY-Hash v1.1. In: Submission to Round 2 of the NIST Lightweight Cryptography Standardization process (2020)
15. Beierle, C., Leander, G., Moradi, A., Rasoolzadeh, S.: CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. IACR Trans. Symmetric Cryptol. 2019(1), 5–45 (2019), https://doi.org/10.13154/tosc.v2019.i1.5-45
16. Benadjila, R., Guo, J., Lomné, V., Peyrin, T.: Implementing lightweight block ciphers on x86 architectures. In: Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers. pp. 324–351 (2013), https://doi.org/10.1007/978-3-662-43414-7_17
17. Berger, T.P., Francq, J., Minier, M.: CUBE cipher: A family of quasi-involutive block ciphers easy to mask. In: Codes, Cryptology, and Information Security - First International Conference, C2SI 2015, Rabat, Morocco, May 26-28, 2015, Proceedings - In Honor of Thierry Berger. pp. 89–105 (2015), https://doi.org/10.1007/978-3-319-18681-8_8
18. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 313–314. Springer (2013)
19. Biham, E.: New types of cryptanalytic attacks using related keys. J. Cryptology 7(4), 229–246 (1994), https://doi.org/10.1007/BF00203965
20. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding. pp. 12–23 (1999), https://doi.org/10.1007/3-540-48910-X_2
21. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptology 4(1), 3–72 (1991), https://doi.org/10.1007/BF00630563
22. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings. pp. 450–466 (2007), https://doi.org/10.1007/978-3-540-74735-2_31
23. Bogdanov, A., Rijmen, V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. Des. Codes Cryptogr. 70(3), 369–383 (2014), https://doi.org/10.1007/s10623-012-9697-z
24. 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., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings. pp. 208–225 (2012), https://doi.org/10.1007/978-3-642-34961-4_14

25. Chakraborti, A., Datta, N., Jha, A., Lopez, C.M., Nandi, M., Sasaki, Y.: LOTUS-AEAD/LOCUS-AEAD. In: Submission to Round 2 of the NIST Lightweight Cryptography Standardization process (2020)
26. Chakraborti, A., Datta, N., Jha, A., Nandi, M.: HYENA. In: Submission to Round 2 of the NIST Lightweight Cryptography Standardization process (2020)
27. Chen, H., Wang, X.: Improved linear hull attack on round-reduced simon with dynamic key-guessing techniques. In: Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. pp. 428–449 (2016), https://doi.org/10.1007/978-3-662-52993-5_22
28. Chen, S., Fan, Y., Fu, Y., Huang, L., Wang, M.: On the design of ANT family block ciphers. Journal of Cryptologic Research 6(6), 748–759 (2019)
29. Cui, T., Jia, K., Fu, K., Chen, S., Wang, M.: New automatic search tool for impossible differentials and zero-correlation linear approximations. IACR Cryptology ePrint Archive 2016, 689 (2016), http://eprint.iacr.org/2016/689
30. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie proposal: NOEKEON. In: First Open NESSIE Workshop. pp. 213–230 (2000)
31. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002), https://doi.org/10.1007/978-3-662-04722-4
32. Diffie, W., Hellman, M.E.: Special feature exhaustive cryptanalysis of the NBS data encryption standard. IEEE Computer 10(6), 74–84 (1977), https://doi.org/10.1109/C-M.1977.217750
33. Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., Biryukov, A.: Design strategies for ARX with provable bounds: Sparx and LAX. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. pp. 484–513 (2016), https://doi.org/10.1007/978-3-662-53887-6_18
34. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon. submission to the caesar competition (2014) (2014)
35. ElSheikh, M., Youssef, A.M.: Related-key differential cryptanalysis of full round CRAFT. In: Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings. pp. 50–66 (2019), https://doi.org/10.1007/978-3-030-35869-3_6
36. Guo, H., Sun, S., Shi, D., Sun, L., Sun, Y., Hu, L., Wang, M.: Differential attacks on CRAFT exploiting the involutory s-boxes and tweak additions. IACR Trans. Symmetric Cryptol. 2020(3), 119–151 (2020), https://doi.org/10.13154/tosc.v2020.i3.119-151
37. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED block cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. pp. 326–341 (2011), https://doi.org/10.1007/978-3-642-23951-9_22
38. Hadipour, H., Sadeghi, S., Niknam, M.M., Song, L., Bagheri, N.: Comprehensive security analysis of CRAFT. IACR Trans. Symmetric Cryptol. 2019(4), 290–317 (2019), https://doi.org/10.13154/tosc.v2019.i4.290-317
39. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Remus v1.0. In: Submission to Round 1 of the NIST Lightweight Cryptography Standardization process (2019)
40. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Romulus v1.2. In: Finalists of the NIST Lightweight Cryptography Standardization process (2021)

41. Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T., Sasaki, Y., Sim, S.M., Sun, L.: Thank Goodness Its Friday (TGIF). In: Submission to Round 1 of the NIST Lightweight Cryptography Standardization process (2019)
42. Jean, J., Peyrin, T., Sim, S.M., Tourteaux, J.: Optimizing implementations of lightweight building blocks. IACR Trans. Symmetric Cryptol. 2017(4), 130–168 (2017), https://doi.org/10.13154/tosc.v2017.i4.130-168
43. Khovratovich, D., Nikolic, I., Pieprzyk, J., Sokolowski, P., Steinfeld, R.: Rotational cryptanalysis of ARX revisited. In: Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers. pp. 519–536 (2015), https://doi.org/10.1007/978-3-662-48116-5_25
44. Knudsen, L.R.: Cryptanalysis of LOKI. In: Advances in Cryptology - ASIACRYPT '91, International Conference on the Theory and Applications of Cryptology, Fujiyoshida, Japan, November 11-14, 1991, Proceedings. pp. 22–35 (1991), https://doi.org/10.1007/3-540-57332-1_2
45. Knudsen, L.R.: Deal - a 128-bit block cipher. In: NIST AES Proposal (1998)
46. Knudsen, L.R., Wagner, D.A.: Integral cryptanalysis. In: Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers. pp. 112–127 (2002), https://doi.org/10.1007/3-540-45661-9_9
47. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON block cipher family. In: Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. pp. 161–185 (2015), https://doi.org/10.1007/978-3-662-47989-6_8
48. Leurent, G., Pernot, C., Schrottenloher, A.: Clustering effect in simon and simeck. Cryptology ePrint Archive, Report 2021/1198 (2021), https://ia.cr/2021/1198
49. Liu, Z., Li, Y., Wang, M.: Optimal differential trails in simon-like ciphers. IACR Trans. Symmetric Cryptol. 2017(1), 358–379 (2017), https://doi.org/10.13154/tosc.v2017.i1.358-379
50. Liu, Z., Li, Y., Wang, M.: The security of simon-like ciphers against linear cryptanalysis. IACR Cryptology ePrint Archive 2017, 576 (2017), http://eprint.iacr.org/2017/576
51. Liu, Z., Li, Y., Wang, M.: The security of simon-like ciphers against linear cryptanalysis. Cryptology ePrint Archive, Report 2017/576 (2017), https://eprint.iacr.org/2017/576
52. Louis, W.: Software for SUPERCOP benchmarking of SIMON and SPECK, https://github.com/lrwinge/simon_speck_supercop
53. Lu, J., Liu, Y., Ashur, T., Sun, B., Li, C.: Rotational-xor cryptanalysis of simon-like block ciphers. In: Information Security and Privacy - 25th Australasian Conference, ACISP 2020, Perth, WA, Australia, November 30 - December 2, 2020, Proceedings. pp. 105–124 (2020), https://doi.org/10.1007/978-3-030-55304-3_6
54. Matsui, M.: Linear cryptanalysis method for DES cipher. In: Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings. pp. 386–397 (1993), https://doi.org/10.1007/3-540-48285-7_33
55. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers. pp. 57–76 (2011), https://doi.org/10.1007/978-3-642-34704-7_5
56. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and

Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4307, pp. 529–545. Springer (2006), https://doi.org/10.1007/11935308_38
57. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptol. 24(2), 292–321 (2011), https://doi.org/10.1007/s00145-010-9085-7
58. Piret, G., Roche, T., Carlet, C.: PICARO - A block cipher allowing efficient higher-order side-channel resistance. In: Applied Cryptography and Network Security - 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings. pp. 311–328 (2012), https://doi.org/10.1007/978-3-642-31284-7_19
59. Sakamoto, K., Minematsu, K., Shibata, N., Shigeri, M., Kubo, H., Funabiki, Y., Bogdanov, A., Morioka, S., Isobe, T.: Tweakable TWINE: building a tweakable block cipher on generalized feistel structure. In: Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings. pp. 129–145 (2019), https://doi.org/10.1007/978-3-030-26834-3_8
60. Sasaki, Y.: Related-key boomerang attacks on full ANU lightweight block cipher. In: International Conference on Applied Cryptography and Network Security. pp. 421–439. Springer (2018)
61. Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. pp. 185–215 (2017), https://doi.org/10.1007/978-3-319-56617-7_7
62. Selçuk, A.A.: On probability of success in linear and differential cryptanalysis. J. Cryptology 21(1), 131–147 (2008), https://doi.org/10.1007/s00145-007-9013-7
63. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An ultra-lightweight blockcipher. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. pp. 342–357 (2011), https://doi.org/10.1007/978-3-642-23951-9_23
64. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher CLEFIA (extended abstract). In: Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers. pp. 181–195 (2007), https://doi.org/10.1007/978-3-540-74619-5_12
65. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. pp. 244–257 (2009), https://doi.org/10.1007/978-3-642-02777-2_24
66. Stefan Kölbl: CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives, https://github.com/kste/cryptosmt
67. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers. In: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I. pp. 158–178 (2014), https://doi.org/10.1007/978-3-662-45611-8_9

68. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: TWINE : A lightweight block cipher for multiple platforms. In: Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. pp. 339–354 (2012), https://doi.org/10.1007/978-3-642-35999-6_22
69. Todo, Y.: Structural evaluation by generalized integral property. In: Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. pp. 287–314 (2015), https://doi.org/10.1007/978-3-662-46800-5_12
70. Todo, Y., Morii, M.: Bit-based division property and application to simon family. In: Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers. pp. 357–377 (2016), https://doi.org/10.1007/978-3-662-52993-5_18
71. Wang, N., Wang, X., Jia, K., Zhao, J.: Differential attacks on reduced SIMON versions with dynamic key-guessing techniques. Sci. China Inf. Sci. 61(9), 098103:1–098103:3 (2018), https://doi.org/10.1007/s11432-017-9231-5
72. Wu, W., Zhang, L.: Lblock: A lightweight block cipher. In: Applied Cryptography and Network Security - 9th International Conference, ACNS 2011, Nerja, Spain, June 7-10, 2011. Proceedings. pp. 327–344 (2011), https://doi.org/10.1007/978-3-642-21554-4_19
73. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. pp. 648–678 (2016), https://doi.org/10.1007/978-3-662-53887-6_24
74. Yang, G., Zhu, B., Suder, V., Aagaard, M.D., Gong, G.: The simeck family of lightweight block ciphers. In: Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings. pp. 307–329 (2015), https://doi.org/10.1007/978-3-662-48324-4_16
75. Zhang, W., Bao, Z., Lin, D., Rijmen, V., Yang, B., Verbauwhede, I.: RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. Sci. China Inf. Sci. 58(12), 1–15 (2015), https://doi.org/10.1007/s11432-015-5459-7

Appendix A Test Vectors

```
/* SAND-64/128 */
K: 0F 1F 2F 3F 4F 5F 6F 7F 8F 9F AF BF CF DF EF FF
P: 0F 1F 2F 3F 4F 5F 6F 7F
C: 4D E9 0F 3B 2B 5E 70 6B

/* SAND-128/128 */
K: 1F 1E 1D 1C 1B 1A 19 18 17 16 15 14 13 12 11 10
P: FF EF DF CF BF AF 9F 8F 7F 6F 5F 4F 3F 2F 1F 0F
C: F1 FB E8 65 BE CE 10 F8 2A 34 C6 C9 9D 6A 73 03
```

Appendix B SAND Reference Implementations

B.1 SAND -64/128 C Reference Code from Bitslice View

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

typedef uint32_t u32;
#define ROUNDS 48
#define SWAP(x, y) ((x) ^= (y), (y) ^= (x), (x) ^= (y))

u32 ROTL(u32 x, int shift){
    return (x << shift) | (x >> (32 - shift));
}

u32 G0(u32 x){
    x ^= (x >> 3) & (x >> 2) & 0x11111111;
    x ^= (x << 3) & (x << 2) & 0x88888888;
    return x;
}

u32 G1(u32 x){
    x ^= (x >> 1) & (x << 1) & 0x44444444;
    x ^= (x << 1) & (x >> 1) & 0x22222222;
    return x;
}

u32 P(u32 x){
    return ROTL(x & 0x0F0F0F0F, 28) | ROTL(x & 0xF0F0F0F0,
   12);
}

