

BAKSHEESH: Similar Yet Different From GIFT

Introducing a Lightweight Cipher

Anubhab Baksi¹, Jakub Breier², Anupam Chattopadhyay¹, Tomáš Gerlich³, Sylvain Guilley⁴, Naina Gupta¹, Takanori Isobe⁵, Arpan Jati¹, Petr Jedlicka³, Hyunjun Kim⁶, Fukang Liu⁵, Zdeněk Martinásek³, Kosei Sakamoto⁵, Hwajeong Seo⁶, Rentaro Shiba⁵, and Ritu Ranjan Shrivastwa⁴



anubhab001@e.ntu.edu.sg, jbreier@jbreier.com, anupam@ntu.edu.sg, xgerli02@stud.feec.vutbr.cz, sylvain.guilley@telecom-paristech.fr, naina003@e.ntu.edu.sg, takanori.isobe@ai.u-hyogo.ac.jp, arpan.jati@ntu.edu.sg, xjedli23@vut.cz, khj930704@gmail.com, liufukangs@gmail.com, martinasek@feec.vutbr.cz, k.sakamoto0728@gmail.com, hwajeong84@gmail.com, rentaro.shiba@gmail.com, ritu-ranjan.shrivastwa@secure-ic.com

Abstract. We propose a lightweight block cipher named BAKSHEESH, which follows up on the popular cipher GIFT-128 (CHES'17). BAKSHEESH runs for 35 rounds, which is 12.50 percent smaller compared to GIFT-128 (runs for 40 rounds) while maintaining the same security claims against the classical attacks. The crux of BAKSHEESH is to use a 4-bit SBox that has a non-trivial Linear Structure (LS). An SBox with one or more non-trivial LS has not been used in a cipher construction until DEFAULT (Asiacrypt'21). DEFAULT is pitched to have inherent protection against the Differential Fault Attack (DFA), thanks to its SBox having 3 non-trivial LS. BAKSHEESH, however, uses an SBox with only 1 non-trivial LS; and is a traditional cipher just like GIFT-128, with no claims against DFA.

The SBox requires a low number of AND gates, making BAKSHEESH suitable for side channel countermeasures (when compared to GIFT-128) and other niche applications. Indeed, our study on the cost of the threshold implementation shows that BAKSHEESH offers a few-fold advantage over other lightweight ciphers. The design is not much deviated from its predecessor (GIFT-128), thereby allowing for easy implementation (such as fix-slicing in software). However, BAKSHEESH opts for the full-round key XOR, compared to the half-round key XOR in GIFT.

Thus, when taking everything into account, we show how a cipher construction can benefit from the unique vantage point of using 1 LS SBox, by combining the state-of-the-art progress in classical cryptanalysis and protection against device-dependent attacks. We, therefore, create a new paradigm of lightweight ciphers, by adequate deliberation on the design choice, and solidify it with appropriate security analysis and ample implementation/benchmark.

Keywords: lightweight cryptography · block cipher · threshold implementation · gift · default · linear structure

Table of Contents

1	Introduction	4
2	Background	5
2.1	Preliminary	5
2.2	Side-Channel Attack and Countermeasure	5
2.3	Countermeasure to Power Analysis	6
2.3.1	Cipher Level Construction: Role of XOR	6
2.3.2	Cipher Level Construction: Specialisation to SBox	6
2.3.3	Resisting Side Channel Attack: Threshold Implementation	6
3	Construction	6
3.1	SubCells (SBox Layer)	7
3.2	PermBits (Bit-permutation as Linear Layer)	7
3.3	AddConstants (Round Constant XOR Layer)	7
3.4	Key Schedule Subroutine and AddRoundKey Layer	7
4	Design Rationale	7
4.1	Motivation and Objective	7
4.2	Rationale behind Choice of Design Components	9
4.2.1	Linear Layer	9
4.2.2	SBox	9
4.2.3	Key Schedule and Round Key XOR	10
4.2.4	Round Constants	11
5	Security Claims and Analysis	11
5.1	Classical Attacks	11
5.1.1	Differential and Linear Attacks	11
5.1.2	Algebraic Attack	12
5.1.3	Integral Attack	13
5.1.4	Impossible Differential Attack	13
5.1.5	Invariant Subspace Attack	14
5.1.6	Other Attacks	15
5.2	Side-Channel Attacks	15
6	Implementation and Performance	16
6.1	Unprotected Benchmark	16
6.1.1	Hardware	16
6.1.2	Software (Fix-slice)	17
6.2	Side-Channel Protected Benchmark: Threshold Implementation	18
6.2.1	SBox	18
6.2.2	Encryption	18
7	Conclusion	19
A	More Cryptographic Properties of BAKSHEESH SBox	20
B	Implementations of BAKSHEESH SBox	20
C	Threshold Implementation (Combinational SBox)	22
D	More Information on BAKSHEESH Construction	23
D.1	Structure and Test Vectors	23
D.2	Details on Round Constants and Tap Positions	23

E	Annex to Security Claims and Analysis	25
E.1	Trivial Differential and Linear Transitions	25
E.2	Key Recovery with Algebraic Methods	25
E.3	Integral Attack: Comparison with GIFT-128	26
F	Postscript	26
F.1	Similarities and Differences between GIFT-128 and BAKSHEESH	26
F.2	BAKSHEESH: Advantages over GIFT-128	26
F.3	Search for BAKSHEESH SBox	27
F.4	Number of Slack Rounds	27
F.5	How BAKSHEESH Improves State-of-the-art/Novelty	27
	References	28

1 Introduction

With the shrinkage of available space for cryptographic modules in an IoT chip, the role and importance of so-called lightweight cryptography are becoming increasingly prominent. The major driving force here is to find interesting cost savings by either employing sophisticated implementations (say [1]), or designing a brand new cipher that attains a lower cost than existing ciphers. In the latter case, the cipher is designed to take leverage of the state-of-the-art progress the research community has access to, thereby streamlining design choices. This ultimately results in reducing the cost of the cipher, while retaining a strong security margin. It is an extremely popular research direction in the community now-a-days, as evident from recent cipher proposals like GIFT [22] or PYJAMASK-128 [63]; as well as two competitions, CAESAR⁷ and LWC⁸.

Going with the trend, here we analyse and improve upon an immensely popular lightweight block cipher, GIFT [22]. Designed in 2017, GIFT-128 has gained serious attention; as evident from the third party analysis (for example, [99, 108]); as well as ciphers which are directly based on GIFT-128 (such as GIFT-COFB [21]).

Although the design of GIFT amasses new concepts to reduce the device footprint, the knowledge and expertise of the community have come a long way in the last half a decade. With that in mind, it stands to reason that a fresh look can yield improvement by pushing the limit even further. Indeed, a deeper look, when equipped with the state-of-the-art progress in related disciplines, reveals potential options to break the efficiency records set by GIFT.

The objective here is to show a more efficient and versatile cipher construction based on the previously untapped potential of SBoxes with non-trivial *Linear Structures* (LS, Definition 1). This has surfaced in the literature merely as a theoretical interest (see, e.g., [82]). The first time the potential of an LS SBox is realised is through DEFAULT [11, Chapter 8], as a way to have an innate protection against the *Differential Fault Attack* (DFA, see [13, Section 5.1]). Now, in our quest to find an answer to the conundrum (*how can we find a more efficient design than GIFT that will not only improve the benchmarks for the unprotected cipher, but also will have an edge for the side-channel countermeasures?*), we decide to entertain the possibility of a mainstream block cipher with full classical security using an LS SBox. At first glance, this may not sound intuitive, however, some advantages start to appear gradually (see Section 4.2.2).

To make BAKSHEESH a proper successor of GIFT, it is imperative not to stray away much from the main design philosophy of GIFT. For this reason, it is decided to use a bit-permutation layer (it takes practically zero area in ASIC, and efficient software implementation is possible [1]). As a self-imposed challenge, we keep the same bit-permutation layer from GIFT-128, this additionally would help easy conversion of existing source codes. We would like to emphasize that keeping the same bit-permutation as GIFT-128 does not make our life easier, if anything it pushes us to a rather disadvantage — as we are facing the GIFT-128 challenge but with less freedom in design. Considering the effort that has been put in finding the lightweight components for GIFT, this unsurprisingly is no easy task. Consequently, the BAKSHEESH SBox, when paired with the linear layer of GIFT-128, has to match (if not outperform) GIFT-128. For this, we ensure that our SBox fulfills two criteria. First, we set the *Linear Branch Number* (LBN, see Definition 3) to be 3, this is the theoretical upper bound for any 4×4 SBox [92]. Save for the newly proposed DEFAULT-LAYER [11, Chapters 7, 8] (or [10, Chapters 7, 8]) no cipher uses an SBox with $LBN > 2$. Second, the SBox is to have the maximum algebraic degree of 2. Indeed, the SBox used in BAKSHEESH possibly has the lowest number of AND operations (among all the traditional ciphers using a 4×4 SBox), it arguably offers unparalleled advantage over any lightweight cipher when it comes to an application that benefits from low number of AND operations (see Section 4.1). Despite using a ‘bad’ SBox, BAKSHEESH passes the classical security requirement with flying colours (see Section 5.1).

Contribution/Organisation

Finding and perfecting novel concepts that can ultimately lead to extremely low cost – without jeopardizing the security – is generally quite challenging as it typically involves studying and selecting a set of novel design choices, choosing efficient parameters for the cipher, asserting security against classical attacks, optimizing the cipher in hardware/software, and finally testing and protecting against the side-channel attacks. To this end, we present a new lightweight block cipher, called BAKSHEESH⁹.

BAKSHEESH consists of 35 rounds; which is 12.5% smaller than its predecessor, GIFT-128 from the beginning. Also, the SBox used in BAKSHEESH is quite lightweight in hardware (as shown in Table 3, it rivals the major

⁷<https://competitions.cr.yep.to/caesar.html>.

⁸<https://csrc.nist.gov/projects/lightweight-cryptography>.

⁹This word in Persian translates to ‘tips’.

lightweight SBoxes in ASIC) and software (requires only 10 instructions on ARM Cortex-M3), on top of having probably the least number of AND operation (only 3, surpassing the SKINNY-64 SBox [24] that requires 4 operations).

Apart from GIFT, some inspiration of BAKSHEESH is drawn DEFAULT [11, Chapters 7, 8]. As DEFAULT uses full round key XOR, the related security claims remain unchallenged. While having an SBox with a linear structure, we do not raise any claims regarding DFA protection in BAKSHEESH.

With some background information covered in Section 2, the construction is described in Section 3, which is then followed by design rationale in Section 4. Thereafter, a formal analysis of BAKSHEESH is given in Section 5 (we claim BAKSHEESH offers sufficient security against the classical attacks in Section 5.1, despite having an SBox with non-trivial LS). In terms of performance, BAKSHEESH offers a noticeable improvement compared to GIFT-128, as detailed in Section 6, in unprotected (Section 6.1) as well as in the threshold implementation (Section 6.2). The paper concludes in Section 7.

All in all, we show how an unusual design choice can lead to improvement in a highly analyzed cipher construction. The range of advantages of our cipher covers the full extent of classical analysis and side channel analysis among others. Every minute detail of the design choice or security claim is scrutinized thoroughly. On top of that, we offer a complete package of various implementations, enabling our user to readily deploy BAKSHEESH¹⁰.

2 Background

2.1 Preliminary

The difference distribution table (DDT) for an $n \times n$ SBox S is basically the $2^n \times 2^n$ matrix; where the row $\delta \in \mathbb{F}_2^n$ and column $\Delta \in \mathbb{F}_2^n$, $\text{DDT}[\delta, \Delta] = \text{number of } x, \ni S(x) \oplus S(x \oplus \delta) = \Delta$. The maximum entry in the DDT except $\text{DDT}[0, 0]$ is known as the differential uniformity (DU). The linear approximation table (LAT) for an $n \times n$ SBox S is the $2^n \times 2^n$ matrix; where the row $\gamma \in \mathbb{F}_2^n$ and column $\Gamma \in \mathbb{F}_2^n$ stores $|\{\gamma \cdot \vec{x} \oplus \Gamma \cdot \vec{y} = 0\}| - 2^{n-1}$; \vec{x} denotes the input variables to the SBox, \vec{y} denotes the output variables, and \cdot is the dot product.

Definition 1 (Linear Structure (LS)). For $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$, an element $a \in \mathbb{F}_2^n$ is called linear structure (LS) of F , if for some constant $c \in \mathbb{F}_2^n$, $F(x) \oplus F(x \oplus a) = c$ holds $\forall x \in \mathbb{F}_2^n$.

Definition 2 (Non-linearity). The non-linearity of the Boolean function $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ is the minimum Hamming distance of f to the set of all affine functions. Further, the non-linearity of $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ is the minimum of the non-linearities of all the component functions of F .

Definition 3 (Differential/Linear Branch Number). The Differential Branch Number (DBN) of F is defined as $\min_{\delta \neq 0} \{\text{HW}(\delta) + \text{HW}(F(x) \oplus F(x \oplus \delta))\}$; and that of the Linear Branch Number (LBN) as $\min\{\text{HW}(\alpha) + \text{HW}(\beta)\}$ given $\alpha \neq 0, \text{LAT}[\alpha, \beta] \neq 0$; where $\text{HW}(\cdot)$ denotes the Hamming weight.

Definition 4 (Coordinate Function and Component Function). Suppose $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ is defined as $F(x) = (f_0(x), \dots, f_{n-1}(x))$ for all $x \in \mathbb{F}_2^n$, where $f_i : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$ for $i = 0, \dots, n-1$. Then each f_i is called a coordinate function of F . The linear combinations of f_i 's are called the component functions of F .

2.2 Side-Channel Attack and Countermeasure

Side channel attacks analyse physical characteristics of cryptographic devices related to the execution of the implementation of a cryptographic algorithm. The physical analysis aims to extract a secret component such as the key. The rationale is that there is a relationship between the manipulated data, the executed operations and the physical properties observed during the execution of the cipher on the device. The physical properties that can be extracted are, for example, the execution time of a cryptographic algorithm [76], the electromagnetic emanation [59] or the power consumption of the device [77]. From a different point of view, side channel attacks lead to extremely effective and successful attacks against industrial products [18, 88].

We focus on the side channel attack based on the power consumption although the same can be applied similarly to other physical properties. This attack is introduced by Kocher [76]. A more systematic approach is to use the so-called *Correlation Power Attack* (CPA) [42], where the correct (sub-)key is recovered by taking the candidate which gives maximum (absolute) correlation.

¹⁰<https://github.com/anubhab001/baksheesh/>.

2.3 Countermeasure to Power Analysis

Power attacks can be prevented by means of countermeasure techniques. The goal of every countermeasure is to make the power consumption of a cryptographic device independent of intermediate values that are processed during its operation phase. Generally, countermeasure techniques are divided into two basic groups: *Hiding* [50] and *Masking* [84]. In the masking approach, which is the common choice in recent times, each intermediate value is concealed by a random mask. The main advantage of this approach is that it can be realised at the implementation level. By contrast, hiding tries to break the link between the power consumption and the processed data values utilizing two approaches. The first approach is to build devices whose power consumption is random and the second is to build devices whose power consumption is constant. However, these goals cannot be reached in practice though there are several methods to get as close as possible to this goal, by affecting amplitude and time dimension of power traces.

2.3.1 Cipher Level Construction: Role of XOR It is reported in [107] (and also in [10, Chapter 4]) that the XOR operation creates ambiguity with respect to the side-channel attacks, at least with respect to CPA [42]. Stated briefly, under noiseless assumption; if x has a correlation of ρ then $\vec{1} \oplus x$ has a correlation of $-\rho$, for all x , where $\vec{1}$ is the Boolean vector (of suitable length) with all bits set at logic high.

