Divide and Rule: DiFA - Division Property Based Fault Attacks on PRESENT and GIFT

Anup Kumar Kundu[1], Shibam Ghosh[2], Dhiman Saha[3], and

Mostafizar Rahman[1]

[1] Indian Statistical Institute, Kolkata 700108, India
anupkundumath@gmail.com,mrahman454@gmail.com
[2]Department of Computer Science, University Of Haifa, Haifa, Israel
sghosh03@campus.haifa.ac.il
[3]de.ci.phe.red Lab, Department of Computer Science & Engineering,
Indian Institute of Technology, Bhilai 492015, India
dhiman@iitbhilai.ac.in

Abstract. The division property introduced by Todo in Crypto 2015 is one of the most versatile tools in the arsenal of a cryptanalyst which has given new insights into many ciphers primarily from an algebraic perspective. On the other end of the spectrum we have fault attacks which have evolved into the deadliest of all physical attacks on cryptosystems. The current work aims to combine these seemingly distant tools to come up with a new type of fault attack. We show how fault invariants are formed under special input division multi-sets and are independent of the fault injection location. It is further shown that the same division trail can be exploited as a multi-round Zero-Sum distinguisher to reduce the key-space to practical limits. As a proof of concept division trails of PRESENT and GIFT are exploited to mount practical key-recovery attacks based on the random nibble fault model. For GIFT-64, we are able to recover the unique master-key with 30 nibble faults with faults injected at rounds 21 and 19. For PRESENT-80, DiFA reduces the key-space from 2^{80} to 2^{16} with 15 faults in round 25 while for PRESENT-128, the unique key is recovered with 30 faults in rounds 25 and 24. This constitutes the best fault attacks on these ciphers in terms of fault injection rounds. We also report an interesting property pertaining to fault induced division trails which shows its inapplicability to attack GIFT-128. Overall, the usage of division trails in fault based cryptanalysis showcases new possibilities and reiterates the applicability of classical cryptanalytic tools in physical attacks.

1 Introduction

Symmetric-key cryptosystems have historically benefited from the public cryptanalysis that adds to the body of results and subsequent improvements in design which imbibe the notion of strength of symmetric ciphers

which are not provably secure like their asymmetric counterparts. The black-box cryptanalysis model has thus seen remarkable progress from the early days of differential and linear attacks to boomerang [45], integral [17], rebound [31], related-key [6], yoyo [8] and recently the division property [40,41] based attacks. All these attacks have contributed to the better understanding of how to design good symmetric ciphers. On the other hand, if we shift focus to the gray-box model, one could easily argue that fault analysis which aims to exploit malicious modifications in runtime execution of a cipher to cryptanalyze it, has received the maximum success. This is perhaps attributed to the ease with which they can be realized in real-world scenarios. Since the inception of this kind of analysis shown by Boneh *et al.* [11], these attacks have evolved in multiple directions and have also been combined with other attacks. Differential Fault Analysis (DFA) has had the most widespread attention from the community though integral fault analysis (IFA), SFA [23], SIFA [20], PFA [50] and very recently SEFA [44] have also been shown as worthy competitors. It can be appreciated that some attacks like DFA and IFA are in principle application of the classical differential and integral cryptanalytic strategies in the context of fault injection capability. Thus it is well-established that classical tools are invaluable aids in devising new physical attacks. The current work is yet another *successful* attempt in this direction.

In this work, we focus on revisiting the well-researched (bit-based) division property in the light of fault analysis attacks. Todo proposed division property as a generalization of the integral property [16] at Eurocrypt 2015 [42] and offered improved integral property for Simon [4], Keccak [5], and Serpent [7]. This initial version of division property was word-based, i.e., the propagation of the division property captured information only at the word level. In FSE 2016, Todo and Morii first introduced the bit-based division property [43] where the propagation captures information at the bit level which naturally exploits more information than the word-based counterpart. Unfortunately, finding bit-based division property of modern block ciphers is computationally expensive. In this case, automatic tools play a significant role. The main idea is to transform this search problem into an optimization problem and use an automatic tool to solve it. In this direction, Xiang et al. first proposed to use Mixed Integer Linear Programming (MILP) based tool in [48]. This approach has been used to attack many ciphers in the last few years [37,47,38]. Researchers have also reported other tools such as SAT/SMT based tools [22,26,24] and Constraint Programming (CP) based tools [39,24] to find bit-based division property.

Our work focuses on finding fault invariant bit-based division property using MILP based tool. *We would like to emphasize that this work is not a variation of the integral fault attack.* Our research leverages some exclusive properties of the division property itself like the evolution of division trails and extends them to devise fault based key-recovery attacks. In doing so, we first look at the MILP modeling of SBoxes and linear layers to search for and identify the trails that are relevant to fault analysis. Here, by *relevance* we imply the creation of *fault-invariants* which are historically known to be vital to fault analysis.

Our primary targets are lightweight SPN based block cipher with a bit-permutation for the linear layer. Obvious choice is the International Lightweight Block Cipher standard PRESENT [10] and NIST Lightweight Crypography contest finalist GIFT [3]. Our findings show that in case of PRESENT and GIFT, division trails exist which are invariant with respect to the input division set and hence can be translated to create fault invariants in intermediate states. In this case the idea stems from the fact that one could use faults to create the input division set in an intermediate state there by triggering the division trail whose output division set constitutes the invariant to be exploited in key-recovery using classical partial decryption.

1.1 Related Work.

Application of classical cryptanalysis to fault attacks started with the inception of DFA. Since then, there have been many new types of fault attacks that have been introduced leveraging properties that are well-studied in classical cryptanalysis literature. Integral Fault Attack (IFA) was introduced in [36] and applied on AES [18] which is based on the 3.5 round integral property of AES. Derbez *et al.* showcased the application of Meet-in-the-Middle and Impossible Differential strategies to mount fault attacks on AES [19]. In CHES 2016, Saha *et al.* introduced the classical popular idea of internal differential cryptanalysis in the fault analysis of CAESAR competition [35] candidate PAEQ [9].
Jeong *et al.* carried out a DFA on PRESENT80 and PRESENT128 by injecting 2 and 3 faults in 2-byte random fault model which reduces the key space to 1.7 and $2^{22.3}$ respectively with a computational complexity of 2^{32} [27]. In [52], DFA is mounted by injecting a random nibble fault in the 29-th round which reduces the key space of PRESENT80 and PRESENT128 to $2^{14.7}$ and $2^{21.1}$ respectively. The DFA attack proposed in [30] injects total 32 random nibble faults in 30^{th} and 31^{st} round of PRESENT80 to recover the master key uniquely. Bagheri *et al.* devise a DFA which uniquely recovers the master key of PRESENT80 by injecting total 18 random bit faults [2]. A DFA by Wang and Wang injects nibble faults in the key schedule to recover the master key [46]. Luo *et al.* mounted a DFA on GIFT128 by injecting random nibble faults in round 25, 26, 27 and 28 and recovers the master key [30]. Previous fault attacks on PRESENT and GIFT along with the results presented in this paper are tabulated in Table 1.

1.2 Our Contribution

The current work is the first attempt in exploring the effectiveness of division trails in mounting fault attacks. The primary contribution is the identification of fault invariants that develop in the intermediate states of the ciphers. Once identified the same can be leveraged to build highly effective distinguishers which are in-turn verified to reduce the sub-keys guessed in partial decryption. The basic aim is to induce input division sets in the internal state using fault-injection. For this purpose, we rely

Table 1: Comparisons of fault attacks on PRESENT and GIFT. Note that, 'Multilevel Key Recovery' refers to the strategy of using the recovered-subkey for partially decrypting the ciphertexts and recover another subkey by repeating the similar fault-injection at a different round. In 'Multi-set Key Recovery', multiple sets comprising of correct ciphertext and its corresponding faulty ciphertexts are used to filter out the wrong key candidates. DFA, PFA and AFA refer to differential fault attack, persistent fault attack and algebraic fault attack respectively. FI stands for fault injection and comma-separated values under the column 'FI Round' refers to the different rounds at which faults are injected during the 'Multilevel Key Recovery'.