void Round(const u32 Pt[], u32 Ct[], const u32 Rk[], int
   CryptRound){
    u32 x = Pt[1], y = Pt[0];
    for(int r = 0; r < CryptRound; r++) {
        y ^= P(G0(x) ^ G1(ROTL(x, 4))) ^ Rk[r];
        SWAP(x, y);
    }
    Ct[1] = y, Ct[0] = x;
}

u32 A8x3(u32 x){
    for(int i = 0; i < 3; i++){
        x = ROTL(x, 28);
        u32 t = x >> 24 & 0xF;
        x ^= (((t << 1) | (t >> 3)) << 28) ^ ((t << 3 & 0xF)
   << 24);
    }
    return x;
}

void KeySchedule(const u32 Mk[], u32 Rk[], int CryptRound,
   int Dec){
    Rk[3] = Mk[3], Rk[2] = Mk[2], Rk[1] = Mk[1], Rk[0] = Mk
   [0];
    for(int r = 0; r < CryptRound - 4; r++)
        Rk[r + 4] = A8x3(Rk[r + 3]) ^ Rk[r] ^ (r + 1);
    if(Dec == 1){
        for(int r = 0; r < (int)(CryptRound / 2); r++)
            SWAP(Rk[r], Rk[CryptRound - r - 1]);
    }
}

int main(int argc, char *argv[])
{
    u32 Pt[2] = {0x4F5F6F7F, 0x0F1F2F3F}, Ct[2];
    u32 Mk[4] = {0xCFDFEFFF, 0x8F9FAFBF, 0x4F5F6F7F, 0
   x0F1F2F3F};
    u32 Rk[ROUNDS];

    printf("Pt: 0x%08X 0x%08X\n", Pt[1], Pt[0]);
    printf("Mk: 0x%08X 0x%08X 0x%08X 0x%08X\n", Mk[3],Mk[2],
   Mk[1],Mk[0]);
    printf("Process Enc\n");
    KeySchedule(Mk, Rk, ROUNDS, 0);
    Round(Pt, Ct, Rk, ROUNDS);
    printf("Ct: 0x%08X 0x%08X\n", Ct[1], Ct[0]);
    printf("Process Dec\n");
    KeySchedule(Mk, Rk, ROUNDS, 1);
    Round(Ct, Pt, Rk, ROUNDS);
    printf("Pt: 0x%08X 0x%08X\n\n", Pt[1], Pt[0]);

    return 0;
}
```

B.2 SAND-128/128 C Reference Code from Bitslice View

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

typedef uint64_t u64;
#define ROUNDS 54
#define SWAP(x, y) ((x) ^= (y), (y) ^= (x), (x) ^= (y))

u64 ROTL(u64 x, int shift){
    return (x << shift) | (x >> (64 - shift));
}

u64 G0(u64 x){
    x ^= (x >> 3) & (x >> 2) & 0x1111111111111111;
    x ^= (x << 3) & (x << 2) & 0x8888888888888888;
    return x;
}

u64 G1(u64 x){
    x ^= (x >> 1) & (x << 1) & 0x4444444444444444;
    x ^= (x << 1) & (x >> 1) & 0x2222222222222222;
    return x;
}

u64 P(u64 x){
    return ROTL(x & 0x00FF00FF00FF00FF, 56) | ROTL(x & 0
   xFF00FF00FF00FF00, 24);
}

void Round(const u64 Pt[], u64 Ct[], const u64 Rk[], int
   CryptRound){
    u64 x = Pt[1], y = Pt[0];
    for(int r = 0; r < CryptRound; r++) {
        y ^= P(G0(x) ^ G1(ROTL(x, 4))) ^ Rk[r];
        SWAP(x, y);
    }
    Ct[1] = y, Ct[0] = x;
}

u64 A16x3(u64 x){
    for(int i = 0; i < 3; i++){
        x = ROTL(x, 60);
        u64 t = x >> 56 & 0xF;
        x ^= (((t << 1) | (t >> 3)) << 60) ^ ((t << 3 & 0xF)
   << 56);
    }
    return x;
}

void KeySchedule(const u64 Mk[], u64 Rk[], int CryptRound,
   int Dec){
    Rk[1] = Mk[1], Rk[0] = Mk[0];
    for(int r = 0; r < CryptRound - 2; r++)
        Rk[r + 2] = A16x3(Rk[r + 1]) ^ Rk[r] ^ (r + 1);
    if(Dec == 1){
        for(int r = 0; r < (int)(CryptRound / 2); r++)
            SWAP(Rk[r], Rk[CryptRound - r - 1]);
    }
}
```

```c
int main(int argc, char *argv[])
{
    u64 Pt[2] = {0x7F6F5F4F3F2F1F0F, 0xFFEFDFCFBFAF9F8F}, Ct
    [2];
    u64 Mk[2] = {0x1716151413121110, 0x1F1E1D1C1B1A1918};
    u64 Rk[ROUNDS];

    printf("Pt: 0x%016lX 0x%016lX\n", Pt[1], Pt[0]);
    printf("Mk: 0x%016lX 0x%016lX\n", Mk[1], Mk[0]);
    printf("Process Enc\n");
    KeySchedule(Mk, Rk, ROUNDS, 0);
    Round(Pt, Ct, Rk, ROUNDS);
    printf("Ct: 0x%016lX 0x%016lX\n", Ct[1], Ct[0]);
    printf("Process Dec\n");
    KeySchedule(Mk, Rk, ROUNDS, 1);
    Round(Ct, Pt, Rk, ROUNDS);
    printf("Pt: 0x%016lX 0x%016lX\n\n", Pt[1], Pt[0]);

    return 0;
}
```