2.3.2 Cipher Level Construction: Specialisation to SBox The notion of inherent side channel security of an SBox is not new, one may look into [48, 90]. However, none of these notions appears to capture the nature of an SBox with non-zero LS (Definition 1). Since such an SBox mimics the XOR operation to some extent, it is possible to generalize the SCA-related property of the XOR operation to an SBox [44]. Under the noiseless assumption, it can be described as follows: For an $n \times n$ SBox S if S^{-1} has a linear structure at $\vec{1}$ (i.e., \mathbf{f} for a 4-bit SBox), the correlation coefficients corresponding to $S(\alpha)$ and $S(\alpha \oplus \Phi)$ will be of equal magnitude (but with opposite signs) $\forall \alpha$, where $\Phi = S^{-1}(x) \oplus S^{-1}(x \oplus \vec{1}) \forall x$. Stated in a different way, S has a linear structure at Φ such that the corresponding output difference is at $\vec{1}$ for all input difference. As a consequence, it will not be possible to uniquely identify which between α and $\alpha \oplus \Phi$ is the correct input to the SBox, for any α .

In the simulation, we observe that the attacker cannot identify which between x and $x \oplus \alpha$ is the correct input up to a certain threshold of noise. Beyond that threshold, the attacker cannot identify any of the inputs. With our experiments with the BAKSHEESH SBox (Section 5.2), we see that indeed that attacker cannot find the correct key to the SBox, but can find the correct key (with the software traces) for the GIFT SBox. Still, we do not claim any inherent side channel security, as one may object based on the precision of the testing equipment and/or the analysis method.

2.3.3 Resisting Side Channel Attack: Threshold Implementation The threshold implementation is among the prominent masking countermeasures against side channel attacks based on secret sharing, threshold cryptography, and multi-party computation protocols [87]. It is shown to be provably secure against higher order DPA attacks [33]. Threshold implementation, although originally proposed for hardware, can also be used for software implementations [96].

3 Construction

BAKSHEESH is a 35-round block cipher that receives a 128-bit plaintext as the state $X = b_{127}b_{126} \cdots b_0$, where b_0 is the least significant bit. The state can also be expressed as $X = w_{31}w_{30} \cdots w_0$, where w_i is a 4-bit (nibble) word. We do not describe the inverse layer here for the sake of brevity, but it can be derived. The round function of BAKSHEESH consists of 4 steps¹¹ (in order): **SubCells** — applying a 4-bit SBox to the state, **PermBits** — permute the bits of the state (same as GIFT-128 [22]), **AddRoundConstants** — XORing a 6-bit constant as well as another bit to the state (same as GIFT-128), and **AddRoundKey** — XORing the round key to the state. One may refer to Figure 1 for an overview where Figure 1(a) shows encryption and Figure 1(b) shows decryption (-Inv indicates the inverse). For more details including test vectors, see Section D.

¹¹Following GIFT [22, Section 2], BAKSHEESH too can be conceptually thought as gift wrapping, i.e., putting an item in a box – wrapping a ribbon around the box – tying a knot; or virtually any sequential process that can be described in three simple steps (like, putting on the socks – slipping feet into the shoes – tying the shoe laces).

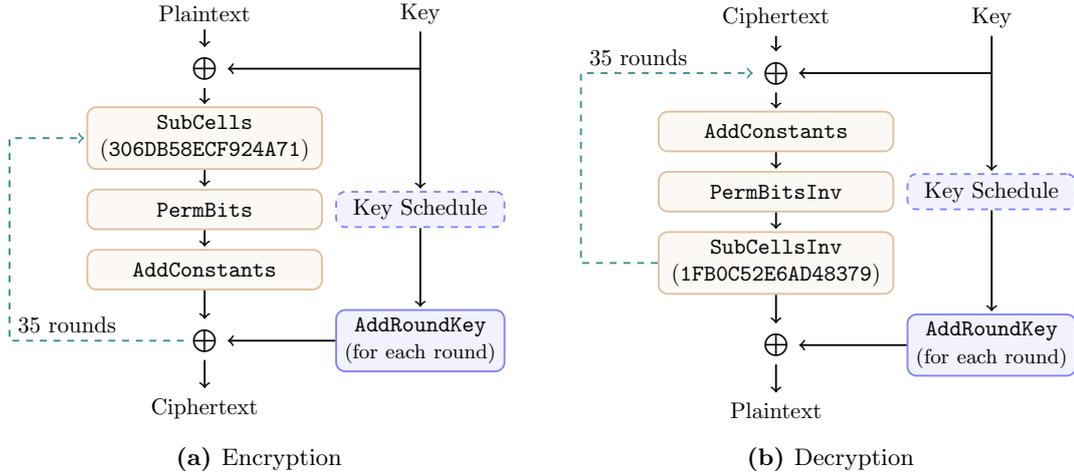


Figure 1: Schematic of BAKSHEESH

3.1 SubCells (SBox Layer)

The SBox layer uses the 4-bit LS SBox $S = 306DB58ECF924A71$ which is applied to every nibble of the state: $w_i \leftarrow S(w_i), \forall i \in \{0, \dots, 31\}$.

3.2 PermBits (Bit-permutation as Linear Layer)

The bit-permutation is the same as permutation P_{128} in GIFT-128 [22]. It maps bits from bit location i of the internal state to bit location $P_{128}(i)$, $i \leftarrow 0, 1, \dots, 127$: $b_{P_{128}(i)} \leftarrow b_i, \forall i \in \{0, \dots, 127\}$; and is given by: (0, 33, 66, 99, 96, 1, 34, 67, 64, 97, 2, 35, 32, 65, 98, 3, 4, 37, 70, 103, 100, 5, 38, 71, 68, 101, 6, 39, 36, 69, 102, 7, 8, 41, 74, 107, 104, 9, 42, 75, 72, 105, 10, 43, 40, 73, 106, 11, 12, 45, 78, 111, 108, 13, 46, 79, 76, 109, 14, 47, 44, 77, 110, 15, 16, 49, 82, 115, 112, 17, 50, 83, 80, 113, 18, 51, 48, 81, 114, 19, 20, 53, 86, 119, 116, 21, 54, 87, 84, 117, 22, 55, 52, 85, 118, 23, 24, 57, 90, 123, 120, 25, 58, 91, 88, 121, 26, 59, 56, 89, 122, 27, 28, 61, 94, 127, 124, 29, 62, 95, 92, 125, 30, 63, 60, 93, 126, 31).

3.3 AddConstants (Round Constant XOR Layer)

The following 6 tap positions are used to XOR the constants at each round: 8, 13, 19, 35, 67, 106. The round constants are given respectively by: (2, 33, 16, 9, 36, 19, 40, 53, 26, 13, 38, 51, 56, 61, 62, 31, 14, 7, 34, 49, 24, 45, 54, 59, 28, 47, 22, 43, 20, 11, 4, 3, 32, 17, 8). Notice that the bit at the last tap location (106) toggles in each round.

3.4 Key Schedule Subroutine and AddRoundKey Layer

The first round key is the same as the master key. The next round keys are generated with 1-bit right rotation, i.e., $k^{j+1} \leftarrow k^j \gg 1$. The round key for the j^{th} round, denoted by k_i^j , is bit-wise XORed to the state: $b_i \leftarrow b_i \oplus k_i^j, \forall i \in \{0, \dots, 127\}$. Unlike GIFT-128, BAKSHEESH has the key XOR at the beginning of encryption/at the end of decryption (can be seen from Figure 1).

4 Design Rationale

4.1 Motivation and Objective

The main motivation for BAKSHEESH comes from the question: *Can we push the classical security bound of GIFT?* Thus, we set the constraint of making a GIFT-like cipher, but with improved efficiency. By extension, BAKSHEESH can be considered as the newest member in the SERPENT [31] \rightsquigarrow PRESENT [38] \rightsquigarrow GIFT [22] line-up,

as shown in Figure 2. For the culture, we could recall that SERPENT is among the five finalists and ultimately ranking 2nd in the NIST AES competition¹². Also note that, PRESENT is an anagram of SERPENT; and most importantly that, PRESENT is an international standard (ISO/IEC 29192-2:2019).

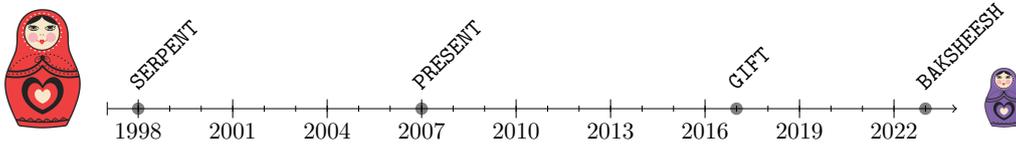


Figure 2: Evolution of ciphers in current context

One of the main differences of BAKSHEESH from its predecessors is that it uses an SBox with 1 non-zero LS. In general, LS SBoxes have a lower cost in hardware and software (follows from [104]), lower number of AND operations, and higher LBN.

However, an attempt to design a cipher with such an SBox meets some obstacles. Most notably, the traditional method of minimum ‘minimum number of active SBoxes’ does not work anymore, since this method will always return trivial classical security. For example, the security proof of AES against the differential attacks is completely based on the fact that its four consecutive rounds have at least 25 active SBoxes and the maximum probability of its SBox is 2^{-6} [53]. Doing an analogous analysis with a cipher with the SBox having 1 or more non-trivial LS would only lead to a trivial upper bound of the differential probability, i.e., 1. This problem is tackled in DEFAULT [11, Chapter 8], by incorporating the recent advancement in usage of automated tools in cipher design/cryptanalysis. In this case, the bulk load is taken up by the tools, thus it is possible to show that the maximum probability is indeed much less than 1 over a large number rounds (the probability is typically 1 for a small number of rounds). BAKSHEESH, too, is leveraged from the recent improvement.

The main objectives for BAKSHEESH can be summarised as follows:

- ① Reuse the same structure as GIFT-128 [22], but replace the half round key XOR with a full round key XOR. Since GIFT-128 is quite lightweight and has gained popularity, we naturally accept the challenge to make it even more streamlined.
- ② Use an SBox with 1 non-zero LS. It can be taken that the GIFT designers have already searched for a vast pool of SBoxes. Now, in all likelihood, the SBoxes with non-zero LS have not been kept within the search space, simply because such an SBox has traditionally been considered too weak (as it somewhat resembles the XOR operation) to have any significance in cipher design [14]. With the advent of DEFAULT [14], however, this status-quo has altered. In terms of advantage, such an SBox typically has less number of AND operations and is more efficient to implement.
- ③ Reduce the cost of unprotected implementation to set a new benchmark for lightweight ciphers. In some sense, BAKSHEESH hits theoretical end-of-the-line research (particularly, the same category as PRESENT [38] or GIFT [22]), since any SBox with lower cost will likely have more (non-zero) LS – thus would most likely require more rounds.
- ④ Take advantage of *fix-slicing* [1] to optimise implementation in software. This is the first cipher to take leverage of it (fix-slicing is proposed after GIFT).
- ⑤ Have low number of AND operations. BAKSHEESH only needs 3 AND operations per SBox. This facilitates a highly efficient adoption of side channel countermeasures (as AND operations are considered the bottleneck [34]). In this sense, BAKSHEESH fits well in the family of ciphers designed to be efficient in adopting side channel countermeasures [61, 64, 74, 91].

Apart from being beneficial for a side channel countermeasure, having a low number of AND operations appears to be one holy grail in symmetric key cryptography, as it would make efficient incorporation of the following niches:

- (a) *Fully Homomorphic Encryption* (FHE, e.g., [2, 3, 43, 56, 68, 86]);
- (b) *Multi-party Computation* (MPC, e.g., [2–4, 56]);
- (c) *Zero Knowledge* (ZK, e.g., [2, 4]);

¹²<https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development>.

(d) *Post-quantum signature* [51, Section 1].

However, one needs to consider certain other aspects while designing a cipher with domain-specific application like FHE/MPC, and hence these directions are kept out-of-scope for this work. We hope future iterations of BAKSHEESH/similar ciphers would be streamlined to for such applications.

4.2 Rationale behind Choice of Design Components

4.2.1 Linear Layer The bit-permutation based linear layer is used in PRESENT [38] or more recently in GIFT [22]. It acts as wiring in hardware, thus making it practically of zero cost. Thus, it comes as a natural choice for BAKSHEESH as we want to keep the cost as minimal as possible. The same bit-permutation as GIFT-128 is chosen, this is to make use of an optimised software implementation of GIFT-128 known as fix-slicing [1]. Keeping the same bit-permutation as GIFT-128 but finding a different SBox is an extra challenge that we have undertaken.

4.2.2 SBox We choose the SBox 306DB58ECF924A71 for BAKSHEESH. It has 1 non-zero LS (at 8), LBN of 3 and DBN of 2 (Definition 3). A comparison of cryptographic properties of few lightweight SBoxes are shown in Table 2 (here ‘AD’ refers to algebraic degree and ‘NI’ refers to non-linearity). The coordinate functions (Definition 4) of this SBox are given respectively (in ANF) by:

$$\begin{aligned} y_0 &= x_0x_2 \oplus x_0 \oplus x_1 \oplus x_3 \oplus 1, \\ y_1 &= x_0 \oplus x_1x_2 \oplus x_3 \oplus 1, \\ y_2 &= x_0x_2 \oplus x_1x_2 \oplus x_1 \oplus x_3, \\ y_3 &= x_0x_1 \oplus x_0x_2 \oplus x_2 \oplus x_3. \end{aligned}$$

It belongs to class AE# 294 (of the 302 AE classes) [55]. The DDT of this SBox is given in Table 1; where row and column corresponding to 0, and zero entries are suppressed for better readability, and it can be seen that the row 8 of the DDT contains only one 16 (at column f).

Table 1: DDT for BAKSHEESH SBox

$\delta \backslash \Delta$	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1		4		4					4		4			4	
2		4	4							4		4			
3				4	4							4	4		
4		4	4			4								4	
5				4	4	4			4						
6							4		4	4	4	4			
7		4		4	4							4			
8															16
9	4		4					4		4					
a		4	4						4	4					
b		4	4						4	4					
c		4				4		4		4	4				
d			4		4	4	4	4							
e		4	4	4		4									
f		4				4	4	4		4					

4.2.2.1 AND Count In the naïve count, BAKSHEESH has by far the least number of AND operations (6 only) compared to all other SBoxes, to our best information. This helps to achieve a highly optimised performance in SCA countermeasure and a few other niches (see Section 4.1). For comparison with similar ciphers, one may notice that PRINCE [39], SKINNY-64 [24] and GIFT [22] SBoxes respectively require 28, 16 and 10 AND

operations in the naïve sense (though this count can be reduced through optimisation). Notice from Table 9 that the naïve AND count plays a direct role in the cost for threshold countermeasure (also noted in [17]).

When optimised, the BAKSHEESH SBox can be implemented using only three AND gates – which finally overtakes the SKINNY-64 SBox (that requires 4 AND operations when optimised), likely becoming the SBox with the least AND count:

$$\begin{aligned} y_0 &= x_0x_2 \oplus x_0 \oplus x_1 \oplus x_3 \oplus 1, \\ y_1 &= x_0 \oplus x_1x_2 \oplus x_3 \oplus 1, \\ y_2 &= x_0x_2 \oplus x_1x_2 \oplus x_1 \oplus x_3, \\ y_3 &= x_0x_1 \oplus x_0x_2 \oplus x_2 \oplus x_3. \end{aligned}$$

Table 2: Cryptographic properties of few lightweight SBoxes

		DBN	LBN	LS	DU	AD (max)	AD (min)	NI
BAKSHEESH	306DB58ECF924A71	2	3	0, 8	16	2	2	0
PRESENT [38]	C56B90AD3EF84712	3	2	0	4	3	2	4
SKINNY-64 [24]	C6901A2B385D4E7F	2	2	0	4	3	2	4
GIFT [22]	1A4C6F392DB7508E	2	2	0	6	3	2	4
PYJAMASK-128 [63]	2D397BA6E0F4851C	2	2	0	4	3	2	4
MIDORI [20]	1053E2F7DA9BC846	2	2	0	4	3	3	4
	CAD3EBF789150246	3	2	0	4	3	2	4

4.2.2.2 Implementation Cost (Unprotected and Side Channel Protected) The SBox is lightweight, as can be seen from the comparative ASIC benchmarks given in Table 3 by using the look-up-based format following [92] and the figures are rounded to the nearest integer. The BAKSHEESH SBox is cheaper compared to recent 4×4 SBoxes, except it is slightly worse than that of SKINNY-64 in Faraday 65nm library.

