BAKSHEESH: Similar Yet Different From GIFT (and ZORRO)

Anubhab Baksi[1], Jakub Breier[2], Anupam Chattopadhyay[3], Tomáš Gerlich[4], Sylvain Guilley[5,6], Naina Gupta[3], Takanori Isobe[7], Arpan Jati[3], Petr Jedlicka[4], Hyunjun Kim[8], Fukang Liu[9], Zdeněk Martinásek[4], Kosei Sakamoto[10], Hwajeong Seo[8] and Rentaro Shiba[10,11]

[1] Indian Statistical Institute, Chennai, India
[2] TTControl GmbH, Vienna, Austria
[3] Nanyang Technological University, Singapore, Singapore
[4] Brno University of Technology, Brno, Czechia
[5] Télécom Paris, Paris, France
[6] Secure-IC, Rennes, France
[7] University of Osaka, Osaka, Japan
[8] Hansung University, Seoul, South Korea
[9] Institute of Science Tokyo, Tokyo, Japan
[10] Mitsubishi Electric Corporation, Tokyo, Japan
[11] Nagoya University, Nagoya, Japan

Abstract. We propose a lightweight block cipher named BAKSHEESH, which draws inspiration from GIFT (CHES'17). BAKSHEESH runs for 35 rounds, which is 12.5 per cent fewer rounds compared to GIFT-128 whilst maintaining the same security claims against classical attacks. However, we also present an alternate (but equivalent) angle of BAKSHEESH that follows the heritage of ZORRO (CHES'13). In the GIFT heritage version, BAKSHEESH uses a 4-bit SBox that has a non-trivial Linear Structure (LS). In the alternate ZORRO heritage specification, it employs a 3-bit SBox and realises a partial non-linear layer.

BAKSHEESH is suitable for efficient hardware and software implementations, and also offers an edge on side channel countermeasures and other niche applications. For instance, our study on the threshold implementation shows that BAKSHEESH offers a few-fold advantage over other lightweight ciphers.

We therefore create a new paradigm of lightweight ciphers through adequate deliberation on the design choice and solidify it with appropriate security analysis and ample implementation/benchmark.

Keywords: lightweight cryptography · block cipher · GIFT · DEFAULT · linear structure · partial non-linear layer · ZORRO · side channel attack · differential fault attack · threshold implementation

Authors are in alphabetical order of (Romanised) surnames. Corresponding author: Anubhab Baksi.

We thank Yong Liu (University of Chinese Academy of Sciences, Beijing, PR China) for taking interest in our cipher and providing valuable feedback on the specification.

E-mail: anubhab@isichennai.res.in, baksianubhab@gmail.com (Anubhab Baksi), jbreier@jbreier.com (Jakub Breier), anupam@ntu.edu.sg (Anupam Chattopadhyay), xgerli02@stud.feec.vutbr.cz (Tomáš Gerlich), sylvain.guilley@telecom-paristech.fr (Sylvain Guilley), naina003@e.ntu.edu.sg (Naina Gupta), takanori.isobe@ist.osaka-u.ac.jp (Takanori Isobe), arpan.jati@ntu.edu.sg (Arpan Jati), xjedli23@vut.cz (Petr Jedlicka), khj930704@gmail.com (Hyunjun Kim), liufukangs@gmail.com (Fukang Liu), martinasek@feec.vutbr.cz (Zdeněk Martinásek), k.sakamoto0728@gmail.com (Kosei Sakamoto), hwajeong84@gmail.com (Hwajeong Seo), rentaro.shiba@gmail.com (Rentaro Shiba)

This work is licensed under a "CC BY 4.0" license.

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 (such as the work of [ANP21]), or designing a brand new cipher that attains a lower cost than existing ciphers. In the latter case, the cipher is designed to take advantage of the state-of-the-art progress the research community has access to, thereby streamlining design choices. It is an extremely popular research direction in the community in (relatively) recent times, as evident from ciphers like `GIFT` [BPP+17a] or `PYJAMASK-128` [GJK+19].

Going with the trend, here we analyse and improve upon two influential design paradigms in lightweight cryptography: The structural efficiency of `GIFT` [BPP+17a] and the masking-friendly concepts introduced by `ZORRO` [GGNS13].

Although the design of `GIFT` amasses new concepts to reduce the device footprint and `ZORRO` opens new avenues for masking efficiency, the knowledge and expertise of the community have come a long way since then. With that in mind, it stands to reason that a fresh look can yield improvement and push the efficiency limit even further. Indeed, a deeper analysis 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 *Substitution Boxes* (SBoxes for short) with non-trivial *Linear Structure*s (LS, Definition 8). This has surfaced in the literature merely as a theoretical interest (see, e.g., [MT14]). The first time the potential of an LS SBox is realised is through `DEFAULT` [Bak22, Chapter 8], as a way to have an innate protection against a type of *Fault Attack* [BBB+23, JT12], namely the *Differential Fault Attack* (DFA).

Now, in our quest to find an answer to the conundrum (*"How can we find a more efficient design that will not only improve the benchmarks, but also will have an edge for the niche applications whilst avoiding the security weakness of partial non-linear designs?"*), 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 when combining structural insights from `GIFT` with masking concepts from `ZORRO`.

To make `BAKSHEESH` a worthy successor drawing from `GIFT`, it is imperative not to stray away much from the proven design philosophies whilst incorporating novel efficiency improvements. Following `GIFT`'s approach, we adopt a bit-permutation layer (it takes practically zero area in ASIC and efficient software implementation is possible [ANP21]). Our bit-permutation is derived from that one used in `GIFT-128`. To reduce the search space for the SBox, we set the *Linear Branch Number* (LBN, see Definition 9) to be 3, this is the theoretical upper bound for any 4×4 SBox [SMS19]. Save for the newly proposed `DEFAULT-LAYER` [Bak22, Chapters 7, 8] (or [Bak21, Chapters 7, 8]) no cipher uses an SBox with LBN > 2. Despite using a *bad* SBox, `BAKSHEESH` passes the classical security requirement with flying colours (see Section 4).

Meanwhile, `ZORRO` [GGNS13], introduced in 2013, is among the first ciphers to introduce the concept of partial non-linear layers, which comes in handy in certain niche applications. Unfortunately, the researchers in the community have been quick to point out security vulnerabilities [RASA14, WWGY14, GNPW14, BODD+14]. Even though our design is originally conceived from `GIFT`, we notice that due to certain design choice made in `BAKSHEESH`, it is possible to consider an alternate (but equivalent) vantage point for `BAKSHEESH` that follows the heritage of `ZORRO`.

This alternate specification employs a 3-bit SBox on the left three-quarters of the state, while the rest of the state is left unchanged. As the full state does not go into a non-linear operation (i.e., SBox) in a particular round, it realises a partial non-linear layer. This alternate specification establishes `BAKSHEESH`'s independence from `GIFT-128`, while also

inheriting the niche-specific efficiency of ZORRO.

The GIFT heritage is considered to be the canonical interpretation of BAKSHEESH[1]. The ZORRO interpretation of BAKSHEESH is found by reverse-engineering, and is not separately analysed (in terms of classical attacks) in this work.

1.1 Contribution/Organisation

Finding and perfecting novel concepts that can ultimately lead to extremely low cost – without jeopardising the security – is generally quite challenging as it typically involves studying and selecting a set of novel design choices, choosing efficient parameters, asserting security against classical attacks, optimising the cipher in hardware/software and finally testing and protecting against the *Side Channel Attack* (SCA) [MOP07, Pee13]. To this end, we present a new lightweight block cipher, called BAKSHEESH[2]. Apart from GIFT-128, significant inspiration for BAKSHEESH is drawn from DEFAULT [Bak22, Chapters 7, 8], which first explores SBoxes with linear structures in a cipher construction; at the same time we show BAKSHEESH resembles ZORRO.

BAKSHEESH consists of 35 rounds, which is 12.5% smaller than its predecessor, GIFT-128 from the get-go. Also, the 4-bit SBox used in BAKSHEESH canonical description is quite lightweight in hardware and software implementations, on top of having a low AND count/depth.

In software implementations, BAKSHEESH achieves 19 to 47 per cent improvements across software platforms and operates with fewer rounds (35 vs. 40). In hardware implementations, BAKSHEESH achieves 1.4× higher operating frequency for unprotected designs and requires 2.43× smaller area in threshold (hardware) implementations.

Some background information is covered in Section 2. The construction and design rationale are described in Section 3 (the canonical specification of GIFT heritage is given in Section 3.2, and the alternate specification of ZORRO heritage is detailed in Section 3.3). Thereafter, formal security analysis is given in Section 4, covering classical attacks (including differential and linear attack (Section 4.1), algebraic attack (Section 4.2), integral attack (Section 4.3) and so on). Device-dependent attack or resistance is addressed in Section 5. Detailed benchmarks are presented in Section 6, covering unprotected hardware (Section 6.1), optimised software implementations (Section 6.2), and threshold implementation for side channel protection (Section 6.3). The paper concludes in Section 7, though we have auxiliary material afterward (namely, acronym collection in Section A, further cryptographic properties of the 4-bit and 3-bit SBoxes in Section B, miscellaneous benchmarks in Section C, and a detailed DFA on combined encryption and decryption walkthrough featuring DEFAULT in Section D).

All in all, we show how an unusual – and traditionally undesirable/overlooked – design choice can lead to improvement in a highly analysed/optimised field of lightweight cipher construction. The relevant resources (containing reference implementations, protected hardware designs, optimised software implementations and experimental data) are publicly available[3].

1.2 Independent Analysis

Several third-party analyses of BAKSHEESH have appeared, such as [JKP25, KGAS25, PLS25, PLS24, XCF+25, GKRS25, WSW+24, GKS25, SKK+25]; we are grateful to the authors for their interest and effort. Among those, the contributions in [JKP25, KGAS25, GKS25, SKK+25] deal with vulnerability against fault attacks [BBB+23], which is out-

[1] Unless otherwise mentioned, the GIFT heritage (i.e., 4-bit SBox with LS) is the version we refer to.
[2] This word in Persian translates to 'tips'.
[3] https://github.com/anubhab001/baksheesh/.

of-scope for our paper (as we do not claim any in-built DFA security). Notably, the papers [PLS24, XCF+25] are entirely dedicated to cryptanalysis of BAKSHEESH.

The authors in [PLS24] go into great depth in terms of the linear attack, as they investigate the optimal linear characteristics of the cipher. They verify the relevant security claims in Section 4.1 in the process. The authors in [XCF+25] do a thorough analysis of BAKSHEESH in terms of three classical attacks; namely differential, linear and integral; and a summary of their results is shown in Table 1 (which is taken from [XCF+25, Table 1]). These independent reviews by the prominent researchers make us confident about the security of BAKSHEESH. We thank for their interest in our research and taking the initiative to examine the security claims of our cipher.

Table 1: Security analysis of BAKSHEESH by Xu, Chen, Feng, Xiang and Zeng (Inscrypt'24)

	Round	Time Complexity	Data Complexity	Memory Complexity
Differential	19	$2^{121.00}$	$2^{122.00}$	$2^{64.00}$
Linear	21	$2^{127.67}$	$2^{129.00}$	$2^{12.00}$
Integral	9	$2^{32.01}$	$2^{15.59}$	Negligible
Integral	10	$2^{32.99}$	$2^{31.59}$	$2^{24.59}$
Integral	15	$2^{127.00}$	$2^{127.00}$	$2^{23.00}$

1.3 (Incremental) Improvement over GIFT and Novelty

At first glance, it may appear that the novelty in BAKSHEESH is rather limited, due to its similarity to GIFT-128. With due respect, however, we would like to argue that despite the superficial resemblance; BAKSHEESH is conceptually a different cipher from GIFT-128.

We admit that it seems like BAKSHEESH merely reduces the number of rounds of GIFT-128 from 40 to 35. However, this so-called *incremental improvement* is not trivial at all. The designers of GIFT have carried out a thorough analysis of the design space. The entire process of GIFT design is very up-to-date and optimised in every way possible, improving carefully upon its predecessor PRESENT [BKL+07]. Therefore, it is not easy to find a better design than GIFT — if it were, then the GIFT designers would have more than likely found that already!

Our design exploration of BAKSHEESH involves usage of an SBox with LS, this paradigm did not exist back when GIFT was designed. Hence it was impossible to design something like this just a few years ago and only became possible through the advent of DEFAULT. One interesting quirk, as can be noticed from Table 9, is that BAKSHEESH allows iterated linear trails (with probability 1) — this goes against the traditional philosophy of limiting the number of high probability iterated differential/linear trails and the design of GIFT specifically aimed to minimise the probability of such trails. Apart from the SBox, we would like to highlight that every single design choice or component of BAKSHEESH is different from that of GIFT:

- **Linear layer:** We use the description of the 128-bit linear layer (bit-permutation) from [PLS24], which is different from that of GIFT-128 [BPP+17a]; even though it is derived from the GIFT-128 bit-permutation.
- **Key schedule and round key XOR:** Unlike GIFT, BAKSHEESH uses a more secure full round key XOR, despite its penalty on benchmarking. This is essential for side channel security[4], even when we ignore the potential problems that may arise in the classical attack setting.

[4]The analysis in [JGC+20] ignores the half round key XOR from GIFT specification and uses full round key XOR instead.

- **Round constants and tap positions:** The method of generating round constants and selecting tap positions is different in BAKSHEESH and all the design choices in BAKSHEESH are clearly stated/justified.

Based on these points, we argue that BAKSHEESH is not a mere incremental improvement over GIFT — it is not possible to improve from GIFT as such. BAKSHEESH follows a generic blue-print of lightweight block cipher construction (for example, TRIFLE-BC [DGM+19] too follows the same blue-print, but it does not mean that *"*BAKSHEESH *incrementally improves upon* TRIFLE-BC*."*).

One novelty of using an SBox with non-trivial LS is that it falls at the confluence of the concept of the partial non-linear layer (this is possible as now the 4-bit BAKSHEESH SBox with 1 LS can be equivalently interpreted as a 3-bit SBox with no non-trivial LS). Although the idea of a partial non-linear layer itself is not new, this always has been a staple in niche-specific ciphers (like [GLSV14, DEG+18, ARS+15, CCF+16]). To the best of our knowledge, BAKSHEESH is the first instance of a general-purpose cipher which uses an SBox with a partial non-linear layer, and thus, BAKSHEESH bridges the gap between the general purpose and niche-specific ciphers. Even though this was not the original motivation, the partial non-linear layer interpretation of BAKSHEESH completely removes any dependence on GIFT-128; rather, BAKSHEESH can be viewed as a natural evolution of ZORRO which makes it more versatile, general-purpose/mainstream and lightweight. This alternate interpretation of BAKSHEESH is detailed in Section 3.3.

Overall, the design and analysis of BAKSHEESH require fundamental rethinking of the problem at hand and opens up a new design space for future ciphers. Several independent third-party analyses of BAKSHEESH have since appeared (see Section 1.2 for details).

2 Background

2.1 Preliminary

Similar to [Bak22], we assume integer representations of bit strings and automatic conversion of elements to/from $\mathbb{F}_2^n$ from/to $\mathbb{F}_{2^n}$ for simplicity. We also use the shorthand hexadecimal string (that shows the look-up values) based notation to indicate an SBox.

Definition 1 (Coordinate Function and Component Function)**.** Suppose $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is defined as $F(x) = (f_0(x), \ldots, f_{n-1}(x))$ for all $x \in \mathbb{F}_2^n$, where $f_i : \mathbb{F}_2^n \to \mathbb{F}_2$ for $i = 0, \ldots, n-1$. 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.

Definition 2 (Algebraic Degree)**.** The algebraic degree of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is the maximum number of variables in any monomial of its *Algebraic Normal Form* (ANF). For a vectorial Boolean function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$, the algebraic degree is the maximum algebraic degree among all its coordinate functions.

Definition 3 (Affine Function)**.** An affine function $g : \mathbb{F}_2^n \to \mathbb{F}_2$ is a function of the form $g(x) = a \cdot x \oplus c$, where $a \in \mathbb{F}_2^n$, $c \in \mathbb{F}_2$ and $\cdot$ denotes the dot product.

Definition 4 (Non-linearity)**.** The non-linearity of the Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is the minimum Hamming distance of f to the set of all affine functions. Further, the non-linearity of $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is the minimum of the non-linearities of all the component functions of F.

Definition 5 (Difference Distribution Table (DDT) and Differential Uniformity (DU))**.** The difference distribution table (DDT) for an $n \times n$ SBox S is the $2^n \times 2^n$ matrix; where for row $\delta \in \mathbb{F}_2^n$ and column $\Delta \in \mathbb{F}_2^n$, the entry $[\delta, \Delta] = |\{x \in \mathbb{F}_2^n : S(x) \oplus S(x \oplus \delta) = \Delta\}|$. The maximum entry in DDT except DDT$[0, 0]$ is called the Differential uniformity (DU).

Definition 6 (Walsh Transform and Walsh Spectrum)**.** The Walsh transform of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as $W_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus a \cdot x}$, where $a \in \mathbb{F}_2^n$ and $\cdot$ denotes the dot product. The Walsh spectrum is the multiset of all Walsh transform values $\{W_f(a) : a \in \mathbb{F}_2^n\}$. The maximum absolute value in the Walsh spectrum (excluding $W_f(0)$) is related to the non-linearity of the function.

Definition 7 (Linear Approximation Table (LAT))**.** 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 $|\{\gamma \cdot \vec{x} \oplus \Gamma \cdot \vec{y} = 0\}| - 2^{n-1}$. Here $\vec{x}$ denotes the input variables to the SBox, $\vec{y}$ denotes the output variables and $\cdot$ is the dot product.

Definition 8 (Linear Structure (LS))**.** Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^m$ be an SBox. A non-zero vector $\alpha \in \mathbb{F}_2^n$ is called a linear structure of S if there exists a vector $\beta \in \mathbb{F}_2^m$ such that $S(x \oplus \alpha) \oplus S(x) = \beta$ for all $x \in \mathbb{F}_2^n$, and β is called the image of the linear structure α.

Definition 9 (Differential/Linear Branch Number)**.** The Differential Branch Number (DBN) of F is defined as $\min_{\delta \neq 0}\{|\delta| + |F(x) \oplus F(x \oplus \delta)|\}$; and that of the Linear Branch Number (LBN) as $\min_{\alpha \neq 0, \mathrm{LAT}[\alpha,\beta] \neq 0}\{|\alpha| + |\beta|\}$; where $|\cdot|$ denotes the Hamming weight throughout the paper.

Definition 10 (Auto-Correlation Table (ACT))**.** The Auto-Correlation Table (ACT) for an $n \times n$ SBox S is a $2^n \times 2^n$ matrix where the entry $\mathrm{ACT}_S[a, \delta]$ for input mask a and output difference δ is given by $\mathrm{ACT}_S[a, \delta] = \sum_{x \in \mathbb{F}_2^n} (-1)^{a \cdot (S(x) \oplus S(x \oplus \delta))}$.

Definition 11 (Boomerang Connectivity Table (BCT))**.** The Boomerang Connectivity Table (BCT) for an $n \times n$ SBox S is a $2^n \times 2^n$ matrix where the entry $\mathrm{BCT}_S[\phi, \Phi]$ for input difference Δ_0 and output difference Φ is given by $\mathrm{BCT}_S[\Delta_0, \Phi] = |\{x \in \mathbb{F}_2^n : S^{-1}(S(x) \oplus \Phi) \oplus S^{-1}(S(x \oplus \Delta_0) \oplus \Phi) = \Delta_0\}|$.