B.3 SAND -64/128 C Reference Code from *SSb* View

```c
/* Just round function */
#include <stdio.h>
#include <stdint.h>

typedef uint32_t u32;
#define ROUNDS 48
#define SWAP(x, y) ((x) ^= (y), (y) ^= (x), (x) ^= (y))

const unsigned char SSb[16] = {
    0x00, 0x11, 0x22, 0xb3, 0x44, 0x57, 0x66, 0xf5,
    0x88, 0x99, 0xae, 0x3d, 0xdc, 0xcf, 0x7a, 0xeb
};

u32 ROTL(u32 x, int shift){
    return (x << shift) | (x >> (32 - shift));
}

u32 S(u32 x){
    u32 g0 = 0, g1 = 0, t = 0;
    for(int i = 0; i < 32; i += 4){
        t = SSb[x >> i & 0xF];
        g0 ^= (t >> 4 & 0xF) << i;
        g1 ^= (t & 0xF) << i;
    }
    return g0 ^ ROTL(g1, 4);
}

u32 P(u32 x){
    return ROTL(x & 0x0F0F0F0F, 28) | ROTL(x & 0xF0F0F0F0,
    12);
}

void Round(const u32 Pt[], u32 Ct[], const u32 Rk[], int
    CryptRound){
    u32 x = Pt[1], y = Pt[0];
    for(int r = 0; r < CryptRound; r++) {
        y ^= P(S(x)) ^ Rk[r];
        SWAP(x, y);
    }
    Ct[1] = y, Ct[0] = x;
}
```

B.4 SAND-128/128 C Reference Code from *SSb* View

```
/* Just round function */
#include <stdio.h>
#include <stdint.h>

typedef uint64_t u64;
#define ROUNDS 54
#define SWAP(x, y) ((x) ^= (y), (y) ^= (x), (x) ^= (y))

const unsigned char SSb[16] = {
    0x00, 0x11, 0x22, 0xb3, 0x44, 0x57, 0x66, 0xf5,
    0x88, 0x99, 0xae, 0x3d, 0xdc, 0xcf, 0x7a, 0xeb
};

u64 ROTL(u64 x, int shift){
    return (x << shift) | (x >> (64 - shift));
}

u64 S(u64 x){
    u64 g0 = 0, g1 = 0, t = 0;
    for(int i = 0; i < 64; i += 4){
        t = SSb[x >> i & 0xF];
        g0 ^= (t >> 4 & 0xF) << i;
        g1 ^= (t & 0xF) << i;
    }
    return g0 ^ ROTL(g1, 4);
}

u64 P(u64 x){
    return ROTL(x & 0x00FF00FF00FF00FF, 56) | ROTL(x & 0
    xFF00FF00FF00FF00, 24);
}

void Round(const u64 Pt[], u64 Ct[], const u64 Rk[], int
    CryptRound){
    u64 x = Pt[1], y = Pt[0];
    for(int r = 0; r < CryptRound; r++) {
        y ^= P(S(x)) ^ Rk[r];
        SWAP(x, y);
    }
    Ct[1] = y, Ct[0] = x;
}
```

Appendix C DDT of *SSb*

For simplicity, we only present the DDT of SSb in this section, see Table 14. More details of SSb could be found in `https://github.com/sand-bar/SAND-Synthetic-Sbox`.