Table 3: ASIC benchmarks for a few lightweight SBoxes

		Cost (Gate Equivalent)		
		UMC 65nm	Faraday 65nm	STM 130nm
GIFT [22]	1A4C6F392DB7508E	28	22	21
PYJAMASK-128 [63]	2D397BA6E0F4851C	28	26	22
SKINNY-64 [24]	C6901A2B385D4E7F	21	16	21
BAKSHEESH	306DB58ECF924A71	21	19	19

4.2.2.3 Branch Numbers The problem of finding SBoxes with higher LBN is studied before [74, 92]. The BAKSHEESH SBox has the LBN of 3, which is the upper limit for any 4×4 SBox and any such SBox has at least 1 non-zero LS [92, follows from Theorem 1]. Understandably, no other 4×4 SBox used in any cipher (save for the LS SBox in DEFAULT-LAYER) has LBN of 3. During our search for a 4×4 SBox with DBN 3 (which is the maximum per [92]), we observe that all such SBoxes have at least 3 non-zero LS (such as, 126CDE39F58BA047), but we are not aware of any theoretical result. Having so many LS would likely slow down the propagation of differential/linear trail significantly, thus requiring more rounds. Overall, 1LS/3LBN/2DBN seems to strike a good balance in cost for implementation and number of rounds, while keeping the AND count considerably low.

4.2.3 Key Schedule and Round Key XOR Initially we consider the key schedule of BAKSHEESH as trivial (i.e., the master key is XORed to the state), akin to LED [67] or PRINCE [39], to keep the implementation cost low. However, an invariant subspace attack [79] is trivially observed for this design. This is partially due to the sparse round constants, i.e., only 6 bits of fixed positions are affected by the constant addition operation. To

resist against this attack, we tweak the key schedule and choose 1-bit right rotation, i.e., $k^{j+1} \leftarrow k^j \gg 1$. A detailed analysis is done thereafter, but no indication of an invariant subspace attack is found.

The full (128-bit) key is XORed to the state, which is changed from half key XOR of GIFT-128. Though the cost is increased because of this choice, we opt for a more conservative design.

4.2.4 Round Constants In GIFT [22, Section 2], a 6-bit *Linear Feedback Shift Register* (LFSR) is used to generate the round constants (which are XORed at 6 tap positions), and the last bit is flipped at each round.

Likewise in BAKSHEESH, we generate the round constants from a 5-bit LFSR along with another bit that toggles in each round are used to generate the round constants (which are XORed at 6 tap positions). Thus, we have one less tap position than GIFT. Further, as can be read from Section D, we provide some justification about the choice of the tap positions.

Our LFSR uses an irreducible polynomial as its feedback function: $x^4 + x^2 + 1$. It is initialized with 00001, and can be realized with one XOR gate. Together with the standalone bit (that toggles at each round), the whole 6-bit state cycles through all integers from [2, 63]. The other popular choice for deriving the round constants, as used in [20, 39, 41], is to use the digits of π . The reason we do not use something similar is, it does not guarantee unique constant for each round.

5 Security Claims and Analysis

In this part, the security claims of BAKSHEESH are analyzed in terms of classical and side-channel attacks. In summary, it offers full (2^{128}) classical security (Section 5.1), with a possibility of a very efficient application of the threshold countermeasure (Section 5.2). Note that, we do not claim any inherent side-channel security, despite the unprotected cipher showing interesting properties with respect to side-channel attacks. An annex to the analysis can be found in Section E. Also, here we presume that full round key XOR for GIFT-128 so that the existing results on classical security are valid.

Table 4: Summary of security claims/analysis and features of BAKSHEESH

		Claim/Bound	Reference
Classical	Differential, Linear	2^{128}	Section 5.1.1
	Algebraic		Section 5.1.2
	Integral		Section 5.1.3
	Impossible Differential		Section 5.1.4
	Invariant Subspace		Section 5.1.5
	Others	\propto GIFT-128 or 2^{128}	Section 5.1.6
Side Channel	Inherent	\times	Section 5.2
	Countermeasure (Threshold [⊗])	Not feasible	Section 6.2

⊗: Easier to implement compared to (traditional) lightweight ciphers like GIFT-128

5.1 Classical Attacks

5.1.1 Differential and Linear Attacks In Table 5, we present the differential¹³ and linear bounds as for BAKSHEESH together with GIFT-128.

The original results on GIFT-128 are taken from [99, Table 5] which inherently assumes full round key XOR). Further, GIFT-128 does not have the initial key XOR during encryption; therefore, the bounds are automatically shifted by 1 place to the round; as shown in Table 5.

For the BAKSHEESH bounds, we follow the same SAT-based model from [99]; and for this purpose Logic Friday¹⁴ is used to construct a *Conjunctive Normal Form* (CNF) of an SBox. Further, results till round 12 are verified independently with the modelling from [9]¹⁵.

¹³Note: As the BAKSHEESH SBox’s DDT only contains multiples of 4, the differential bounds are even.

¹⁴Obtained from https://download.cnet.com/Logic-Friday/3000-20415_4-75848245.html.

¹⁵As a side note, it can be mentioned that this modelling is used in the initial search for the SBox (see Section F.3 for more details).

Table 5: Optimal differential and linear bounds for GIFT-128 (full key XOR) and BAKSHEESH

	Round	1	2	3	4	5	6	7	8	9	10	11
GIFT-128	Differential	1.415 0.000	3.415 1.415	7.000 3.415	11.415 7.000	17.000 11.415	22.415 17.000	28.415 22.415	39.000 28.415	45.415 39.000	49.415 45.415	54.415 49.415
	Linear	1.000 0.000	2.000 1.000	3.000 2.000	5.000 3.000	7.000 5.000	10.000 7.000	13.000 10.000	17.000 13.000	22.000 17.000	26.000 22.000	31.000 26.000
BAKSHEESH	Differential	0.000	2.000	4.000	8.000	14.000	20.000	30.000	40.000	48.000	54.000	60.000
	Linear	0.000	1.000	3.000	5.000	8.000	12.000	14.000	18.000	22.000	26.000	30.000
	Round	12	13	14	15	16	17	18	19	20	21	22
GIFT-128	Differential	60.415 54.415	67.830 60.415	79.000 67.830	85.415 79.000	90.415 85.415	96.415 90.415	103.415 96.415	110.83 103.415	121.415 110.83	126.415 121.415	132.415 126.415
	Linear	36.000 31.000	38.000 36.000	41.000 38.000	45.000 41.000	48.000 45.000	51.000 48.000	56.000 51.000	59.000 56.000	64.000 59.000	68.000 64.000	74.000 68.000
BAKSHEESH	Differential	70.000	76.000	84.000	92.000	100.000	110.000	120.000	126.000	132.000	140.000	148.000
	Linear	34.000	37.000	40.000	43.000	46.000	49.000	52.000	55.000	58.000	61.000	64.000

If the absence of key XOR of a round key at the first round of encryption is not taken into account, then the following observations can be drawn. It can be seen that BAKSHEESH has better differential bounds up to the 22nd round. Besides, linear bounds of BAKSHEESH are also better than GIFT-128 up to the 8th round. However, the linear bound becomes equal at the 9th round and finally drops at the 11th round, but is still competitive to GIFT-128. Since the actual design of GIFT-128 misses the key XOR at the beginning of encryption, BAKSHEESH outperforms GIFT-128 throughout (at least till 22 rounds) in differential, and falls behind in linear starting at round 19 and does not recover at least till round 22.

In a nutshell, BAKSHEESH achieves the upper differential bound of 2^{-128} and the upper linear bounds of 2^{-64} at the 20th and 22nd rounds, respectively. Based on this result, we argue that 35 rounds of BAKSHEESH would be sufficient against differential and linear attacks.

An optimal differential and an optimal linear trail for reduced 22-round BAKSHEESH are shown in Table 6. It can be noted that the trivial paths are taken for differential (i.e., input difference of 8) and linear (i.e., input mask of 8) trails¹⁶. Also, it can be seen from the optimal linear trail that an input mask is iterated (there exists the one-round iterated linear trail from the 4th to 21th round with 3 active bits)¹⁷.

5.1.2 Algebraic Attack The algebraic degree of the SBox and its inverse is 2 in either cases. Using Sage¹⁸, the inverse of the SBox can be represented as follows:

$$\begin{aligned}
x_0 &= (y_0 \oplus y_3)(y_1 \oplus y_2) \oplus y_2 \oplus y_3 \oplus 1, \\
x_1 &= y_0 y_2 \oplus y_3(y_0 \oplus y_2) \oplus y_0 \oplus y_1 \oplus y_3, \\
x_2 &= y_0(y_1 \oplus y_2) \oplus y_0 \oplus y_1 y_2 \oplus y_2 \oplus y_3, \\
x_3 &= y_0 \oplus y_1 \oplus y_2.
\end{aligned}$$

It can be found that x_3 is linear in the output bits. Combined with the special construction of the linear layer, after reversing r_0 rounds, it is feasible to obtain at least $128 \div 4 = 32$ expressions in terms of the key whose algebraic degree is upper bounded by 2^{r_0-1} . Moreover, it is found with Sage that the SBox can be represented as 21 quadratic Boolean expressions and 1 linear equation in terms of the input bits and output bits. Therefore, for a fixed key, the primitive can be represented as $32 \times 21 \times 32 = 21504$ quadratic Boolean expressions in terms of $32 \times 7 \times 32 = 7168$ variables. Even though the system of equations is over-defined, as far as we know, there does not exist an efficient algorithm to solve it.

With the help of Sage again, we also compute the Boolean expressions (in ANF) of output bits of BAKSHEESH after 1, 2, 3 and 4 rounds. The average numbers of monomials are respectively 4.25, 29.00, 1481.75 and 2572441.00, while the lowest algebraic degrees are 2, 4, 8 and 16, respectively. Both figures show an exponential growth.

¹⁶Note that the input differences/masks 0 and 8 behave analogously with respect to differential/linear attack, despite being the polar opposite of each other (i.e., tautology and contradiction). This boils down to the fact that, as far as the attacker is concerned, the two events ('always happens' and 'never happens') are deterministic in nature.

¹⁷From a traditional point-of-view, such a property is typically considered undesirable in a cipher.

¹⁸<https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/sbox.html>.

Table 6: Optimal differential and linear trails for 22-round BAKSHEESH

Round	Differential				Linear			
1 [‡]	00000000	00000000	90049000	00000000	00000000	00000000	00000409	0000000a
2	0000c400	00000000	00000000	00000000	00000000	00000000	0000a000	00000401
3	00000000	00000000	09000000	00000000	00000000	00000000	0000000a	00000405
4	00000000	00001000	00000000	00000000	00000000	00000000	0000000a	00000105
5	00040000	00020000	00000000	00000000	00000000	00000000	0000000a	00000105
6	00000000	00000000	20200000	10100000	00000000	00000000	0000000a	00000105
7	00005010	000000a0	0000a010	000000a0	00000000	00000000	0000000a	00000105
8	00000000	00000000	04040404	0a000a00	00000000	00000000	0000000a	00000105
9	00000000	00000000	0000aa00	00001144	00000000	00000000	0000000a	00000105
10	0000060e	0000000b	00000000	00000000	00000000	00000000	0000000a	00000105
11	00000000	00000000	0a020000	00000000	00000000	00000000	0000000a	00000105
12	00000000	00000000	0000a000	00005000	00000000	00000000	0000000a	00000105
13	00000404	00000002	00000000	00000000	00000000	00000000	0000000a	00000105
14	00000000	00000000	0a020000	01010000	00000000	00000000	0000000a	00000105
15	000000a0	00004010	000000a0	00005010	00000000	00000000	0000000a	00000105
16	00000000	00000000	04040404	000a000a	00000000	00000000	0000000a	00000105
17	0000aa00	00001144	00000000	00000000	00000000	00000000	0000000a	00000105
18	060e0000	000b0000	00000000	00000000	00000000	00000000	0000000a	00000105
19	00000000	00000000	a0100000	00000000	00000000	00000000	0000000a	00000105
20	00005000	00000000	00000000	00005000	00000000	00000000	0000000a	00000105
21	00000004	00000000	00000000	08000000	00000000	00000000	0000000a	00000105
22	08000020	00000010	00000080	00000040	00000808	00000000	00000008	00000005

‡: Plaintext

5.1.3 Integral Attack The concept of the integral attack is first proposed in [52], then it is formalised in [75]. Relatively recently, the *division property* that can find the integral distinguisher more efficiently, is proposed in [102]. We first search the integral distinguisher based on the bit-based division property by the MILP searching tool proposed by in [106] from both forward and backward directions.

In the MILP model of BAKSHEESH, we only need to construct the propagation of the division property through the SBox since the bit permutation just permutes variables. The propagation table of the division property for the SBox is shown in Section A, where $u = (u_3, u_2, u_1, u_0)$ and $v = (v_3, v_2, v_1, v_0)$ donate the input and output division property, respectively. Following this, we construct the constraints for the propagation of the division property for the SBox as follows:

$$\begin{aligned}
 -2u_3 - u_2 - u_1 - u_0 + 2v_3 + 2v_2 + 2v_1 + 2v_0 &\geq 0, \\
 -u_3 - 2u_2 - 3u_1 - 3u_0 + 2v_3 + v_2 + v_1 + v_0 + 4 &\geq 0, \\
 -2u_3 - 3u_2 - 2u_1 - 3u_0 + v_3 + v_2 + 2v_1 + v_0 + 5 &\geq 0, \\
 -u_3 - 3u_2 - 3u_1 - 2u_0 + v_3 + 2v_2 + 2v_1 + v_0 + 4 &\geq 0, \\
 -u_3 - u_2 - 2u_1 - u_0 + v_3 + v_0 + 3 &\geq 0, \\
 -2u_3 - u_2 - u_1 - 2u_0 + 3v_3 + 3v_2 + 2v_1 + 3v_0 &\geq 0.
 \end{aligned}$$

As a result of the forward direction, we observe 13-round integral distinguishers with 127 active bits (these 127 bits taking all possible 2^{127} values). Besides, there is no integral distinguisher on the 14-round of BAKSHEESH with 127 active bits. In the backward direction, we obtain the same results as the forward direction, i.e., we find 13-round integral distinguishers with 127 active bits. No integral distinguisher on the 14-round of BAKSHEESH with 127 active bits is found. One of the 13-round integral distinguishers on the forward direction is that the 0th input bit is constant and all 128 output bits are balanced.

5.1.4 Impossible Differential Attack The impossible differential attack [32] exploits the pair of input and output differences that the input difference never reaches the output difference. In this context, it is important to talk about “full diffusion” along with that. Full diffusion means that any 1-bit at the input can affect all 128-bits at the output. In other words, to achieve full diffusion implies that algebraic expression of all output bits contain all input bits. Therefore, we can roughly estimate the maximum number of rounds that can have the impossible differences by that. Since BAKSHEESH respectively achieves full diffusion after 4 and 5 rounds

Equations (1) and (2) can not hold for the full round. Consequently, the above weak-key distinguisher cannot hold.

To prove the resistance against the non-linear invariant subspace attack [103], we search for the quadratic non-linear invariant for the SBox and no such invariant is found. This demonstrates that our primitive is secure against the non-linear invariant subspace attack.