Primitive	Fault Attack		#Faults	Reduced Key Space Size	FI Round	Ref.	Remarks
	Type	Model					
GIFT-64	DFA	Random Nibble Fault	81	1		[32]	Multilevel Key Recovery
	DiFA	Random Nibble Fault	30*	1	19,21	Sec. 4	
PRESENT-80	DFA	2-Byte Random Fault	2	1.7	28	[27]	Multi-Set Key Recovery
		Random Nibble Fault	8	$2^{14.7}$	29	[52]	
			32	1	30, 31	[30]	Multilevel Key Recovery
		Random Bit Fault	96	1	30, 31	[2]	
			18	1	28, 29		
		Nibble Fault	64	2^{29}		[46]	
	PFA		98	1		[51]	
	AFA		1	2^{30}		[49]	
	DiFA	Random Nibble Fault	15†	2^{16}	25	Sec. 5	Multilevel Key Recovery
PRESENT-128	DFA	2-Byte Random Fault	3	$2^{22.3}$	28	[27]	Multi-Set Key Recovery
		Random Nibble Fault	16	$2^{21.1}$	29	[52]	
	DiFA	Random Nibble Fault	30*	1	24,25	Sec. 5	Multilevel Key Recovery

*Expected #faults using CCP is 100. †Expected #faults using CCP is 50

on the random nibble fault model which is aligned with 4-bit SBox based substitution layer of PRESENT and GIFT. The input division set requires all the possible nibble faults to be induced in the same SBox. This is similar to the requirement of IFA and is well-studied in contemporary literature as a feasible and realistic fault model. In terms of hardware capability of the fault injection mechanism this translates to the ability

to induce faults at the same location. Recent advances in optical fault injection has made this an easily achievable feat [14,21,15]. We further also capitalize on the Coupon Collector Problem (CCP), to make this even more aligned to the random nibble faults on a random but fixed SBox for the desired division set to be created. Though this increases the fault count, it helps leverage the random nibble faults to satisfy the constraints of the input division set. Figure 1 summarizes the approach. It is worth mentioning that the current work gives the best profile in terms of fault injection round and constitutes the best results considering the level of penetration of fault inside the cipher.

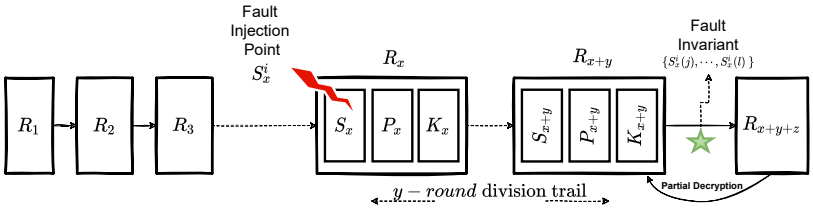

Fig. 1: Fault Invariant Division Trails. MILP based trail search helps find the positions $\{S_x^i(j), \cdots, S_x^i(l)\}$ which remain invariant to the position of the fault injection point (i.e. the SBox) $S_x^i(j)$

While the first contribution constitutes the application of division property in fault analysis, the second contribution is related to finding fault invariant division trails which is done through Mixed Integer Linear Programming (MILP) based automated tools. These tools have garnered a lot of interest in the symmetric crypto community since Mouha *et al.* [33] showcased its effectiveness in automating trail search. Table 3 summarizes the findings of this automated trail search which help to find the balance bit positions that constitute the fault invariant as shown in Figure 1. One of the interesting observations that we made was that the balance bit positions were *independent of the position of the SBox* where the input division set was induced using nibble faults. Finally, we have fault induced zero-sum distinguishers which can be verified to recover key bits via partial decryption. In case of GIFT, we also show how we can actually form multi-round zero-sum distinguishers exploiting the quotient-remainder groups that form an integral part of the linear layer of GIFT. Consolidating all the findings, we introduce DiFA or Division property based Fault Analysis which adds to the body of results that exploit classical cryptanalysis techniques in physical attacks.
Finally, we also stumble upon an interesting property due to which DiFA becomes inapplicable on GIFT-128. Our in-depth analysis shows that though faults invariants develop in the intermediate state, they become practically useless since the zero-sum distinguisher that is formed has a structure where ciphertext parts occur in even numbers and hence all key-guesses trivially pass the zero-sum filter. This phenomenon is due

to low diffusion of the division trail of GIFT-128 and is unavoidable in the context of DiFA. Interestingly, this equips GIFT-128 with implicit protection against the kind of fault analysis introduced in this work. Details of this are furnished Section 6.

Organization of the paper In section 2, we define the notation for the paper as well as provide a short introduction of the division property technique and zero-sum distinguisher. We also discuss an overview of various fault models and the targeted ciphers, GIFT and PRESENT in section 2. Then, in section 3, we introduce DiFA as our main contribution. section 4 and section 5 discuss the applicability of DiFA on GIFT and PRESENT, respectively. We also discuss inaplicability of DiFA on GIFT-128 in section 6. Finally, we conclude the paper in section 7.

2 Preliminaries and Background

2.1 Notations

In this section, we introduce the notations that we use to illustrate the properties exploited to mount the fault attacks described later. The size of a set X is denoted as $|X|$. We use bold lowercase letters to represent vectors in a binary field. For any n-bit vector $\mathbf{x} \in \mathbb{F}_2^n$, its i-th coordinate is denoted by x_i, thus we have $\mathbf{x} = (x_{n-1}, ..., x_0)$. We represent the binary vector with all elements being 0 as $\mathbf{0}$. The Hamming weight of $\mathbf{x} \in \mathbb{F}_2^n$ is $wt(x) = \sum_{i=0}^{n-1} x_i$.

For any two vectors $\mathbf{u}, \mathbf{x} \in \mathbb{F}_2^n$, we define the *bit product* as $\mathbf{x}^\mathbf{u} = \prod_{i=0}^{n-1} {x_i}^{u_i}$. We will often refer to $\mathbf{x}^\mathbf{u}$ as a monomial. For any two vector $\mathbf{k}, \mathbf{k}' \in \mathbb{F}_2^n$, we define $\mathbf{k} \succeq \mathbf{k}'$ if $k_i \geq k_i'$ for all $i = 0, 1, ..., n-1$. Note that if two n-bit vectors $\mathbf{u}, \mathbf{v}$, if $\mathbf{u} \succeq \mathbf{v}$, then the monomial $\mathbf{x}^\mathbf{v}$ divides $\mathbf{x}^\mathbf{u}$ or in other words $\mathbf{x}^\mathbf{u}$ contains $\mathbf{x}^\mathbf{v}$.

For any bit-vector, $\mathbf{a}$, we often call the bit positions with value 1 as the *active* bit positions. Let $Y \subseteq \mathbb{F}_2^n$ be a multi-set of vectors. A coordinate position $0 \leq i < n$ is called a *balanced position* if $\bigoplus_{\mathbf{y} \in Y} y_i = 0$.

The Algebraic Normal Form (ANF) of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ can be defined as $f(\mathbf{x}) = \bigoplus_{\mathbf{u} \in \mathbb{F}_2^n} a_\mathbf{u}^f \mathbf{x}^\mathbf{u}$ and the degree of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is d if d is the degree of the largest monomial in the ANF of f, i.e., $d = \max_{\mathbf{u} \in \mathbb{F}_2^n, a_\mathbf{u}^f \neq 0} wt(\mathbf{u})$.

In this paper, the notation R^ℓ is used to represent the ℓ-th round function of an iterative cipher consisting of r rounds, where ℓ takes values from 0 to $r-1$. The input state and round key of R^ℓ are denoted by S^ℓ and K^ℓ, respectively. The focus of the current work is on the ciphers PRESENT and GIFT, both of which have 4-bit SBox-es and are represented based on nibbles. Specifically, the i-th nibble of S^ℓ is denoted by N_i^ℓ, and the j-th bit of this nibble is denoted by $N_{i,j}^\ell$. In terms of endianness, the right most nibble is considered as the 0-th nibble.

2.2 Zero-sum distinguishers and Division Property

The bit-product of the output bits of a symmetric cryptographic scheme can be considered as a polynomial over $\mathbb{F}_2$, denoted as

$$f(k_{n-1}, ..., k_0, x_{m-1}, ..., x_0),$$

where $k_0, ..., k_{n-1}$ are the secret variables and $x_0, ..., x_{m-1}$ are the public variables, usually plaintext bits. Zero-sum distinguishers [1,12,13] distinguish a cryptographic Boolean function from a random function based on the XOR-sum of this polynomial representation. The attacker aims to find a subset of public variables $I \subset \{x_0, ..., x_{m-1}\}$. The central concept is to create a set of inputs ν by considering all possible combinations for the variables in I, while the remaining bits have fixed values. Thus, the input set forms an affine vector space ν of dimension $|I|$. Therefore, the resulting output sets are the $|I|$-th derivative of the corresponding Boolean function with respect to ν. This approach was first suggested as the higher-order differential attack [28,29].
Consider the monomial $\mathbf{x^u}$, where $u_i = 1$ if $x_i \in I$ and $u_i = 0$ otherwise. If we can prove that no monomial in the Algebraic Normal Form (ANF) of the Boolean function f contains the monomial $\mathbf{x^u}$, then for any fixed value of $\mathbf{k}$, the following holds:

$$\bigoplus_{\mathbf{x}\in\nu} f(\mathbf{k}, \mathbf{x}) = 0$$

A random function should not possess such a property and consequently this gives us a distinguisher. The time and data complexity of the distinguisher is $2^{|I|}$, and the memory complexity is negligible. Consider the following example.