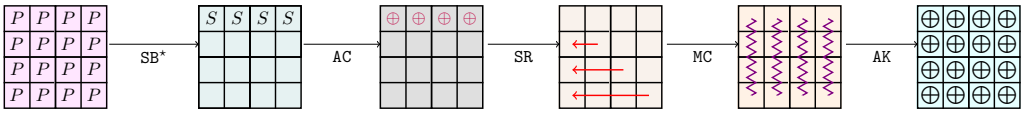
Definition 12 (Division Property)**.** Let $\mathcal{X}$ be a multiset of n-bit vectors. The multiset $\mathcal{X}$ has the division property $\mathcal{D}_{\mathcal{K}}^n$ where $\mathcal{K} \subseteq \mathbb{F}_2^n$ if for any $u \in \mathbb{F}_2^n$ such that there does not exist $k \in \mathcal{K}$ with $u \succeq k$ (where $u \succeq k$ means $u_i \geq k_i$ for all i), the XOR of $u \cdot x$ over all $x \in \mathcal{X}$ is 0. Each vector $u \in \mathbb{F}_2^n$ represents which input bits are considered, where $u_i = 1$ means the i-th bit is involved in the monomial.

Definition 13 (Division Property Table (DPT))**.** The Division Property Table (DPT) for an $n \times n$ SBox S is a $2^n \times 2^n$ table where rows correspond to input division property vectors $\omega \in \mathbb{F}_2^n$ and columns correspond to output division property vectors $\Omega \in \mathbb{F}_2^n$. An entry $\mathrm{DPT}[\omega, \Omega]$ is marked (feasible) if there exists a monomial x^ω in the ANF of the component function $x \mapsto \Omega \cdot S(x)$ with a non-zero coefficient, where $x^\omega = \prod_{i:\omega_i=1} x_i$. The DPT characterises how division properties propagate through the SBox and is essential for constructing MILP models in integral cryptanalysis.

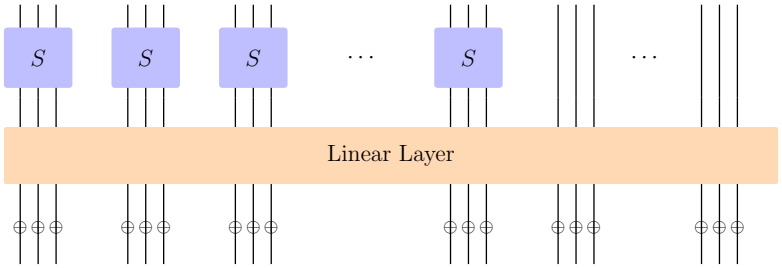
Definition 14 (Affine Equivalence (AE))**.** Two SBoxes $S_1, S_2 : \mathbb{F}_2^n \to \mathbb{F}_2^n$ are said to be affine equivalent if there exist invertible affine transformations $A : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and $B : \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that $S_2 = B \circ S_1 \circ A$. Here, $A(x) = M_A x \oplus c_A$ and $B(x) = M_B x \oplus c_B$ where M_A, M_B are invertible $n \times n$ binary matrices and $c_A, c_B \in \mathbb{F}_2^n$ are constant vectors.

Notice that $\mathrm{DDT}_S[0, 0] = 2^n$ as $S(x) \oplus S(x \oplus 0) = 0$ holds for all x and also $\mathrm{LAT}[0, 0] = 2^{n-1}$. This indicates, 0 is the trivial linear structure. If an SBox has a linear structure at α ($\neq 0$), then the αth of its DDT contains one 2^n and rest of this row contains 0. Also 0 is the trivial LS for any SBox. Further, note that the DU for a 4-bit SBox's DDT can be 16, but the maximum entry in its LAT can be ± 8. In general, the sign of the LAT entries is not useful, so we generally consider the absolute LAT. As a side-remark, notice that LS (i.e., where the DU is maximum) in the field of SCA has already been leveraged in [CG13]. For simplicity, we generally omit the leftmost column and the topmost row and all the zero entries of all the cryptographic tables.

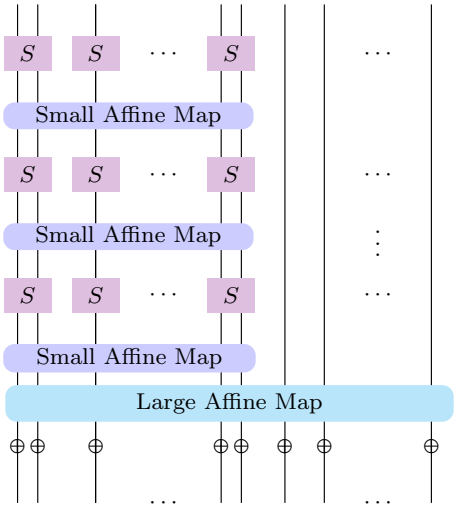
2.2 Partial Non-Linear Layer



(a) ZORRO round structure



(b) LOWMC round structure



(c) Generic round structure

Figure 1: Partial non-linear layer in block cipher construction

Mainstream block ciphers traditionally apply non-linear operations (SBoxes) uniformly across the entire state in each round. However, some niche-specific ciphers (cf. Section 3.1) explore partial non-linear layers where only a subset of the state undergoes non-linear transformation per round. The concept here involves applying SBoxes to only part of the cipher state in each round.

We show two representative examples of this approach in Figure 1a for ZORRO [GGNS13] and for LOWMC [ARS+15] (Figure 1b), where one can see that the non-linear operations (SBoxes) do not cover the full state. The generic structure of such partial non-linear layers is illustrated in Figure 1c[5]. Other examples of ciphers with partial non-linear layers include ARMADILLO [BDJ+10][6], FANTOMAS [GLSV14], PICARO [PRC12], SCREAM and

[5]We call *affine* map here instead of the more commonly used *linear* map, since the map may include a constant XOR (aside from a non-singular matrix multiplication).

[6]Historically, ARMADILLO was ahead of time compared to ZORRO [GGNS13], though it is stated in

ISCREAM [GLS+14].

2.3 Fault Attack (Selected Models)

2.3.1 Differential Fault Analysis (DFA)

Arguably DFA [BS97] is the most common type of fault attack. In DFA, the adversary induces faults during the tail end of the encryption process and analyses the differences between the correct and faulty ciphertexts to extract information about the secret key.

To provide a general overview, we describe the most basic form of DFA applied to a typical block cipher which follows the *Substitution Permutation Network* (SPN) structure. In this scenario, the adversary introduces faults into the final round's SBoxes individually (or it could be some other round near the final round). For each SBox S, the adversary induces a fault $\delta \neq 0$ and establishes a relationship of the form: $S(x) \oplus S(x \oplus \delta) = \Delta(x, \delta)$; where Δ represents a known value to the adversary, x denotes the unknown input value to be recovered, and δ may be either known (in deterministic scenarios) or unknown (in random scenarios). Nevertheless, by exploiting the input-output differential properties of S, the adversary can constrain the possible values for x. Subsequently, through accumulating several such relationships (and possibly by brute-force at a later stage), the adversary can determine x with certainty. Once the state preceding the final SBox layer is known, the final round key can be derived by performing an XOR operation between the ciphertext and the output from the last SBox layer.

A schematic representation of this attack is illustrated in Figure 2. For more details, one may refer to [BBB+23, Section 5].

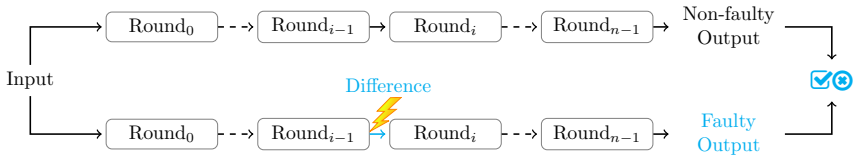


Figure 2: Schematic of DFA

2.3.2 Towards Fault Analysis (ToFA)

A recent work by Kundu et al. [KGAS25] introduces *Towards Fault Analysis* (ToFA) a new fault attack that represents the first successful application of impossible differential cryptanalysis to bit-oriented ciphers. The fundamental innovation of ToFA lies in its exploitation of truncated differential paths induced by fault injection in intermediate rounds. The attack methodology is grounded in the observation that fault-induced truncated differences propagate through the cipher structure in deterministic patterns.

Quotient-Remainder (QR) Group In the context of bit-permutation layers, Quotient-Remainder (QR) groups constitute independent sub-permutations operating on disjoint bit positions. For the 128-bit variant, the bit-permutation decomposes into eight independent 16-bit sub-permutations.

The core filtering mechanism computes intermediate difference values at the input of round $r - 1$ and verifies whether these differences admit impossible truncated patterns. When such impossibilities are detected, the corresponding partial key candidates are systematically eliminated from the search space. This process is enhanced through cross-validation between adjacent QR groups, where truncated patterns across QR group

[ARS+15] that "Together with ARMADILLO [BDJ+10], ZORRO [GGNS13] is the first SPN cipher in the literature that uses a non-full Sbox layer".

boundaries are verified for consistency, leading to the elimination of additional inconsistent key combinations. A summary of ToFA's performance can be seen from Table 2.

Table 2: Summary of ToFA on `BAKSHEESH` and `GIFT`

	`BAKSHEESH`	`GIFT`
Required number of faults	4	6
Time complexity	2^{20}	2^{24}
Key recovery success rate	100%	95%
Attack rounds covered	5th last to last	4th last to last

2.4 Side Channel Attack (SCA) and Countermeasure/Resistance

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 [Koc96], the electromagnetic emanation [GMO01] or the power consumption of the device [KJJ99].

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 [Koc96]. A more systematic approach is to use the so-called *Correlation Power Attack* (CPA) [BCO04], where the correct (sub-)key is recovered by taking the candidate which gives maximum (absolute) correlation.

2.4.1 Correlation Power Analysis (CPA)

Correlation Power Analysis (CPA) is an unprofiled side channel attack methodology that makes use of statistical correlations between measured power consumption traces and hypothetical intermediate values. Introduced by Brier et al. [BCO04], CPA employs the Pearson's correlation coefficient as a distinguisher to identify the correct key hypothesis among all candidates.

The Pearson's correlation coefficient for two random variables X and Y is defined as: $\rho = \mathrm{Cov}(X, Y) \div \sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)}$, where $\mathrm{Cov}(X, Y)$ represents the covariance between X and Y, and $\mathrm{Var}(\cdot)$ denotes the variance of a random variable. Let $\hat{\rho}(\vec{x}, \vec{y})$ denote the estimated correlation coefficient computed from observed data vectors $\vec{x}$ and $\vec{y}$. In the context of CPA, $\vec{x}$ represents the measured power leakage traces collected from the target device, whilst $\vec{y}$ represents hypothetical leakage values computed based on a key hypothesis and an assumed leakage model. The correlation coefficient ranges between -1 and $+1$, where the sign indicates the direction of correlation. For side channel analysis, we focus exclusively on the correlation magnitude, thus taking the absolute value $|\hat{\rho}|$.

The effectiveness of CPA depends critically on the leakage model. The two most commonly employed models are the *Hamming Weight* (HW) model and the Hamming Distance (HD) model. The HW model assumes that power consumption is proportional to the number of bits set to 1 in the manipulated data value at a given time instant. The HD model, by contrast, assumes that power consumption is proportional to the number of bit transitions ($0 \rightarrow 1$ or $1 \rightarrow 0$) between consecutive states. Under the Hamming weight model, the CPA score for key candidate k_c is computed as: $s_{k_c} = \max_j |\hat{\rho}(\vec{t}_{*,j}, \vec{h}_{k_c})|$, where $h_{k_c,n} = \mathrm{HW}(v(k_c, p_{n,c}))$ for each trace index n, with $v(k_c, p_{n,c})$ denoting the intermediate value computed using key candidate k_c and plaintext $p_{n,c}$, and $\vec{t}_{*,j}$ representing a vector consisting of the jth sample point across all power traces. The key candidate yielding the maximum correlation magnitude is identified as the most likely correct key value.

Countermeasure 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* [Cor00] and *Masking* [MTT+05]. 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.

2.4.2 Threshold Implementation (TI)

The *Threshold Implementation* is among the prominent masking countermeasures against side channel attacks [NRR06]. It is shown to be provably secure against higher order *Differential Power Analysis* (DPA) [Bil15].

The core principle of threshold implementation relies on secret sharing schemes where sensitive values are split into multiple shares such that knowledge of a subset of shares reveals no information about the original secret. This is achieved through internal randomisation, which is done in a way that ensures that the effect of the randomness is nullified at the end.

2.4.3 Test Vector Leakage Assessment (TVLA)

Test Vector Leakage Assessment (TVLA) [SM15] offers a statistically rigorous technique for identifying side channel leakage without requiring knowledge of the embedded secret key. The method contrasts two populations of traces, typically collected under fixed and random plaintexts, and determines whether their mean leakage differs beyond the background noise.

Welch's unequal-variance t-test is executed independently at every sampling instant. For a time index t the hypotheses are $H_0: \quad \mu_1(t) = \mu_2(t)$ vs. $H_1: \quad \mu_1(t) \neq \mu_2(t)$; with $\mu_i(t)$ denoting the expected leakage for population i. The test statistic is:

$$t(t) = \frac{\bar{X}_1(t) - \bar{X}_2(t)}{\sqrt{\frac{s_1^2(t)}{n_1} + \frac{s_2^2(t)}{n_2}}},$$

where $\bar{X}_i(t)$ is the sample mean of population i, $s_i^2(t)$ its unbiased variance estimate, and n_i the corresponding number of traces. A detection cut-off of $\tau = 4.5$ (i.e., $|t(t)| > \tau \Rightarrow$ leakage detected at time t) is employed, this provides a false-positive probability below 10^{-5} and is also aligned with industry practice.

3 Construction and Design Rationale

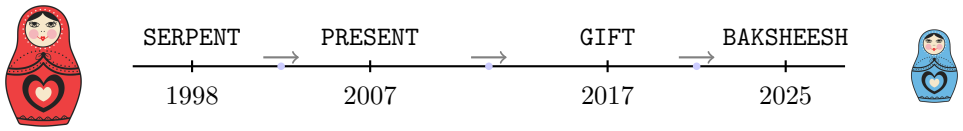
3.1 Motivation and Objective

The main motivation for `BAKSHEESH` originally comes from the question: *"Can we push the best aspects of* `GIFT`*'s design to push the benchmarks even further?"* Thus, `BAKSHEESH` is a young but a proud member of the lineage `SERPENT` [BAK98] $\rightsquigarrow$ `PRESENT` [BKL+07] $\rightsquigarrow$ `GIFT` [BPP+17a, BPP+17b], as shown in Figure 3a. For the culture, we could recall that `SERPENT` is among the five finalists and ultimately ranking 2nd in the NIST AES competition[7]. Also note that, `PRESENT` is an anagram of `SERPENT`; and most importantly that, `PRESENT` is an international standard (ISO/IEC 29192-2:2019).

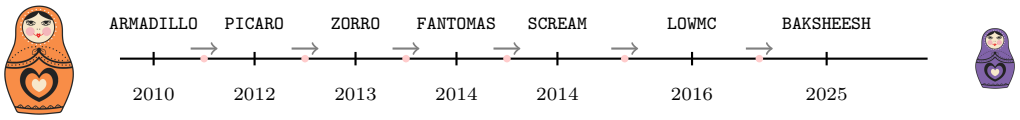
[7] https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development.

However, somewhat unexpectedly, we observe BAKSHEESH can be alternately interpreted as drawing inspiration from ZORRO, as shown in Figure 3b.[8] ZORRO is a lightweight block cipher that is designed with a focus on ease of masking against the side channel attack. It employs a partial non-linear layer and a linear diffusion layer to achieve both security and efficiency.

By standing at the confluence of both the GIFT and ZORRO lineages, BAKSHEESH aims to achieve a balance between general-purpose and niche-specific ciphers, making it a versatile choice for various applications. The evolution of ciphers in the context of partial non-linear layer is not as straightforward as in Figure 3a, so we follow the chronology in Figure 3b[9].



(a) GIFT lineage (canonical)



(b) ZORRO lineage (alternate)

Figure 3: 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 [Ull10]), 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 the minimum number of active SBoxes does not work anymore[10], since this method will always return a classical attack with trivial complexity. 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} [DR02]. 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 [Bak22, Chapter 8], by incorporating advancements 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).

The main objectives for BAKSHEESH can be summarised as follows:

1. Start with the same generic structure as GIFT-128 [BPP+17b], but replace the half round key XOR with a full round key XOR.
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 to have any significance in cipher design [Bak22]. With the advent of DEFAULT [Bak22], however, this status-quo has been altered. In terms of advantage, such an SBox typically has less number of AND operations and offers more efficient circuits.

[8] This alternate (ZORRO heritage) interpretation of BAKSHEESH is indicated in [Bak24].
[9] This is a simplistic and probably non-exhaustive representation.
[10] That said, the ZORRO heritage specification of BAKSHEESH uses a 3-bit SBox without a trivial LS, and hence it may be possible to use that concept with respect to this specification.

❸ 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` [BKL+07] or `GIFT` [BPP+17b]). Thus, `BAKSHEESH` makes the passing torch from `GIFT` glow brigher as it explores (arguably) the last frontier of using an SBox with a non-trivial LS in a cipher construction.

❹ Take advantage of *fix-slicing* [ANP21] to optimise implementation in software. This is probably the first cipher to take leverage of it (fix-slicing is proposed after `GIFT`).

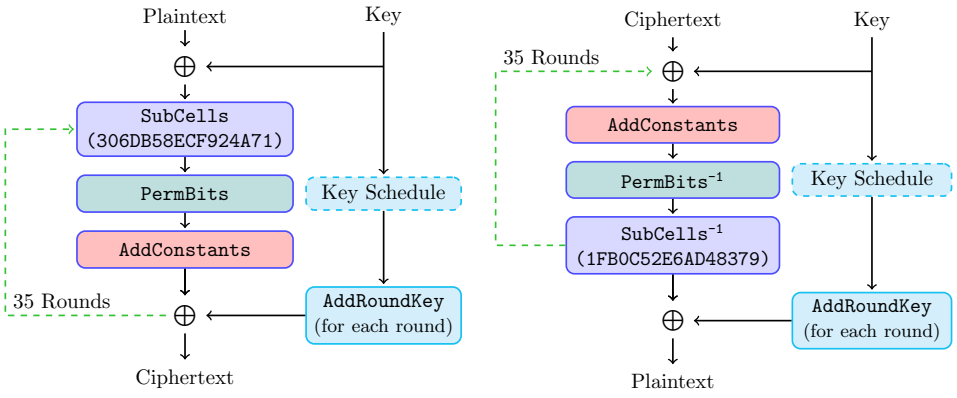
❺ Have low number of AND gate/depth. `BAKSHEESH` only needs 3 AND operations per SBox. This facilitates a highly efficient adoption of side channel countermeasures (as AND operations are considered the bottleneck [BDMD+20]). In this sense, `BAKSHEESH` fits well in the family of ciphers designed to be efficient in adopting side channel countermeasures [PRC12, GGNS13, GLSV14, KJK+20].

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: *Fully Homomorphic Encryption* (FHE) [CCF+16, MJSC16, ARS+15, DEG+18, AGR+16, HKL+22], *Multi-party Computation* (MPC, e.g., [DEG+18, ARS+15, AABS+20, AGR+16]), *Zero Knowledge* (ZK, e.g., [AABS+20, AGR+16]).

3.2 Construction Details: `GIFT` Heritage (Canonical)

3.2.1 Overview

`BAKSHEESH` is a 35-round block cipher that receives a 128-bit plaintext as the state $b_{127}b_{126}\cdots b_0$, where b_0 is the least significant bit. The state can also be expressed as $w_{31}w_{30}\cdots w_0$, where w_i is a 4-bit (nibble) word. We do not describe the inverse layer here for the sake of brevity. The round function of `BAKSHEESH` consists of 4 steps (in order): `SubCells` — applying a 4-bit SBox to the state, `PermBits` — permute the bits of the state (the same permutation as `GIFT-128` [BPP+17b], expressed here in the within-nibble bit-ordering convention clarified in [PLS24]), `AddRoundConstants` — XORing a 6-bit round constant to the state at six tap positions (`GIFT-128` uses seven) and `AddRoundKey` — XORing the round key to the state. One may refer to Figure 4 for an overview where Figure 4a shows encryption and Figure 4b shows decryption.