In aggregate, the tap positions, round constants and key scheduling are carefully chosen to prevent invariant subspace attack (and similar attacks). In other words, our choice prevents self-similarity of key scheduling.

5.1.6 Other Attacks

5.1.6.1 Machine Learning Assisted Attacks Machine learning (ML) has recently emerged as a prominent method for classical cryptanalysis [15, 16, 62]. In that respect, we note that the design of BAKSHEESH directly follows from that of GIFT-128; and at the moment, we are not aware of any impeding threat to GIFT-128 by ML. In future, if such an attack becomes a practicality, GIFT-128 and BAKSHEESH will likely be affected equally. Keeping this in mind, we would recommend to increment the number of rounds for BAKSHEESH as a quick patch, should the need arise. If the same patch is adopted by the GIFT designers, we are confident that the (patched) BAKSHEESH will retain its 20% efficiency over (patched) GIFT-128.

5.1.6.2 Related Key Attack Similar to GIFT, we do not claim any related key security, even though we are not aware of any such attack. Following the designers' statement [22, Section 2], "in case one wants to protect against related-key attacks as well, we advise to double the number of rounds"; we also recommend to double the number of rounds (the round constants would reiterate from 2 to 8), should there be any necessity. In that case too, BAKSHEESH would maintain its 12.50% smaller number of rounds over GIFT-128.

5.1.6.3 Meet-in-the-Middle (MitM) Attack BAKSHEESH achieves full diffusion after 4 and 5 rounds from the forward (encryption) and backward (decryption) directions, respectively. It enables us to claim that any inserted key bit non-linearly affects all bits of the state after 4 and 5 rounds in the forward and the backward directions, respectively. Thus, the number of rounds used for the *Partial Matching* (PM) [6] is upper bounded by 8 ($= (4 - 1) + (5 - 1) + 1$). The condition for the *Initial Structure* (IS) [94] is that key differential trails in the forward direction and those in the backward direction do not share active non-linear components. For BAKSHEESH, since any key differential affects all SBoxes after at least 6 rounds in the forward and the backward directions, there is no such differential which shares active SBox in more than 6 rounds. Thus, the number of rounds used for IS is upper bounded by 5. Assuming that the splice-and-cut technique [6] allows an attacker to add more 5 rounds in the worst case, at most 18-round ($= 8 + 5 + 5$) MitM attack may be feasible. Thus, we expect that the full-round BAKSHEESH is secure against MitM attacks.

5.1.6.4 Biclique Attack The claim against the biclique attack [36] on BAKSHEESH is analogous to that of SKINNY [24, Section 4.2]. More specifically, this attack slightly improves the complexity of exhaustive search by computing only a part of encryption algorithm, i.e., the improved factor is often evaluated by the ratio of the number of SBoxes involved in the partial computation to all SBoxes in the cipher. This attack is unavoidable (which is also the case for GIFT-128), and we do not think that such attack will be serious vulnerability in future. Therefore, we do not claim any security against the biclique attack, although the impact of it against BAKSHEESH will be marginal.

5.2 Side-Channel Attacks

Theoretically speaking, it can be shown under CPA [42] with noiseless assumption, the BAKSHEESH SBox creates an ambiguity between the correct input x and $x \oplus 8$, $\forall x \in \{0, \dots, \mathbf{f}\}$, thanks to the property mentioned in Section 2.3.2. In practice, it appears that the input recovery from the SBox by CPA is not possible with our experimental set-up (described subsequently as an interesting/non-trivial observation). However, this does not constitute any inherent side channel security, since it is, in theory, possible to retrieve the key by using more sophisticated equipment or more traces. That said, the main claim of BAKSHEESH against SCA comes from its unique advantage over other traditional ciphers when adopting an existing SCA countermeasure, as described in Section 6.2. This stems from the facts that the design is lightweight, and most notably it uses an SBox with a low AND count.

Side-Channel Profile of Unprotected Cipher We show an interesting observation relating to the SCA profile of the SBox used in BAKSHEESH compared to that of GIFT [22]. In our analysis, it seems not possible to recover the key from the leakage of the BAKSHEESH SBox; but the same method recovers the key from the leakage of the GIFT SBox. Note again that, this does not amount for any inherent SCA security BAKSHEESH, and a suitable countermeasure is to be adopted for this purpose. In the following, we describe two experiments, one in software (key recovery is possible for GIFT-128, but not possible for BAKSHEESH despite using the same set-up/method), and the other in hardware (key recovery is not successful in either cipher).

Software Our test-bed consists of the following components: Sakura GII, Sakura W (Funcard including ATMega8515) and MSO9104A digital oscilloscope (sampling frequency of 1 GHz with 16 bits/sample). The signal-to-noise ratio (SNR) for 8-bit AVR-based smart cards is known to be high and therefore we expect a high level of leakage coming from unprotected cipher implementations. We measure power traces of GIFT-128 and BAKSHEESH encryption processes where the observed intermediate value is $S(p \oplus k)$ for $32 \times$ GIFT SBox and $32 \times$ BAKSHEESH SBox, where p is the known input to the SBox S and k is the unknown key. Note that, we assume the 4-bit key k is XORed for the simplicity of our analysis, that is an anachronism as GIFT only uses half round key XOR. Altogether, $100,000 \approx 2^{16.61}$ power traces are measured for each case with the best possible resolution on the oscilloscope. The attack is performed for all the 32 SBoxes, and based on that, we infer the following — no part of the unknown key for BAKSHEESH is revealed but the full key is retrieved for GIFT-128.

Hardware In order to study SCA including the hardware implementations, we utilize Sakura X (Kintex-7 FPGA) board in our test-bed. We implement the whole GIFT-128 and BAKSHEESH encryption processes in VHDL and measure the power consumption. However, being lightweight, GIFT-128 and BAKSHEESH provides substantially small leakage; and therefore, it is not possible to measure the power consumption utilizing our test-bed directly. Consequently, CPA on any of the ciphers are not successful. Following the literature [60, 69], it is necessary to use some tricks that can bring the signal above the noise level in such a situation, for instance to implement more parallel encryption cores or some amplifier (the Sakura X board is lacking an on-board amplifier). This experiment is conducted just to gain some insight on the BAKSHEESH SBox with respect to SCA, and as such, we leave a more sophisticated study as a future work.

6 Implementation and Performance

6.1 Unprotected Benchmark

6.1.1 Hardware In the following, we tabulate the synthesis results for unprotected hardware implementation of BAKSHEESH and compare it with that of GIFT-128 (full round key XOR). For the purpose of benchmark, we report costs incurred in FPGA (Xilinx Ultrascale+ and Xilinx Kintex) as well as ASIC (TSMC 65nm) platforms. The ASIC synthesis is performed using Synopsys Design Compiler version R-2020.09-SP5, place and route using Cadence Innovus V19.10-P002 and FPGA synthesis utilising Xilinx Vivado 2020.2. See Table 7 for an overview. The same HDL implementation is used to benchmark in ASIC and FPGA, the round keys are computed on-the-fly.

Table 7: Unprotected hardware benchmarks BAKSHEESH and GIFT-128 (half round key XOR)
(a) ASIC (TSMC 65nm) (b) FPGA

	Area (GE)	Critical Path (ns)	Max. Freq. (MHz)	LUT	F/F	UltraScale+*	Kintex*
BAKSHEESH	3554	1.282	780	268	269	847	543
GIFT-128	3264	1.818	550	235	262	830	546

*: Maximum frequency (MHz)

ASIC One may note from Table 7(a) that BAKSHEESH takes more area but offers higher frequency.

FPGA We observe from Table 7(b) that the number of LUT utilised for BAKSHEESH is very close to that of GIFT-128²⁰.

6.1.2 Software (Fix-slice) The intricate bit permutation utilized in both GIFT and BAKSHEESH tends to be less efficient in software compared to its operation in hardware. However, an approach known as fix-slicing, as used in [1], addresses this issue, implementing it more effectively in software. Within the framework of fix-slicing, the cipher state is depicted as a collection of slice matrices. Each slice matrix embodies bits of the cipher state. A singular slice matrix is persistently stationary, thereby ensuring it never moves. In contrast, the other slice matrices undergo a mixing process where their rows and columns are interchanged. This procedure entails transposing each individual slice matrix and then shuffling the left and right matrices of each slice. It is simpler and more efficient than the operations traditionally used in the GIFT representation. Furthermore, the round keys and round constants are adjusted to comply with the new manner in which the bits are arranged. As BAKSHEESH also employs the same permutation as Gift, it can be implemented using fix-slicing, thereby making it more efficient.

We implement and benchmark both GIFT-128 and BAKSHEESH in fix-sliced manner [1] which is probably the most efficient software representation for 32-bit devices. For this benchmark, we use ARM Cortex-M3 (Atmel SAM3X8E) device clocked at 84 MHz. The results are consolidated in Table 8. See Section A for the BAKSHEESH SBox implementation.

Similar to the fix-slicing of GIFT [1], the key scheduling is not taken into account as it is assumed that round keys are precomputed and stored in RAM. We present embedded C (with compiler flag `-O3`) and two assembly implementations (the fast assembly implementation considers best execution time, and the compact assembly implementation considers best code size). The results are stated in Table 8(a). For a fair comparison, we replaced the half round key of GIFT-128 with a full round key and measured its performance. We observed that BAKSHEESH outperforms GIFT-128 in both C and assembly. This advantage seems to derive from the efficiency of BAKSHEESH in XORing the full state with the round key, which uses fewer rounds (35 vs. 40).

6.1.2.1 Reference Implementation When fix-slice is not considered, we observe from our reference C code that BAKSHEESH encryption achieves the speed of 13310 cycles/bytes whereas that of GIFT-128 (half round key XOR) is 19214 on ARM Cortex-M3. In both, the round keys are computed on-the-fly. GIFT-128 is 44% slower than BAKSHEESH because of the advantages of BAKSHEESH: simple key schedule and fewer rounds. The reference code is on-the-fly, so there seems to be a big difference. Compared to the A benchmark, BAKSHEESH works efficiently in software when fix-slicing is applied. And in fix-slicing, the key scheduling is pre-computed, so the performance difference with GIFT-128 is reduced.

6.1.2.2 Masked Implementation We further implement the first-order masking, by adopting the secure operations from [1, Section 6.2]. Similar to that work, we do not consider the cost of random number generation, nor the efficacy is verified by us. The results are stated in Table 8(b), whence the advantage of BAKSHEESH becomes apparent (GIFT-128 is 48% slower). This is where BAKSHEESH using an SBox that requires 3 non-linear gates (one less than that of GIFT-128), and having 5 fewer rounds comes into play. The scale of efficiency will increase with the order of masking.

Table 8: Fix-sliced benchmarks of BAKSHEESH and GIFT-128 (full round key XOR) on ARM Cortex-M3
(a) Unprotected (b) First order masking

		BAKSHEESH	GIFT-128			BAKSHEESH	GIFT-128
C (-O3)	Clocks (Cycles/Byte)	128 (0.82×)	156 (1.00×)	ASM	Clocks	210	312
	Code Size (Bytes)	1240 (0.82×)	1504 (1.00×)		(Cycles/Byte)	(0.67×)	(1.00×)
ASM (compact)	Clocks (Cycles/Byte)	126 (0.82×)	154 (1.00×)	Code Size (Bytes)	2088	2504	
	Code Size (Bytes)	688 (0.97×)	712 (1.00×)		(0.83×)	(1.00×)	
ASM (fast)	Clocks (Cycles/Byte)	113 (0.81×)	140 (1.00×)				
	Code Size (Bytes)	4656 (0.83×)	5582 (1.00×)				

²⁰The excess usage of resources comes from the extra cost for the full round key XOR.

6.2 Side-Channel Protected Benchmark: Threshold Implementation

6.2.1 SBox The coordinate functions of the BAKSHEESH SBox (306DB58ECF924A71) have the maximum algebraic degree of 2 (see Section 4.2.2). Therefore, 3-sharing is sufficient (which is rare, if not unique, among other commonly used 4×4 SBoxes).

For a comparative study, a benchmark for several 4×4 SBoxes are given in Table 9, where the number of shares is taken as the algebraic degree plus 1. The prerequisite theory of threshold and the implementation details are omitted here for the sake of brevity, and can be found in [17]. Each of the SBoxes is used in a cipher construction, except for 048AFC691EBD7532 which is presented in [60, Section 3.4]. It can be seen from Table 9 that the cost is few fold less than other SBoxes; in terms of total number of monomials, and GE in the STM 130nm ASIC library (it is mostly reproduced from [17, Table 2]). The Boolean polynomial expressions for the implementation of the BAKSHEESH SBox; consisting of 71 XOR operations, 54 AND operations and 83 monomials; are given in Section C. Also note that, the figures given in Table 9 are done for the minimal order.

Table 9: Minimal order threshold (without decomposition) cost of some 4×4 SBoxes

		# Naïve AND	Shares	# Monomials	ASIC*	FPGA*
BAKSHEESH	306DB58ECF924A71	6	3	83 (1.00×)	166 (1.00×)	16 (1.00×)
GIFT [22]	1A4C6F392DB7508E	10	4	265 (3.19×)♣	526 (3.17×)	102 (6.38×)
PRESENT [38]	C56B90AD3EF84712	23		666 (8.02×)	1132 (6.82×)	56 (3.50×)
PRINCE [39]	BF32AC916780E5D4	28		731 (8.81×)	991 (5.97×)	77 (4.81×)
PICCOLO [98]	E4B238091A7F6C5D	16		399 (4.81×)	645 (3.89×)	55 (3.44×)
SKINNY-64 [24]	C6901A2B385D4E7F	16		398 (4.80×)	723 (4.36×)	139 (8.69×)
TWINE [100]	COFA2B9583D71E64	25		626 (7.54×)	723 (4.36×)	41 (2.56×)
PYJAMASK-128 [63]	2D397BA6E0F4851C	15		373 (4.49×)	640 (3.86×)	60 (3.75×)
Gao-Roy-Oswald [60]	048AFC691EBD7532	36		988 (11.90×)	1658 (9.99×)	162 (10.13×)

★: Gate equivalent in STM 130nm (HCMOS9GP, Cadence v14.20 2016)

*: Number of LUTs in Kintex-7 (Vivado 2020)

♣: Used in [69, Appendix B]

One may expect the relative cost of the BAKSHEESH SBox will be more significant as the order increases.

6.2.2 Encryption The threshold protected benchmark for BAKSHEESH encryption is given (under Kintex FPGA TSMC 65nm ASIC libraries) in Table 10. The round keys are computed on-the-fly and the implementation are optimised. Indeed, both implementations directly follow the same design principles of the 4-share implementation (Profile 6) of GIFT-128 [69], and hence are comparable.

Table 10: Threshold benchmarks for BAKSHEESH and GIFT-128 (half key XOR)
(a) ASIC (TSMC 65nm) (b) FPGA

	Area (GE)	Critical Path (ns)	Max. Freq. (MHz)	LUT	F/F	Kintex*
BAKSHEESH	11811	2.09	506	1247	526	598
GIFT-128	28709	1.602	624	3254	646	463

*: Maximum frequency (MHz)

By looking at the benchmark from [69, Table III] and comparing it with Table 10(a), we see that GIFT-128 takes 28709 GE in TSMC 65nm ASIC library. Thus, GIFT-128 takes about $2.43\times$ area than BAKSHEESH in a comparable implementation. Further, at the operating frequency of 10 MHz for the protected 65nm design, the power consumption is only 0.115 mW. Likewise, Table 10(b) shows significantly improved performance of BAKSHEESH compared to GIFT-128 in threshold.

Figure 3 shows ASIC implementation results after place and route on the same ASIC library. Comparing the two images, it is evident that the SBox takes a large amount of area in the protected implementation. Further, Figure 4 shows component-wise break-up for area requirement for the same, the unprotected version in Figure 4(a) and the protected (3-share threshold) in Figure 4(b).