Example 1. Let us consider a Boolean function $f(k_2, k_1, k_0, x_2, x_1, x_0) = k_1x_1 + k_2x_2x_0 + k_0x_0$. If we take $I = \{x_0, x_1\}$ and construct the corresponding set ν, then for any value of k_0, k_1, k_2 and x_2 we have

$$\bigoplus_{\mathbf{x}\in\nu} f(\mathbf{k}, \mathbf{x}) = 0.$$

Whereas, if we take $I = \{x_0, x_2\}$, we cannot guarantee such property.

However, finding algebraic properties, such as the polynomial expressions of the output bits of a real-life cipher, is usually very difficult due to computational complexity. The bit-based division property provides a systematic way to determine whether a particular monomial is contained in some monomial of the polynomial representation corresponding to the bit-product function of the output bits. Let us recall the definition of the bit-based division property.

Definition 1. *(Bit-based Division Property. [43]) A multi-set $X \subseteq \mathbb{F}_2^n$ is said to have the division property $\mathcal{D}_{\mathbb{K}}^n$ for some set of m-dimensional vectors $\mathbb{K}$ if for all $\mathbf{u} \in \mathbb{F}_2^n$, if it fulfills the following conditions:*

$$\bigoplus_{\mathbf{x}\in X} \mathbf{x^u} = \begin{cases} \textit{unknown}, & \textit{if there is } \mathbf{k} \in \mathbb{K} \textit{ s.t. } \mathbf{u} \succeq \boldsymbol{k} \\ 0 & \textit{otherwise} \end{cases}$$

Propagation of Division Property The bit-based division property can help us to determine if a particular monomial is contained in some monomial (the "unknown" case) or not (the "zero" case) in the polynomial representation of the bit-product function of the output bits. Suppose that we have two n-bit functions f and g such that $\mathbf{y} = f(\mathbf{x})$ and $\mathbf{z} = g(\mathbf{y}) = g \circ f(\mathbf{x})$. The division property captures that if all monomials of the bit-product $\mathbf{y}^{\mathbf{w}}$ appearing in $\mathbf{z}^{\mathbf{w}'}$ do not involve a monomial $\mathbf{x}^{\mathbf{u}}$, then $\mathbf{z}^{\mathbf{w}'}$ does not either. Therefore, we can study how the division property propagates through basic operations of a cipher, such as SBox, a linear function, or even a round function. We are interested in how the division property can propagate through these functions. If the input set with division property $\mathcal{D}_{\{\mathbf{k}_0^n\}}$ propagates to the output set with division property $\mathcal{D}_{\{\mathbf{k}_1^n\}}$ through some function, we call $(\mathbf{k}_0, \mathbf{k}_1)$ a valid *division trail*. In other words, if $(\mathbf{k}_0, \mathbf{k}_1)$ is a valid division trail through the function f and $\mathbf{y} = f(\mathbf{x})$, then at least one monomial in the ANF of $\mathbf{y}^{\mathbf{k}_1}$ contains $\mathbf{x}^{\mathbf{k}_0}$. A formal definition of a division trail was given in [48], and we recall the definition here.

Definition 2. *Let f_r denote the round function of an r round iterative primitive. Suppose the initial division property is $\mathcal{D}^n_{\mathbf{k}_0}$ and after $(i-1)$-round propagation, the division property is $\mathcal{D}^n_{\mathbb{K}_i}$. Then we have the following chain of division property propagations:*

$$\{\mathbf{k}_0\} := \mathbb{K}_0 \xrightarrow{f_0} \mathbb{K}_1 \xrightarrow{f_1} \mathbb{K}_2 \xrightarrow{f_2} \cdots$$

Moreover, for any vector $\mathbf{k}_i \in \mathbb{K}_i$ $(i \geq 1)$, there must exist a vector $\mathbf{k}_{i-1} \in \mathbb{K}_{i-1}$ such that $\mathbf{k}_{i-1}$ can propagate to $\mathbf{k}_i$ by division property propagation rules. For $(\mathbf{k}_0, \mathbf{k}_1, \cdots, \mathbf{k}_{r-1})$, if $\mathbf{k}_{i-1}$ can propagate to $\mathbf{k}_i$ for all $i \in \{1, 2, \cdots, r\}$, we call $(\mathbf{k}_0, \mathbf{k}_1, \cdots, \mathbf{k}_r)$ an r-round division trail.

Our main goal is to find valid division trails through a given function, and we aim to automate this task. To achieve this, a Mixed Integer Linear Programming (MILP) approach was proposed by the authors in [48]. The approach models the valid division trails of a function with linear inequalities so that only the valid trails satisfy the system. In this paper, we focus on two lightweight ciphers: PRESENT and GIFT. Both of these ciphers are designed based on the SPN structure, which consists of a substitution layer with parallel applications of SBox, followed by a bit-permutation layer. Together, these form the round function of the ciphers. We provide a brief introduction to modeling a SBox, and for more information on MILP modeling, please refer to [48].

A division trail may suggest some balanced position based on the output division properties. After obtaining a set of balanced positions for an input division property $\mathbf{k}$, we can distinguish the cipher E from a random function. To achieve this, we construct an output set Y from a set X of plaintexts, where $Y = \{y = E(\mathbf{x}) \mid \mathbf{x} \in X\}$. The set X is an affine subspace, constructed based on the input division property $\mathbf{k}$. For each vector $\mathbf{x} = (x_0, \cdots, x_{n-1}) \in X$, if the i-th coordinate of $\mathbf{k}$ is 1, then x_i can take any possible value from $\{0, 1\}$, and if the i-th coordinate of $\mathbf{k}$ is 0, then x_i is set to a fixed constant $c_i \in \{0, 1\}$. As the size of the set X is $2^{wt(\mathbf{k})}$, the data complexity is $2^{wt(\mathbf{k})}$.

Modeling SBox Xiang et al. [48] proposed a method to accurately compute the propagation of bit-based division property through an SBox. The process is recalled in Algorithm 2 given in Appendix B. This algorithm takes an input division property vector $\mathbf{k} = (k_0, k_1, \cdots, k_{n-1})$ and computes all possible bit-product functions of the output bits. It then checks if the monomial corresponding to the input property is contained in any of the bit-product functions. For instance, if some monomial in the polynomial representation of the bit-product function $\mathbf{y}^{\mathbf{v}}$ contains $\mathbf{x}^{\mathbf{k}}$ then $(\mathbf{k}, \mathbf{v})$ is included in the set of output division property. The algorithm outputs a set of vectors $\mathbb{K}_k$ such that the output multi-set has division property $\mathcal{D}^n_{\mathbb{K}_k}$.

MILP Modeling of Substitution Permutation Network Here we will discuss the modeling of division property propagation rules for SPN constructions where the permutation layer only consists of bit permutation. It should be noted that both the ciphers PRESENT and GIFT fall into this category. We consider an r-round SPN with a state size of n. To model such a construction in MILP, we define the MILP variables $\mathbf{a}^{i-1}$ and $\mathbf{a}^i$ to denote the input and output property of the SBox layer for $i = 1, 2, ..., r$. Each $\mathbf{a}^i$ is of the form $a^i_{n-1} \cdots a^i_0$, where $a^i_j \in \{0, 1\}$. Then, $\mathbf{a}^i$ is rotated according to the bit-permutation, and we obtain another set of variables, $\mathbf{b}^i$. Note that as this is only a permutation of variables, we do not need to introduce new variables for $\mathbf{b}^i$. Instead, we can just connect $\mathbf{a}^i$ and $\mathbf{b}^i$ according to the bit-permutation. The propagation chain is depicted as follows, where we have omitted the last bit-permutation.

$$a^0 \xrightarrow{\text{SBox}} a^1 \xrightarrow{\text{rotation}} b^1 \xrightarrow{\text{SBox}} a^2 \xrightarrow{\text{rotation}} \cdots b^{r-1} \xrightarrow{\text{SBox}} a^r.$$

Once the MILP model is prepared (carried out once per cryptosystem), we set the values of a^0 to the selected input division property (active bit positions). Once the initial input division property is provided to the MILP solver, the choice of the output division property depends on when we want to terminate the search, i.e., when we obtain a set without an integral property. This is outlined in the following proposition.

Proposition 1. *([48]) Let $\mathbb{X}$ be a multi-set with bit-based division property $\mathcal{D}^n_{\mathbb{K}}$, then $\mathbb{X}$ does not have integral property iff $\mathbb{K}$ contains all vectors of weight* 1.