Appendix D 7-round Optimal Differential and Linear Characteristics for SAND-64/128

We give the following 7-round optimal differential and linear characteristics in Table 15 and Table 16 for SAND-64/128 as examples.

Appendix E Experiments of Differential and Linear Properties for 7-round Trails

In order to give a verification for the model under the S-box transformation and also an evaluation of clustering effect, we consider the differential and linear hull effect and distributions under random keys of the example trails presented in Appendix D. Although, these experiments are still limited with regard to the number of rounds and the number of trails, we just want to show the differential and linear properties (differential and linear hull effect, the probability distributions over random keys) of SAND block cipher to some extent.

Clustering Trails by Solver. For the optimal 7-round differential trail listed in Table 15, we fix the input and output differences in the search program and enumerate the number of trails within this differential by the SAT solver [65]. The clustering result indicates that the differential effect for this 7-round differential is not significant. Similarly, we also study the linear hull effect of the 7-round linear trail listed in Table 16, which is not notable.

Δ_{in}	Δ_{out} (with Prob.)
0x0	0x00(1)
0x1	0x11(2^{-2}) 0x13(2^{-2}) 0x91(2^{-2}) 0x93(2^{-2})
0x2	0x22(2^{-2}) 0x24(2^{-3}) 0x26(2^{-3}) 0xA2(2^{-2}) 0xA4(2^{-3}) 0xA6(2^{-3})
0x3	0x31(2^{-3}) 0x33(2^{-3}) 0x37(2^{-2}) 0xB1(2^{-3}) 0xB3(2^{-3}) 0xB5(2^{-2})
0x4	0x44(2^{-2}) 0x46(2^{-2}) 0x54(2^{-3}) 0x56(2^{-3}) 0xD4(2^{-3}) 0xD6(2^{-3})
0x5	0x45(2^{-2}) 0x47(2^{-2}) 0x55(2^{-3}) 0x57(2^{-3}) 0xD5(2^{-3}) 0xD7(2^{-3})
0x6	0x66(2^{-2}) 0x72(2^{-2}) 0xE4(2^{-2}) 0xF2(2^{-2})
0x7	0x61(2^{-3}) 0x63(2^{-3}) 0x75(2^{-3}) 0x77(2^{-3}) 0xE1(2^{-3}) 0xE3(2^{-3}) 0xF5(2^{-3}) 0xF7(2^{-3})
0x8	0x1C(2^{-3}) 0x1E(2^{-3}) 0x88(2^{-2}) 0x8C(2^{-3}) 0x8E(2^{-3}) 0x98(2^{-2})
0x9	0x1D(2^{-3}) 0x1F(2^{-3}) 0x8B(2^{-2}) 0x8D(2^{-3}) 0x8F(2^{-3}) 0x99(2^{-2})
0xA	0x2A(2^{-3}) 0x2C(2^{-3}) 0x3A(2^{-3}) 0x3E(2^{-3}) 0xAA(2^{-3}) 0xAE(2^{-3}) 0xBA(2^{-3}) 0xBC(2^{-3})
0xB	0x29(2^{-3}) 0x2D(2^{-3}) 0x3B(2^{-3}) 0x3D(2^{-3}) 0xA9(2^{-3}) 0xAF(2^{-3}) 0xBB(2^{-3}) 0xBF(2^{-3})
0xC	0x58(2^{-2}) 0xC8(2^{-2}) 0xCC(2^{-3}) 0xCE(2^{-3}) 0xDC(2^{-3}) 0xDE(2^{-3})
0xD	0x5B(2^{-2}) 0xC9(2^{-2}) 0xCD(2^{-3}) 0xCF(2^{-3}) 0xDD(2^{-3}) 0xDF(2^{-3})
0xE	0x6A(2^{-3}) 0x6C(2^{-3}) 0x7A(2^{-3}) 0x7C(2^{-3}) 0xEA(2^{-3}) 0xEE(2^{-3}) 0xFA(2^{-3}) 0xFE(2^{-3})
0xF	0x6B(2^{-3}) 0x6F(2^{-3}) 0x79(2^{-3}) 0x7D(2^{-3}) 0xEB(2^{-3}) 0xED(2^{-3}) 0xF9(2^{-3}) 0xFF(2^{-3})

Table 14: DDT of *SSb* (4-bit input and 8-bit output)

Rounds	x	y	Probability
0	`09000890`	`88880230`	2^{-6}
1	`80000900`	`09000890`	2^{-4}
2	`80000000`	`80000900`	2^{-2}
3	`00000000`	`80000000`	2^{-0}
4	`80000000`	`00000000`	2^{-2}
5	`80000800`	`80000000`	2^{-4}
6	`08000880`	`80000800`	2^{-6}
7	`88890000`	`08000880`	-

Table 15: 7-round optimal differential characteristic with probability 2^{-24}.