(a) Encryption **(b)** Decryption

Figure 4: Schematic of `BAKSHEESH` (`GIFT` heritage)

3.2.2 `SubCells` (4-bit SBox Layer)

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

3.2.3 `PermBits` (Bit-Permutation as Linear Layer)

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

3.2.4 `AddConstants` (Round Constant XOR Layer)

The following 6 tap positions are used to XOR the constants at each round: 11, 14, 16, 32, 64, 105. 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). The least significant bit of each round constant is XORed at the first tap position (11), the next bit at the second tap position (14), and so on. In particular, the least significant bit (at position 11) alternates between 0 and 1 in successive rounds (refer to Section 3.7 for more information).

3.2.5 Key Schedule Subroutine and `AddRoundKey` Layer

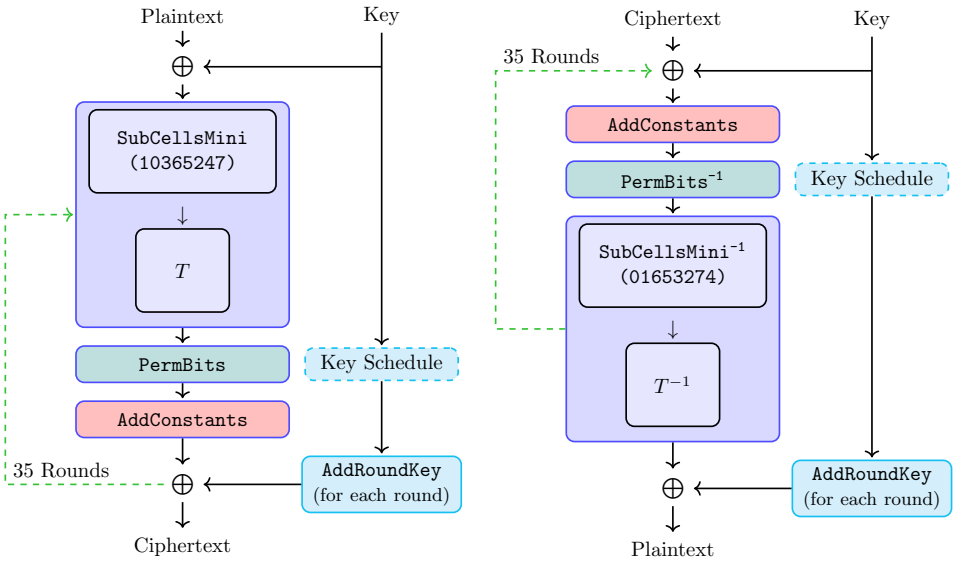
The first round key is the same as the master key. The next round keys are generated with $k^{j+1} \leftarrow k^j \ggg 1$. The rotation is taken in the word-oriented layout of the reference implementation; it is not a rotation of the plain hexadecimal string. The key as written in Table 5 splits into eight words of four hexadecimal characters, the leftmost word is the least significant word of a 128-bit integer, this integer is rotated right by one bit, and the words are written back in the same order. For example, the key `25798b6c2b24cd83281de76786213b86` from Table 5 has the word-reversed value `3b868621e767281dcd832b248b6c2579`, its rotation is `9dc34310f3b3940ee6c1959245b612bc`, and restoring the word order gives the next round key `12bc45b69592e6c1940ef3b343109dc3`. The reference implementation and the test vectors fix this convention. The round key for the jth round, denoted by k_i^j, is bit-wise XORed to the state: $b_i \leftarrow b_i \oplus k_i^j, \forall i \in \{0, \ldots, 127\}$. Unlike `GIFT-128`, `BAKSHEESH` has the key XOR at the beginning of encryption/at the end of decryption (can be seen from Figure 4).

3.3 Construction Details: `ZORRO` Heritage (Alternate)

3.3.1 Overview

We retain the 128-bit state $b_{127}b_{126}\cdots b_0$ (where b_0 is the least significant bit) arranged as thirty-two nibbles $w_{31}\cdots w_0$; but the non-linear layer now relies on the 3-bit SBox s, `10365247`. In this way, `BAKSHEESH` joins the ranks of ciphers with a 3-bit SBox such as `LOWMC` [ARS+15], `SEA` [SPGQ06] and `PRINTCIPHER` [KLPR10].

The linear layer `PermBits` (Section 3.2.3), `AddConstants` (Section 3.2.4) and the key schedule plus `AddRoundKey` layer (Section 3.2.5) remain unchanged from `GIFT` heritage description. Figure 5 illustrates the full round schematic of the alternate specification. The two heritages are equivalent interpretations of one cipher.



(a) Encryption **(b)** Decryption

Figure 5: Schematic of `BAKSHEESH` (`ZORRO` heritage)

3.3.2 `SubCellsMini` (3-bit SBox Layer)

The non-linear layer `SubCellsMini` applies the 3-bit SBox s (`10365247`) to the lower 3 bits of each nibble, whilst leaving the most significant bit x_3 of each nibble unchanged as in $(x_3, z_2, z_1, z_0) \leftarrow (x_3, s(x_2, x_1, x_0))$, $\forall i \in \{0, \ldots, 31\}$, where nibble $w_i = (x_3, x_2, x_1, x_0)$ and $(z_2, z_1, z_0) = s(x_2, x_1, x_0)$. Note that every 4th bit (i.e., 3, 7, 11, ..., 127) remains unchanged after this layer.

3.3.3 Linear Layer (128×128 Matrix T)

The entire 128-bit state is multiplied by a 128×128 block-diagonal matrix T (T is pre-multiplied to the state). This is applied after the 3-bit SBox layer to introduce the missing most significant bit linearly back into the state. The matrix T is given by:

$$T = \begin{pmatrix} T_4 & 0 & \cdots & 0 \\ 0 & T_4 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & T_4 \end{pmatrix}^{32\times 32} \quad \text{where} \quad T_4 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}.$$

Figure 6 depicts the action on one nibble: the SBox s transforms the lower 3 bits, then T_4 XORs the unchanged bit x_3 into all the 4 output bits (and $z_1 \oplus z_0$ into y_0).

3.3.4 Rationale/Justification

The `SubCells` layer from the `GIFT` heritage can be realised through a two-stage process combining `SubCellsMini` (3-bit SBox) and the linear map T as shown in Algorithm 1.

The coordinate functions of the 4-bit SBox demonstrate that x_3 appears only linearly; this enables factorisation into a nonlinear 3-bit core s that operates on (x_2, x_1, x_0), with the linear layer reintroducing x_3 linearly. This can be realised through the per-nibble using the $T_4^{4\times 4}$ matrix applied independently to each of 32 nibbles, which then can be described globally using the block-diagonal $T^{128\times 128}$ linear layer applied to the entire 128-bit state.

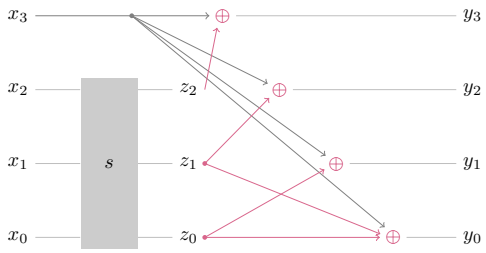


Figure 6: One-nibble view of 3-bit SBox layer and matrix T_4

Algorithm 1: `SubCells` (of `ZORRO` heritage) via `SubCellsMini` and T

Require: $x \in \{0,1\}^{128}$ represented as 32 nibbles $(w_{31}, \ldots, w_0)$
Ensure: `SubCells` (x) ▷ Realise `SubCells` as in Section 3.2.2
▷ Stage 1: Apply `SubCellsMini` (3-bit SBox to lower bits of a nibble)
1: **for** $i \in \{0, \ldots, 31\}$ **do** ▷ Process each nibble
2: $(x_3, x_2, x_1, x_0) \leftarrow w_i$ ▷ Extract nibble bits
3: $(z_2, z_1, z_0) \leftarrow s(x_2, x_1, x_0)$ ▷ Apply 3-bit SBox s
4: $w_i \leftarrow (x_3, z_2, z_1, z_0)$ ▷ Store intermediate result
▷ Stage 2: Apply T (128 × 128 linear transformation)
5: $y \leftarrow T \cdot (w_{31}, \ldots, w_0)$
6: **return** y as 128-bit state

Apart from making `BAKSHEESH` independent of `GIFT-128`, the `ZORRO` heritage specification connects with the partial non-linear layer[11] concept from `ZORRO` and similar niche-specific ciphers. Thus, `BAKSHEESH` sits at the crossroads of general-purpose and niche-specific ciphers.

3.4 Rationale behind Choice of 4-bit (`GIFT` Heritage) SBox

We choose the SBox `306DB58ECF924A71` for `BAKSHEESH` for its canonical `GIFT` heritage specification. It has 1 non-zero LS (at `8`), LBN of 3 and DBN of 2 (Definition 9). A comparison of cryptographic properties of few lightweight SBoxes are shown in Table 3. The coordinate functions (Definition 1) of this SBox are given respectively in ANF by:

$$y_0 = x_0x_2 \oplus x_0 \oplus x_1 \oplus x_3 \oplus 1, \quad y_1 = x_0 \oplus x_1x_2 \oplus x_3 \oplus 1,$$
$$y_2 = x_0x_2 \oplus x_1x_2 \oplus x_1 \oplus x_3, \quad y_3 = x_0x_1 \oplus x_0x_2 \oplus x_2 \oplus x_3.$$

It belongs to AE class #294 (of the 302 AE classes) [De 07]. The DDT of this SBox is given in Table 7.

3.4.1 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 3.1). For comparison with similar ciphers, one may notice that `PRINCE` [BCG+12a, BCG+12b], `SKINNY-64` [BJK+16] and `GIFT` [BPP+17b] SBoxes respectively require 28, 16 and 10 AND operations in the naïve sense (though this count can be reduced through optimisation).

[11] It seems the concept is a convergent evolution, a trait that independently appears more than once.

Notice from Table 15 that the naïve AND count plays a direct role in the cost for threshold countermeasure (also noted in [BGST23]).

When optimised, the BAKSHEESH SBox can be implemented using only three AND operations — which finally overtakes the SKINNY-64 SBox (that requires four AND operations when optimised).

3.4.2 Implementation Cost (Unprotected and Side Channel Protected)

The SBox is lightweight, as can be seen from the comparative ASIC benchmarks given in Table 4 by using the look-up-based format following [SMS19] and the figures are rounded to the nearest integer; the cost is expressed in Gate Equivalent (GE). 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: Cryptographic properties of few lightweight SBoxes

		DBN	LBN	LS	DU	AD Max.	AD Min.	Nl
BAKSHEESH	306DB58ECF924A71	2	3	0, 8	16	2	2	0
PRESENT [BKL+07]	C56B90AD3EF84712	3	2	0	4	3	2	4
SKINNY-64 [BJK+16]	C6901A2B385D4E7F	2	2	0	4	3	2	4
GIFT [BPP+17a]	1A4C6F392DB7508E	2	2	0	6	3	2	4
PYJAMASK-128 [GJK+19]	2D397BA6E0F4851C	2	2	0	4	3	2	4
MIDORI [BBI+15]	1053E2F7DA9BC846	2	2	0	4	3	3	4
	CAD3EBF789150246	3	2	0	4	3	2	4

AD: Algebraic Degree
Nl: Non-linearity

Table 4: ASIC benchmarks for a few lightweight SBoxes

		Cost (GE)		
		UMC 65nm	Faraday 65nm	STM 130nm
GIFT [BPP+17a]	1A4C6F392DB7508E	28	22	21
PYJAMASK-128 [GJK+19]	2D397BA6E0F4851C	28	26	22
SKINNY-64 [BJK+16]	C6901A2B385D4E7F	21	16	21
BAKSHEESH	306DB58ECF924A71	21	19	19

3.4.3 Branch Numbers

The problem of finding SBoxes with higher LBN is studied before [SMS19, KJK+20]. 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 [SMS19, 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 [SMS19]), 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, whilst keeping the AND count considerably low.

3.4.4 Search for BAKSHEESH SBox

At the top level, the search for the SBox consists of expanding the chosen AE classes (to generate several potential 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, even though we later replace it another permutation in the current version following the advice in [PLS24]) in the model to compute the differential and linear bounds

of [Bak20a] until a suitable SBox is found. Further, we do not need to consider the BOGI [BPP+17a, Section 3] or BOGI+ [SSS20] properties[12], as the automated modeling discards the SBoxes not suitable for our purpose.

Note that a desirable protection against SCAwe would want that the LS of the inverse SBox is at `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 `GIFT` and `DEFAULT` The filtering with 3-LBN is comparable to the filtering criteria in `GIFT` [BPP+17a, Section 3.3] or in `DEFAULT` as described in [Bak22, 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.

3.5 Rationale behind Choice of Linear Layer

The bit-permutation based linear layer is used in `PRESENT` [BKL+07] or more recently in `GIFT` [BPP+17a], among other ciphers. 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. However, we do not reuse the bit-permutation from `GIFT-128`, rather something different is chosen (although that is derived from `GIFT-128` and is suggested by [PLS24]).

Our choice of the bit-permutation also breaks away from the trope of using a *regular-looking* bit-permutation (probably for the first time). This contrasts the usual choice of the bit-permutation in ciphers like `PRESENT` [BKL+07] or `GIFT-128` or `TRIFLE-BC` [DGM+19].

3.6 Justification for 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` [GPPR11] or `PRINCE` [BCG+12b], to keep the implementation cost low. However, an invariant subspace attack [LMR15] 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., $k^{j+1} \leftarrow k^j \gg 1$ (the rotation is taken in the layout of Section 3.2.5). 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.

3.7 Generation of Round Constant and Tap Position

In `GIFT` [BPP+17a, 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 a separate toggle bit (forming 6-bit round constants that are XORed at 6 tap positions). Thus, we have one less tap position than `GIFT`[13].

[12] For instance, we allow one-round iterated trails in optimal differential/linear characteristics (see Table 9 for an example on the linear characteristics, which is confirmed in [PLS24, Figure 8]).

[13] As we know from [BBB+22], if the choice of the round constants is not justified, it may indicate the presence of a hidden backdoor. Somewhat unexpectedly, this seems to be missing in `GIFT`.

LFSR and Tap Position

Our LFSR is 5-bit with an irreducible primitive polynomial as its feedback function, $x^5 + x^2 + 1$ (taps at positions 2 and 0). It is initialised with 00001, uses right-shift operation and can be realised with one XOR gate. The 6-bit round constant is constructed by left-shifting the LFSR state by 1 position and placing a toggle bit at position 0 (LSB). The complete LFSR implementation is given in Algorithm 2. In each round, the 6-bit round constant[14] is applied by XORing individual bits to specific state bit positions.

Algorithm 2: `BAKSHEESH` round constants generation (via LFSR and toggle bit)

```
lfsr ← 00001_2                                  ▷ Initialise 5-bit LFSR
toggle ← 0                                      ▷ Initialise toggle bit
for r = 0 to 34 do                              ▷ Iterate for all rounds
    RC[r] ← (lfsr ≪ 1) | toggle                 ▷ 6-bit round constant with LFSR and toggle bit
    feedback ← lfsr[2] ⊕ lfsr[0]                ▷ Feedback polynomial x^5 + x^2 + 1
    lfsr ← (feedback ≪ 4) | (lfsr ≫ 1)          ▷ Right-shift with feedback to MSB
    lfsr ← lfsr ∧ 11111_2                       ▷ Keep 5 bits
    toggle ← 1 − toggle                         ▷ Adjust toggle bit
return RC                                       ▷ Round constants generated
```

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 [BBI+15, BCG+12b, BEK+21], is to use the digits of π. The reason we do not use something similar is, it does not guarantee a unique constant for each round.

Further Details

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 toggle 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; however, we read the positions from right-to-left as: 11, 14, 16, 32, 64, 105.

To be more precise, the round constant application follows a bit-wise XOR pattern as follows. For round r with round constant $RC[r]$, state bit is toggled if the corresponding bit of $RC[r]$ is set. Specifically, state positions 11, 14, 16, 32, 64 and 105 are XORed with bits 0, 1, 2, 3, 4 and 5 of $RC[r]$ respectively.

Implication on Fix-slicing

Going through the fix-slicing of `GIFT` [ANP21], 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 sub-routine (with justification and also with verification against the invariant subspace attack), despite its toll on fix-slice benchmark.

[14] Note that, the LFSR itself is 5-bit (initialised with 00001). The toggle bit is separate (1 bit, initialised with 0). The round constant is 6-bit (constructed as (lfsr ≪ 1) | toggle).

3.8 Complete Construction

3.8.1 Number of 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 rounds, 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 XOR.

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).

We keep the total number of rounds to be 35, this makes it faster than GIFT-128 (40 rounds) by 12.5%. For easier applicability of fix-slicing [ANP21], we keep the number of rounds to be multiple of 5, see Figure 12 for the quintuple-round structure in fix-slicing.