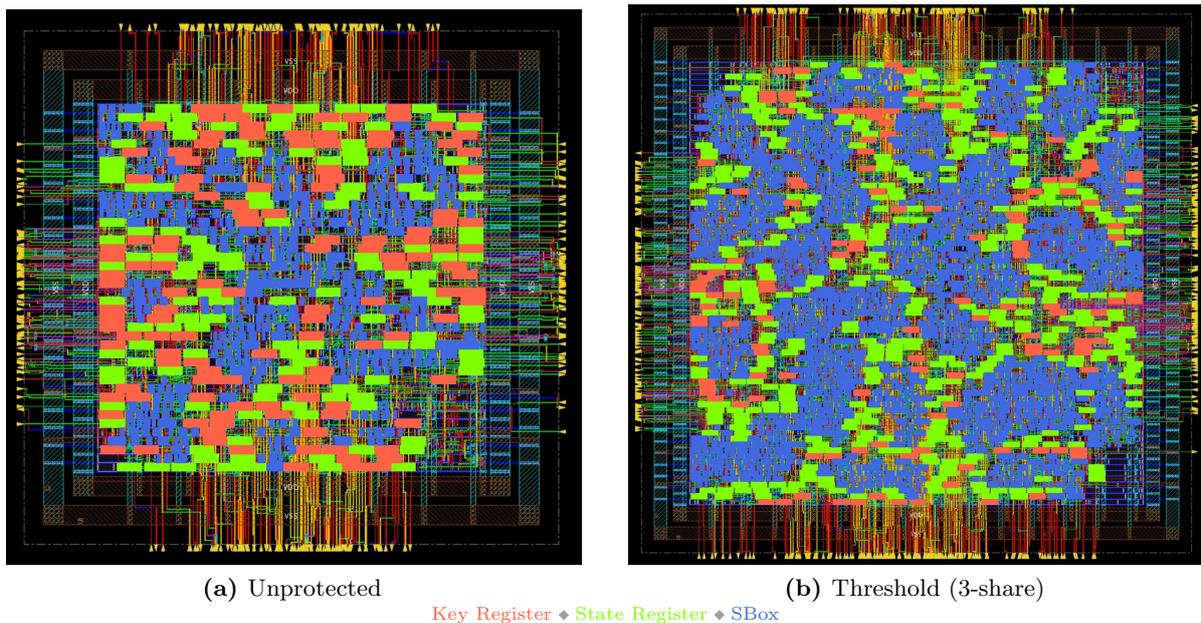


Figure 3: BAKSHEESH implementation as ASIC layout (after place and route)

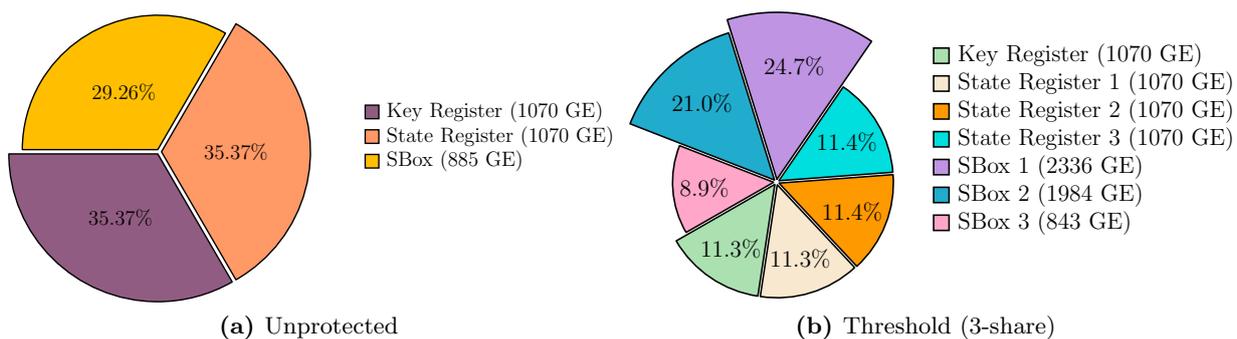


Figure 4: Area comparison of BAKSHEESH implementations in ASIC

7 Conclusion

Pushing the envelope of GIFT-like design, here we present a lightweight cipher, BAKSHEESH. It brings certain unique features to the table; most notably, this is the first cipher to make use an SBox with a non-trivial LS in the classical setting. Not only it improves in terms of number of rounds, but also reduces the cost for implementation on multiple hardware and software platforms over its predecessor GIFT-128 [22]. Overall, BAKSHEESH offers the following advantages over other lightweight ciphers:

- Lightweight.** BAKSHEESH takes design ideas from GIFT-128 [22] (hence optimisations like [1] are readily adoptable) and takes 12.50% smaller number of rounds compared to GIFT-128. Thanks to the meticulously crafted design, BAKSHEESH stands toe-to-toe against other lightweight ciphers, if not outperforming those in hardware as well as software benchmarks.
- Low AND Count/Efficient Side Channel Protection.** Symmetric key ciphers with low AND complexity are pitched suitable for certain applications, most notably for easier adoption of side channel countermeasures. In general, the direction is not new as some ciphers like [61, 64, 91] are designed specifically to fill in that niche. Additionally, some other ciphers [3, 43, 56] are designed to have lower AND cost, although the primary objective is to have leverage in other domain-specific niches like FHE/MPC/ZK. BAKSHEESH has one of the least, if not the least, AND counts among traditional lightweight ciphers; and as shown here in Section 6.2 offers an edge in efficient adoption of side channel countermeasure. It probably

does not compare well against the domain specific ciphers in terms of AND complexity as of yet, but we expect that the future iterations would be further streamlined to gradually close-in/match.

For the consideration of the community, a few interesting research problems are listed next. First, one may consider a cipher with a mirror-like design like PRINCE [39], QARMA [8] or MANTIS [24] for faster encryption + decryption²¹. Second, a design with a non-bit-permutation-based linear layer like SKINNY [24] can be considered in conjunction with an SBox having 1 non-trivial LS.

A More Cryptographic Properties of BAKSHEESH SBox

The LAT for the BAKSHEESH SBox (306DB58ECF924A71) is given in Table 11 with the same formatting as the DDT. Denoting the input bits as (x_3, x_2, x_1, x_0) and output bits as (y_3, y_2, y_1, y_0) , it holds with probability 1 that, (corresponding to input mask = 8, output mask = 7): $x_2 \oplus y_2 \oplus y_1 \oplus y_0 = 0$.

Similar to GIFT [22, Table 12], the division property table of the BAKSHEESH SBox is given in Table 12, where the position marked with ✖ means the propagation from ω to Ω is feasible (a trivial propagation is marked by ✕).

Further, the autocorrelation table (ACT) is shown in Table 13. For a mapping $F : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^n$, the ACT is defined as the $2^m \times 2^n$ matrix: $\text{ACT}[\psi, \Psi] = \sum_{x \in \mathbb{F}_2^m} (-1)^{\Psi \cdot F(x) \oplus \psi \cdot F(x \oplus \psi)}$, $\forall \psi \in \mathbb{F}_2^m, \forall \Psi \in \mathbb{F}_2^n$, where \cdot denotes the dot product.

Table 11: LAT for BAKSHEESH SBox

$\Gamma \backslash \gamma$	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1			+4		-4					-4		-4			
2			+4			-4				+4					+4
3					-4	-4						+4			-4
4									+4	-4		+4			+4
5			-4		-4					-4					+4
6			+4			+4				-4		+4			
7					+4	-4				-4	-4				
8							+8								
9		-4		+4							-4		-4		
a	-4			+4				+4						+4	
b	-4	-4						-4			+4				
c								+4			+4		-4	+4	
d		-4		-4				+4							-4
e	+4			+4							+4				-4
f	-4	+4											-4	-4	

B Implementations of BAKSHEESH SBox

The CPU-optimised implementation (for ARM Cortex-M3²²) is given in Code 1.1, which is obtained by using the LIGHTER tool [72] and requires 10 instructions. The ASIC (TSMC 65nm) implementation is given in Code 1.2, which incurs cost of 20.5 GE.

Similarly, a reversible implementation with 8 gates (4 Toffoli, 3 CNOT, 1 NOT) is presented in Code 1.3, which is optimised for gate count by using LIGHTER-R [54].

²¹The construction makes the ciphers suitable for memory encryption (since the critical path is the maximum between encryption/decryption).

²²It can be stated that, NOR2 is included in the instruction set, but NAND2 is not.

Table 12: DPT for BAKSHEESH SBox

$\omega \backslash \Omega$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
3			x		x	x	x	x	x	x	x	x	x	x	x	x
4	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
5	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x
6			x	x	x	x	x	x	x	x	x	x	x	x	x	x
7												x		x	x	x
8	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
9			x		x	x	x		x	x	x	x	x	x	x	x
a			x		x	x	x		x	x	x	x	x	x	x	x
b								x		x	x	x	x	x	x	x
c			x		x	x	x		x	x	x	x	x	x	x	x
d			x		x	x			x	x		x	x	x	x	x
e			x		x	x			x	x		x	x	x	x	x
f																x

Table 13: ACT for BAKSHEESH SBox

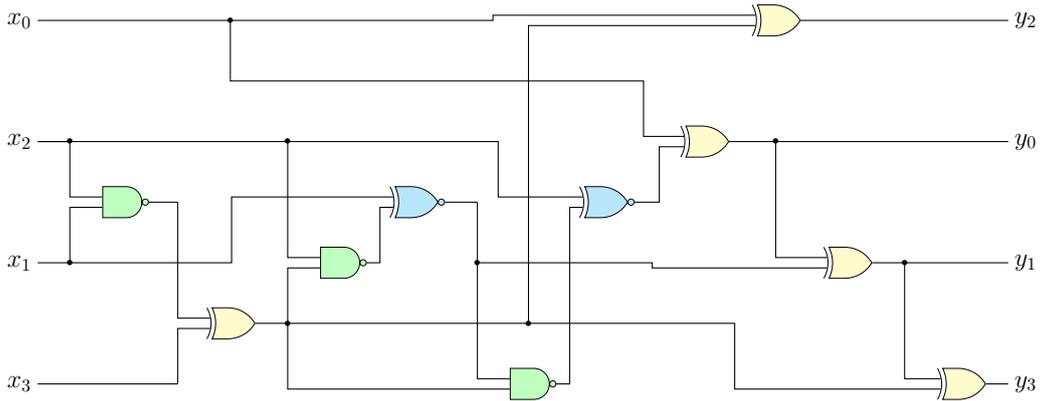
$\psi \backslash \Psi$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	+16	+16	+16	+16	+16	+16	+16	+16	+16	+16	+16	+16	+16	+16	+16	+16
1	+16	0	-16	0	0	-16	0	+16	0	0	0	0	0	0	0	0
2	+16	-16	0	0	0	0	-16	+16	0	0	0	0	0	0	0	0
3	+16	0	0	-16	-16	0	0	+16	0	0	0	0	0	0	0	0
4	+16	0	0	0	0	0	0	+16	0	-16	0	0	0	0	-16	0
5	+16	0	0	0	0	0	0	+16	0	0	0	-16	-16	0	0	0
6	+16	0	0	0	0	0	0	+16	-16	0	0	0	0	0	0	-16
7	+16	0	0	0	0	0	0	+16	0	0	-16	0	0	-16	0	0
8	+16	-16	-16	+16	-16	+16	+16	-16	-16	+16	+16	-16	+16	-16	-16	+16
9	+16	0	+16	0	0	-16	0	-16	0	0	0	0	0	0	0	0
a	+16	+16	0	0	0	0	-16	-16	0	0	0	0	0	0	0	0
b	+16	0	0	-16	+16	0	0	-16	0	0	0	0	0	0	0	0
c	+16	0	0	0	0	0	0	-16	0	-16	0	0	0	0	+16	0
d	+16	0	0	0	0	0	0	-16	0	0	0	+16	-16	0	0	0
e	+16	0	0	0	0	0	0	-16	+16	0	0	0	0	0	0	-16
f	+16	0	0	0	0	0	0	-16	0	0	-16	0	0	+16	0	0

```

1 // Initial reordering
2 F[0] = X[0];
3 F[1] = X[2];
4 F[2] = X[3];
5 F[3] = X[1];
6 // Actual implementation (10 lines)
7 t0 = NOR2 (F[1], F[2]);
8 F[3] = XOR2(F[3], t0);
9 t1 = NOR2 (F[2], F[3]);
10 F[1] = XOR2(F[1], t1);
11 t2 = NOR2 (F[1], F[3]);
12 F[2] = XOR2(F[2], t2);
13 F[1] = XOR2 (F[1], F[0]);
14 F[3] = XOR2 (F[3], F[1]);
15 F[0] = XOR2 (F[0], F[2]);
16 F[2] = XOR2 (F[2], F[3]);
17 // Final reordering
18 X[0] = F[1];
19 X[1] = F[0];
20 X[2] = F[2];
21 X[3] = F[3];

```

Code 1.1: BAKSHEESH SBox in software (ARM Cortex-M3) using LIGHTER

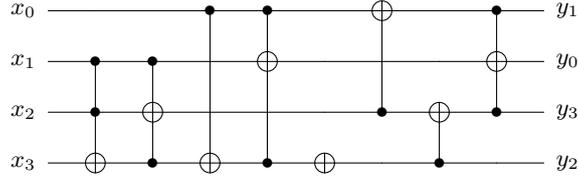


Code 1.2: BAKSHEESH SBox in ASIC (TSMC 65nm) using LIGHTER

C Threshold Implementation (Combinational SBox)

The algorithm used to find the BAKSHEESH SBox’s threshold implementation, as well as that of the other SBoxes’ is based on what is described in [17]. This takes as input the given SBox in its coordinate function representation. It follows the *without-decomposition* method discussed in [69, Section III.A]. In other words, the threshold implementation of the SBox only produces combinational circuits.