Rounds	x	y	Correlation
0	`2440e004`	`40004004`	2^{-3}
1	`40004004`	`44000000`	2^{-2}
2	`44000000`	`00000400`	2^{-1}
3	`00000400`	`00000000`	2^{-0}
4	`00000000`	`00000400`	2^{-1}
5	`00000400`	`44000000`	2^{-2}
6	`44000000`	`40004004`	2^{-3}
7	`40004004`	`04406004`	-

Table 16: 7-round optimal linear characteristic with correlation 2^{-12}.

Distribution Tests over Random Sampling Keys. To perform the probability distribution tests of the above trails over random sampling keys, we start by randomly selecting 10000 keys. Then, with each selected key, we encrypt 2^{30} blocks and count the corresponding number of right pairs for differential (resp., the absolute linear bias for linear). The results are compared to the expected normal distributions for differential and linear from [62], as shown in Figure 16 and 17 respectively, which indicates a nice match between the experimental distributions and the expected.

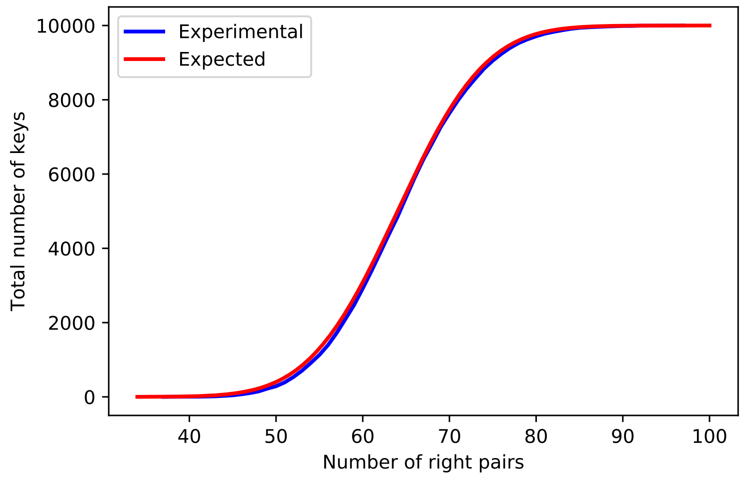


Figure 16: Cumulative distributions of number of right pairs over 10000 random keys (blue) and expected normal distribution (red).

Appendix F Threshold Implementation of the SAND Cipher

In this section, we show that the threshold implementation of SAND cipher can take advantage of its Toffoli gates-based structure.

The concept of threshold implementation is to randomly encode each secret-dependent bit (say, x) into several shares (say, $x_1, \ldots, x_n$) such that $x = x_1 \oplus \ldots \oplus x_n$, and accordingly perform the cryptographic algorithms in the shared form (rather than the raw secret). In the rest of this section, unless otherwise noted, we consider the most efficient first-order secure case, where the number of shares is 3. As any cryptographic algorithm can be represented by a composition of XOR and AND gates, the shared form implementation can be achieved by

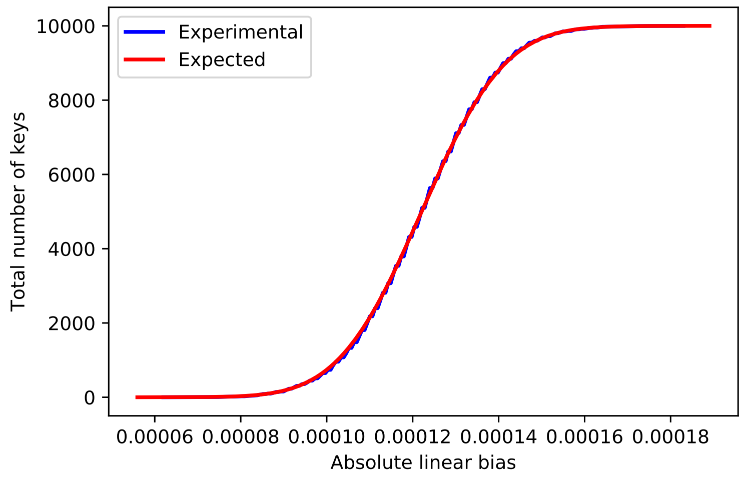


Figure 17: Cumulative distributions of absolute linear bias over 10000 random keys (blue) and expected normal distribution (red).

transforming each gate into their shared correspondence. The shared XOR gate can be trivially constructed by XORing pairs of input shares separately. However, the construction of shared AND gate is non-trivial. Previous work [57] has shown that it is impossible to construct a secure AND gate[8] with 3 shares without introducing any randomness.