Table 5: Test vectors for BAKSHEESH

1	Key	00000000000000000000000000000000
	Plaintext	00000000000000000000000000000000
	Ciphertext	c002be5e64c78a72ab9a3439518352aa
2	Key	00000000000000000000000000000000
	Plaintext	00000000000000000000000000000007
	Ciphertext	1ba3363734c09a29f67c23bbb2cccc05
3	Key	00000000000000000000000000000000
	Plaintext	70000000000000000000000000000000
	Ciphertext	6f7d7746eaf0d97a154079f6bd846438
4	Key	00000000000000000000000000000000
	Plaintext	44444444444444444444444444444444
	Ciphertext	7ad3303667b2af6deef434dd110d7fb8
5	Key	ffffffffffffffffffffffffffffffff
	Plaintext	11111111111111111111111111111111
	Ciphertext	806f0cf45b94f0370206975fe78ac10f
6	Key	76543210032032032032032032032032
	Plaintext	789a789a789a789a789a789a789a789a
	Ciphertext	f568e4e628f723ee6d5fbacccf811523
7	Key	23023023023023023023023001234567
	Plaintext	b6e4789ab6e4789ab6e4789ab6e4789a
	Ciphertext	77acdeed645e49da25591bf5f32658a2
8	Key	5920effb52bc61e33a98425321e76915
	Plaintext	e6517531abf63f3d7805e126943a081c
	Ciphertext	ea5e0713249400a86ec61bb6cdffb364
9	Key	d9bb750877be6eebc339a2258c874fc9
	Plaintext	3325cfadce142135242f8e04752614ab
	Ciphertext	2b599d496ee6938cec0bae00c4947b7d
10	Key	25798b6c2b24cd83281de76786213b86
	Plaintext	57cf5777053928343047ebe8dbfc374f
	Ciphertext	62b9d55d799a3e784583e7377229a623

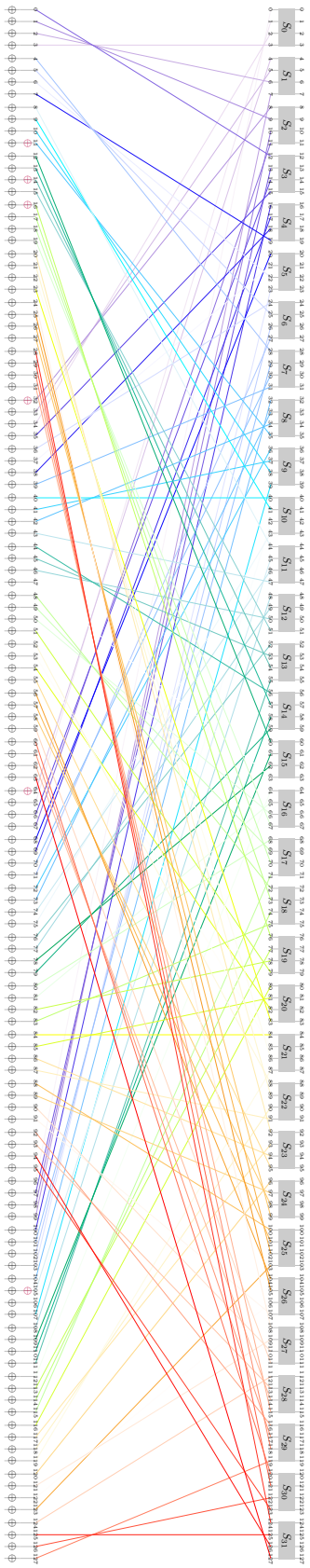


Figure 7: Structure of BAKSHEESH GIFT heritage encryption (one round)

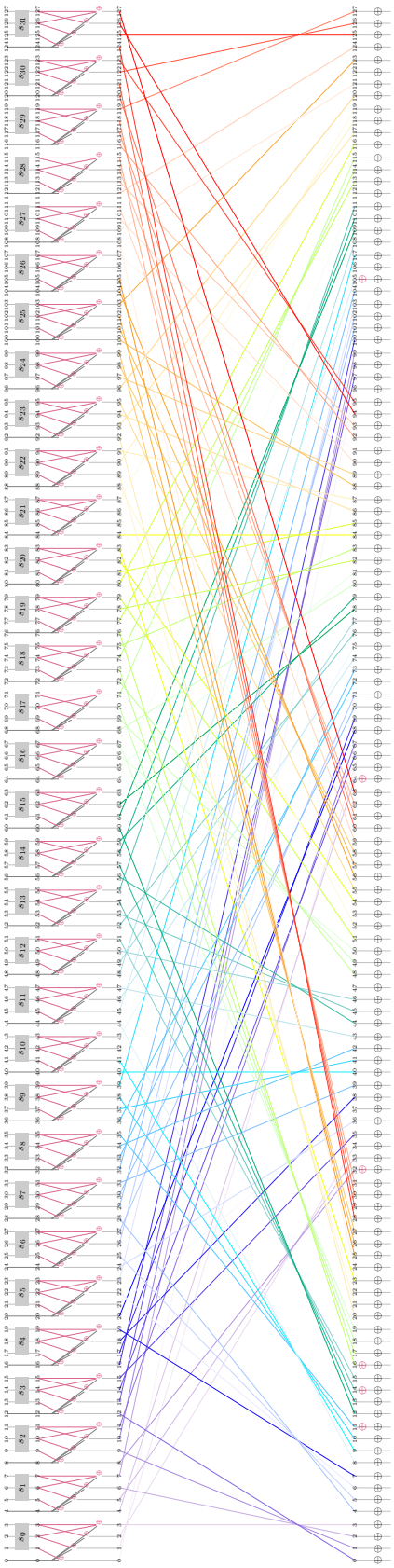


Figure 8: Structure of BAKSHEESH ZORRO heritage encryption (one round)

3.8.2 Structure and Test Vector

A few test vectors for BAKSHEESH are given in Table 5. The hexadecimal strings there follow the convention of the reference implementation: the plaintext and the key are read character by character, from left to right, into the 32 nibbles of the state and the key registers; the ciphertext is written with this nibble order reversed. For example, the last vector of Table 5 ends with the state 326a9227737e385487e3a997d55d9b26 in this reading order; the ciphertext is written as the reversed string 62b9d55d799a3e784583e7377229a623. A complete round-by-round encryption trace in this convention accompanies the public codes. Additionally the structures of BAKSHEESH GIFT heritage (one round) and BAKSHEESH ZORRO heritage (one round) are given in Figures 7 and 8, respectively. There the round constant XORs are shown in $\bigoplus$, and the round key XORs are shown with $\bigoplus$. In Figure 8, the MSB of each nibble bypasses its 3-bit SBox on the left, and the XOR operations for $T^{128\times128}$ are shown with $\bigoplus$ symbols (all the connections of T stay within the corresponding nibble).

As it can be seen from Figures 7 and 8, 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)[15].

3.9 Security Claim/Expectation against DFA

The main claim of DEFAULT [Bak22, Chapter 8] comes from the non-trivial 2^{64} search complexity against DFA, this stems from the 2 non-zero linear structures at the SBox. BAKSHEESH, however, use an SBox that has only 1 non-zero linear structure in its canonical GIFT heritage, which amounts to a trivial 2^{32} search complexity against DFA. This does not tantamount to a tangible solution, and thus, we do not claim any inherent protection against DFA in BAKSHEESH.

Apart from that, there have been some recent research works that seem to question the effectiveness of the linear structure based DFA protection [JKP25, KGAS25, NDE21]. We remain independent of these claims and do not take a position on applicability/validity of these papers[16].

4 Classical Security Claim/Analysis (GIFT Heritage)

4.1 Differential and Linear Attacks

In Table 6, we present the differential[17] and linear bounds as for BAKSHEESH together with GIFT-128. For the BAKSHEESH bounds, we follow the same SAT-based model from [SWW21]; and for this purpose Logic Friday[18] is used to construct a *Conjunctive Normal Form* (CNF) of an SBox. Further, results till round 12 are verified independently with the modelling from [Bak20a, Bak20b][19]. It is worth noting that, the optimal differential bound given in this paper is independently verified in [XCF+25, Table 7] and that of the linear bound in both [PLS24, Figure 3] and [XCF+25, Table 9]. On top of that, the linear bound in [PLS24, Figure 3] and [XCF+25, Table 9] actually cover till the full (35) rounds.

[15] Here the taps are named in the within-nibble bit-ordering used in the figures (the convention of [PLS24]); the positions 8, 13, 19, 35, 67 and 106 are the same as the positions 11, 14, 16, 32, 64 and 105 of Section 3.2.4 under this ordering.

[16] However, one may notice that the combined DFA on encryption and decryption, a model proposed/used in [NDE21], is outside the scope of DEFAULT [Bak22]. Indeed, we show how this model can break DEFAULT with an example in Section D.

[17] As the BAKSHEESH SBox's DDT_S $[\delta, \Delta]$ only contains multiples of 4, the differential bounds are even.

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

[19] As a side note, it can be mentioned that this modelling is used in the initial search for the SBox.

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 9. 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) trails[20]. Also, it can be seen from the optimal linear trail that an input mask is iterated (there exists a one-round iterated linear trail from the 4th to 21th round with 3 active nibbles).

Table 6: Optimal differential and linear bounds for GIFT-128 and BAKSHEESH

	Round	1	2	3	4	5	6	7	8	9	10	11
GIFT-128	Differential	1.415	3.415	7.000	11.415	17.000	22.415	28.415	39.000	45.415	49.415	54.415
	Linear	1.000	2.000	3.000	5.000	7.000	10.000	13.000	17.000	22.000	26.000	31.000
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	67.830	79.00	85.415	90.415	96.415	103.415	110.83	121.415	126.415	132.415
	Linear	36.000	38.000	41.000	45.000	48.000	51.000	56.000	59.000	64.000	68.000	74.000
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

Table 7: DDT for BAKSHEESH (4-bit) SBox

δ \ Δ	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															16
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			

Probability 1 Transition

At the surface level, 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 attacks[21]. 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 [Bak22, Section 7] for relevant discussion).

[20] Note 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 the attacker is concerned, the two events (*always happens* and *never happens*) are deterministic in nature.

[21] This is the case for DEFAULT [Bak22, Chapter 8] as well.

Table 8: LAT for BAKSHEESH (4-bit) SBox

γ \ Γ	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	

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 → 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 → 0 or 8 → 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 $8 \notin \{f, 0\}$, 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 much lower than the 4-round optimal differential probability, 2^{-8}.

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

4.2 Algebraic Attack

The algebraic degree of the SBox and its inverse is 2 in either cases. Using Sage[22], the inverse of the SBox can be represented as follows:

$$\begin{aligned} x_0 &= (y_0 \oplus y_3)(y_1 \oplus y_2) \oplus y_2 \oplus y_3 \oplus 1, \\ x_1 &= y_0y_2 \oplus y_3(y_0 \oplus y_2) \oplus y_0 \oplus y_1 \oplus y_3, \\ x_2 &= y_0(y_1 \oplus y_2) \oplus y_0 \oplus y_1y_2 \oplus y_2 \oplus y_3, \\ x_3 &= y_0 \oplus y_1 \oplus y_2. \end{aligned}$$

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

[22] https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/sbox.html.

Table 9: Optimal differential and linear trails for 22-round BAKSHEESH

Round	Differential	Linear
1[♀]	00000000 00000000 90049000 00000000	00000000 00000000 00004409 0000000a
2	0000c400 00000000 00000000 00000000	00000000 00000000 00000a00 00004401
3	00000000 00000000 09000000 00000000	00000000 00000000 0000000a 00004405
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	000060e 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

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, whilst the lowest algebraic degrees are 2, 4, 8 and 16, respectively. Both figures show an exponential growth.

Key Recovery with Algebraic Method

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 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 [BFP12], 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.

4.3 Integral Attack

The concept of the integral attack is first proposed in [DKR97], then it is formalised in [KW02]. Relatively recently, the division property that can find the integral distinguisher more efficiently, is proposed in [Tod15]. We first search the integral distinguisher based on

Table 10: DPT for BAKSHEESH (4-bit) SBox

ω \ Ω	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖	✖
1		✖	✖		✖				✖							
2		✖	✖		✖				✖							
3				✖		✖	✖		✖							
4		✖	✖		✖				✖							
5		✖			✖				✖							
6			✖		✖					✖						
7												✖		✖	✖	
8		✖	✖		✖				✖							
9				✖		✖	✖			✖	✖		✖			
a				✖		✖	✖			✖	✖		✖			
b								✖		✖	✖		✖			
c				✖		✖	✖			✖	✖		✖			
d				✖			✖				✖			✖		
e				✖		✖					✖		✖			
f																✖

the bit-based division property by the MILP searching tool proposed in [XZBL16] 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 given in Table 10, where $u = (u_3, u_2, u_1, u_0)$ and $v = (v_3, v_2, v_1, v_0)$ denote the input and output division property, respectively.

Similar to GIFT [BPP+17a, Table 12], a position marked with ✖ means the propagation from ω to Ω is feasible (a trivial propagation is marked by ✖). 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.

GIFT-128 has the 11-round integral distinguisher as the longest integral distinguisher by the bit-based division property [BPP+17a] whilst 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.

4.4 Impossible Differential Attack

The impossible differential attack [BBS99] makes use of the pair of (non-zero) input and output differences that the input difference never reaches the output difference. In this context, we talk about *full diffusion* first. 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 from the forward (encryption) and backward (decryption) directions, impossible differences may exist over 9 rounds of BAKSHEESH.

Following [ST17], we only check for Hamming weight 1 (input difference, output difference) pairs. Thus, there are $\binom{128}{1} \times \binom{128}{1} = 2^{14}$ such pairs. For 8-round BAKSHEESH, we find that some impossible differences like $0000000000000000000000000000000\mathtt{1} \longrightarrow 0000000000000000000000000000000\mathtt{1}$ exists. However, such impossible difference patterns disappear starting the 9th round. Therefore, we expect there is no impossible difference in more than or equal to 9-rounds of BAKSHEESH.

4.5 Invariant Subspace Attack

The invariant subspace attack [LAAZ11, LMR15] is considered a major cryptanalytic technique. Its feasibility relies on the existence of a subspace V and a constant u which are invariant for the round transformation.

In our preliminary design, the round key is kept identical to the master key. For such a scheme, the algorithm to detect the invariant subspace proposed in [LMR15] can be trivially applied. Due to the sparse round constants, i.e., only 6 bits of fixed positions will be affected by the constant addition operation, it is soon reported by the code provided in [LMR15] that the scheme is vulnerable against the invariant subspace attack.

To resist this attack vector, we tweak the key schedule and choose to introduce 1-bit right rotation. In this way, the algorithm in [LMR15] does not apply any more. Intuitively, such a way makes the round keys sufficiently different, though still being the permutation on each other. To make all the round keys belong to the detected invariant subspace, there will be many bit conditions on the key bits and the number of these conditions will exceed 128, which causes that contradiction easily occurs or that the space of weak keys is rather small, i.e., when all key bits are all 0 or all 1. Thus, our scheme is secure against the invariant subspace attack.

Similar to the analysis of GIFT, we first search for the subspace transitions through the SBox. It is found that there are 8 transitions with $V's$ dimension 1 and 1 transition with $V's$ dimension 2, as listed next:

$$\begin{aligned}
\{\mathtt{0},\mathtt{a}\} \oplus 0 &\rightarrow \{\mathtt{0},\mathtt{a}\} \oplus \mathtt{3},\\
\{\mathtt{0},\mathtt{5}\} \oplus \mathtt{3} &\rightarrow \{\mathtt{0},\mathtt{5}\} \oplus \mathtt{d},\\
\{\mathtt{0},\mathtt{c}\} \oplus \mathtt{3} &\rightarrow \{\mathtt{0},\mathtt{c}\} \oplus \mathtt{d},\\
\{\mathtt{0},\mathtt{9}\} \oplus \mathtt{6} &\rightarrow \{\mathtt{0},\mathtt{9}\} \oplus \mathtt{8},\\
\{\mathtt{0},\mathtt{c}\} \oplus \mathtt{7} &\rightarrow \{\mathtt{0},\mathtt{c}\} \oplus \mathtt{e},\\
\{\mathtt{0},\mathtt{9}\} \oplus \mathtt{7} &\rightarrow \{\mathtt{0},\mathtt{9}\} \oplus \mathtt{e},\\
\{\mathtt{0},\mathtt{5}\} \oplus \mathtt{b} &\rightarrow \{\mathtt{0},\mathtt{5}\} \oplus \mathtt{2},\\
\{\mathtt{0},\mathtt{7},\mathtt{8},\mathtt{f}\} \oplus \mathtt{1} &\rightarrow \{\mathtt{0},\mathtt{8},\mathtt{f},\mathtt{7}\} \oplus 0.
\end{aligned}$$

For dimension 3, no subspace transition exists.

An interesting property of the linear layer of GIFT-128 is that if there does not exist key addition or constant addition operations, when the inputs to all the SBoxes are set as the same value α_0, after 1-round permutation, the inputs to all the SBoxes must be the same and the values must be $S(\alpha_0)$. Therefore, it is interesting if there exists $\alpha_0 = S(S(\alpha_0))$. Indeed, there are some useful fixed point[23] satisfying such a constraint, as: $7 = S(e), e = S(7), 5 = S(5)$. Hence, to identify a meaningful weak key space, i.e., the number of weak keys is larger than 1, we can impose the following constraint on the round constant rc_i and round key rk_i for each round:

$$(rc_i \oplus rk_i) \in \{000\ldots0, 999\ldots9\}. \tag{1}$$

Due to the influence of whitening key addition, the whitening key wk should also satisfy

$$wk \in \{000\ldots0, 999\ldots9\}. \tag{2}$$

If the above constraints can hold, for the plaintext $P \in \{777\ldots7, eee\ldots e\}$ each nibble of the ciphertext will be the same and its value is either 7 or e. However, due to our choice of round constants and the key schedule,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 [TLS16], 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.

4.6 Other

4.6.1 Machine Learning Assisted Attack

Machine Learning (ML) has recently emerged as a prominent method for classical cryptanalysis [Goh19,BBCD20,BBD+23]. 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 12.5% efficiency over (patched) GIFT-128.

4.6.2 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 [BPP+17a, Section 2], we recommend to double the number of rounds (the round constants would reiterate to accommodate for the additional rounds), should there be any necessity. In that case too, BAKSHEESH would maintain its 12.5% advantage (in less number of rounds) over GIFT-128. That said, we refer to [XCF+25] for further analysis in the related key setting.

[23] Note that 5 is a fixed point for the 4-bit BAKSHEESH SBox, i.e., $S(5) = 5$.

4.6.3 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) [AS09] is upper bounded by 8 $(= (4-1)+(5-1)+1)$. The condition for the *Initial Structure* (IS) [SA09] 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 [AS09] 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.

4.6.4 Biclique Attack

The claim against the biclique attack [BKR11] on BAKSHEESH is analogous to that of SKINNY [BJK+16, Section 4.2]. More specifically, this attack slightly improves the complexity of exhaustive search by computing only a part of the 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 a serious vulnerability in the future. Therefore, we do not claim any security against the biclique attack, although the impact of it against BAKSHEESH will be marginal.

5 Device Dependent Attack with Countermeasure

5.1 Fault Attack: Resisting DFA and ToFA

Fault attacks (FA) constitute a serious threat to small-scale devices performing cryptographic operations by altering the regular flow of execution [BBB+23]. We do not claim any in-built FA security for BAKSHEESH (hence a standard countermeasure is recommended).

In general, the simplest countermeasure against FA is to perform redundant computations and compare the results, then follow a recovery procedure if a mismatch is detected. This is the so-called *detection*-based approach. However, this approach comes with the bottleneck that the variable checking for mismatch itself can be faulted, even though that is more challenging. To mitigate this, one can use *infection*-based to propagate the fault to mask the computation [BSS20].

More sophisticated countermeasures exist, but for our case, we focus on protection against DFA and ToFA. For this purpose, we recommend either detection-based or infection-based countermeasures, depending on the presumed strength of the adversary. With our discussion (in a private communication) with the proponents of ToFA [KGAS25], we can confirm that either are sufficient to prevent DFA and ToFA on BAKSHEESH (no specialised countermeasure would be necessary). In fact, this was the recommendation from their side.

It is not clear how to design a cipher that is inherently resistant to DFA and ToFA. There have been differential fault attack related claims on DEFAULT (such as, [JKP25, KGAS25, NDE21]). Even though we stay neutral about the claims made in those papers, we can say that the DFA security of DEFAULT is in question. Therefore, we remain careful not to make any claim about the inherently applicable (i.e., purely from the algorithmic perspective) DFA/ToFA security of BAKSHEESH.

5.2 Side Channel Attack: Power and Prevention

5.2.1 Idealistic Scenario with LS SBox

The XOR operation exhibits inherent ambiguity in side channel attacks under CPA analysis [YO19, Bak21]. Under noiseless assumptions, if input x produces correlation ρ, then $\vec{1} \oplus x$ yields correlation $-\rho$, where $\vec{1}$ represents the all-ones vector. Whilst XOR provides some intrinsic resistance, constructing ciphers purely from linear operations is cryptographically infeasible. The non-linear SBox layer remains the primary target for side channel attackers.

The concept of SBoxes with linear structures extends this XOR ambiguity property. For an $n \times n$ SBox S where S^{-1} has a linear structure at $\vec{1}$, the theoretical correlation coefficients for $S(\alpha)$ and $S(\alpha \oplus A)$ exhibit equal magnitude with opposite signs for all α, where $A = S^{-1}(x) \oplus S^{-1}(x \oplus \vec{1})$ represents the image of the linear structure. This suggests potential key-candidate ambiguity analogous to one XOR operation.

Practicality Our experimental results (presented subsequently in Section 5.2.2) demonstrate that the unprotected `BAKSHEESH` SBox (possessing one non-trivial linear structure with $\alpha = 8$) exhibits reduced but still exploitable leakage compared to `GIFT` (which has no non-trivial linear structures). Specifically, `BAKSHEESH` software implementations show 36% lower correlation magnitude ($|\rho| = 0.564$ vs. 0.884 for `GIFT`), yet key recovery remains feasible with 100000 traces. Consequently, we do not claim inherent side channel security from linear structures alone.