According to Proposition 1, if all the unit vectors are contained in $\mathbb{K}_r$, we can terminate the search. On the other hand, if the i-th unit vector is not in $\mathbb{K}_r$, then the i-th bit is balanced based on the definition of the bit-based division property. Therefore, we obtain a comprehensive MILP model that can be solved using the freely available tools such as Gurobi [25]. An interested reader can refer [48] for further details of the constraints, variables and objective function used in the MILP model.

2.3 Fault Attacks

A Fault Attack is an attack to break the cryptosystem by exploiting its hardware design. A successful fault attack consists of two things Fault

Injection and Fault Propagation. Fault injection depends upon the hardware and the fault model. Fault propagation depends upon the property and the design structure of the cipher. Fault attacks are most useful in a practical scenario. If an attacker gets access to the device, he can manipulate the device and get the hidden information from there.

Fault Model The Fault Model shows the type and nature of a fault. Depending upon the impact of the fault, a fault model can be of bit level, or nibble model or, byte level. Also, it can be categorized by seeing whether its distribution is uniform or random. The attack we present here falls under the random nibble level fault model. After choosing the nibble randomly we give faults to generate some ciphertexts and try to recover the key from there.

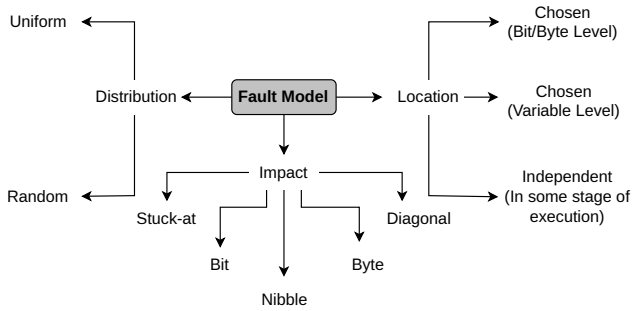

Fig. 2: Fault Model

Differential Fault Attack (DFA) Differential Fault Attack is one of the basic fault attack techniques to recover the original key of the cipher using the fault attack and the classical differential cryptanalysis technique. Here the attacker takes a message and computes its corresponding ciphertext through the oracle. Then he takes the same message and injects a particular difference in the bit, nibble, or byte location at some round r. The difference then propagates through the cipher and the attacker receives a ciphertext as an output at the end. This ciphertext differs from the original one as the fault is injected in some intermediate round r at the propagation of the original plaintext. The newly generated ciphertext is called the *faulty ciphertext*. Now the difference between the original and faulty ciphertext C and C' can be viewed as if the difference propagates through the $n - r$ rounds of the cipher and generates the outputs at last. From these two ciphertexts, C and C' the attacker can try to recover the original key of the cipher by using the classical differential cryptanalysis technique.

Integral Fault Attack (IFA) In an Integral Fault Attack the attacker tries to recover the key by combining the classical integral cryptanalysis

technique with the fault attack. Here initially the attacker generates the ciphertext of the original message. Then he gives faults to generate the All property in a particular bit, nibble or, byte position. Depending upon the impact of the fault attack the number of faults can vary. As an example to generate the All property in a byte the attacker has to give 2^8 many faults whereas for nibble he has to give 2^4 many faults. Depending upon the given faults in a particular intermediate round, he gets some ciphertexts C_i, for $i \in \mathbb{N}$ as the output. The generated ciphertexts can be viewed as a propagation of the All property from the intermediate round r. From the original and the faulty ciphertexts C and C_i''s, attacker tries to recover the key of the cipher. An important aspect of IFA is the fault-induced input set creation which essentially means getting the *All* property at an intermediate state of the cipher. While in the known fault model this is easily captured, the random fault model needs a special treatment. To explain this we borrow a very well-known result from combinatorics.

Definition 3 (Coupon Collector Problem - CCP [34]). *Given a set of n coupons, the collector draws randomly $l(1 \leq l \leq n)$ many coupons at each trial with replacement. Then the expected number of trials necessary to collect at least one of each coupon of the n coupons is given as below.*

$$E(X) = nH_n, \tag{1}$$

where $H_n = \sum_{i=1}^{n} 1/i$ is the n-th Harmonic number, and for large n, this equals with $\log(n) + O(1)$.

In the context of IFA, the coupons map to distinct faults required to cover the All property. This generally implies that to create the requisite set, the attacker needs to induce a higher number of faults than the cardinality of the set and the expected number is given by Equation 1 as per CCP. In the current work we use CCP to capture expected number of faults under random nibble fault model as shown in Table 1 and 4.

2.4 PRESENT [10]

PRESENT is an ultra-lightweight SPN structured cipher with 31 rounds. Both the versions of PRESENT consist of a 64-bit state size (i.e. sixteen 4-bit words) and a key size of 80 or 128 depending upon its two variants 80 or 128 respectively. Each round of this cipher contains the addRoundKey, SBox, and permutation layer operations. The key schedule takes the whole 80/128-bit key depending upon the version and generates the round keys of size 64 by bit rotation, applying SBox and adding round counter at each round. The round key XORs with the state at the initial phase of each round. Then the non-linear SBox layer applies to the state. PRESENT uses 4-bit SBox (given in Table 2) and the SBox applied to each 16 words of the state. The state bits are then permuted among themselves through the permutation layer and go as the input bits in the next round. After the 31st round, the final key K^{32} is XOR-ed with the state and returns as the ciphertext of the cipher.

Primitive	SBox															
PRESENT	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2
GIFT	1	a	4	c	6	f	3	9	2	d	b	7	5	0	8	e

Table 2: PRESENT and GIFT SBox

2.5 GIFT [3]

GIFT family is designed to reduce the hardware area even more than PRESENT without compromising its security. GIFT is also an SPN structured block cipher with two versions 64 and 128. Both the ciphers use 128-bit key but the state size and the number of rounds differs depending upon the versions. GIFT-64 consists of 28 rounds with a 64-bit state size and GIFT-128 has 40 rounds with a state size of 128-bit. Each round of the cipher contains SubCells, Bit Permutation, Addition of round keys, and Addition of Round Constants. Initially, the cipher takes the master key of 128 bits and generates 32 or 64-bit round keys for each round for GIFT-64 and GIFT-128 respectively. In each round, at first the 4 bit to 4 bit SBox (given in Table 2) is applied on the state bits. The bits are then permuted through the permutation layer and after that, the round keys and round constants are XOR-ed. For GIFT-64 the round keys are XOR-ed with the 0th and 1st bit whereas for GIFT-128 the round keys are XOR-ed with the 1st and 2nd bit of each nibble. The 6-bit round constants are XOR-ed at some specific positions with the states for both the versions and after 28 rounds or 40 rounds depending upon GIFT-64 or GIFT-128 it returns the ciphertext as output.

3 DiFA: Division Property Based Fault Analysis

This section presents our main contribution DiFA, which establishes the prospect of exploiting the bit-based division property in the context of fault attacks. In the subsequent sections, we apply this contribution to GIFT-64 and PRESENT-80/128. As previously stated, the fault model used is the *random* nibble fault. To create the input division set in a way that closely approximates a practical set-up, we leverage the Coupon Collector Principle to estimate the expected number of faults. Next we furnish the details of DiFA which proceeds in three steps.

Search For Input Division Set Invariant Division Trails The input division set is an affine subspace ν that, when induced in the input, leads to a specific division trail. For example, in a b-bit block cipher with an s-bit SBox, the input division set would consist of the input bits corresponding to a particular SBox. When the input division set is induced, these bits will take all values $\in \{0,1\}^s$, while the remaining bits will take a fixed value $\in \{0,1\}^{b-s}$. The size of the overall plaintext set is therefore 2^s. The chosen SBox is called the active SBox. By the virtue of the bit-based division property introduced by Todo [43], the division

Table 3: Here bb denotes Balanced bit (excluding the last $p-$layer) and Inv. denotes Fault Invariant

Primitive	round	#sets	#bb	bb position	Inv.
PRESENT-80/128	4	1	64	$\{0, \cdots, 63\}$	✓
	5	1	4	$\{0, 4, 8, 12\}$	✓
GIFT-64	4	1	64	$\{0, \cdots, 63\}$	✓
	5	2	16, 17	$\cup_{j=0}^{15}\{4j\}$	✓
				$\cup_{j=0}^{15}\{4j\} \cup \{53\}$	✗
GIFT-128	4	1	128	$\{0, ..., 128\}$	✓
	5	2	80	section A	✓

set propagates through various layers of the cipher, generating division trails (see Definition 2). The output division set corresponding to the trail indicates bit positions that are balanced and therefore admit a zero-sum. This zero-sum forms the distinguisher. *Our primary observation is that there exist output-division sets that are invariant with respect to the position of the active SBox chosen for the input division set.* To identify these sets, we reuse the MILP models developed by Zhang *et al.* [48] for GIFT and PRESENT and made slight modifications. We then performed an automated search and results are furnished in Table 3. The left part of Figure 3 summarizes the formation of division trails. The next step is to adapt this property to FA.