In the following, we show a threshold implementation of the BAKSHEESH SBox with 3 shares. The input variables (x_0, x_1, x_2, x_3) are each split into 3 shares: $x_i = x_{i,0} \oplus x_{i,1} \oplus x_{i,2}$. Similarly the output variables (y_0, y_1, y_2, y_3) are each split into 3 shares: $y_i = y_{i,0} \oplus y_{i,1} \oplus y_{i,2}$:



Code 1.3: BAKSHEESH SBox in reversible logic using LIGHTER-R

$$\begin{aligned}
y_{0,0} &= x_{0,1}x_{2,1} \oplus x_{0,1}x_{2,2} \oplus x_{0,1} \oplus x_{0,2}x_{2,1} \oplus x_{0,2}x_{2,2} \oplus x_{0,2} \oplus x_{1,1} \oplus x_{1,2} \oplus x_{3,2} \oplus 1 \\
y_{0,1} &= x_{0,0}x_{2,0} \oplus x_{0,0}x_{2,2} \oplus x_{0,0} \oplus x_{0,2}x_{2,0} \oplus x_{1,0} \oplus x_{3,0} \\
y_{0,2} &= x_{0,0}x_{2,1} \oplus x_{0,1}x_{2,0} \oplus x_{3,1} \\
y_{1,0} &= x_{0,1} \oplus x_{0,2} \oplus x_{1,1}x_{2,1} \oplus x_{1,1}x_{2,2} \oplus x_{1,2}x_{2,1} \oplus x_{1,2}x_{2,2} \oplus x_{3,2} \oplus 1 \\
y_{1,1} &= x_{0,0} \oplus x_{1,0}x_{2,0} \oplus x_{1,0}x_{2,2} \oplus x_{1,2}x_{2,0} \oplus x_{3,0} \\
y_{1,2} &= x_{1,0}x_{2,1} \oplus x_{1,1}x_{2,0} \oplus x_{3,1} \\
y_{2,0} &= x_{0,1}x_{2,1} \oplus x_{0,1}x_{2,2} \oplus x_{0,2}x_{2,1} \oplus x_{0,2}x_{2,2} \oplus x_{1,1}x_{2,1} \oplus x_{1,1}x_{2,2} \oplus x_{1,1} \oplus x_{1,2}x_{2,1} \oplus x_{1,2}x_{2,2} \oplus x_{1,2} \oplus x_{3,2} \\
y_{2,1} &= x_{0,0}x_{2,0} \oplus x_{0,0}x_{2,2} \oplus x_{0,2}x_{2,0} \oplus x_{1,0}x_{2,0} \oplus x_{1,0}x_{2,2} \oplus x_{1,0} \oplus x_{1,2}x_{2,0} \oplus x_{3,0} \\
y_{2,2} &= x_{0,0}x_{2,1} \oplus x_{0,1}x_{2,0} \oplus x_{1,0}x_{2,1} \oplus x_{1,1}x_{2,0} \oplus x_{3,1} \\
y_{3,0} &= x_{0,1}x_{1,1} \oplus x_{0,1}x_{1,2} \oplus x_{0,1}x_{2,1} \oplus x_{0,1}x_{2,2} \oplus x_{0,2}x_{1,1} \oplus x_{0,2}x_{1,2} \oplus x_{0,2}x_{2,1} \oplus x_{0,2}x_{2,2} \oplus x_{2,1} \oplus x_{2,2} \oplus x_{3,2} \\
y_{3,1} &= x_{0,0}x_{1,0} \oplus x_{0,0}x_{1,2} \oplus x_{0,0}x_{2,0} \oplus x_{0,0}x_{2,2} \oplus x_{0,2}x_{1,0} \oplus x_{0,2}x_{2,0} \oplus x_{2,0} \oplus x_{3,0} \\
y_{3,2} &= x_{0,0}x_{1,1} \oplus x_{0,0}x_{2,1} \oplus x_{0,1}x_{1,0} \oplus x_{0,1}x_{2,0} \oplus x_{3,1}
\end{aligned}$$

D More Information on BAKSHEESH Construction

D.1 Structure and Test Vectors

A schematic of BAKSHEESH for 2 rounds (out of 35) is given in Figure 5. The SBoxes are numbered (from 0 to 31), so are the bit positions (from 0 to 127), for better clarity. The coloured lines indicate the permutation (linear) layer. By \oplus , it is indicated that the corresponding key bits are XORed. The round constant XORs are shown by \oplus (at tap positions: 8, 13, 19, 35, 67, 106). A few test vectors for BAKSHEESH are given in Table 14.

D.2 Details on Round Constants and Tap Positions

This is the full sequence of the 6-bit round constants (permutation of integers from 2 to 63): 2, 33, 16, 9, 36, 19, 40, 53, 26, 13, 38, 51, 56, 61, 62, 31, 14, 7, 34, 49, 24, 45, 54, 59, 28, 47, 22, 43, 20, 11, 4, 3, 32, 17, 8, 37, 18, 41, 52, 27, 12, 39, 50, 57, 60, 63, 30, 15, 6, 35, 48, 25, 44, 55, 58, 29, 46, 23, 42, 21, 10, 5. It misses 0 (due to the LFSR) and 1 (due to the appended bit which toggles at each round). Without the toggling bit, the 5-bit LFSR produces the following sequence: 1, 16, 8, 4, 18, 9, 20, 26, 13, 6, 19, 25, 28, 30, 31, 15, 7, 3, 17, 24, 12, 22, 27, 29, 14, 23, 11, 21, 10, 5, 2.

To decide on the tap positions for the round constant addition, the digits of π are used. The digits (after the decimal point) when grouped into consecutive 12-bits, respectively result in: 243, f6a, 888, 5a3, 08d, 313. Then, each of the groups is taken $\cdot \pmod{128}$, and finally sorted to yield: 8, 13, 19, 35, 67, 106.

As it can be seen from Figure 5, the tap positions are located on all the 4-bits with respect to an SBox (position 8 is at the 0th bit of the third SBox; position 13 is at the 1st bit of the fourth SBox; positions 19, 35 and 67 are at the 3rd bit of the corresponding SBoxes, and position 106 is at the 2nd bit of the twenty-seventh SBox). This is different from GIFT, where the round constants are always added at the 3rd bit of the corresponding SBoxes.

Going through the fix-slicing of GIFT [1], it seems that having the taps at 3 (mod 4) position has the following advantage that the round constants can be processed with only one XOR instruction in the fix-slicing expression. BAKSHEESH deliberately chooses a different round constant subroutine (with justification and also with verification against the invariant subspace attack), despite its toll on fix-slice benchmark.

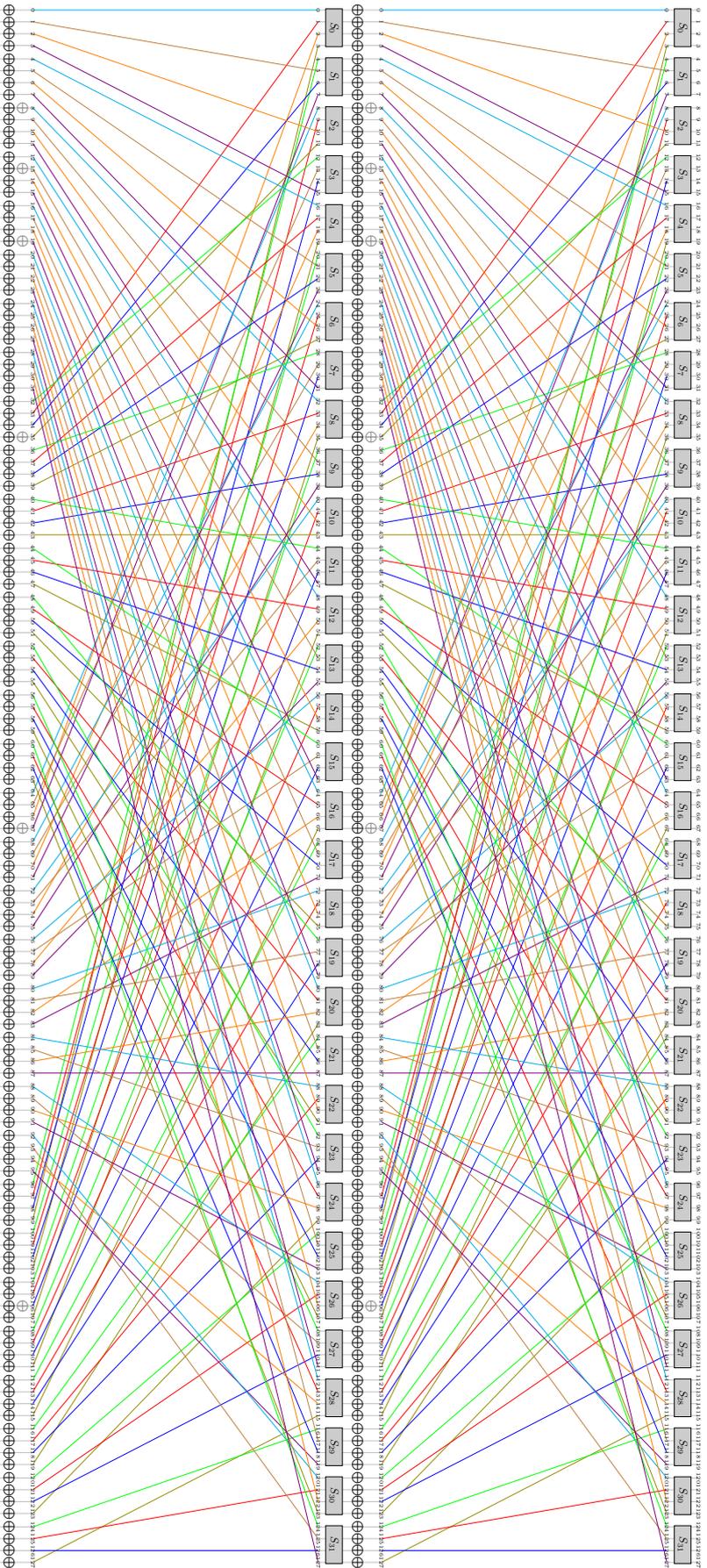


Figure 5: Structure of BAKSHEESH encryption (two rounds only)

Table 14: Test vectors for BAKSHEESH

	Key	00000000000000000000000000000000
1	Plaintext	00000000000000000000000000000000
	Ciphertext	c002be5e64c78a72ab9a3439518352aa
	Key	00000000000000000000000000000000
2	Plaintext	00000000000000000000000000000007
	Ciphertext	6f7d7746eaf0d97a154079f6bd846438
	Key	00000000000000000000000000000000
3	Plaintext	70000000000000000000000000000000
	Ciphertext	1ba3363734c09a29f67c23bbb2cccc05
	Key	00000000000000000000000000000000
4	Plaintext	44444444444444444444444444444444
	Ciphertext	7ad3303667b2af6deef434dd110d7fb8
	Key	ffffffffffffffffffffffffffffffffffff
5	Plaintext	11111111111111111111111111111111
	Ciphertext	806f0cf45b94f0370206975fe78ac10f
	Key	76543210032032032032032032032032
6	Plaintext	789a789a789a789a789a789a789a789a
	Ciphertext	ae654b5333b876584f8e8dd54f4e490a
	Key	23023023023023023023023001234567
7	Plaintext	b6e4789ab6e4789ab6e4789ab6e4789a
	Ciphertext	3dbbdf7fe254cc0be396a753442dccad
	Key	5920effb52bc61e33a98425321e76915
8	Plaintext	e6517531abf63f3d7805e126943a081c
	Ciphertext	fc7e61fee3d587308ca7bc594ebf3244

E Annex to Security Claims and Analysis

E.1 Trivial Differential and Linear Transitions

At the first glance, it may appear that the 1 probability difference transitions and the $\pm\frac{1}{2}$ linear transitions from the SBox will make BAKSHEESH readily susceptible to differential and linear attacks²³. However, as revealed by our automated search for optimum differential and linear bounds, this is indeed not a problem when a larger number of rounds is taken into account. The main reason we believe is that the interaction between the SBox and the linear layer is not weak even though the SBox is weak in some sense (refer to [11, Section 7] for relevant discussion).

To demonstrate that using an SBox with a 1-probability difference transition does not necessarily make the primitive vulnerable against the differential attack, we take a closer look at how the deterministic non-zero difference $8 \rightarrow \mathbf{f}$ propagates through multiple rounds in forward and backward directions.

Consider the case when there is a middle round r_i where all the difference 1 transitions through the SBox are deterministic, i.e., either $0 \rightarrow 0$ or $8 \rightarrow \mathbf{f}$. Due to the property of the linear layer, in round r_{i-1} , the output difference of each SBox must be either 8 or 0 . As $8 \notin \{\mathbf{f}, 0\}$, there must be active SBoxes in round r_{i-1} . To keep the number of active SBoxes as small as possible, we assume that there is only one active SBox in round r_{i-1} . This way, the input difference of this active SBox belongs to the set $\{4, 5, 6, 7\}$. As a result, in round r_{i-2} , there will be at least 2 active SBoxes with probabilistic difference transitions. In the forward direction, i.e., in round r_{i+1} , there must be probabilistic difference transitions in 3 SBoxes. Hence, in these 4 consecutive rounds, the optimal differential probability is $2^{-2 \times 6} = 2^{-12}$, which is of course much lower than the 4-round optimal differential probability, 2^{-8} .

Regarding the influence on the linear attack, a similar conclusion can be derived.

E.2 Key Recovery with Algebraic Methods

As discussed, there exist at least 32 expressions in terms of the key of algebraic degree upper bounded by 2^{r_0-1} when reversing r_0 rounds from the ciphertext. To mount a key-recovery attack on r_1 rounds of BAKSHEESH

²³This is the case for DEFAULT [11, Chapter 8] as well.

with the integral distinguisher, we first collect the 2^{127} ciphertexts. Then, we reverse $r_1 - 13$ rounds for each ciphertext and obtain the expressions for the output after 13 rounds of encryption. Based on the 13-round integral distinguisher, at least 32 Boolean equations can be set up and their algebraic degrees are upper bounded by 2^{r_1-14} .

Though there is a method to bound the time complexity to solve such a system of non-linear Boolean equations [29], we note that the equation system is sparse due to the sparse linear layer used in BAKSHEESH when r_1 is not that large. Hence, we turn to consider when all key bits will be involved in the calculation of a certain bit after reversing $r_1 - 13$ rounds. As 5 rounds are needed to achieve the full diffusion in the backward direction, it is reasonable to draw the conclusion that the number of key-recovery rounds is upper bounded by 5.

E.3 Integral Attack: Comparison with GIFT-128

GIFT-128 has the 11-round integral distinguisher as the longest integral distinguisher by the bit-based division property [22] while BAKSHEESH has the 13-round integral distinguisher that is 2 rounds longer than the one of GIFT-128. Though the number of rounds is more in BAKSHEESH, it is well within the recommended 35 rounds. Therefore, we conclude that the 35-round BAKSHEESH is not susceptible to integral attack.

F Postscript

F.1 Similarities and Differences between GIFT-128 and BAKSHEESH

The design of BAKSHEESH is much alike to GIFT-128. Indeed, BAKSHEESH is, for the most part, inspired by that of GIFT-128. The 128-bit permutation is the same in both ciphers. The SBoxes used in the ciphers are different. BAKSHEESH runs for 35 rounds, whereas GIFT-128 runs for 40 rounds (thus BAKSHEESH is generally 12.50% “smaller”).

GIFT-128 uses half the key XOR (key bits are XORed at the middle two bits of each SBox input at each round), but the full round key is XORed in BAKSHEESH. There are some claims against half-key addition (see [12, 97]), and hence we opt for the conservative full round key XOR. Apart from that, GIFT-128 does not use the initial key XOR in encryption/final round key XOR in decryption, but BAKSHEESH uses round key XOR in all the rounds.

Also, BAKSHEESH uses a different key schedule than that of GIFT-128. The round constants as well as the tap positions (reduced to 6 from 7) are different in BAKSHEESH. We also provide some justification on the choice round constants and the tap positions, which seems to be missing from GIFT-128.

One may also note from the specification of GIFT-128 that the ordering of design components inside a round is slightly altered in BAKSHEESH. In BAKSHEESH, `AddConstants` is put after `PermBits`/before `AddRoundKey` (`AddRoundConstants` is after `AddRoundKey` in GIFT-128 [22, Section 2]). This essentially does not affect the output, as all these operations use XOR. The motivation for this alteration is to make the basic software implementation simpler. In the basic implementation, one extra conversion bit \leftrightarrow nibble is required for GIFT-128 setting (where `AddConstants` operates on bits and `AddRoundKey` operates on nibbles). This is similar to DEFAULT [11, Chapter 8].

F.2 BAKSHEESH: Advantages over GIFT-128

As with the passage of time, the state-of-the-art in the field of lightweight cryptography matures, and it becomes increasingly hard to reduce the device footprint. Indeed, the authors of GIFT claim that it attempts, if not achieves, “towards reaching the limit of lightweight encryption” [22, Title].

The SBox used in GIFT-128 is more lightweight in software as well as in hardware, thereby leading to more efficient implementation. Still, since the basic design principle in both ciphers is the same, any optimization (e.g., [1]) will almost readily apply to BAKSHEESH. Thanks to the low AND count (and lower number of rounds), adopting BAKSHEESH for side-channel countermeasures is more efficient. For instance, the threshold countermeasure of GIFT-128 presented in [69, Section III.A] is more efficient for BAKSHEESH by a factor of a few folds. Apart from that, certain seemingly unrelated research areas like fully homomorphic encryption, multi-party computation, and zero knowledge can also benefit.