5.2.2 SCA Evaluation

Set-up We evaluate implementations on Sakura G-II and Sakura W boards. Software experiments target an ATMega8515 microcontroller, whilst hardware experiments employ a Xilinx Spartan-6 FPGA. Power traces are acquired using a Keysight MSOX4154A oscilloscope at 1 GS/s sampling rate. Experiments use 100000 power traces per implementation.

Methodology CPA targets the SBox output $S(p \oplus k)$ using Hamming weight models, testing all 16 key hypotheses (4-bit key nibble). TVLA employs Welch's t-test comparing fixed versus random plaintext groups (100000 traces each). *Partial Guessing Entropy* (PGE) tracks the rank of the correct key candidate across trace counts. For fair comparison, both `BAKSHEESH` and `GIFT` implementations use full round key XOR at the SBox input (even though this is not the specification of `GIFT`). PGE measures attack effectiveness; rank 1 indicates successful key recovery, whilst rank 16 (worst) indicates complete protection. TVLA t-statistics $|t| > 4.5$ detect first-order leakage [SM15].

5.2.3 Result on Software Implementation

Analysis of unprotected software implementations using 100000 traces reveals distinct leakage profiles between the two SBoxes (see Table 11). The `GIFT` implementation exhibits maximum correlation $|\rho| = 0.884$ with PGE rank 1/16, enabling successful key recovery through standard CPA methodology. In comparison, `BAKSHEESH` shows maximum correlation $|\rho| = 0.564$ with PGE rank 1/16, representing 36% reduced correlation magnitude whilst remaining vulnerable to key extraction. Both implementations exhibit t-statistics exceeding the 4.5 cut-off in TVLA testing (`GIFT`: 5.12, `BAKSHEESH`: 4.89), confirming first-order leakage detection consistent with successful CPA attacks.

5.2.4 Result on Hardware Implementation

Hardware evaluation on FPGA substrates demonstrates consistent vulnerability patterns across both SBox implementations (see Table 11 for detailed results). The `GIFT` hardware implementation achieves maximum correlation $|\rho|= 0.767$ with PGE rank 1/16, confirming successful key recovery. The `BAKSHEESH` hardware implementation shows maximum correlation $|\rho|= 0.628$ with PGE rank 1/16, representing 18% reduced leakage compared to `GIFT` hardware whilst remaining vulnerable. TVLA t-statistics for hardware implementations also exceed the cut-off (`GIFT`: 4.93, `BAKSHEESH`: 4.71), consistent with FPGA leakage characteristics.

Figure 9 presents CPA profiles for `BAKSHEESH` across software and hardware platforms, both unprotected and TI-protected. Here we use the (secret) key nibble `5`. The threshold implementation exhibits reduced distinguishability between key hypotheses compared to unprotected baselines. Figure 10 shows TVLA t-statistic profiles for all implementations.

We evaluate a 3-share (first-order) masking scheme for `BAKSHEESH` on both software and hardware platforms using 100000 traces per implementation (see Table 12). The software protected implementation exhibits maximum correlation $|\rho|= 0.037$ with PGE rank 16/16, representing 93.4% reduction in correlation magnitude compared to the unprotected software baseline. The hardware protected implementation achieves maximum correlation $|\rho|= 0.023$ with PGE rank 15/16, demonstrating 96.3% reduction from the unprotected hardware baseline. TVLA verification confirms all protected implementations remain below the 4.5 cut-off (software: 3.083, hardware: 2.979), with no statistically significant first-order leakage detected at the measured trace count. Standard CPA attacks fail to recover key material from protected implementations, with the correct key consistently ranking poorly among candidates. The details of the TI can be found in Section 6.3.

Table 11: CPA results on unprotected `GIFT` and `BAKSHEESH`

		Maximum $\|\rho\|$	Key Recovery	PGE Rank
`GIFT`	Software	0.884	✓	1/16
	Hardware	0.767	✓	1/16
`BAKSHEESH`	Software	0.564	✓	1/16
	Hardware	0.628	✓	1/16

Lower $|\rho|$ 👍

Table 12: TVLA results on `GIFT` and `BAKSHEESH`

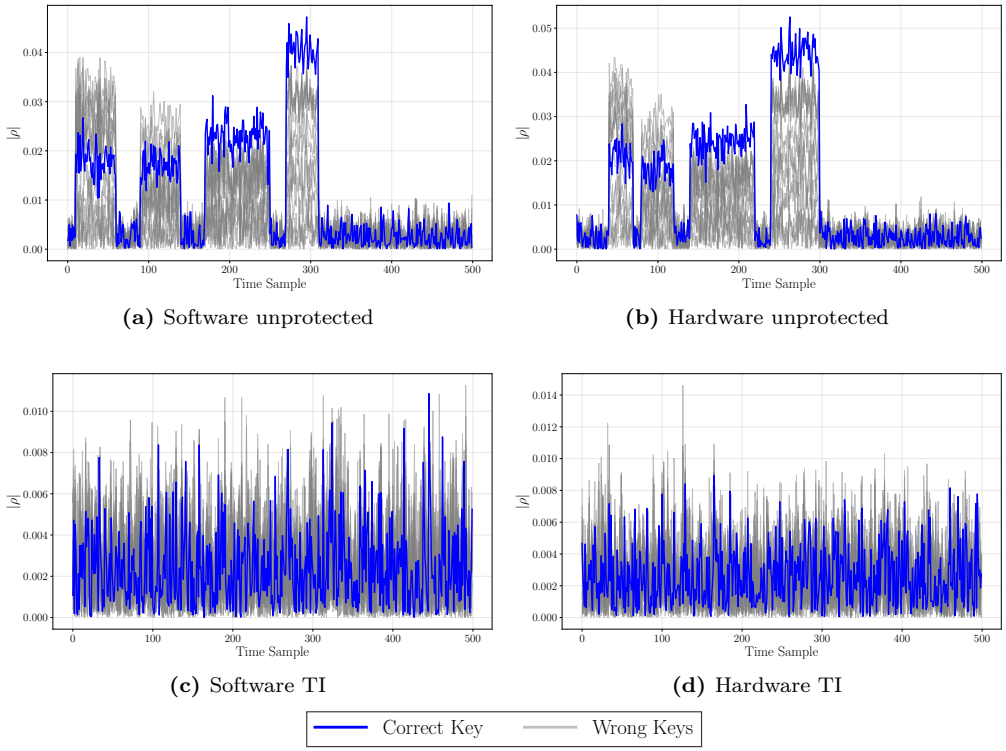
	Max. $\|\rho\|$	Key Rec.	PGE Rank	TVLA ($\|t\|$)
Unprotected `GIFT` (SW)	0.884	✓	1/16	5.12
Unprotected `GIFT` (HW)	0.767	✓	1/16	4.93
Unprotected `BAKSHEESH` (SW)	0.564	✓	1/16	4.89
Unprotected `BAKSHEESH` (HW)	0.628	✓	1/16	4.71
TI `BAKSHEESH` (SW)	0.037	✗	16/16	3.08
TI `BAKSHEESH` (HW)	0.023	✗	15/16	2.98

SW: Software

HW: Hardware

Lower $|\rho|$ and TVLA 👍

TVLA analysis with 100000 traces reveals clear distinction between unprotected and protected implementations (detailed metrics are given in Table 12). Figure 10 illustrates TVLA t-statistic profiles. Unprotected implementations exhibit t-statistics ranging from 4.71 to 5.12 across software and hardware platforms, exceeding the conventional 4.5 cut-off and confirming detectable first-order leakage consistent with successful CPA-based key recovery.



(a) Software unprotected

(b) Hardware unprotected

(c) Software TI

(d) Hardware TI

Figure 9: CPA (correlation) profiles for `BAKSHEESH`

In contrast, protected implementations yield t-statistics of 3.08 (software) and 2.98 (hardware), remaining below the detection threshold and confirming absence of detectable first-order leakage under the masking scheme. These results demonstrate that whilst unprotected implementations are vulnerable to both CPA and TVLA-based analysis, protected implementations successfully resist first-order attacks under identical experimental conditions.

Figure 11 provides a direct comparative analysis between `BAKSHEESH` and `GIFT`, and it illustrates the relative leakage characteristics and CPA correlation profiles for both ciphers. Across both software and hardware platforms, `BAKSHEESH` consistently exhibits lower correlation peaks compared to `GIFT`.

In Tables 11 and 12, results marked in green indicate expected behaviour under the respective security assumptions. For unprotected implementations, successful key recovery (✓) is expected, whilst for protected implementations, failed key recovery (✗) demonstrates the effectiveness of the countermeasure. In Table 12, TVLA values in red indicate leakage detected ($|t|> 4.5$).

6 Implementation and Performance

6.1 Hardware (Unprotected)

In the following, we tabulate the synthesis results for unprotected hardware implementation of `BAKSHEESH` and compare it with that of `GIFT-128`. For the purpose of benchmark, we report costs incurred in FPGA, namely 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

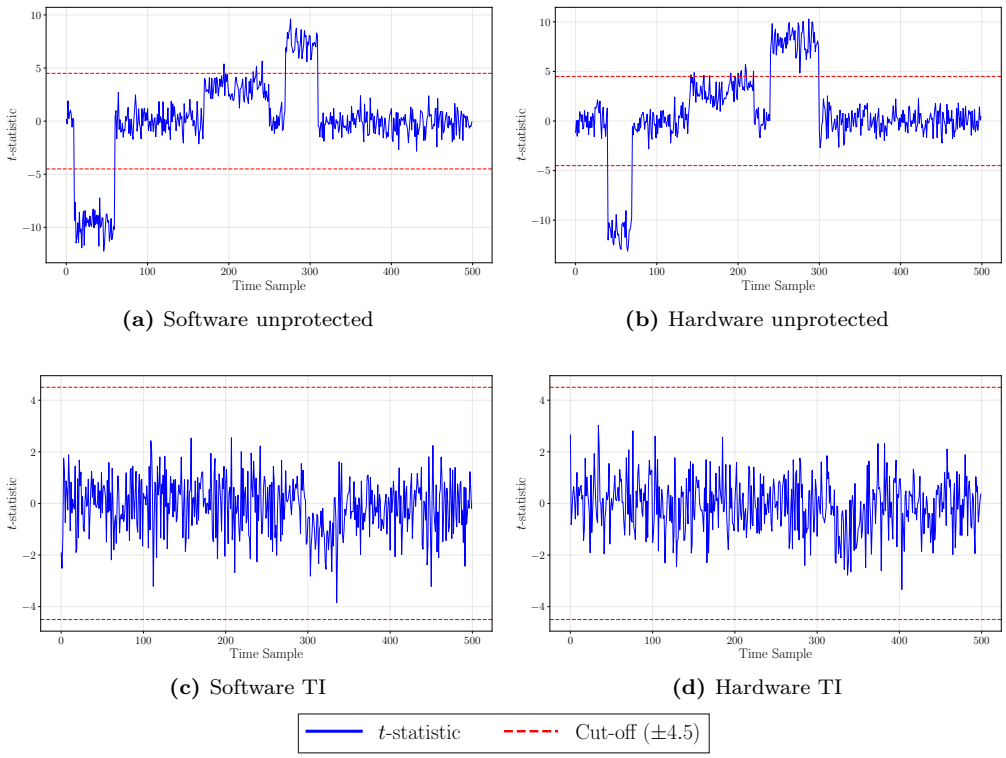


Figure 10: TVLA (t-statistic) profiles for `BAKSHEESH` implementations

and FPGA synthesis utilising Xilinx Vivado 2020.2. See Table 13 for an overview. The same HDL implementation is used to benchmark in ASIC and FPGA, the round keys are computed on-the-fly.

Table 13: Hardware benchmarks (unprotected): `BAKSHEESH` vs. `GIFT-128`

(a) ASIC (TSMC 65nm)

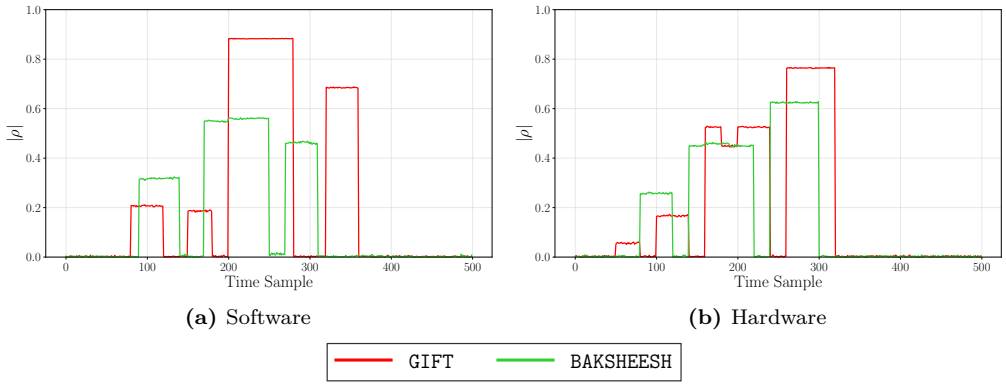
	Area (GE)	Critical Path (ns)	Max. Freq. (MHz)
`BAKSHEESH`	3554	1.282	780
`GIFT-128`	3264	1.818	550

Lower critical path and higher frequency 👍

(b) FPGA (UltraScale+ and Kintex)

	LUT	F/F	Maximum Frequency (MHz)	
			UltraScale+	Kintex
`BAKSHEESH`	252	270	847	454
`GIFT-128`	235	262	830	546

ASIC One may note from Table 13a that `BAKSHEESH` takes more area (3554 GE vs. 3264 GE) but offers significantly higher frequency (780 MHz vs. 550 MHz, a 41.8% improvement). Whilst the unprotected area cost is higher for `BAKSHEESH`, this cost is more than compensated by the substantial frequency improvement, and also the significantly better performance for protected implementation as given in the threshold implementation results in Section 6.3.



(a) Software **(b)** Hardware

Figure 11: CPA (correlation) profiles of `BAKSHEESH` and `GIFT`

FPGA We observe from Table 13b that the number of LUT utilised for `BAKSHEESH` is very close to that of `GIFT-128`. Number of computation cycles for unprotected implementation of `BAKSHEESH` is 17.

6.2 Software

Our benchmarks employ pre-computed round keys. The `BAKSHEESH` implementation maintains full round key XOR (128 bits) as specified in its design, whilst `GIFT` utilises half round key XOR (64 bits) following its original specification. All the software benchmarks reported here are for the `GIFT` heritage interpretation. Although the `GIFT` and `ZORRO` heritages describe the same cipher, the `ZORRO` heritage combines a 3-bit SBox with a 4×4 binary T-matrix, for which a suitable fix-slice pattern is not yet known, so a direct implementation stays in the nibble domain and is markedly slower. We therefore do not report `ZORRO` benchmarks, and leave its optimised fix-slice as future work.

For ARM benchmarks, we use ARM Cortex-M3 embedded microcontrollers with the `arm-linux-gnueabi-gcc` toolchain and GCC 13.3.0 with `-O3` optimisation for comprehensive performance characterisation across ARM architectural paradigms. Additionally, we evaluate performance using QEMU ARM emulation (configured with ARM926EJ-S processor, 2GB RAM, Ubuntu 22.04) for broader ARM ecosystem validation. Table 14 presents results across unprotected and masked implementations.

The assembly implementations employ register-level optimisations and instruction scheduling to maximise throughput whilst minimising memory access overhead. Results demonstrate that `BAKSHEESH` achieves consistent performance advantages across all implementation approaches, as shown in Table 14a. With 7.31 cycles/byte speed, we achieve throughput of 98.57 MB/s for ARM QEMU emulation (running on ARM926EJ-S 720MHz, Ubuntu 22.04), we observe that `BAKSHEESH` achieves 1.47× throughput improvement over `GIFT-128`.

First-order masking implementation follows established fix-slice methodologies [ANP21]. Results in Table 14b demonstrate significant performance advantages for `BAKSHEESH`, with `GIFT-128` requiring 48% additional computational overhead.

Fix-slice Pattern

As we do not reuse the same bit-permutation as `GIFT-128`, we define new fix-slice patterns for `BAKSHEESH`. We split the state into 4 words of 32 bits each. The `BAKSHEESH` fix-slice implementation employs a quintuple-round approach where five consecutive permutation layers are grouped and optimised using a combination of rotations and mask operations

Table 14: Fix-sliced benchmarks of BAKSHEESH and GIFT-128

(a) Performance

		BAKSHEESH	GIFT-128
ARM Cortex-M3	C (-O3)	128 cyc/byte (0.82×)	156 cyc/byte
	Compact	126 cyc/byte (0.82×)	154 cyc/byte
	Fast	113 cyc/byte (0.81×)	140 cyc/byte
QEMU ARM	Fix-slice	7.31 cyc/byte (0.68×)	10.75 cyc/byte

Lower cycles per byte 👍

(b) First-order masking

		BAKSHEESH	GIFT-128
Assembly	Clocks (Cycles/Byte)	210 (0.67×)	312 (1.00×)
	Code Size (Bytes)	2088 (0.83×)	2504 (1.00×)

similar to GIFT-128, but adapted for BAKSHEESH's unique bit-permutation structure. Figure 12 illustrates the complete quintuple-round fix-slice pattern.

The fix-slicing of the bit-permutation of BAKSHEESH is ported from that of the GIFT-128 bit-permutation. The quintuple-round structure is implemented through (where $\ggg$ denotes right rotation; $\gg$ denotes logical right shift; and $\texttt{SWAPMOVE}(a, b, \text{mask}, n)$ is described as $\text{tmp} \leftarrow (b \oplus (a \gg n)) \wedge \text{mask}, b \leftarrow b \oplus \text{tmp}, a \leftarrow a \oplus (\text{tmp} \ll n)$):

- **Nibble rotation:** Applied to subsets of state words with different rotation amounts: $x \ggg 1$, $x \ggg 2$, $x \ggg 3$ within 4-bit boundaries;
- **Half-word rotation:** Targeting 16-bit boundaries within 32-bit words: $x \ggg 4$, $x \ggg 8$, $x \ggg 12$ within 16-bit segments;
- **Byte rotation:** Operating on 8-bit boundaries with conditional masking: $x \ggg 2$, $x \ggg 4$, $x \ggg 6$ within 8-bit segments;
- **Word rotation:** Full 32-bit rotations combined with bit swapping operations: $x \ggg 8$, $x \ggg 16$, $x \ggg 24$;
- **Conditional swap:** Using SWAPMOVE operations for irregular bit reorderings with masks 55555555, 00005555, 55550000.

6.3 Side Channel Protected Benchmark: Threshold Implementation

6.3.1 SBox

The coordinate functions of the BAKSHEESH SBox (306DB58ECF924A71) have the maximum algebraic degree of 2 (see Section 3.4). Therefore, 3-sharing is sufficient (which is rare, if not unique, among other commonly used 4×4 SBoxes). The following sharing is chosen for our case (generated using the code from [BGST23][24]):

$$
\begin{aligned}
x_0 &= x_{0,0} \oplus x_{0,1} \oplus x_{0,2}, & y_0 &= y_{0,0} \oplus y_{0,1} \oplus y_{0,2},\\
x_1 &= x_{1,0} \oplus x_{1,1} \oplus x_{1,2}, & y_1 &= y_{1,0} \oplus y_{1,1} \oplus y_{1,2},\\
x_2 &= x_{2,0} \oplus x_{2,1} \oplus x_{2,2}, & y_2 &= y_{2,0} \oplus y_{2,1} \oplus y_{2,2},\\
x_3 &= x_{3,0} \oplus x_{3,1} \oplus x_{3,2}; & y_3 &= y_{3,0} \oplus y_{3,1} \oplus y_{3,2};
\end{aligned}
$$

[24] https://github.com/anubhab001/sbox-threshold-public.