Formation of Fault Induced Intermediate Division Set The property discussed above inspired us to apply it to fault analysis. This property allows us to inject random faults in a fixed but unknown SBox and induce the input set, thereby generating division trails that result in fault invariants in terms of the output division set. These fault invariants help us recover the key using the zero-sum distinguisher. Faults are therefore useful for inducing division trails in the intermediate state.
Consider a $(n+p)$-round cipher, in which we aim to identify the division property at the output of n rounds. Suppose that, the fault injection is required at the input of $(n-r)$-th round to find a division trail for r rounds. In such cases, the cipher needs to be replayed on a fixed plaintext for 2^s times (assuming $s \times s$ SBox), with a random nibble fault injected in a fixed but unknown SBox every time (except for one fault-free run). The replay count increases when the Coupon Collector Principle is used to exhaust all values of the input set. This process is depicted in the right half of Figure 3.

Key-Recovery Leveraging Fault Invariants In the end, we utilize a fault-invariant zero-sum distinguisher to recover the round keys for the last p rounds after n rounds. Our approach involves guessing the round keys in a specific manner that takes advantage of the linear layers of both GIFT and PRESENT (described in the subsequent subsections).

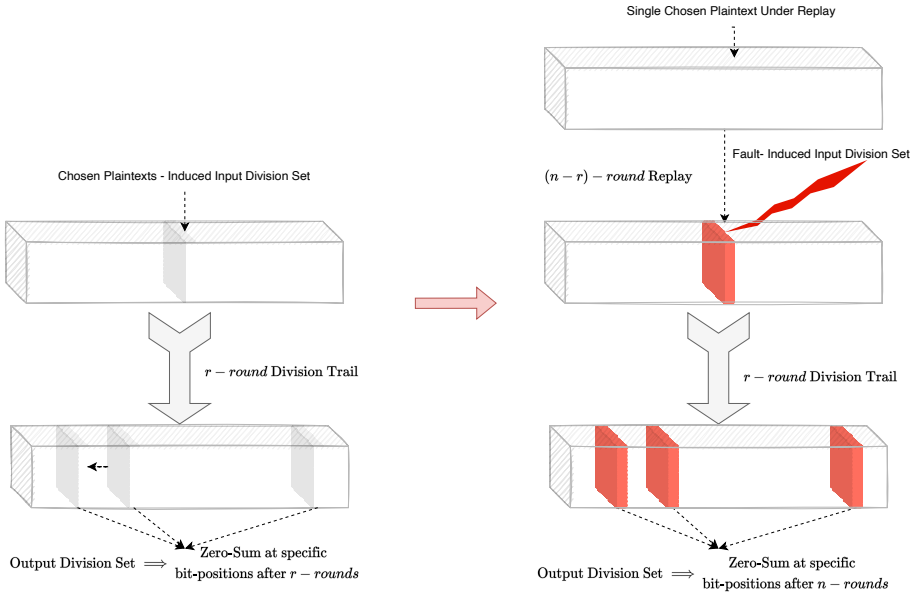

Fig. 3: Fault induced extension of the zero-sum distinguisher exploiting the bit-based division property

We then decrypt the ciphertext through p rounds to identify the zero-sum. This process is illustrated in Figure 1. Notably, the zero-sum distinguisher might be exploitable in multiple rounds, as we demonstrate for both GIFT and PRESENT. Finally, we employ the *Onion-Peeling Strategy* as the last phase of the key-recovery process, which is needed based on the size of the reduced key-space.

The Onion-Peeling Strategy The strategy is essentially employed when one fault-induced division set is unable to reduce the key-space to practical limits and when recovering one round key is not-enough to invert the key-schedule. In such cases, the attack is repeated with an additional fault-induced division set at a preceding round. The round where the fault needs to be induced in the second iteration is based on the strategy used in the first iteration. For example, for an n-round cipher if we are exploiting a r-round division trail and the zero-sum across two-rounds, then the first iteration will induce faults in round $(n-r-2)$ while the second iteration in round $(n-r-4)$. For a one round zero-sum exploit, the rounds will be $(n-r-1)$ and $(n-r-2)$ for first and second iterations respectively.

The next sections will delve into the cipher-specific details on how to utilize the division property in key-recovery attacks for GIFT and PRESENT.

4 Mounting DiFA on GIFT64

To apply DiFA to GIFT64, we utilize the 4 and 5-round division properties to recover the round keys of the last two rounds, which demonstrates the ability to leverage multi-round zero-sum distinguishers. It is important to note that this is achievable due to the specific design of the linear layer in GIFT. The basic idea is to make a guess of the last two round keys and partially decrypt the set of ciphertexts to verify the balanced property. The attack is carried out in two steps, making use of the balanced bits obtained from the 4 and 5-round division properties for the active nibble of the input division set, as detailed in Table 3. We first guess the last round keys to partially decrypt a set of ciphertexts and filter the keys based on the 5-round division property. Subsequently, we guess the penultimate round keys and decrypt one more round to verify the 4-round division property for all the key suggestions obtained from the last round.

Exploiting Quotient-Remainder ($\mathcal{QR}$) group structures The bit-permutation layer of GIFT's round function has an interesting property called the *quotient-remainder* or $\mathcal{QR}$ group structure, as defined by Banik *et al.* [3]. This structure directly benefits our attack by enabling us to exploit the multi-round zero-sum distinguisher. According to this structure, the bit-permutation layer maps the output bits of 4 SBox-es from a *quotient group* to the input bits of 4 SBox-es in the corresponding *remainder group*. Specifically, the q-th quotient group contains the p-th SBox if $p = 4q + r$, and the r-th remainder group contains the p-th SBox if $p = 4q + r$. For instance, in the i-th round, the nibbles $\{N_0^i, N_1^i, N_2^i, N_3^i\}$ form the 0-th quotient group, while the nibbles $\{N_0^{i+1}, N_4^{i+1}, N_8^{i+1}, N_{12}^{i+1}\}$ form the corresponding remainder group for the next round. Consequently, the SBox-es of two consecutive rounds can be grouped together, forming a super SBox. This property allows us to recover the round keys partially (independently for each $\mathcal{QR}$ group) for two rounds, rather than guessing the full round keys at once. This significantly improves the time complexity of the attack and enables us to reduce the key-space using only one fault-induced division set. In the following, we provide a detailed description of the attacks and their complexity analysis.

Key Recovery Attack The GIFT-64 cipher is composed of 28 rounds, with R^i representing the i-th round function for $i \in \{0, 1, ..., 27\}$. The objective is to recover the last two round keys, K^{27} and K^{26}. This is achieved by constructing an input division property at the input of R^{22}. To do so, a random nibble N_i^{22} is selected for $i \in \{0, \cdots, 15\}$ and 15 faults are injected to activate the nibble N_i^{22}. As per the CCP (refer to Definition 3), the attacker would need to inject 50 random nibble faults in some fixed but unknown nibble to obtain these distinct 15 faults. If the i-th nibble is active at R^{22}, the resulting input division property takes the form:

$$a^{22} = (0, 0, 0, 0, ..., 1, 1, 1, 1, 0, 0, 0, 0, ..., 0, 0, 0, 0)$$

where the four consecutive 1's are positioned at $4i$, $4i + 1$, $4i + 2$, and $4i + 3$ for some $i \in \{0, ..., 15\}$. From this input property, we get balanced

bits after the SBox layer of R^{25} and R^{26} from the 4-round and 5-round division properties, respectively. The positions of the balanced bits can be found in Table 3. This step of the attack allows for the recovery of the last round key K^{27}. The attack is carried out independently for each $\mathcal{QR}$ group (as shown in Figure 4). Furthermore, if only R^{27} is considered, each SBox in the remainder group can be addressed separately, leading to a reduction in time complexity. The following discussion refers to Figure 4, which illustrates the key-recovery attack for the 0-th $\mathcal{QR}$ group. The balanced bits are highlighted in red.