While the AND complexity of BAKSHEESH is lower than most (if not all) lightweight ciphers, including GIFT-128, it remains to see how it compares with other domain-specific ciphers in general. Still, a quick

comparison of BAKSHEESH with ROBIN [65] (which is specifically designed to efficiently support side-channel countermeasures), as given in Table 15; reveals that the former performs equally well. The ‘minimum AND’ reflects the minimum number of AND operations that are needed to run the full cipher, and ‘AND per bit’ is calculated by dividing the former by the state size. Note that, ROBIN is based on a new/experimental design paradigm that involves a partial non-linear layer, and is subsequently attacked [80]. BAKSHEESH does not involve such experimental construction.

Table 15: AND complexity comparison of BAKSHEESH with ROBIN

	Security	Key Size	Block Size	Minimum AND	AND per Bit
BAKSHEESH	128-bit	128-bit	128-bit	3072	24
ROBIN [65]	128-bit	128-bit	128-bit	3072	24

F.3 Search for BAKSHEESH SBox

At the top level, the search for the SBox consists of expanding the chosen AE classes (to generate a few hundred candidates); then keeping only 3-LBN SBoxes; then picking one SBox at a time from the large pool of SBoxes; fitting it (together with the GIFT-128 bit-permutation) in the model to compute the differential and linear bounds of [9] until a suitable one is found. Further, we do not need to consider the BOGI [22, Section 3] or the BOGI+ [93] properties²⁴, as the automated modelling discards the SBoxes not suitable for our purpose.

Note that a desirable protection against SCA, we would want that the LS of the inverse SBox is at \mathbf{f} (i.e., the constant output difference corresponding to the LS of the SBox). This is not kept as a hard criterion (as we do not claim inherent side channel security). It is more like a happenstance that this SBox satisfies this property.

Comparison with SBox Search Procedures in GIFT-128 and DEFAULT The SBox search procedure in GIFT [22], and particularly in DEFAULT [11, Chapter 8] are mostly comparable. It is also worth noting that no source material for the SBox search is made public by the GIFT or DEFAULT designers.

The filtering with 3-LBN is comparable to the filtering criteria in GIFT [22, Section 3.3] or in DEFAULT as described in [11, Chapter 8.4.3]. Note that the filtering criteria in these two ciphers are more rigorous than that of BAKSHEESH. This is due to the fact that the automated tool (to find the differential and linear bounds) is used only after the SBox is finalised – the tool merely reports the exact bounds. However, in BAKSHEESH, the tool is integrated during the selection of the SBox.

F.4 Number of Slack Rounds

The slack for BAKSHEESH is of 15 rounds with respect to the differential attack and 13 rounds with respect to the linear attack. In other words, we keep 15 (respectively, 13) extra rounds²⁵, even after the cipher attains the desired differential (respectively, linear) security. Other classical attacks have larger slack. Furthermore, BAKSHEESH uses the conservative full-round key addition.

There does not appear to be any hard-and-fast rule to decide the number of rounds to be kept as slack. For instance, DEFAULT-LAYER and DEFAULT-CORE have narrow slack (respectively 8 and 2 rounds against the linear attack).

F.5 How BAKSHEESH Improves State-of-the-art/Novelty

Given the progress of lightweight cryptography in the last few years, it is very hard to come up with a drastically new concept that reduces the device footprint while ensuring security. Therefore, it is understandable that the new/upcoming ciphers will follow a generic blue-print of pre-existing ciphers and specialise for niche

²⁴For instance, we allow iterated paths in differential/linear trails.

²⁵It is likely the case that BAKSHEESH attains the desired differential and linear bounds in lower number rounds. The current bounds are extrapolated based on the results obtained for lower rounds.

applications²⁶, in lieu of opting for a drastically new/experimental structure. In practice, it is quite common for one cipher to borrow component(s) and/or take inspiration from another cipher – including²⁷ – but not limited to:

- ◊ GIFT [22] is inspired from PRESENT [38]. The latter in turn is inspired from SERPENT [31].
- ◊ MANTIS [24], CRAFT [25] and WARP [19] borrow the SBox from MIDORI [20].
- ◊ Ciphers like LED [67], PHOTON [66] reuse the same SBox from PRESENT.
- ◊ GIFT-COFB [21], LOTUS-LOCUS [45], ESTATE [46] use same structure as GIFT.
- ◊ PHOTON-BEETLE [23] is based on PHOTON [66] and BEETLE [47]. It takes the SBox from PRESENT [38].
- ◊ The SKINNY-64 [24] SBox is engineered from that of TWINE [101].
- ◊ LOWMC-M [89] instantiates LOWMC [3].
- ◊ Some ciphers adopt AES-like design, e.g., SMALL-SCALE-AES [49], FORK-AES [5]. Some other ciphers take some component from AES; KIASU [70], AEGIS [105], SNOW-V [58] to name a few.
- ◊ JOLTIK [71] and DEOXYS [73] are quite similar in structure.
- ◊ As per the specification of PRINCE-v2 [40], the ShiftRow is taken from AES [53], and the SBox taken from PRINCE [39].
- ◊ SYCON [83] is inspired from ASCON [57]; CHASKEY [85] is inspired from SIPHASH [7]; CHACHA [27] is inspired from SALSA [26].
- ◊ ELEPHANT [30] uses components from SPONGENT [37] and KECCAK [28].
- ◊ Last, but not the least, DEFAULT [11, Chapter 8] is inspired from/similar to GIFT-128.

Now to put into perspective, BAKSHEESH uses a similar structure from GIFT-128, though the only component reused is the linear layer. The rest of the components are designed from scratch and are different from that of GIFT-128. The linear layer is kept the same to readily apply a software optimisation (namely, fix-slicing [1]) and also as an extra challenge.

Apart from being a general-purpose lightweight cipher like GIFT-128, BAKSHEESH also suitably fits into the niche of ciphers that are easier to protect against side-channel attacks. This is owing to the use of an SBox with a low AND count. At the same time, having a low AND count is also desirable for applications such as FHE/MPC/ZK.

BAKSHEESH uses an already established and highly analysed cipher design paradigm, rather than opting for new/experimental construction (as can be seen from the niche-specific ciphers like [2, 3, 56]). Thus, BAKSHEESH bridges the gap between a general-purpose cipher and a niche-specific cipher for the first time, to the best of our knowledge.

The idea of using a bad SBox (i.e., that has a non-zero linear structure) in a cipher construction is relatively fresh, the only cipher to deploy this is DEFAULT [11, Chapter 8]. This comes out from the necessity of the niche application of cipher-level DFA protection, and the structure of DEFAULT (both its components, DEFAULT-CORE and DEFAULT-LAYER) are very much similar to GIFT-128. BAKSHEESH shows, for the first (and only) time, how such a bad SBox can be incorporated in the design of a lightweight cipher with a lower device footprint without compromising security. This opens up a previously unexplored direction where future lightweight ciphers can be designed with an edge over certain other applications that benefit from low AND count (including, but not limited to, side-channel countermeasures).

To summarise, BAKSHEESH invents and proves an interesting concept. It improves over the highly popular/acclaimed cipher GIFT-128 [22] specially in the side channel-protected version. It shows a potential direction, from where one can expect more ciphers to come out in the near future.

References

1. Adomnicai, A., Najm, Z., Peyrin, T.: Fixslicing: A new gift representation. Cryptology ePrint Archive, Report 2020/412 (2020), <https://eprint.iacr.org/2020/412> 4, 8, 9, 17, 19, 23, 26, 28
2. Albrecht, M., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Report 2016/492 (2016), <https://eprint.iacr.org/2016/492> 8, 28

²⁶For example, MIDORI [20] or SPEEDY [81] target minimal energy consumption.

²⁷Note that we only consider the general-purpose symmetric key ciphers here. If we consider, for example, the post-quantum ciphers (which typically use AES or SHA-3 in some form) or niche-specific ciphers (like SPACE [35], which is claimed to be useful in context of white-box cryptography and uses AES in some form), the list becomes unmanageably big.

3. Albrecht, M., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for mpc and fhe. Cryptology ePrint Archive, Report 2016/687 (2016), <https://eprint.iacr.org/2016/687> 8, 19, 28
4. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Report 2019/426 (2019), <https://eprint.iacr.org/2019/426> 8
5. Andreeva, E., Reyhanitabar, R., Varici, K., Vizár, D.: Forking a blockcipher for authenticated encryption of very short messages. Cryptology ePrint Archive (2018) 28
6. Aoki, K., Sasaki, Y.: Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In: Halevi, S. (ed.) CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5677, pp. 70–89. Springer (2009), https://doi.org/10.1007/978-3-642-03356-8_5 15
7. Aumasson, J., Bernstein, D.J.: Siphash: A fast short-input PRF. In: INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings. pp. 489–508 (2012). https://doi.org/10.1007/978-3-642-34931-7_28, https://doi.org/10.1007/978-3-642-34931-7_28 28
8. 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. Cryptology ePrint Archive, Report 2016/444 (2016), <https://eprint.iacr.org/2016/444> 20
9. Baksi, A.: New insights on differential and linear bounds using mixed integer linear programming (full version). Cryptology ePrint Archive, Report 2020/1414 (2020), <https://eprint.iacr.org/2020/1414> 11, 27
10. Baksi, A.: Classical and Physical Security of Symmetric Key Cryptographic Algorithms. Ph.D. thesis, School of Computer Science & Engineering, Nanyang Technological University, Singapore (2021), <https://dr.ntu.edu.sg/handle/10356/152003> 4, 6
11. Baksi, A.: Classical and Physical Security of Symmetric Key Cryptographic Algorithms. Springer, Singapore (2022), https://doi.org/10.1007/978-981-16-6522-6_4, 5, 8, 25, 26, 27, 28
12. Baksi, A.: The problem of half round key xor. Cryptology ePrint Archive, Paper 2023/730 (2023), <https://eprint.iacr.org/2023/730> 26
13. Baksi, A., Bhasin, S., Breier, J., Jap, D., Saha, D.: Fault attacks in symmetric key cryptosystems. Cryptology ePrint Archive, Report 2020/1267 (2020), <https://eprint.iacr.org/2020/1267> 4
14. Baksi, A., Bhasin, S., Breier, J., Khairallah, M., Peyrin, T., Sarkar, S., Sim, S.M.: DEFAULT: cipher level resistance against differential fault attack. IACR Cryptology ePrint Archive **2021**, 712 (2021), <https://eprint.iacr.org/2021/712> 8
15. Baksi, A., Breier, J., Chen, Y., Dong, X.: Machine learning assisted differential distinguishers for lightweight ciphers. Cryptology ePrint Archive, Report 2020/571 (2020), <https://eprint.iacr.org/2020/571> 15
16. Baksi, A., Breier, J., Dasu, V.A., Hou, X., Kim, H., Seo, H.: New results on machine learning based distinguishers. Cryptology ePrint Archive, Paper 2023/235 (2023). <https://doi.org/10.1109/ACCESS.2023.3270396>, <https://eprint.iacr.org/2023/235> 15
17. Baksi, A., Guilley, S., Shrivastwa, R.R., Takarabt, S.: From substitution box to threshold. Cryptology ePrint Archive, Paper 2023/633 (2023), <https://eprint.iacr.org/2023/633> 10, 18, 22
18. Balasch, J., Gierlichs, B., Verdult, R., Batina, L., Verbauwhede, I.: Power analysis of atmel cryptomemory - recovering keys from secure eeproms. In: Dunkelman, O. (ed.) CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012. Lecture Notes in Computer Science, vol. 7178, pp. 19–34. Springer (2012), https://doi.org/10.1007/978-3-642-27954-6_2 5
19. 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. Cryptology ePrint Archive, Report 2020/1320 (2020), <https://eprint.iacr.org/2020/1320> 28
20. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy (extended version). Cryptology ePrint Archive, Report 2015/1142 (2015), <https://eprint.iacr.org/2015/1142> 10, 11, 28
21. Banik, S., Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT-COFB. IACR Cryptol. ePrint Arch. p. 738 (2020), <https://eprint.iacr.org/2020/738> 4, 28
22. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: Gift: A small present. Cryptology ePrint Archive, Report 2017/622 (2017), <https://eprint.iacr.org/2017/622> 4, 6, 7, 8, 9, 10, 11, 15, 16, 18, 19, 20, 26, 27, 28
23. Bao, Z., Chakraborti, A., Datta, N., Guo, J., Nandi, M., Peyrin, T., Yasuda, K.: Photon-beetle authenticated encryption and hash family, submission to the nist lightweight cryptography standardization process (2019) 28
24. 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. IACR Cryptology ePrint Archive **2016**, 660 (2016), <http://eprint.iacr.org/2016/660> 5, 9, 10, 15, 18, 20, 28
25. 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>, <https://doi.org/10.13154/tosc.v2019.i1.5-45> 28

26. Bernstein, D.J.: The salsa20 family of stream ciphers (2007), <http://cr.yp.to/snuffle/salsafamily-20071225.pdf> 28
27. Bernstein, D.J.: Chacha, a variant of salsa20 (2008), <http://cr.yp.to/chacha/chacha-20080128.pdf> 28
28. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Keccak. Cryptology ePrint Archive, Paper 2015/389 (2015). https://doi.org/10.1007/978-3-642-38348-9_19, <https://eprint.iacr.org/2015/389>, <https://eprint.iacr.org/2015/389> 28
29. Bettale, L., Faugère, J., Perret, L.: Solving polynomial systems over finite fields: improved analysis of the hybrid approach. In: van der Hoeven, J., van Hoeij, M. (eds.) International Symposium on Symbolic and Algebraic Computation, ISSAC'12, Grenoble, France - July 22 - 25, 2012. pp. 67–74. ACM (2012). <https://doi.org/10.1145/2442829.2442843>, <https://doi.org/10.1145/2442829.2442843> 26
30. Beyne, T., Chen, Y.L., Mennink, C.D.R.B.: Elephant v2 (2021), <https://www.esat.kuleuven.be/cosic/elephant/>, <https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/elephant-spec-final.pdf> 28
31. Biham, E., Anderson, R., Knudsen, L.: Serpent: A new block cipher proposal. In: Vaudenay, S. (ed.) Fast Software Encryption. pp. 222–238. Springer Berlin Heidelberg, Berlin, Heidelberg (1998) 7, 28
32. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: 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 13
33. Bilgin, B.: Threshold Implementations As Countermeasure Against Higher-Order Differential Power Analysis. Ph.D. thesis, Katholieke Universiteit Leuven and University of Twente (2015), <https://www.esat.kuleuven.be/cosic/publications/thesis-256.pdf> 6
34. Bilgin, B., De Meyer, L., Duval, S., Levi, I., Standaert, F.X.: Low and depth and efficient inverses: a guide on s-boxes for low-latency masking. IACR Transactions on Symmetric Cryptology 2020(1), 144–184 (2020) 8
35. Bogdanov, A., Isobe, T.: White-box cryptography revisited: Space-hard ciphers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. p. 1058–1069. CCS '15, Association for Computing Machinery, New York, NY, USA (2015). <https://doi.org/10.1145/2810103.2813699>, <https://doi.org/10.1145/2810103.2813699> 28
36. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique cryptanalysis of the full AES. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7073, pp. 344–371. Springer (2011), https://doi.org/10.1007/978-3-642-25385-0_19 15
37. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: Spongent: The design space of lightweight cryptographic hashing. Cryptology ePrint Archive, Paper 2011/697 (2011), <https://eprint.iacr.org/2011/697>, <https://eprint.iacr.org/2011/697> 28
38. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: CHES. vol. 4727, pp. 450–466. Springer (2007) 7, 8, 9, 10, 18, 28
39. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knežević, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçın, T.: Prince - a low-latency block cipher for pervasive computing applications (full version). Cryptology ePrint Archive, Report 2012/529 (2012), <https://eprint.iacr.org/2012/529> 9, 10, 11, 18, 20, 28
40. Božilov, D., Eichlseder, M., Knežević, M., Lambin, B., Leander, G., Moos, T., Nikov, V., Rasoolzadeh, S., Todo, Y., Wiemer, F.: Princev2. In: International Conference on Selected Areas in Cryptography. pp. 483–511. Springer (2020) 28
41. Božilov, D., Eichlseder, M., Knežević, M., Lambin, B., Leander, G., Moos, T., Nikov, V., Rasoolzadeh, S., Todo, Y., Wiemer, F.: Princev2 - more security for (almost) no overhead. Cryptology ePrint Archive, Paper 2020/1269 (2020), <https://eprint.iacr.org/2020/1269> 11
42. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. pp. 16–29 (2004), https://doi.org/10.1007/978-3-540-28632-5_2 5, 6, 15
43. Canteaut, A., Carpov, S., Fontaine, C., Lepoint, T., Naya-Plasencia, M., Paillier, P., Sirdey, R.: Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. Cryptology ePrint Archive, Report 2015/113 (2015), <https://eprint.iacr.org/2015/113> 8, 19
44. Carlet, C., Guilley, S.: Side-Channel Indistinguishability. In: HASP. pp. 9:1–9:8. ACM, Tel Aviv, Israel (Jun 2013), <https://hal.archives-ouvertes.fr/hal-00826618> 6
45. Chakraborti, A., Datta, N., Jha, A., Lopez, C.M., Nandi, M., Sasaki, Y.: Lotus and locus aead: Hardware benchmarking and security (2019) 28
46. Chakraborti, A., Datta, N., Jha, A., Mancillas-López, C., Nandi, M., Sasaki, Y.: Estate: A lightweight and low energy authenticated encryption mode. IACR Transactions on Symmetric Cryptology pp. 350–389 (2020) 28
47. Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle family of lightweight and secure authenticated encryption ciphers. Cryptology ePrint Archive (2018) 28
48. Chakraborty, K., Maitra, S., Sarkar, S., Mazumdar, B., Mukhopadhyay, D.: Redefining the transparency order. IACR Cryptol. ePrint Arch. p. 367 (2014), <https://eprint.iacr.org/2014/367> 6