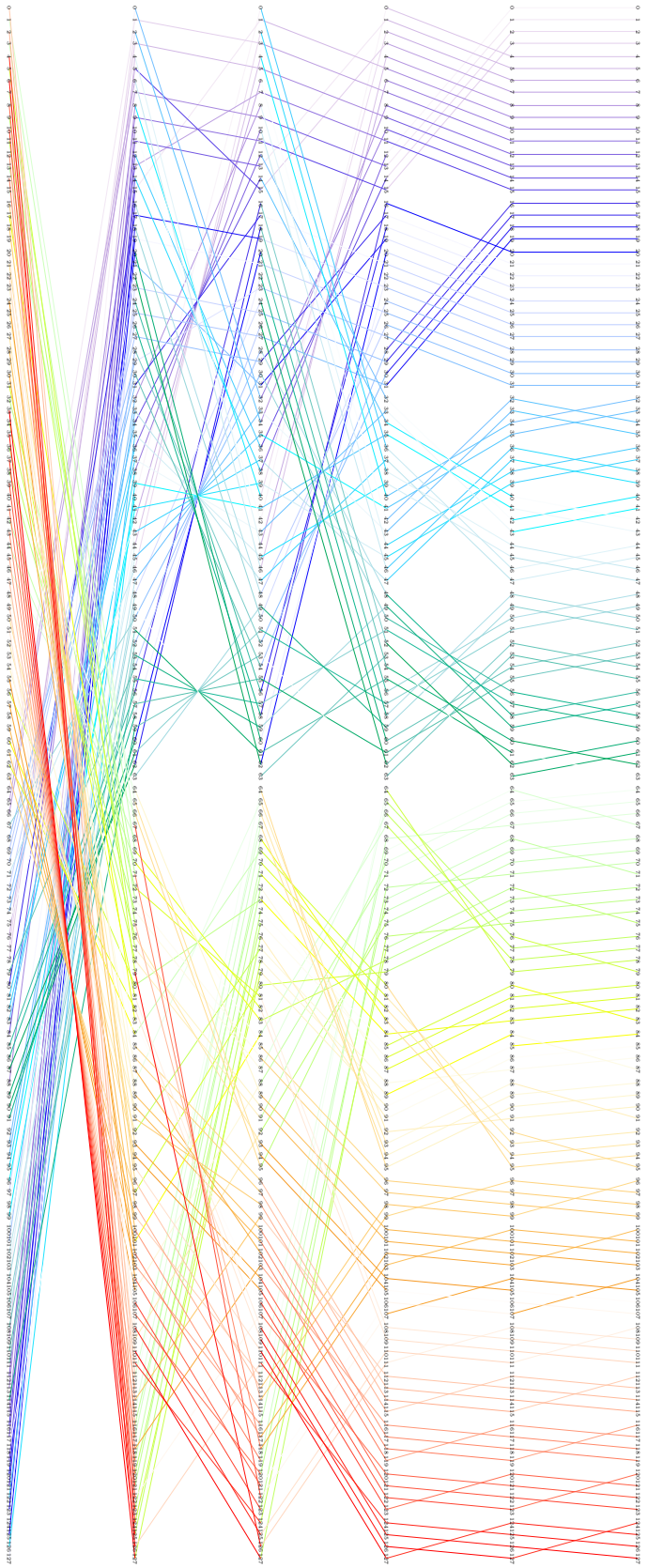


Figure 12: Quintuple-round fix-slice pattern for BAKSHEESH

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

Table 15: 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 [BPP+17b]	1A4C6F392DB7508E	10	4	265 (3.19×)*	526 (3.17×)	102 (6.38×)
PRESENT [BKL+07]	C56B90AD3EF84712	23		666 (8.02×)	1132 (6.82×)	56 (3.50×)
PRINCE [BCG+12b]	BF32AC916780E5D4	28		731 (8.81×)	991 (5.97×)	77 (4.81×)
PICCOLO [SIH+11]	E4B238091A7F6C5D	16		399 (4.81×)	645 (3.89×)	55 (3.44×)
SKINNY-64 [BJK+16]	C6901A2B385D4E7F	16		398 (4.80×)	723 (4.36×)	139 (8.69×)
TWINE [SMMK11]	C0FA2B9583D71E64	25		626 (7.54×)	723 (4.36×)	41 (2.56×)
PYJAMASK-128 [GJK+19]	2D397BA6E0F4851C	15		373 (4.49×)	640 (3.86×)	60 (3.75×)
Gao+ [GRO19]	048AFC691EBD7532	36		988 (11.90×)	1658 (9.99×)	162 (10.13×)

†: GE in STM 130nm (HCMOS9GP, Cadence v14.20 2016)
‡: Number of LUTs in Kintex-7 (Vivado 2020)
∗: Used in [JGC+20, Appendix B]

For a comparative study, a benchmark for several 4×4 SBoxes is given in Table 15, 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 [BGST23]. Each of the SBoxes is used in a cipher construction, except for `048AFC691EBD7532` which is presented in [GRO19, Section 3.4]. It can be seen from Table 15 that the cost is a 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 [BGST23, 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 documented in [BGST23]. Also note that, the figures given in Table 15 are done for the minimal order. One may expect the relative cost of the `BAKSHEESH` SBox will be more significant as the order increases.

6.3.2 Encryption

The threshold protected benchmark for `BAKSHEESH` encryption is given (under Kintex FPGA and TSMC 65nm ASIC libraries) in Table 16. The round keys are computed on-the-fly. Both implementations follow the same design principles of the 4-share implementation (Profile 6) of `GIFT-128` [JGC+20], and hence are comparable.

By looking at the benchmark from [JGC+20, Table III] and comparing it with Table 16a, we see that `GIFT-128` takes 28709 GE in TSMC 65nm ASIC library. Thus, `GIFT-128` takes about 2.43× area than `BAKSHEESH` in a comparable implementation. Further, at the

Table 16: Threshold benchmark for `BAKSHEESH` and `GIFT-128`

(a) ASIC (TSMC 65nm)

	Area (GE)	Critical Path (ns)	Max. Freq. (MHz)
`BAKSHEESH`	11811	2.09	506
`GIFT-128`	28709	1.602	624

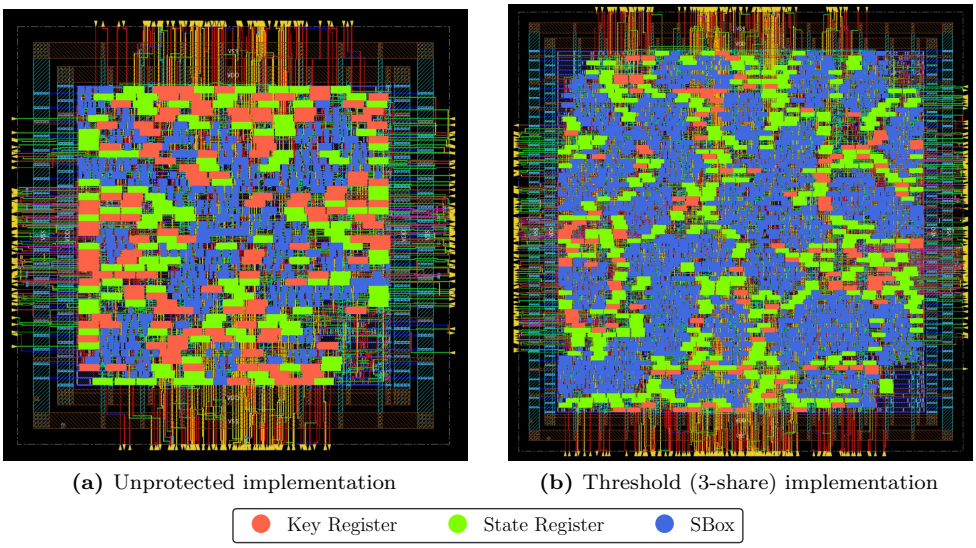
Smaller area 👍

(b) FPGA (Kintex)

	LUT	F/F	Max. Freq. (MHz)
`BAKSHEESH`	870	526	344
`GIFT-128`	3254	646	463

Smaller area (LUT) 👍

operating frequency of 10 MHz for the protected 65nm design, the power consumption is only 0.115 mW. Likewise, Table 16b shows significantly improved performance of `BAKSHEESH` compared to `GIFT-128` in threshold. The number of computation cycles for threshold implementation of `BAKSHEESH` is 17.



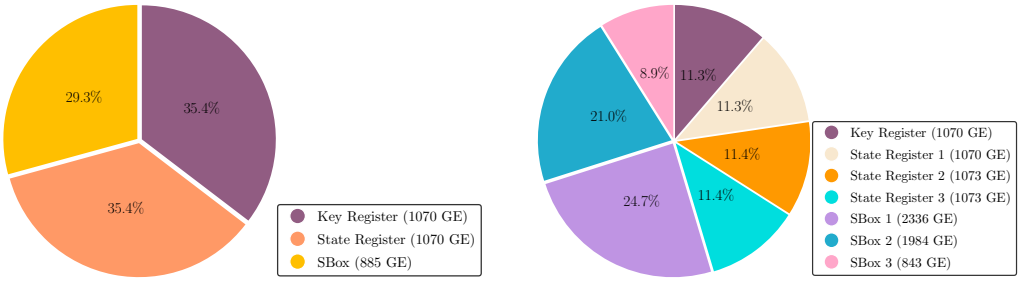
(a) Unprotected implementation **(b)** Threshold (3-share) implementation

Figure 13: `BAKSHEESH` implementation as ASIC layout (after place-and-route)

Figure 13 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. In particular, Figure 13b shows the detailed threshold implementation layout. Further, Figure 14 shows component-wise break-up for area requirement for the same, the unprotected version in Figure 14a and the protected (3-share threshold) in Figure 14b.

7 Conclusion

Pushing the envelope of modern cipher design, we propose `BAKSHEESH`, a 128-bit lightweight block cipher. `BAKSHEESH` brings certain unique features to the table; most notably, this is the first mainstream cipher to make use of an SBox with a non-trivial LS. Overall,



(a) Unprotected **(b)** Threshold (3-share)

Figure 14: ASIC area usage of BAKSHEESH components

BAKSHEESH offers the following advantages over other (comparable) ciphers:

- **Lightweight:** BAKSHEESH synthesises design ideas from both GIFT-128 [BPP+17b] (structural efficiency and bit-permutation layers) and ZORRO [GGNS13] (masking-friendly partial non-linearity), taking 12.5% fewer rounds compared to GIFT-128 whilst avoiding the security vulnerabilities that plague ZORRO.

  Thanks to its 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 (and partial non-linear layer):** Symmetric key ciphers with low AND complexity are pitched suitable for certain applications, most notably for easier adoption of side channel countermeasures. This direction builds upon concepts introduced by ZORRO [GGNS13] and similar ciphers. BAKSHEESH advances this research direction by achieving similar masking benefits through linear structures whilst maintaining full classical security.

  A quick comparison of BAKSHEESH with ROBIN [GLSV15] (which is specifically designed to efficiently support side channel countermeasures), as given in Table 17, reveals that BAKSHEESH does a commendable job.

Table 17: Comparison of AND complexity of BAKSHEESH with niche-specific ciphers

	Security	# Key	# Block	AND depth	Min. AND	AND per bit	# Rounds
BAKSHEESH	128	128	128	32	3072	24	35
ROBIN [GLSV14]	128	128	128	16	3072	24	16

For the consideration of the community, we list the two interesting research problems. First, one may construct a cipher for faster encryption plus decryption[25]. Second, a design with a more complex linear layer than bit-permutation can be considered in conjunction with an SBox having 1 non-trivial LS.

A Acronym

ACT Auto-Correlation Table ↪ A cryptanalytic table that measures the correlation between a function and its shifted versions

AE Affine Equivalence ↪ A relationship between SBoxes where one can be transformed into another through invertible affine transformations

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

ANF Algebraic Normal Form ↪ A canonical representation of Boolean functions as polynomials over $\mathbb{F}_2$

BCT Boomerang Connectivity Table ↪ A cryptanalytic table that captures the properties against the boomerang attack

CNF Conjunctive Normal Form ↪ A logical formula structure where expressions are written as conjunctions (AND) of disjunctions (OR)

CPA Correlation Power Analysis ↪ A side channel attack that makes use of correlation between power traces and intermediate cryptographic values

DBN Differential Branch Number ↪ The minimum sum of active input and output differences

DDT Difference Distribution Table ↪ A table containing the frequency distribution of output differences for each possible input difference in an SBox

DFA Differential Fault Attack ↪ A cryptanalytic technique that injects faults during cipher execution to extract secret information

DPA Differential Power Analysis ↪ A pioneering side channel attack technique using difference-based analysis of power traces

DU Differential Uniformity ↪ The maximum entry in the DDT (excluding the trivial case)

FA Fault Attack ↪ A broad class of attacks that exploit computational errors induced in cryptographic devices

FHE Fully Homomorphic Encryption ↪ A cryptographic scheme enabling computation on encrypted data without requiring decryption

GE Guessing Entropy ↪ A metric quantifying the average number of key guesses required by an attacker after a successful side channel attack

IS Initial Structure ↪ A cryptanalytic concept in Meet-in-the-Middle attacks referring to the initial configuration and constraints that define the attack's starting conditions

LAT Linear Approximation Table ↪ A table storing correlation values between linear combinations of input and output bits for an SBox

LBN Linear Branch Number ↪ The minimum sum of active input and output masks

LFSR Linear Feedback Shift Register ↪ A shift register with feedback characterised by a linear function, commonly used for generating pseudo-random sequences

LS Linear Structure ↪ An input difference that produces a constant output difference for all inputs of an SBox

MILP Mixed-Integer Linear Programming ↪ An optimisation technique used for automated cryptanalytic search (e.g., finding optimal differential/linear trails)

MitM Meet-in-the-Middle Attack ↪ A cryptanalytic technique that attacks a cipher by analysing it from both ends

ML Machine Learning ↪ Computational techniques that use statistical methods to enable systems to learn and improve from data, recently applied to classical cryptanalysis

MPC Multi-party Computation ↪ A cryptographic protocol enabling multiple parties to jointly compute a function over their inputs while keeping those inputs private

PGE Partial Guessing Entropy ↪ A metric ranking the correct key among all candidates based on correlation magnitude (rank 1 indicates successful key recovery)

PM Partial Matching ↪ A cryptanalytic technique used in Meet-in-the-Middle attacks that allows matching of partial state information to reduce computational complexity

QR Quotient-Remainder Group ↪ Independent 16-bit sub-permutations in `GIFT` or `BAKSHEESH` bit-permutation layer that enable efficient fault analysis through parallel attack vectors

SBox Substitution Box ↪ A non-linear transformation in symmetric key ciphers, often the only source of non-linearity

SCA Side Channel Attack ↪ An attack exploiting physical information leakage (power, timing, electromagnetic emissions) from cryptographic devices

SPN Substitution-Permutation Network ↪ A cryptographic construction alternating substitution (collection of SBoxes) and permutation layers

TI Threshold Implementation ↪ A masking countermeasure against side channel attacks using secret sharing to hide intermediate values

ToFA Towards Fault Analysis ↪ A new fault attack that represents the first successful application of impossible differential cryptanalysis to bit-oriented ciphers

TVLA Test Vector Leakage Assessment ↪ A standardised statistical methodology for detecting side channel leakage in cryptographic implementations using Welch's t-test

ZK Zero Knowledge ↪ A cryptographic proof system where a prover can convince a verifier that a statement is true without revealing any additional information

B Further Cryptographic Properties of `BAKSHEESH` SBoxes

B.1 `GIFT` Heritage (4-bit)

Table 18: ACT for `BAKSHEESH` (4-bit) SBox

ψ \ Ψ	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1		−16			−16		+16								
2	−16					−16	+16								
3			−16	−16			+16								
4							+16		−16					−16	
5							+16				−16	−16			
6							+16	−16							−16
7							+16			−16			−16		
8	−16	−16	+16	−16	+16	+16	−16	−16	+16	+16	−16	+16	−16	−16	+16
9		+16			−16		−16								
a	+16					−16	−16								
b			−16	+16			−16								
c							−16		−16					+16	
d							−16				+16	−16			
e							−16	+16							−16
f							−16			−16			+16		

The ACT and BCT for the 4-bit SBox used in the `GIFT` heritage of `BAKSHEESH` are respectively presented in Tables 18 and 19.

Table 19: BCT for BAKSHEESH (4-bit) SBox

ϕ \ Φ	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1	4		4	4		4			4		4	4		4	16
2		4	4	4	4					4	4	4	4		16
3	4	4			4	4			4	4			4	4	16
4	4		4		4		4			4		4		4	16
5				4	4	4	4		4	4	4				16
6	4	4		4			4				4		4	4	16
7		4	4			4	4		4			4	4		16
8	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
9	4		4	4		4			4		4	4		4	16
a		4	4	4	4					4	4	4	4		16
b	4	4			4	4			4	4			4	4	16
c	4		4		4		4			4		4		4	16
d				4	4	4	4		4	4	4				16
e	4	4		4			4				4		4	4	16
f		4	4			4	4		4			4	4		16

B.2 ZORRO Heritage (3-bit)

We show the coordinate functions of 3-bit SBox s (10365247) used in the ZORRO heritage of BAKSHEESH in the following. The DDT, LAT, DPT, ACT and BCT are respectively presented in Table 21, 22, 20, 23 and 24.

$$y_0 = 1 \oplus x_0 \oplus x_1x_2,$$
$$y_1 = x_1 \oplus x_0x_2 \oplus x_1x_2,$$
$$y_2 = x_2 \oplus x_0x_1 \oplus x_0x_2.$$

Table 20: DPT for BAKSHEESH (3-bit) SBox

ω \ Ω	0	1	2	3	4	5	6	7
0	✖	✖	✖	✖	✖	✖	✖	✖
1		✖	✖		✖			
2		✖	✖		✖			
3				✖	✖			
4		✖	✖		✖			
5			✖		✖			
6		✖	✖					
7								✖

Table 21: DDT for BAKSHEESH (3-bit) SBox

δ \ Δ	1	2	3	4	5	6	7
1	2		2		2		2
2	2	2			2	2	
3		2	2			2	2
4	2	2		2			2
5		2	2	2	2		
6				2	2	2	2
7	2		2	2		2	

Table 22: LAT for BAKSHEESH (3-bit) SBox

γ \ Γ	1	2	3	4	5	6	7
1	−2	+2			−2	−2	
2		+2	−2	+2	+2		
3	−2		−2	−2		+2	
4				+2	−2	+2	+2
5	−2	−2		+2			−2
6		+2	+2			+2	−2
7	+2		−2		−2		−2

Table 23: ACT for BAKSHEESH (3-bit) SBox

ψ \ Ψ	1	2	3	4	5	6	7
1	−8						
2			−8				
3		−8					
4							−8
5						−8	
6				−8			
7					−8		

Table 24: BCT for BAKSHEESH (3-bit) SBox

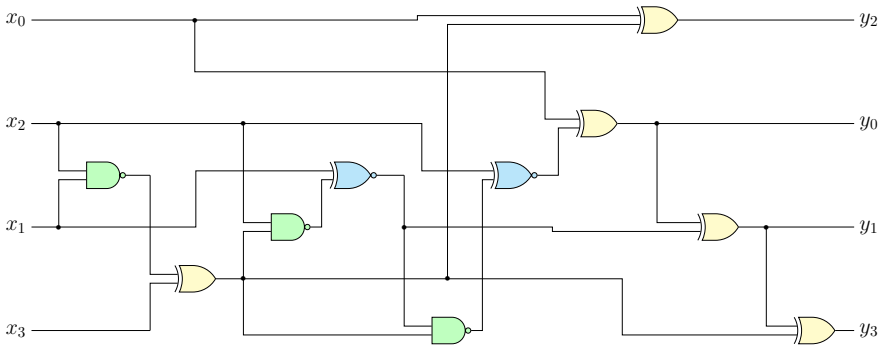
ϕ \ Φ	1	2	3	4	5	6	7
1	2		2		2		2
2	2	2			2	2	
3		2	2			2	2
4	2	2		2			2
5		2	2	2	2		
6				2	2	2	2
7	2		2	2		2	