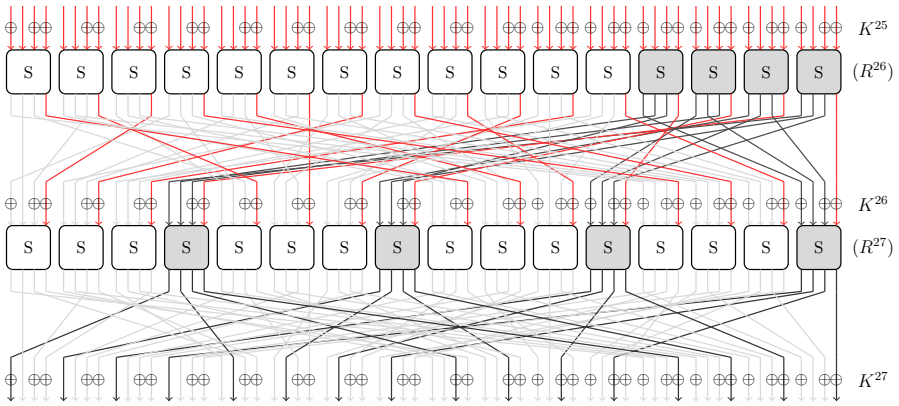

Fig. 4: Key recovery phase of DiFA on GIFT-64

From Table 3, it can be observed that the 0-th output bit of each SBox is balanced at R^{26}. Due to the bit-permutation layer, these balanced bits go to the 0-th input bit of each SBox at R^{27}. To utilize this zero-sum distinguisher, we first select four SBox-es from a specific remainder group. Next, we partially decrypt four nibbles of the 16 ciphertexts (15 faulty and one fault-free ciphertext) through the bit-permutation layer and the four SBox-es mentioned above. This one-round decryption is carried out independently for each SBox. Note that only two key bits are XOR-ed at each nibble of the state. Thus, we need to guess 2-bit keys for each SBox inversion. For each possible 2-bit key, we decrypt the 4 bits of ciphertext and check for zero-sum at the 0-th bit of each nibble. This step yields a reduced key-space for the 8 bits guessed in total.

For each decrypted ciphertext from the first step, we proceed to the second step of the attack, which involves decryption through one more round. Based on Table 3, we observe that all the output bits of each SBox are balanced at R^{25}, implying that all the input bits at R^{26} are balanced as well. We can leverage this zero-sum distinguisher by guessing another 8 bits of the key. Then, for each key guess, we decrypt through 4 SBox-es in the quotient group (corresponding to the remainder group used in the first step) and check for zero-sum at the 4 input bits of each SBox. Once

again, it is not necessary to guess the 8 key bits corresponding to the quotient group all at once. Instead, we can decrypt each SBox separately by only guessing the 2 key bits that affect it.
We repeat this whole process independently for the other quotient-remainder groups. The attack procedure is summarized in Algorithm 1.

Results The results of extensive software simulations indicate that using a single fault-induced division set at round 21, it is possible to uniquely recover the last two round keys of GIFT-64. However, according to the GIFT-64 key-schedule, all four last round keys are needed to recover the master key. Therefore, we utilize the onion peeling strategy to exploit the second fault-induced division set in round $(21 - 2) = 19$. By doing so, we can use two sets (equivalent to 30 faults) to uniquely recover the master key of GIFT-64. Following the CCP (refer to Definition 3), this corresponds to approximately 100 random nibble faults.
Complexity of the attack: The attack is executed by employing the onion peeling strategy, which requires two fault-induced division sets. For each division set, a total of 16 plaintext-ciphertext pairs are required (15 fault-induced and one original). Therefore, the overall data complexity is $2 \times 2^4 = 2^5$ encryption queries.
For each of the 16 ciphertexts in a set, we decrypt two rounds independently for each $\mathcal{QR}$ group, rather than decrypting the whole 64-bit ciphertext. This significantly reduces the time complexity and can be done in parallel for each $\mathcal{QR}$ group. In the first step of the attack, we make a guess of a 2-bit key for each SBox in a remainder group and decrypt 4 SBox-es in that group. Thus, we need to perform 4×2^2 SBox inversions for each ciphertext. This gives us a complexity of $16 \times (4 \times 2^2) = 2^8$ SBox inversions for each remainder group. We then use a 1-bit zero-sum for each SBox to filter the 2-bit key. The expected number of keys that pass through this filter is 2 for each SBox. For each of these keys, we proceed with the second step of the attack.
In the second step of the attack, we focus on the corresponding quotient group and decrypt each SBox of the quotient group separately by guessing the corresponding 2-bit key. Thus, we need to perform $16 \times 2^4 \times (4 \times 2^2)$ SBox inversions for the corresponding quotient group. Therefore, the complexity of the first two steps of the attack is $2^8 + 2^{12} \approx 2^{12}$ SBox inversions for each $\mathcal{QR}$ group. Taking all of these into account, the overall complexity of the final two round key-recovery amounts to 2^{14} SBox inversions, with each of the four groups being worked on separately. The attack is summarized in Table 4.

5 Mounting DiFA on PRESENT-80/128

In this section, we discuss key recovery attacks on PRESENT-80. We want to emphasize that all the attacks we discuss in this section can also be extended to PRESENT-128, as the difference between the two constructions is only in the size of the master-key.
We exploit the propagation of the division property for 4 rounds for the cipher. We discuss one attack on PRESENT-80 in this section to recover

Algorithm 1 KeyRecoveryGIFT-64

Input: A list of ciphertexts $\mathcal{L}_c$
Output: A list of keys $\mathcal{L}_k$
Initialize four empty lists $\mathcal{L}_k^i$ for $i \in \{0,1,2,3\}$ ▷ for each $\mathcal{QR}$ group
for $i = 0$ to 16 **do**
    $\mathcal{L}_c[i] = P^{-1}(\mathcal{L}_c[i])$ ▷ Invert through bit-permutation layer
Prepare four lists $\mathcal{L}_c^i$ for $i \in \{0,1,2,3\}$ of 16 bit values from $\mathcal{L}_c$ according to $\mathcal{QR}$ groups
for $i = 0$ to 3 **do** ▷ for each $\mathcal{QR}$ group
    Initialize tables $\mathcal{T}_j$ for $j = 0,1,2,3$ to store 2-bit key
    and corresponding list of sixteen 4-bit decrypted bits.
    for $s = 0$ to 3 **do** ▷ for each SBox in the i-th remainder group
        for $k \in \{00,01,10,11\}$ **do**
            $S = 0 \quad \mathcal{M} = \phi$
            for $c \in \mathcal{L}_c^i$ **do**
                $m = SBox^{-1}(c[s] \oplus k)$
                $S = S \oplus m$
                $\mathcal{M} = \mathcal{M} \cup \{m\}$
            if S & `0x1= 0x0` **then** $\mathcal{T}_s = \mathcal{T}_s \cup \{(k, \mathcal{M})\}$
    Construct a table $\mathcal{T}^{27}$ to store 8-bit key
    and corresponding list of sixteen 16-bit decrypted bits

    Construct a table $\mathcal{T}^{26}$ to store 8-bit key
    of 27-th round and 8-bit key of 26-th round
    for $s = 0$ to 3 **do** ▷ for each SBox in the i-th quotient group
        for $(k^{27}, \mathcal{M}^{27})$ in $\mathcal{T}^{27}$ **do**
            $Flag = False$
            for $k \in \{00,01,10,11\}$ **do**
                $S = 0$
                for $m \in \mathcal{M}^{27}$ **do**
                    $S = S \oplus SBox^{-1}(m[s] \oplus k_s)$
                if S&`0xF=0x0` **then**
                    $\mathcal{T}^{26}[k^{27}][s] = k$
                    $Flag = True$
                    Break
            if $Flag = False$ **then**
                Remove $(k^{27}, \mathcal{M}^{27})$ from $\mathcal{T}^{27}$
        construct k^{26} and k^{27} from $\mathcal{T}^{26}$
        $\mathcal{L}_k^i = \mathcal{L}_k^i \cup \{k^{27} || k^{26}\}$
Construct full round keys for R^{26} and R^{27} from $\mathcal{L}_k^0, \mathcal{L}_k^1, \mathcal{L}_k^2, \mathcal{L}_k^3$ and store in to $\mathcal{L}_k$
return $\mathcal{L}_k$

the last round key using the 4-round division property. Another attack that recovers the partial round keys of the last as well as the second last round using both 4 and 5-round properties simultaneously is discussed in the Appendix C. The basic idea of the attack is similar to the one used in GIFT, as discussed in the previous section. We guess the last round key and partially decrypt a set of ciphertexts to check the balanced property. Next, we explain more about our attacks and show how we can use the faults along with division property to recover the key of the last round.