49. Cid, C., Murphy, S., Robshaw, M.J.: Small scale variants of the aes. In: International Workshop on Fast Software Encryption. pp. 145–162. Springer (2005) [28](#)
50. Coron, J.: On the exact security of full domain hash. In: Bellare, M. (ed.) CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1880, pp. 229–235. Springer (2000). https://doi.org/10.1007/3-540-44598-6_14, https://doi.org/10.1007/3-540-44598-6_14 [6](#)
51. Cui, J., Hu, K., Wang, Q., Wang, M.: Integral attacks on pyjamask-96 and round-reduced pyjamask-128. Cryptology ePrint Archive, Report 2021/1572 (2021), <https://eprint.iacr.org/2021/1572> [9](#)
52. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher square. In: Biham, E. (ed.) Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer (1997), <https://doi.org/10.1007/BFb0052343> [13](#)
53. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag Berlin Heidelberg (2002), <https://www.springer.com/gp/book/9783540425809> [8](#), [28](#)
54. Dasu, V.A., Baksi, A., Sarkar, S., Chattopadhyay, A.: LIGHTER-R: optimized reversible circuit implementation for sboxes. In: 32nd IEEE International System-on-Chip Conference, SOCC 2019, Singapore, September 3-6, 2019. pp. 260–265 (2019). <https://doi.org/10.1109/SOCC46988.2019.1570548320>, <https://doi.org/10.1109/SOCC46988.2019.1570548320> [20](#)
55. De Cannière, C.: Analysis and Design of Symmetric Encryption Algorithms. Ph.D. thesis, Katholieke Universiteit Leuven, Belgium (2007), <https://www.esat.kuleuven.be/cosic/publications/thesis-139.pdf> [9](#)
56. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: A cipher with low anddepth and few ands per bit. Cryptology ePrint Archive, Report 2018/181 (2018), <https://eprint.iacr.org/2018/181> [8](#), [19](#), [28](#)
57. Dobraunig, C., Eichlseder, M., Mendel, F., Schl affer, M.: Ascon v1.2. Submission to NIST (2019), <https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf> [28](#)
58. Ekdahl, P., Johansson, T., Maximov, A., Yang, J.: A new snow stream cipher called snow-v. IACR Transactions on Symmetric Cryptology **2019**(3), 1–42 (Sep 2019). <https://doi.org/10.13154/tosc.v2019.i3.1-42>, <https://tosc.iacr.org/index.php/ToSC/article/view/8356> [28](#)
59. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Ko ,  .K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2162, pp. 251–261. Springer (2001), https://doi.org/10.1007/3-540-44709-1_21 [5](#)
60. Gao, S., Roy, A., Oswald, E.: Constructing ti-friendly substitution boxes using shift-invariant permutations. In: Matsui, M. (ed.) Topics in Cryptology – CT-RSA 2019. pp. 433–452. Springer International Publishing, Cham (2019) [16](#), [18](#)
61. G rard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.: Block ciphers that are easier to mask: How far can we go? IACR Cryptol. ePrint Arch. **2013**, 369 (2013), <http://eprint.iacr.org/2013/369> [8](#), [19](#)
62. Gohr, A.: Improving attacks on round-reduced speck32/64 using deep learning. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 150–179. Springer International Publishing, Cham (2019) [15](#)
63. Goudarzi, D., Jean, J., K lbl, S., Peyrin, T., Rivain, M., Sasaki, Y., Sim, S.M.: Pyjamask v1.0 (2019), <https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/pyjamask-spec-round2.pdf> [4](#), [10](#), [18](#)
64. Grosso, V., Leurent, G., Standaert, F.X., Varic , K.: LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In: Fast Software Encryption - FSE 2014. Londres, United Kingdom (Mar 2014), <https://hal.inria.fr/hal-01093491> [8](#), [19](#)
65. Grosso, V., Leurent, G., Standaert, F.X., Varic , K.: Ls-designs: Bitslice encryption for efficient masked software implementations. In: Cid, C., Rechberger, C. (eds.) Fast Software Encryption. pp. 18–37. Springer Berlin Heidelberg, Berlin, Heidelberg (2015) [27](#)
66. Guo, J., Peyrin, T., Poschmann, A.: The photon family of lightweight hash functions. In: Annual cryptology conference. pp. 222–239. Springer (2011) [28](#)
67. 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), http://dx.doi.org/10.1007/978-3-642-23951-9_22 [10](#), [28](#)
68. Ha, J., Kim, S., Lee, B., Lee, J., Son, M.: Rubato: Noisy ciphers for approximate homomorphic encryption (full version). Cryptology ePrint Archive, Paper 2022/537 (2022), <https://eprint.iacr.org/2022/537> [8](#)
69. Jati, A., Gupta, N., Chattopadhyay, A., Sanadhya, S.K., Chang, D.: Threshold implementations of gift: A trade-off analysis. IEEE Trans. Inf. Forensics Secur. **15**, 2110–2120 (2020), <https://doi.org/10.1109/TIFS.2019.2957974> [16](#), [18](#), [22](#), [26](#)
70. Jean, J., Nikolic, I., Peyrin, T.: Kiasu v1. Submission to the CAESAR competition (2014), <https://competitions.cr.yt.to/round1/kiasuv1.pdf> [28](#)
71. Jean, J., Nikoli , I., Peyrin, T.: Joltik v1.3 (2015), Submission to the CAESAR competition, <http://www1.spms.ntu.edu.sg/~syllab/Joltik> [28](#)

72. 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>, <https://doi.org/10.13154/tosc.v2017.i4.130-168> **20**
73. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: The deoxys aead family. *Journal of Cryptology* **34** (2021). <https://doi.org/10.1007/s00145-021-09397-w> **28**
74. Kim, H., Jeon, Y., Kim, G., Kim, J., Sim, B.Y., Han, D.G., Seo, H., Kim, S., Hong, S., Sung, J., Hong, D.: A new method for designing lightweight s-boxes with high differential and linear branch numbers, and its application. *Cryptology ePrint Archive, Report 2020/1582* (2020), <https://eprint.iacr.org/2020/1582> **8, 10**
75. 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 **13**
76. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) *CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. Lecture Notes in Computer Science*, vol. 1109, pp. 104–113. Springer (1996), https://doi.org/10.1007/3-540-68697-5_9 **5**
77. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: *CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. pp. 388–397 (1999), https://doi.org/10.1007/3-540-48405-1_25 **5**
78. Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A cryptanalysis of printcipher: The invariant subspace attack. In: Rogaway, P. (ed.) *CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Lecture Notes in Computer Science*, vol. 6841, pp. 206–221. Springer (2011), https://doi.org/10.1007/978-3-642-22792-9_12 **14**
79. Leander, G., Minaud, B., Rønjom, S.: A generic approach to invariant subspace attacks: Cryptanalysis of robin, iscream and zorro. In: Oswald, E., Fischlin, M. (eds.) *EUROCRYPT 2015. Lecture Notes in Computer Science*, vol. 9056, pp. 254–283. Springer (2015), https://doi.org/10.1007/978-3-662-46800-5_11 **10, 14**
80. Leander, G., Minaud, B., Rønjom, S.: A generic approach to invariant subspace attacks: Cryptanalysis of robin, iscream and zorro. In: Oswald, E., Fischlin, M. (eds.) *Advances in Cryptology – EUROCRYPT 2015*. pp. 254–283. Springer Berlin Heidelberg, Berlin, Heidelberg (2015) **27**
81. Leander, G., Moos, T., Moradi, A., Rasoolzadeh, S.: The speedy family of block ciphers - engineering an ultra low-latency cipher from gate level for secure processor architectures. *Cryptology ePrint Archive, Report 2021/960* (2021), <https://eprint.iacr.org/2021/960> **28**
82. Makarim, R.H., Tezcan, C.: Relating undisturbed bits to other properties of substitution boxes. *Cryptology ePrint Archive, Paper 2014/855* (2014), <https://eprint.iacr.org/2014/855> **4**
83. Mandal, K., Saha, D., Sarkar, S., Todo, Y.: Sycon: a new milestone in designing ascon-like permutations. *Journal of Cryptographic Engineering* pp. 1–23 (2021) **28**
84. Mesquita, D., Techer, J., Torres, L., Sassatelli, G., Cambon, G., Robert, M., Moraes, F.: A new hardware countermeasure for masking power signatures of crypto cores. In: Sassatelli, G., Glesner, M., Torres, L., Indrusiak, L.S., Hollstein, T. (eds.) *Proceedings of the 1st International Workshop on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2005, Montpellier, France, June 2005*. pp. 169–176. Univ. Montpellier II (2005) **6**
85. Mouha, N., Mennink, B., Herrewewe, A.V., Watanabe, D., Preneel, B., Verbauwhede, I.: Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In: *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*. pp. 306–323 (2014), https://doi.org/10.1007/978-3-319-13051-4_19 **28**
86. Méaux, P., Journault, A., Standaert, F.X., Carlet, C.: Towards stream ciphers for efficient fhe with low-noise ciphertexts. *Cryptology ePrint Archive, Report 2016/254* (2016), <https://eprint.iacr.org/2016/254> **8**
87. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: *International conference on information and communications security*. pp. 529–545. Springer (2006) **6**
88. Oswald, D.F., Richter, B., Paar, C.: Side-channel attacks on the yubikey 2 one-time password generator. In: Stolfo, S.J., Stavrou, A., Wright, C.V. (eds.) *Research in Attacks, Intrusions, and Defenses - 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings. Lecture Notes in Computer Science*, vol. 8145, pp. 204–222. Springer (2013), https://doi.org/10.1007/978-3-642-41284-4_11 **5**
89. Peyrin, T., Wang, H.: The malicious framework: Embedding backdoors into tweakable block ciphers. *Cryptology ePrint Archive, Paper 2020/986* (2020). https://doi.org/10.1007/978-3-030-56877-1_9, <https://eprint.iacr.org/2020/986>, <https://eprint.iacr.org/2020/986> **28**
90. Picek, S., Papagiannopoulos, K., Ege, B., Batina, L., Jakobovic, D.: Confused by confusion: Systematic evaluation of DPA resistance of various s-boxes. In: Meier, W., Mukhopadhyay, D. (eds.) *INDOCRYPT 2014. Lecture Notes in Computer Science*, vol. 8885, pp. 374–390. Springer (2014), https://doi.org/10.1007/978-3-319-13039-2_22 **6**
91. Piret, G., Roche, T., Carlet, C.: Picaro - a block cipher allowing efficient higher-order side-channel resistance – extended version –. *Cryptology ePrint Archive, Report 2012/358* (2012), <https://eprint.iacr.org/2012/358> **8, 19**
92. Sarkar, S., Mandal, K., Saha, D.: On the relationship between resilient boolean functions and linear branch number of s-boxes. *Cryptology ePrint Archive, Report 2019/1427* (2019), <https://eprint.iacr.org/2019/1427> **4, 10**

93. Sarkar, S., Sasaki, Y., Sim, S.M.: On the design of bit permutation based ciphers - the interplay among s-box, bit permutation and key-addition. Cryptology ePrint Archive, Report 2020/680 (2020), <https://eprint.iacr.org/2020/680> 27
94. Sasaki, Y., Aoki, K.: Finding preimages in full MD5 faster than exhaustive search. In: Joux, A. (ed.) EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 134–152. Springer (2009), https://doi.org/10.1007/978-3-642-01001-9_8 15
95. Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017. Lecture Notes in Computer Science, vol. 10212, pp. 185–215 (2017), https://doi.org/10.1007/978-3-319-56617-7_7 14
96. Sasdrich, P., Bock, R., Moradi, A.: Threshold implementation in software - case study of PRESENT. IACR Cryptol. ePrint Arch. p. 189 (2018), <http://eprint.iacr.org/2018/189> 6
97. Shi, D., Sun, S., Song, L., Hu, L., Yang, Q.: Exploiting non-full key additions: Full-fledged automatic demirci-selcuk meet-in-the-middle cryptanalysis of skinny. Cryptology ePrint Archive, Paper 2023/255 (2023), <https://eprint.iacr.org/2023/255> 26
98. 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 18
99. Sun, L., Wang, W., Wang, M.: Accelerating the search of differential and linear characteristics with the sat method. Cryptology ePrint Archive, Report 2021/213 (2021), <https://eprint.iacr.org/2021/213> 4, 11
100. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: Twine : A lightweight , versatile block cipher. ECRYPT (2011), https://www.nec.com/en/global/rd/tg/code/symenc/pdf/twine_LC11.pdf 18
101. Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E.: Twine: A lightweight, versatile block cipher. In: ECRYPT workshop on lightweight cryptography. vol. 2011 (2011) 28
102. Todo, Y.: Structural evaluation by generalized integral property. In: EUROCRYPT 2015. pp. 287–314 (2015), https://doi.org/10.1007/978-3-662-46800-5_12 13
103. Todo, Y., Leander, G., Sasaki, Y.: Nonlinear invariant attack - practical attack on full scream, iscream, and midori64. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. Lecture Notes in Computer Science, vol. 10032, pp. 3–33 (2016), https://doi.org/10.1007/978-3-662-53890-6_1 15
104. Ullrich, M.: The design and efficient software implementation of S-boxes. Ph.D. thesis, KU Leuven (2010), <https://mouha.be/wp-content/uploads/ullrich-thesis.pdf> 8
105. Wu, H., Preneel, B.: Aegis: A fast authenticated encryption algorithm. In: International Conference on Selected Areas in Cryptography. pp. 185–201. Springer (2013) 28
106. 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: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. Lecture Notes in Computer Science, vol. 10031, pp. 648–678 (2016), https://doi.org/10.1007/978-3-662-53887-6_24 13
107. Yan, Y., Oswald, E.: Examining the Practical Side Channel Resilience of ARX-boxes. Cryptology ePrint Archive, Report 2019/335 (2019), <https://eprint.iacr.org/2019/335> 6
108. Zhu, B., Dong, X., Yu, H.: MILP-based differential attack on round-reduced gift. Cryptology ePrint Archive, Report 2018/390 (2018), <https://eprint.iacr.org/2018/390> 4