C Miscellaneous Benchmarks

C.1 GIFT Heritage (4-bit) SBox

```
// Initial reordering
F[0]=X[0]; F[1]=X[2]; F[2]=X[3]; F[3]=X[1];
// Actual circuit (10 lines)
t0 = NOR2 (F[1], F[2]);
F[3] = XOR2(F[3], t0);
t1 = NOR2 (F[2], F[3]);
F[1] = XOR2(F[1], t1);
t2 = NOR2 (F[1], F[3]);
F[2] = XOR2(F[2], t2);
F[1] = XOR2 (F[1], F[0]);
F[3] = XOR2 (F[3], F[1]);
F[0] = XOR2 (F[0], F[2]);
F[2] = XOR2 (F[2], F[3]);
// Final reordering
X[0]=F[1]; X[1]=F[0]; X[2]=F[2]; X[3]=F[3];
```

Code 1: BAKSHEESH (4-bit) SBox in software (ARM Cortex-M3) using LIGHTER

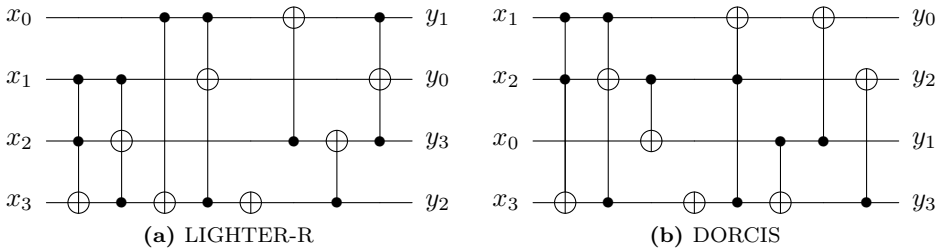


Code 2: BAKSHEESH (4-bit) SBox in ASIC (TSMC 65nm) using LIGHTER

The CPU-optimised implementation (for ARM Cortex-M3[26]) is given in Code 1, which is obtained by using the LIGHTER tool [JPST17] and requires 10 instructions. The ASIC

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

(TSMC 65nm) implementation is given in Code 2, which incurs cost of 20.5 GE. Similarly, reversible implementations are presented in Code 3a using LIGHTER-R [DBSC19] with 8 gates (4 Toffoli, 3 CNOT, 1 NOT), and in Code 3b using DORCIS [BCC+24, CBC23] (this tool specifically optimises for the quantum technology) with 8 gates (3 Toffoli, 4 CNOT, 1 NOT) that achieves T-depth 3 and full depth 24.



(a) LIGHTER-R **(b)** DORCIS

Code 3: `BAKSHEESH` (4-bit) SBox in reversible logic

C.2 `ZORRO` Heritage Linear Map (T)

Using the tool from [BCC+24], we find that the T matrix from the `BAKSHEESH ZORRO` heritage can be implemented with 128 CNOT (i.e., in-place XOR) gates, along with 96 in-place SWAP operations prior to the CNOT sequence. The circuit incurs classical depth of 2. This tool counts SWAP operations as conflicts for qunatum depth, and with that constraint the quantum depth is 4. This is given in Code 4.

D DFA on Encryption and Decryption: Featuring `DEFAULT`

D.1 SBox

Before delving into more details on the DFA security, one may be interested in the situation where the attacker mounts DFA on both encryption and decryption. Each round of `DEFAULT-LAYER` applies the SBox layer, then the bit permutation, and then the round-key addition, so a fault on an SBox leaks information about the round key that follows it. We track a single nibble of that key, written $k = (k_3\, k_2\, k_1\, k_0)$. The four linear structures of the `DEFAULT` LS SBox (`037ED4A9CF18B265`) are `0`, `6`, `9`, `f`, and those of its inverse (`0AD15FE2B76C8439`) are `0`, `5`, `a`, `f`.

A fault in the encryption direction, repeated over enough fault values, leaks the two bits $k_0 \oplus k_2$ and $k_1 \oplus k_3$; equivalently, it confines k to a coset of the output linear structures $\{0, 5, a, f\}$. A fault in the decryption direction leaks the complementary bits $k_0 \oplus k_3$ and $k_1 \oplus k_2$, and confines k to a coset of the input linear structures $\{0, 6, 9, f\}$ [NDE21, JKP25]. The two spaces meet only in $\{0, f\}$. Combining the two directions therefore determines k up to 2 candidates, namely k and $k \oplus f$. Over the 32 nibbles this leaves 2^{32} candidates for the round key. Consequently, `DEFAULT` [Bak22, Chapter 8] does not inherently protect against the model that combines information from encryption and decryption.

This combined model, however, is considered out-of-scope for `DEFAULT` per [Bak22, Chapter 8.2.3]. It is moreover unrealistic. Although small-scale devices such as sensors are subject to fault attacks in the field, the base stations are typically protected well enough to leave little scope for a fault attack. It is hard to find a realistic scenario where both the recipient and the sender are vulnerable to DFA.

```
001:  x000 ↔ x032   002:  x001 ↔ x033   003:  x002 ↔ x034   004:  x003 ↔ x035
005:  x004 ↔ x036   006:  x005 ↔ x037   007:  x006 ↔ x038   008:  x007 ↔ x039
009:  x008 ↔ x040   010:  x009 ↔ x041   011:  x010 ↔ x042   012:  x011 ↔ x043
013:  x012 ↔ x044   014:  x013 ↔ x045   015:  x014 ↔ x046   016:  x015 ↔ x047
017:  x016 ↔ x048   018:  x017 ↔ x049   019:  x018 ↔ x050   020:  x019 ↔ x051
021:  x020 ↔ x052   022:  x021 ↔ x053   023:  x022 ↔ x054   024:  x023 ↔ x055
025:  x024 ↔ x056   026:  x025 ↔ x057   027:  x026 ↔ x058   028:  x027 ↔ x059
029:  x028 ↔ x060   030:  x029 ↔ x061   031:  x030 ↔ x062   032:  x031 ↔ x063
033:  x032 ↔ x064   034:  x033 ↔ x065   035:  x034 ↔ x066   036:  x035 ↔ x067
037:  x036 ↔ x068   038:  x037 ↔ x069   039:  x038 ↔ x070   040:  x039 ↔ x071
041:  x040 ↔ x072   042:  x041 ↔ x073   043:  x042 ↔ x074   044:  x043 ↔ x075
045:  x044 ↔ x076   046:  x045 ↔ x077   047:  x046 ↔ x078   048:  x047 ↔ x079
049:  x048 ↔ x080   050:  x049 ↔ x081   051:  x050 ↔ x082   052:  x051 ↔ x083
053:  x052 ↔ x084   054:  x053 ↔ x085   055:  x054 ↔ x086   056:  x055 ↔ x087
057:  x056 ↔ x088   058:  x057 ↔ x089   059:  x058 ↔ x090   060:  x059 ↔ x091
061:  x060 ↔ x092   062:  x061 ↔ x093   063:  x062 ↔ x094   064:  x063 ↔ x095
065:  x064 ↔ x096   066:  x065 ↔ x097   067:  x066 ↔ x098   068:  x067 ↔ x099
069:  x068 ↔ x100   070:  x069 ↔ x101   071:  x070 ↔ x102   072:  x071 ↔ x103
073:  x072 ↔ x104   074:  x073 ↔ x105   075:  x074 ↔ x106   076:  x075 ↔ x107
077:  x076 ↔ x108   078:  x077 ↔ x109   079:  x078 ↔ x110   080:  x079 ↔ x111
081:  x080 ↔ x112   082:  x081 ↔ x113   083:  x082 ↔ x114   084:  x083 ↔ x115
085:  x084 ↔ x116   086:  x085 ↔ x117   087:  x086 ↔ x118   088:  x087 ↔ x119
089:  x088 ↔ x120   090:  x089 ↔ x121   091:  x090 ↔ x122   092:  x091 ↔ x123
093:  x092 ↔ x124   094:  x093 ↔ x125   095:  x094 ↔ x126   096:  x095 ↔ x127
097:  x062 += x126  098:  x057 += x121  099:  x058 += x122  100:  x054 += x118
101:  x056 += x120  102:  x034 += x098  103:  x044 += x108  104:  x050 += x114
105:  x037 += x101  106:  x043 += x107  107:  x047 += x111  108:  x035 += x099
109:  x053 += x117  110:  x063 += x127  111:  x039 += x103  112:  x055 += x119
113:  x051 += x115  114:  x049 += x113  115:  x040 += x104  116:  x046 += x110
117:  x045 += x109  118:  x033 += x097  119:  x036 += x100  120:  x048 += x112
121:  x059 += x123  122:  x042 += x106  123:  x041 += x105  124:  x060 += x124
125:  x061 += x125  126:  x038 += x102  127:  x032 += x096  128:  x052 += x116
129:  x024 += x056  130:  x108 += x076  131:  x031 += x063  132:  x001 += x033
133:  x098 += x066  134:  x116 += x084  135:  x019 += x051  136:  x084 += x052
137:  x103 += x071  138:  x022 += x054  139:  x113 += x081  140:  x117 += x085
141:  x025 += x057  142:  x120 += x088  143:  x028 += x060  144:  x125 += x093
145:  x071 += x039  146:  x008 += x040  147:  x009 += x041  148:  x115 += x083
149:  x101 += x069  150:  x014 += x046  151:  x083 += x051  152:  x030 += x062
153:  x100 += x068  154:  x016 += x048  155:  x127 += x095  156:  x010 += x042
157:  x122 += x090  158:  x068 += x036  159:  x005 += x037  160:  x095 += x063
161:  x111 += x079  162:  x006 += x038  163:  x096 += x064  164:  x102 += x070
165:  x124 += x092  166:  x121 += x089  167:  x070 += x038  168:  x079 += x047
169:  x099 += x067  170:  x085 += x053  171:  x112 += x080  172:  x020 += x052
173:  x126 += x094  174:  x029 += x061  175:  x119 += x087  176:  x087 += x055
177:  x023 += x055  178:  x107 += x075  179:  x075 += x043  180:  x064 += x032
181:  x018 += x050  182:  x114 += x082  183:  x104 += x072  184:  x089 += x057
185:  x092 += x060  186:  x017 += x049  187:  x015 += x047  188:  x027 += x059
189:  x067 += x035  190:  x105 += x073  191:  x021 += x053  192:  x123 += x091
193:  x109 += x077  194:  x097 += x065  195:  x011 += x043  196:  x072 += x040
197:  x093 += x061  198:  x088 += x056  199:  x066 += x034  200:  x069 += x037
201:  x077 += x045  202:  x003 += x035  203:  x118 += x086  204:  x073 += x041
205:  x086 += x054  206:  x094 += x062  207:  x081 += x049  208:  x080 += x048
209:  x012 += x044  210:  x000 += x032  211:  x090 += x058  212:  x026 += x058
213:  x076 += x044  214:  x002 += x034  215:  x082 += x050  216:  x013 += x045
217:  x110 += x078  218:  x007 += x039  219:  x004 += x036  220:  x091 += x059
221:  x078 += x046  222:  x106 += x074  223:  x065 += x033  224:  x074 += x042
```

Code 4: In-place implementation of T (from `BAKSHEESH ZORRO` heritage)

D.2 Round-Key Recovery

The single-SBox observation extends to the recovery of a round key of DEFAULT-LAYER. Each round transforms the state by the SBox layer S, then the bit permutation P, and then the addition of the round key $K^{(r)}$. Writing s_{r-1} for the input to round r, the round computes $a_r = S(s_{r-1})$, $b_r = P(a_r)$, and $s_r = b_r \oplus K^{(r)}$, with the plaintext as s_0 and the ciphertext as the final state. We consider independent round keys and omit the round constants without loss of generality. Each round constant is public and fixed across every key hypothesis, so it cancels in every fault-induced difference and does not affect the recovered candidate sets or the resulting intersection. We use subscript $j \in \{0, \ldots, 31\}$ for nibble indices, with the rightmost nibble at $j = 0$.

Worked Example on One Nibble

We illustrate the recovery of the last round key on a single nibble. Let the corresponding SBox input be $x = \mathtt{7}$ and the key nibble be $k = \mathtt{2}$, both unknown to the attacker. Once the final permutation is inverted, the two are tied to the observed ciphertext nibble through $c = S(x) \oplus k = \mathtt{9} \oplus \mathtt{2} = \mathtt{b}$ (known).

The attacker injects a known fault δ ($\neq 0$) into the nibble, targeting the SBox input in the encryption direction and the SBox output in the decryption direction. Consider first an encryption fault $\delta = \mathtt{2}$. It changes x from $\mathtt{7}$ to $\mathtt{7} \oplus \mathtt{2} = \mathtt{5}$, so the SBox output changes by $S(\mathtt{7}) \oplus S(\mathtt{5}) = \mathtt{9} \oplus \mathtt{4} = \mathtt{d}$, which the attacker reads off after inverting the permutation. The differential distribution table gives the solutions of $S(x) \oplus S(x \oplus \mathtt{2}) = \mathtt{d}$ as the eight values $\{\mathtt{1}, \mathtt{3}, \mathtt{5}, \mathtt{7}, \mathtt{8}, \mathtt{a}, \mathtt{c}, \mathtt{e}\}$, since every non-trivial entry of this table equals 8. Repeating with further fault values narrows these to the input linear-structure coset $\{\mathtt{1}, \mathtt{7}, \mathtt{8}, \mathtt{e}\}$ of x, the tightest set the linear structures permit. The key nibble then follows from $k = c \oplus S(x)$, and lies in $c \oplus S(\{\mathtt{1}, \mathtt{7}, \mathtt{8}, \mathtt{e}\}) = \{\mathtt{2}, \mathtt{7}, \mathtt{8}, \mathtt{d}\}$, a coset of the output linear structures.

A decryption fault, analysed the same way through the inverse SBox, confines k to a coset of the input linear structures, here $\{\mathtt{2}, \mathtt{4}, \mathtt{b}, \mathtt{d}\}$ [NDE21, JKP25]. The two cosets intersect as $\{\mathtt{2}, \mathtt{7}, \mathtt{8}, \mathtt{d}\} \cap \{\mathtt{2}, \mathtt{4}, \mathtt{b}, \mathtt{d}\} = \{\mathtt{2}, \mathtt{d}\}$, so the attacker deduces $k \in \{\mathtt{2}, \mathtt{d}\}$, the true key nibble together with $k \oplus \mathtt{f}$.

Every pairing of an encryption coset with a decryption coset meets in 0 or 2 elements (Table 25), so the combined attack always leaves exactly 2 candidates for the key nibble.

Table 25: Intersection of encryption cosets and decryption cosets

	E_0={0, 5, a, f}	E_1={1, 4, b, e}	E_2={2, 7, 8, d}	E_3={3, 6, 9, c}
D_0={0, 6, 9, f}	{0, f}	∅	∅	{6, 9}
D_1={1, 7, 8, e}	∅	{1, e}	{7, 8}	∅
D_2={2, 4, b, d}	∅	{4, b}	{2, d}	∅
D_3={3, 5, a, c}	{5, a}	∅	∅	{3, c}

E_k: encryption coset, from the output linear structures

D_i: decryption coset, from the input linear structures

Repeating this across all 32 nibbles leaves 2^{32} candidates for the round key.

Full-Key Recovery

The last round key is thus known up to 2^{32} candidates. Peeling it off exposes the preceding SBox layer, and the same combined attack recovers the previous round key; the round keys are stripped one layer at a time. A second known plaintext-ciphertext pair removes the residual per-nibble ambiguity. The full multi-round procedures, with the fault counts for the simple, rotating, and independent round-key schedules of DEFAULT, are given by Nageler et al. [NDE21] and Jana et al. [JKP25], who reach unique key recovery.

References

[AABS+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symmetric Cryptol.*, 2020(3):1–45, 2020. doi:10.13154/tosc.v2020.i3.1-45. 12

[AGR+16] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219. Springer, 2016. doi:10.1007/978-3-662-53887-6_7. 12

[ANP21] Alexandre Adomnicai, Zakaria Najm, and Thomas Peyrin. Fixslicing: A new GIFT representation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):402–425, 2021. Preprint: https://eprint.iacr.org/2020/412. doi:10.46586/tches.v2021.i1.402-425. 2, 12, 18, 19, 34

[ARS+15] Martin Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015. doi:10.1007/978-3-662-46800-5_17. 5, 7, 12, 13

[AS09] Kazumaro Aoki and Yu Sasaki. Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In Shai Halevi, editor, *CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009. URL: https://doi.org/10.1007/978-3-642-03356-8_5. 29

[BAK98] Eli Biham, Ross Anderson, and Lars Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *Fast Software Encryption*, pages 222–238, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. doi:10.1007/3-540-69710-1_15. 10

[Bak20a] Anubhab Baksi. New insights on differential and linear bounds using mixed integer linear programming. In *Innovative Security Solutions for Information Technology and Communications – SecITC 2020*, volume 12596 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 2020. doi:10.1007/978-3-030-69255-1_4. 17, 22

[Bak20b] Anubhab Baksi. 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. 22

[Bak21] Anubhab Baksi. *Classical and Physical Security of Symmetric Key Cryptographic Algorithms*. PhD thesis, School of Computer Science & Engineering, Nanyang Technological University, Singapore, 2021. https://dr.ntu.edu.sg/handle/10356/152003. 2, 30

[Bak22] Anubhab Baksi. *Classical and Physical Security of Symmetric Key Cryptographic Algorithms*. Springer, Singapore, 2022. https://doi.org/10.1007/978-981-16-6522-6. 2, 3, 5, 11, 17, 22, 23, 44

[Bak24] Anubhab Baksi. A study of partial non-linear layers with DEFAULT and BAKSHEESH. Cryptology ePrint Archive, Report 2024/1072, 2024. https://eprint.iacr.org/2024/1072. 11

[BBB+22] Anubhab Baksi, Arghya Bhattacharjee, Jakub Breier, Takanori Isobe, and Mridul Nandi. Big brother is watching you: A closer look at backdoor construction. Cryptology ePrint Archive, Paper 2022/953, 2022. https://eprint.iacr.org/2022/953. URL: https://eprint.iacr.org/2022/953. 17

[BBB+23] Anubhab Baksi, Shivam Bhasin, Jakub Breier, Dirmanto Jap, and Dhiman Saha. A survey on fault attacks on symmetric key cryptosystems. *ACM Computing Surveys*, 55(4):86:1–86:34, 2023. doi:10.1145/3530054. 2, 3, 8, 29

[BBCD20] Anubhab Baksi, Jakub Breier, Yi Chen, and Xiaoyang Dong. Machine learning assisted differential distinguishers for lightweight ciphers. Cryptology ePrint Archive, Report 2020/571, 2020. https://eprint.iacr.org/2020/571. 28

[BBD+23] Anubhab Baksi, Jakub Breier, Vishnu Asutosh Dasu, Xiaolu Hou, Hyunji Kim, and Hwajeong Seo. New results on machine learning based distinguishers. Cryptology ePrint Archive, Paper 2023/235, 2023. https://eprint.iacr.org/2023/235. doi:10.1109/ACCESS.2023.3270396. 28

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *ASIACRYPT 2015*, pages 411–436, 2015. Extended version: https://eprint.iacr.org/2015/1142. doi:10.1007/978-3-662-48800-3_17. 16, 18