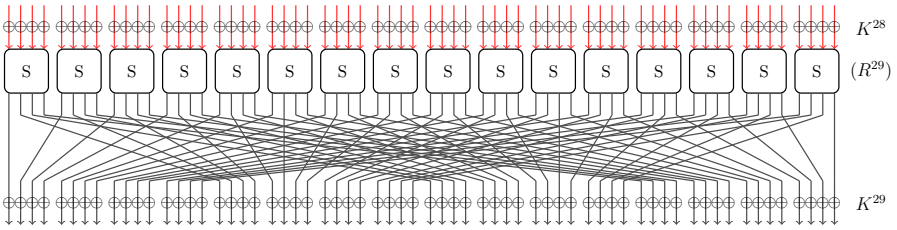

Fig. 5: Round-Key recovery verifying balancedness at the end of Round 28. Here, the key-bits corresponding to every sbox is guessed and reduced separately

5.1 Last Round Key Recovery

The PRESENT cipher consists of 31 rounds, with R^i denoting the i-th round function. The goal of our attack is to recover the last round key K^{30} by exploiting the 4-round division property. To achieve this, we inject 15 faults to generate the input division property at a random but fixed nibble at the input R^{26}. The CCP shows that generating these distinct 15 faults requires injecting 50 random nibble faults in some fixed but unknown nibble. This results in the following input division property:

$$a^{26} = (0, 0, 0, 0, ..., 1, 1, 1, 1, 0, 0, 0, 0, ..., 0, 0, 0, 0)$$

The four consecutive 1's are placed at $4i, 4i + 1, 4i + 2, 4i + 3$ if the i-th nibble is active at R^{26} for some $i \in \{0, ..., 15\}$. Using this input property, we observe that all bits are balanced after the SBox layer of R^{29} (see Table 3). Therefore, all input bits to R^{30} are also balanced, and we can exploit this zero-sum property to recover K^{30}.
To reduce the time complexity of the attack, we recover each nibble of the key independently. For each possible 4-bit key, we decrypt the 4 bits of the 16 ciphertexts (including 15 faulty and the original ciphertexts) and verify the zero-sum at 4 bits. The algorithmic details are provided in Algorithm 3 in Appendix C.1.

Results Based on experiments using software simulations, we found that by inducing one division set fault at round 25, the last round key of

PRESENT-80/128 can be recovered. However, the attack differs between PRESENT-80 and PRESENT-128. In the case of PRESENT-80, the reduced key-space is 2^{16} after the last round key is recovered, which makes the attack practical and therefore it ends there. However, in the case of PRESENT-128, the reduced key-space is 2^{64}, and hence, we require the onion peeling technique. The second fault-induced division set is located at round 24 which helps to recover the penultimate round key, and from the key-schedule, the master key can be obtained. As a result, for PRESENT-80 and PRESENT-128, 15 and 30 faults are required, respectively, leading to reduced key-spaces of 2^{16} and one, respectively.

Complexity of the attack: The attack for recovering the master key follows the onion peeling technique, which requires two phases and for each phase, a total of 16 plaintext-ciphertext pairs are required (15 fault-induced and one original). Therefore, the overall data complexity is $2 \times 2^4 = 2^5$ encryption queries. Instead of decrypting the entire ciphertext, we decrypt each nibble independently. In the first step of the attack, we guess a 4-bit key for each nibble. Then, we decrypt one SBox layer for each guessed key. Thus, for one nibble, we require 16 SBox inversions and 16 XOR operations of 4 bits. As a result, the complexity for the whole attack is $16 \times 16 = 2^8$ SBox inversions and 16 XOR operations of the entire state, which is equivalent to 2^4 rounds of decryption. Table 4 provides a summary of the attacks.

Table 4: Summary of our work on PRESENT-80/128 and GIFT-64; Here F: faults, F_{CCP}: Expected Faults using Coupon Collector Problem , FI: Fault Injection

Primitive	$\#F$ (n)	$\#F_{CCP}$ ($n.H(n)$)	FI Round		Reduced Keyspace	Complexities		
			1st Iteration	2nd Iteration		Data	Time*	Memory†
PRESENT-80	15	50	25	na	2^{16}	2^4	2^8	neg.
PRESENT-128	30	100	25	24	1	2^5	2^9	
GIFT-64	30	100	21	19		2^5	2^{15}	2^{12}

*unit of time is SBox inversion. †unit of memory is state size.

6 On The Inapplicability of DiFA on GIFT-128

Here we discuss the division properties that we have observed for the 4 and 5 rounds of GIFT-128. GIFT-128 is composed of 40 rounds, where R^i denotes the i-th round function for $i \in \{0, 1, ..., 39\}$. To construct the input division property at the input of R^{34}, we randomly choose a nibble N_i^{34} for $i \in \{0, ..., 15\}$ and inject 15 faults to activate the nibble N_i^{34}. Similar to the attack of GIFT-64, we have the following input division property:

$$a^{34} = (0, 0, 0, 0, ..., 1, 1, 1, 1, 0, 0, 0, 0, ..., 0, 0, 0, 0),$$

where the four consecutive 1's are placed at $4i, 4i+1, 4i+2, 4i+3$ if the i-th nibble is active at R^{34}. From this input property, we obtain balanced bits after the SBox layer of R^{37} and R^{38} from the 4-round and 5-round division property, respectively. The positions of the balanced bits are given in Table 3.

For GIFT-128, we have observed that the indices of the balanced bits after 5 rounds form two sets $\mathcal{A}$ and $\mathcal{B}$, depending on the location of the active nibble. If the active nibble at the input of R^{34} belongs to $\{N_0^{34}, ..., N_{15}^{34}\}$, we obtain the set $\mathcal{A}$, and for nibble in $\{N_{16}^{34}, ..., N_{31}^{34}\}$, we obtain the set $\mathcal{B}$. The sets $\mathcal{A}$ and $\mathcal{B}$ are listed in Appendix A. To obtain a fault-invariant property, we take the intersection of the two sets $\mathcal{A}$ and $\mathcal{B}$, resulting in the following:

- $\mathcal{A} \cap \mathcal{B} = \{0, 1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 25, 28, 29, 32, 33, 36, 37, 40, 41, 44, 45, 48, 49, 52, 53, 56, 57, 60, 61, 64, 65, 68, 69, 72, 73, 76, 77, 80, 81, 84, 85, 88, 89, 92, 93, 96, 97, 100, 101, 104, 105, 108, 109, 112, 113, 116, 117, 120, 121, 124, 125\}$.

The indices that belong to $\mathcal{A} \cap \mathcal{B}$ are balanced after 5 rounds, regardless of the active nibble number at the input of R^{34}. Therefore, this property is fault-invariant. Additionally, we observe that from $\mathcal{A} \cap \mathcal{B}$, the first and second input bits of each SBox-es at R^{39} are balanced.

To use this zero-sum distinguisher, we can use a similar method as in the attack of GIFT-64. However, we observe that the zero-sum based filter does not work for GIFT-128. Our investigations in this direction lead to some non-trivial results for GIFT-128, which we discuss here.

Even-nibble property We have observed that during the encryption of 2^4 plaintexts using the single nibble fault model in the first 6 rounds of GIFT-128 (excluding the last linear layer), most of the nibbles exhibit values in even numbers. We refer to this phenomenon as the *even-nibble property*.

Definition 4 (Even-nibble). *Let $\mathcal{C}$ be a set of ciphertexts generated by the 6-rounds of GIFT-128 (excluding the last linear layer). A nibble N_i for some $i \in \{0, ..., 31\}$ is said to exhibit the even-nibble property according to $\mathcal{C}$ if all the values for N_i occur an even number of times.*

For instance, if the values in the 0-th nibble follow the pattern given in Table 5 for some set of ciphertexts of size $\mathcal{C}$, then we can say that the 0-th nibble exhibits the even-nibble property. If such an event occurs, then we have $\oplus_{c \in \mathcal{C}} S^{-1}(c \oplus k) = 0$ for any choice of key, implying that all keys trivially pass through the zero-sum filter.

Values	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
Occurence	0	0	0	0	0	6	0	2	8	0	0	0	0	0	0	0

Table 5: Example of Even-nibble

Experimental Verification We have conducted experiments to verify our claim regarding the even-nibble property in GIFT-128. Specifically, we have used a set of 2^{16} plaintexts and counted the number of nibbles that

exhibit this property. Our results show that, on average, more than 92% of the SBox-es satisfy the even-nibble property. Furthermore, we have observed that for all the SBox-es exhibiting the even-nibble property, all 4 key guesses trivially pass through the zero-sum filter.