A Toffoli gate is a composite gate that takes 3 input bits (say, x, y and z) and outputs $z = x \oplus yz$. Although a single AND gate is not quite friendly to the threshold implementation, a secure shared Toffoli gate can be constructed as follows:

$$\begin{aligned} z_1 &= x_2 \oplus y_2z_2 \oplus y_2z_3 \oplus y_3z_2 \\ z_2 &= x_3 \oplus y_3z_3 \oplus y_3z_1 \oplus y_1z_3 \\ z_3 &= x_2 \oplus y_2z_2 \oplus y_2z_3 \oplus y_3z_2 \end{aligned} \tag{1}$$

This structure contributes to an efficient threshold implementation of the SAND cipher.

Figure 18 shows the threshold implementation of the SAND round function, where all the shared linear gates (XOR, and rotation and P_n) are performed on each share separately and TI Toffoli gate can be calculated by Equation 1. Note that each of G_0 and G_1 includes two Toffoli gates that need to be separated by a register to prevent the Glitches. Thus, the calculation of the round function requires two cycles.

[8] Informally speaking, a secure shared gate satisfied the properties of correctness, noncompleteness and uniformity [56,57]

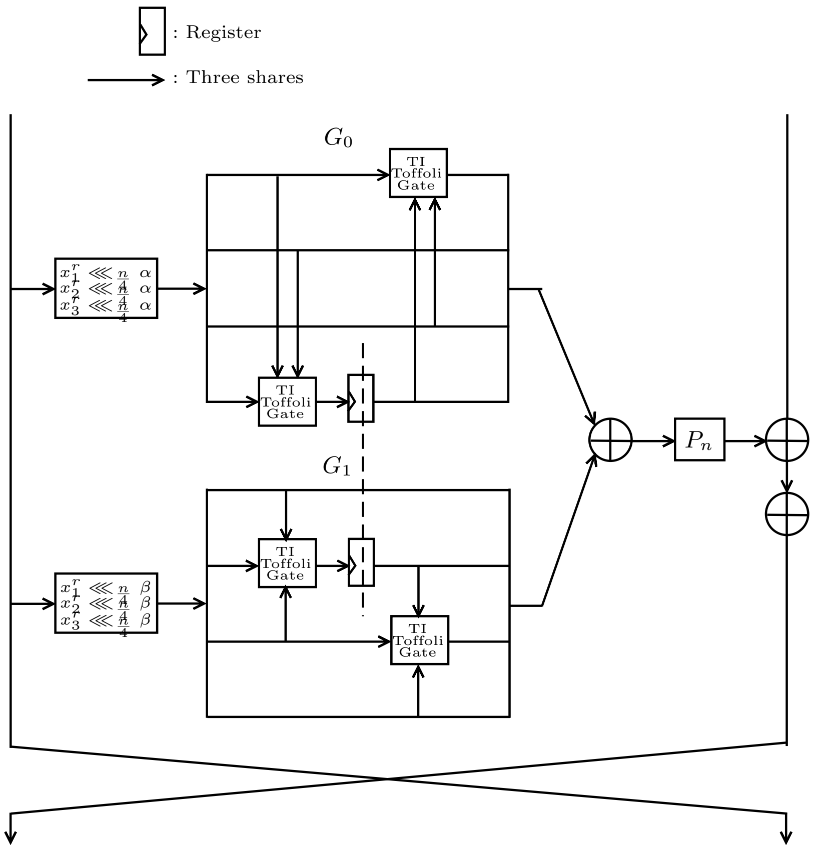

: Register
: Three shares
G_0
G_1
TI Toffoli Gate
$x_1^r \lll \frac{n}{4}\ \alpha$
$x_2^r \lll \frac{n}{4}\ \alpha$
$x_3^r \lll \frac{n}{4}\ \alpha$
$x_1^r \lll \frac{n}{4}\ \beta$
$x_2^r \lll \frac{n}{4}\ \beta$
$x_3^r \lll \frac{n}{4}\ \beta$
P_n

Figure 18: Threshold implementation of the SAND round function