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. 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*, pages 12–23, 1999. URL: https://doi.org/10.1007/3-540-48910-X_2. 27

[BCC+24] Anubhab Baksi, Sumanta Chakraborty, Anupam Chattopadhyay, Matthew Chun, SK Hafizul Islam, Kyungbae Jang, Hyunji Kim, Yujin Oh, Soham Roy, Hwajeong Seo, and Siyi Wang. Quantum implementation of linear and non-linear layers. In Diana Göhringer, Uwe Gäbler, Tanja Harbaum, and Klaus Hofmann, editors, *37th IEEE International System-on-Chip Conference, SOCC 2024, Dresden, Germany, September 16-19, 2024*, pages 1–6. IEEE, 2024. doi:10.1109/SOCC62300.2024.10737862. 44

[BCG+12a] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications. In *ASIACRYPT 2012*, pages 208–225, 2012. doi:10.1007/978-3-642-34961-4_14. 15

[BCG+12b] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. 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. 15, 17, 18, 37

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. 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*, pages 16–29, 2004. doi:10.1007/978-3-540-28632-5_2. 9

[BDJ+10] Stéphane Badel, Nilay Dagtekin, Jorge Nakahara Jr., Khaled Ouafi, Nicolas Reffé, Pouyan Sepehrdad, Petr Susil, and Serge Vaudenay. ARMADILLO: A multi-purpose cryptographic primitive dedicated to hardware. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2010. doi:10.1007/978-3-642-15031-9_27. 7

[BDMD+20] Begül Bilgin, Lauren De Meyer, Sébastien Duval, Itamar Levi, and François-Xavier Standaert. 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. doi:10.13154/tosc.v2020.i1.144-184. 12

[BEK+21] Dušan Božilov, Maria Eichlseder, Miroslav Knežević, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. PRINCEv2 - more security for (almost) no overhead. In *Selected Areas in Cryptography - SAC 2020*, pages 483–511, 2021. Preprint: https://eprint.iacr.org/2020/1269. doi:10.1007/978-3-030-81652-0_19. 18

[BFP12] Luk Bettale, Jean-Charles Faugère, and Ludovic Perret. Solving polynomial systems over finite fields: improved analysis of the hybrid approach. In Joris van der Hoeven and Mark van Hoeij, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC'12, Grenoble, France - July 22 - 25, 2012*, pages 67–74. ACM, 2012. doi:10.1145/2442829.2442843. 25

[BGST23] Anubhab Baksi, Sylvain Guilley, Ritu-Ranjan Shrivastwa, and Sofiane Takarabt. From substitution box to threshold. Cryptology ePrint Archive, Paper 2023/633, 2023. https://eprint.iacr.org/2023/633. 16, 35, 37

[Bil15] Begül Bilgin. *Threshold Implementations As Countermeasure Against Higher-Order Differential Power Analysis*. PhD thesis, Katholieke Universiteit Leuven and University of Twente, 2015. https://www.esat.kuleuven.be/cosic/publications/thesis-256.pdf. 10

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *CRYPTO 2016*, pages 123–153, 2016. Extended version: https://eprint.iacr.org/2016/660. doi:10.1007/978-3-662-53008-5_5. 15, 16, 29, 37

[BKL+07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *CHES*, volume 4727, pages 450–466. Springer, 2007. doi:10.1007/978-3-540-74735-2_31. 4, 10, 12, 16, 17, 37

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-25385-0_19. 29

[BODD+14] Achiya Bar-On, Itai Dinur, Orr Dunkelman, Virginie Lallemand, and Boaz Tsaban. Improved analysis of zorro-like ciphers. In *Selected Areas in Cryptography - SAC 2014*, pages 194–210, 2014. Preprint: https://eprint.iacr.org/2014/228. doi:10.1007/978-3-319-13051-4_12. 2

[BPP+17a] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present. In *Cryptographic Hardware and Embedded Systems - CHES 2017*, pages 321–345, 2017. doi:10.1007/978-3-319-66787-4_16. 2, 4, 10, 16, 17, 26, 28

[BPP+17b] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present (extended version). Cryptology ePrint Archive, Report 2017/622, 2017. https://eprint.iacr.org/2017/622. 10, 11, 12, 15, 37, 39

[BS97] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In Jr. Kaliski, BurtonS., editor, *CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer Berlin Heidelberg, 1997. URL: http://dx.doi.org/10.1007/BFb0052259, doi:10.1007/BFb0052259. 8

[BSS20] Anubhab Baksi, Dhiman Saha, and Sumanta Sarkar. To infect or not to infect: A critical analysis of infective countermeasures in fault attacks. *Journal of Cryptographic Engineering*, 2020. URL: https://link.springer.com/article/10.1007/s13389-020-00224-9, doi:10.1007/s13389-020-00224-9. 29

[CBC23] Matthew Chun, Anubhab Baksi, and Anupam Chattopadhyay. DORCIS: Depth optimized quantum implementation of substitution boxes. Cryptology ePrint Archive, Paper 2023/286, 2023. URL: https://eprint.iacr.org/2023/286. 44

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In *FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 313–333. Springer, 2016. doi:10.1007/978-3-662-52993-5_16. 5, 12

[CG13] Claude Carlet and Sylvain Guilley. Side-channel indistinguishability. *HAL Open Science*, 2013. URL: https://hal.archives-ouvertes.fr/hal-00826618v6/document. 6

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer, 2000. doi:10.1007/3-540-44598-6_14. 10

[DBSC19] Vishnu Asutosh Dasu, Anubhab Baksi, Sumanta Sarkar, and Anupam Chattopadhyay. LIGHTER-R: optimized reversible circuit implementation for sboxes. In *32nd IEEE International System-on-Chip Conference, SOCC 2019, Singapore, September 3-6, 2019*, pages 260–265, 2019. doi:10.1109/SOCC46988.2019.1570548320. 44

[De 07] Christophe De Cannière. *Analysis and Design of Symmetric Encryption Algorithms*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2007. https://www.esat.kuleuven.be/cosic/publications/thesis-139.pdf. 15

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low ANDdepth and few ANDs per bit. In *CRYPTO 2018*, volume 10991 of *Lecture Notes in Computer Science*, pages 662–692. Springer, 2018. doi:10.1007/978-3-319-96884-1_22. 5, 12

[DGM+19] Nilanjan Datta, Ashrujit Ghoshal, Debdeep Mukhopadhyay, Sikhar Patranabis, Stjepan Picek, and Rajat Sadhukhan. TRIFLE. Submission to NIST Lightweight Cryptography Competition, March 2019. Available at https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-1/spec-doc/trifle-spec.pdf. URL: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-1/spec-doc/trifle-spec.pdf. 5, 17

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher square. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997. URL: https://doi.org/10.1007/BFb0052343. 25

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag Berlin Heidelberg, 2002. URL: https://www.springer.com/gp/book/9783540425809. 11

[GGNS13] Benoît Gérard, Vincent Grosso, María Naya-Plasencia, and François-Xavier Standaert. Block ciphers that are easier to mask: How far can we go? In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, pages 383–399, 2013. doi:10.1007/978-3-642-40349-1_22. 2, 7, 12, 39

[GJK+19] Dahmun Goudarzi, Jérémy Jean, Stefan Kölbl, Thomas Peyrin, Matthieu Rivain, Yu Sasaki, and Siang Meng Sim. Pyjamask v1.0, 2019. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/pyjamask-spec-round2.pdf. 2, 16, 37

[GKRS25] Shibam Ghosh, Anup Kumar Kundu, Mostafizar Rahman, and Dhiman Saha. Bizness: Bit invariant zero-sum property based on division trail. In Sourav Mukhopadhyay and Pantelimon Stănică, editors, *Progress in Cryptology – INDOCRYPT 2024*, pages 134–155, Cham, 2025. Springer Nature Switzerland. doi:10.1007/978-3-031-80311-6_7. 3

[GKS25] Shibam Ghosh, Anup Kumar Kundu, and Dhiman Saha. DIFA-rent: Division property based fault attacks on DEFAULT and BAKSHEESH. Cryptology ePrint Archive, Paper 2025/2053, 2025. URL: https://eprint.iacr.org/2025/2053. 3

[GLS+14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, Kerem Varici, François Durvaux, Lubos Gaspar, and Stéphanie Kerckhof. SCREAM & iSCREAM: Side-channel resistant authenticated encryption with masking. CAESAR Competition Submission, March 2014. URL: https://competitions.cr.yp.to/round1/screamv1.pdf. 8

[GLSV14] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varıcı. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In *Fast Software Encryption - FSE 2014*, Londres, United Kingdom, March 2014. URL: https://hal.inria.fr/hal-01093491. 5, 8, 12, 39

[GLSV15] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varıcı. Ls-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption*, pages 18–37, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-46706-0_2. 39

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001. URL: https://doi.org/10.1007/3-540-44709-1_21. 9

[GNPW14] Jian Guo, Ivica Nikolić, Thomas Peyrin, and Lei Wang. Cryptanalysis of zorro. In *Fast Software Encryption - FSE 2014*, pages 383–405, 2014. Preprint: https://eprint.iacr.org/2013/713. doi:10.1007/978-3-662-46706-0_21. 2

[Goh19] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 150–179, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-26951-7_6. 28

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 326–341, 2011. URL: http://dx.doi.org/10.1007/978-3-642-23951-9_22, doi:10.1007/978-3-642-23951-9_22. 17

[GRO19] Si Gao, Arnab Roy, and Elisabeth Oswald. Constructing ti-friendly substitution boxes using shift-invariant permutations. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, pages 433–452, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-12612-4_22. 37

[HKL+22] Jincheol Ha, Seongkwang Kim, Byeonghak Lee, Jooyoung Lee, and Mincheol Son. Rubato: Noisy ciphers for approximate homomorphic encryption (full version). Cryptology ePrint Archive, Paper 2022/537, 2022. https://eprint.iacr.org/2022/537. 12

[JGC+20] Arpan Jati, Naina Gupta, Anupam Chattopadhyay, Somitra Kumar Sanadhya, and Donghoon Chang. Threshold implementations of gift: A trade-off analysis. *IEEE Trans. Inf. Forensics Secur.*, 15:2110–2120, 2020. URL: https://doi.org/10.1109/TIFS.2019.2957974. 4, 37

[JKP25] Amit Jana, Anup Kumar Kundu, and Goutam Paul. More vulnerabilities of linear structure sbox-based ciphers reveal their inability to resist dfa. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024*, pages 168–203, Singapore, 2025. Springer Nature Singapore. doi:10.1007/978-981-96-0944-4_6. 3, 22, 29, 44, 46

[JPST17] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Trans. Symmetric Cryptol.*, 2017(4):130–168, 2017. doi:10.13154/tosc.v2017.i4.130-168. 43

[JT12] Marc Joye and Michael Tunstall, editors. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012. doi:10.1007/978-3-642-29656-7. 2

[KGAS25] Anup Kumar Kundu, Shibam Ghosh, Aikata Aikata, and Dhiman Saha. Tofa: Towards fault analysis of gift and gift-like ciphers leveraging truncated impossible differentials. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(3):614–643, Jun. 2025. URL: https://tches.iacr.org/index.php/TCHES/article/view/12237, doi:10.46586/tches.v2025.i3.614-643. 3, 8, 22, 29

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999. URL: https://doi.org/10.1007/3-540-48405-1_25. 9

[KJK+20] Hangi Kim, Yongjin Jeon, Giyoon Kim, Jongsung Kim, Bo-Yeon Sim, Dong-Guk Han, Hwajeong Seo, Seonggyeom Kim, Seokhie Hong, Jaechul Sung, and Deukjo Hong. 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. 12, 16

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. Printcipher: A block cipher for ic-printing. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010. URL: https://www.iacr.org/archive/ches2010/62250016/62250016.pdf, doi:10.1007/978-3-642-15031-9_2. 13

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. URL: https://doi.org/10.1007/3-540-68697-5_9. 9

[KW02] Lars R. Knudsen and David A. Wagner. Integral cryptanalysis. In *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, pages 112–127, 2002. URL: https://doi.org/10.1007/3-540-45661-9_9. 25

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A cryptanalysis of printcipher: The invariant subspace attack. In Phillip Rogaway, editor, *CRYPTO 2011 - 31st Annual Cryptology Conference,*

Santa Barbara, CA, USA, August 14-18, 2011. Proceedings, volume 6841 of *Lecture Notes in Computer Science*, pages 206–221. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-22792-9_12. 27

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. A generic approach to invariant subspace attacks: Cryptanalysis of robin, iscream and zorro. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 254–283, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. doi:10.1007/978-3-662-46800-5_11. 17, 27

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *Advances in Cryptology – EUROCRYPT 2016*, volume 9665 of *Lecture Notes in Computer Science*, pages 311–343. Springer, 2016. doi:10.1007/978-3-662-49890-3_13. 12

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - Revealing the secrets of smart cards*. Springer, 2007. 3

[MT14] Rusydi H. Makarim and Cihangir Tezcan. Relating undisturbed bits to other properties of substitution boxes. Cryptology ePrint Archive, Paper 2014/855, 2014. https://eprint.iacr.org/2014/855. 2

[MTT+05] Daniel Mesquita, Jean-Denis Techer, Lionel Torres, Gilles Sassatelli, Gaston Cambon, Michel Robert, and Fernando Moraes. A new hardware countermeasure for masking power signatures of crypto cores. In Gilles Sassatelli, Manfred Glesner, Lionel Torres, Leandro Soares Indrusiak, and Thomas Hollstein, editors, *Proceedings of the 1st International Workshop on Reconfigurable Communication-centric Systems-on-Chip, ReCoSoC 2005, Montpellier, France, June 2005*, pages 169–176. Univ. Montpellier II, 2005. URL: https://dblp.org/rec/conf/recosoc/MesquitaTTSCRM05. 10

[NDE21] Marcel Nageler, Christoph Dobraunig, and Maria Eichlseder. Information-combining differential fault attacks on default. Cryptology ePrint Archive, Report 2021/1374, 2021. https://eprint.iacr.org/2021/1374. 22, 29, 44, 46

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *International conference on information and communications security*, pages 529–545. Springer, 2006. doi:10.1007/11935308_38. 10

[Pee13] Eric Peeters. *Advanced DPA Theory and Practice: Towards the Security Limits of Secure Embedded Circuits*. Springer-Verlag New York, 1 edition, 2013. URL: https://www.springer.com/gp/book/9781461467823, doi:10.1007/978-1-4614-6783-0. 3

[PLS24] Yuxuan Peng, Jinpeng Liu, and Ling Sun. Investigation of the optimal linear characteristics of BAKSHEESH (full version). Cryptology ePrint Archive, Paper 2024/1801, 2024. URL: https://eprint.iacr.org/2024/1801. 3, 4, 12, 17, 22

[PLS25] Yuxuan Peng, Jinpeng Liu, and Ling Sun. Investigation of the optimal linear characteristics of BAKSHEESH. In *Inscrypt 2024*, volume 15529 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2025. Full version: https://eprint.iacr.org/2024/1801. doi:10.1007/978-981-96-5566-3_2. 3

[PRC12] Gilles Piret, Thomas Roche, and Claude Carlet. PICARO - a block cipher allowing efficient higher-order side-channel resistance – extended version –. Cryptology ePrint Archive, Paper 2012/358, 2012. URL: https://eprint.iacr.org/2012/358. 8, 12

[RASA14] Shahram Rasoolzadeh, Zahra Ahmadian, Mahmoud Salmasizadeh, and Mohammad Reza Aref. Total break of zorro using linear and differential attacks. Cryptology ePrint Archive, Paper 2014/220, 2014. https://eprint.iacr.org/2014/220. URL: https://eprint.iacr.org/2014/220. 2

[SA09] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Antoine Joux, editor, *EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009. URL: https://doi.org/10.1007/978-3-642-01001-9_8. 29

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 342–357, 2011. URL: https://doi.org/10.1007/978-3-642-23951-9_23, doi:10.1007/978-3-642-23951-9_23. 37

[SKK+25] Hanbeom Shin, Insung Kim, Sunyeop Kim, Byoungjin Seok, Dongjae Lee, Deukjo Hong, Jaechul Sung, Seokhie Hong, and Sangjin Lee. MIFA: An MILP-based framework for improving differential fault attacks. Cryptology ePrint Archive, Paper 2025/2070, 2025. URL: https://eprint.iacr.org/2025/2070. 3

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology – a clear roadmap for side-channel evaluations. In *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2015. doi:10.1007/978-3-662-48324-4_25. 10, 30

[SMMK11] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. Twine : A lightweight , versatile block cipher. ECRYPT, 2011. https://www.nec.com/en/global/rd/tg/code/symenc/pdf/twine_LC11.pdf. 37

[SMS19] Sumanta Sarkar, Kalikinkar Mandal, and Dhiman Saha. On the relationship between resilient boolean functions and linear branch number of S-boxes. In *INDOCRYPT 2019*, volume 11898 of *Lecture Notes in Computer Science*, pages 361–374. Springer, 2019. doi:10.1007/978-3-030-35423-7_18. 2, 16

[SPGQ06] François-Xavier Standaert, Gilles Piret, Neil Gershenfeld, and Jean-Jacques Quisquater. Sea: A scalable encryption algorithm for small embedded applications. In *Smart Card Research and Advanced Applications*, pages 222–236, 03 2006. doi:10.1007/11733447_16. 13

[SSS20] Sumanta Sarkar, Yu Sasaki, and Siang Meng Sim. On the design of bit permutation based ciphers – the interplay among s-box, bit permutation and key-addition. In *IWSEC 2020*, volume 12231 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2020. doi:10.1007/978-3-030-58208-1_1. 17

[ST17] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017*, volume 10212 of *Lecture Notes in Computer Science*, pages 185–215, 2017. URL: https://doi.org/10.1007/978-3-319-56617-7_7. 27

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. 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. 22

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear invariant attack - practical attack on full scream, iscream, and midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10032 of *Lecture Notes in Computer Science*, pages 3–33, 2016. URL: https://doi.org/10.1007/978-3-662-53890-6_1. 28

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In *EUROCRYPT 2015*, pages 287–314, 2015. URL: https://doi.org/10.1007/978-3-662-46800-5_12. 25

[Ull10] Markus Ullrich. *The design and efficient software implementation of S-boxes*. PhD thesis, KU Leuven, 2010. https://mouha.be/wp-content/uploads/ullrich-thesis.pdf. 11

[WSW+24] Libo Wang, Ling Song, Baofeng Wu, Mostafizar Rahman, and Takanori Isobe. Revisiting the boomerang attack from a perspective of 3-differential. *IEEE Transactions on Information Theory*, 70(7):5343–5357, 2024. doi:10.1109/TIT.2024.3381655. 3

[WWGY14] Yanfeng Wang, Wenling Wu, Zhiyuan Guo, and Xiaoli Yu. Differential cryptanalysis and linear distinguisher of full-round zorro. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, pages 308–323, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-07536-5_19. 2

[XCF+25] Shengyuan Xu, Siwei Chen, Xiutao Feng, Zejun Xiang, and Xiangyong Zeng. Cryptanalysis of BAKSHEESH block cipher. In *Inscrypt 2024*, volume 15529 of *Lecture Notes in Computer Science*, pages 280–299. Springer, 2025. Full version: https://eprint.iacr.org/2024/1926. doi:10.1007/978-981-96-4734-7_19. 3, 4, 22, 28

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678, 2016. URL: https://doi.org/10.1007/978-3-662-53887-6_24. 25

[YO19] Yan Yan and Elisabeth Oswald. Examining the Practical Side Channel Resilience of ARX-boxes. In *CF '19: Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 373–379. ACM, 2019. doi:10.1145/3310273.3323399. 30