7 Conclusion and Open Problems

In this work, the first fault attack based on the bit-based division property - DiFA is proposed on bit-oriented SPN ciphers. The idea of inducing fault based input division sets is exploited to admit division trails in the intermediate state of a cipher to devise zero-sum distinguishers using balanced bit position in the output division set. Primary observation exploited is the formation of fault invariants which are independent of the nibble fault injection position thereby facilitating a fault-injection enabled adversary. MILP based search models are developed and executed to search for fault invariants. Key-recovery strategies devised here exploit multi-round zero-sum distinguishers leveraging the linear layer structures like quotient-remainder groups. Simulation models allow DiFA to be applied on GIFT-64 which leads to unique key-recovery with 30 faults (two fault induced division sets one at Round-21 and subsequently another at Round-19). For PRESENT-80, DiFA reduces the key-space to 16 bits with 15 faults (one fault induced division set at the Round-25). We also report an interesting result that makes GIFT-128 DiFA-resistant and investigate the event that allows this while also supporting this with empirical data. We believe that primary cause for this is linked to the cipher design and warrants further investigation. In terms of fault inject round penetration, this work breaks all previous records and hence gives the best attacks in that context. To conclude, DiFA adds a new tool in the arsenal of cryptanalysts for physical attacks and is expected to be applicable on a large class of ciphers.

References

1. Aumasson, J.P., Meier, W.: Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. rump session of Cryptographic Hardware and Embedded Systems-CHES **2009**, 67 (2009)
2. Bagheri, N., Ebrahimpour, R., Ghaedi, N.: New differential fault analysis on PRESENT. EURASIP J. Adv. Signal Process. **2013**, 145 (2013)
3. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: CHES. Lecture Notes in Computer Science, vol. 10529, pp. 321–345. Springer (2017)
4. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK lightweight block ciphers. In: Proceedings of the 52nd Annual Design Automation Conference. DAC '15, Association for Computing Machinery, New York, NY, USA (2015)

5. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The keccak reference. Submission to NIST **3**(30), 320–337 (2011)
6. Biham, E.: New types of cryptanalytic attacks using related keys. J. Cryptol. **7**(4), 229–246 (1994)
7. Biham, E., Anderson, R.J., Knudsen, L.R.: Serpent: A new block cipher proposal. In: FSE. Lecture Notes in Computer Science, vol. 1372, pp. 222–238. Springer (1998)
8. Biham, E., Biryukov, A., Dunkelman, O., Richardson, E., Shamir, A.: Initial observations on Skipjack: Cryptanalysis of Skipjack-3XOR. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 1556, pp. 362–376. Springer (1998)
9. Biryukov, A., Khovratovich, D.: PAEQ: parallelizable permutation-based authenticated encryption. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S. (eds.) Information Security - 17th International Conference, ISC 2014, Hong Kong, China, October 12-14, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8783, pp. 72–89. Springer (2014). https://doi.org/10.1007/978-3-319-13257-0_5, https://doi.org/10.1007/978-3-319-13257-0_5
10. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: CHES. Lecture Notes in Computer Science, vol. 4727, pp. 450–466. Springer (2007)
11. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: Advances in Cryptology – EUROCRYPT. Lecture Notes in Computer Science, vol. 1233, pp. 37–51. Springer (1997)
12. Boura, C., Canteaut, A.: Zero-sum distinguishers for iterated permutations and application to Keccak-f and Hamsi-256. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 6544, pp. 1–17. Springer (2010)
13. Boura, C., Canteaut, A., Cannière, C.D.: Higher-order differential properties of Keccak and *Luffa*. In: FSE. Lecture Notes in Computer Science, vol. 6733, pp. 252–269. Springer (2011)
14. Breier, J., He, W., Bhasin, S., Jap, D., Chef, S., Ong, H.G., Gan, C.L.: Extensive laser fault injection profiling of 65 nm FPGA. J. Hardw. Syst. Secur. **1**(3), 237–251 (2017). https://doi.org/10.1007/s41635-017-0016-z, https://doi.org/10.1007/s41635-017-0016-z
15. Colombier, B., Grandamme, P., Vernay, J., Chanavat, É., Bossuet, L., de Laulanié, L., Chassagne, B.: Multi-spot laser fault injection setup: New possibilities for fault injection attacks. In: 20th Smart Card Research and Advanced Application Conference-CARDIS 2021 (2021)
16. Daemen, J., Knudsen, L., Rijmen, V.: The block cipher SQUARE. In: Proceedings of FSE 1997. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer (1997)
17. Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher square. In: FSE. Lecture Notes in Computer Science, vol. 1267, pp. 149–165. Springer (1997)

18. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)
19. Derbez, P., Fouque, P., Leresteux, D.: Meet-in-the-middle and impossible differential fault analysis on AES. In: CHES. Lecture Notes in Computer Science, vol. 6917, pp. 274–291. Springer (2011)
20. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 547–572 (2018)
21. Dutertre, J., Beroulle, V., Candelier, P., Castro, S.D., Faber, L., Flottes, M., Gendrier, P., Hély, D., Leveugle, R., Maistri, P., Natale, G.D., Papadimitriou, A., Rouzeyre, B.: Laser fault injection at the CMOS 28 nm technology node: an analysis of the fault model. In: 2018 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2018, Amsterdam, The Netherlands, September 13, 2018. pp. 1–6. IEEE Computer Society (2018). https://doi.org/10.1109/FDTC.2018.00009, `https://doi.org/10.1109/FDTC.2018.00009`
22. Eskandari, Z., Kidmose, A.B., Kölbl, S., Tiessen, T.: Finding integral distinguishers with ease. In: SAC. Lecture Notes in Computer Science, vol. 11349, pp. 115–138. Springer (2018)
23. Fuhr, T., Jaulmes, É., Lomné, V., Thillard, A.: Fault attacks on AES with faulty ciphertexts only. In: FDTC. pp. 108–118. IEEE Computer Society (2013)
24. Ghosh, S., Dunkelman, O.: Automatic search for bit-based division property. In: LATINCRYPT. Lecture Notes in Computer Science, vol. 12912, pp. 254–274. Springer (2021)
25. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), `https://www.gurobi.com`
26. Hu, K., Wang, Q., Wang, M.: Finding bit-based division property for ciphers with complex linear layers. IACR Trans. Symmetric Cryptol. **2020**(1), 396–424 (2020)
27. Jeong, K., Lee, Y., Sung, J., Hong, S.: Improved differential fault analysis on PRESENT-80/128. Int. J. Comput. Math. **90**(12), 2553–2563 (2013)
28. Knudsen, L.R.: Truncated and higher order differentials. In: FSE. Lecture Notes in Computer Science, vol. 1008, pp. 196–211. Springer (1994)
29. Lai, X.: Higher order derivatives and differential cryptanalysis. In: Communications and Cryptography: Two Sides of One Tapestry. pp. 227–233. Springer (1994)
30. Luo, H., Chen, W., Ming, X., Wu, Y.: General differential fault attack on PRESENT and GIFT cipher with nibble. IEEE Access **9**, 37697–37706 (2021)
31. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The rebound attack: Cryptanalysis of reduced whirlpool and grøstl. In: FSE. Lecture Notes in Computer Science, vol. 5665, pp. 260–276. Springer (2009)
32. Min, X., Feng, T., Jiaqi, L.: Differential fault attack on GIFT. Chinese Journal of Electronics **30**(4), 669–675 (2021)

33. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Inscrypt. Lecture Notes in Computer Science, vol. 7537, pp. 57–76. Springer (2011)
34. Ross, S.M.: A First Course in Probability. Prentice Hall, Upper Saddle River, N.J., fifth edn. (1998)
35. Saha, D., Chowdhury, D.R.: Encounter: On breaking the nonce barrier in differential fault analysis with a case-study on PAEQ. In: CHES. Lecture Notes in Computer Science, vol. 9813, pp. 581–601. Springer (2016)
36. Sakiyama, K., Sasaki, Y., Li, Y.: Security of Block Ciphers - From Algorithm Design to Hardware Implementation. Wiley (2015)
37. Sun, L., Wang, W., Liu, R., Wang, M.: Milp-aided bit-based division property for ARX ciphers. Sci. China Inf. Sci. **61**(11), 118102:1–118102:3 (2018)
38. Sun, L., Wang, W., Wang, M.: Milp-aided bit-based division property for primitives with non-bit-permutation linear layers. IET Inf. Secur. **14**(1), 12–20 (2020)
39. Sun, S., Gérault, D., Lafourcade, P., Yang, Q., Todo, Y., Qiao, K., Hu, L.: Analysis of aes, skinny, and others with constraint programming. IACR Trans. Symmetric Cryptol. **2017**(1), 281–306 (2017)
40. Todo, Y.: Integral cryptanalysis on full MISTY1. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 9215, pp. 413–432. Springer (2015)
41. Todo, Y.: Structural evaluation by generalized integral property. In: Advances in Cryptology – EUROCRYPT. Lecture Notes in Computer Science, vol. 9056, pp. 287–314. Springer (2015)
42. Todo, Y.: Structural evaluation by generalized integral property. In: Advances in Cryptology – EUROCRYPT. Lecture Notes in Computer Science, vol. 9056, pp. 287–314. Springer (2015)
43. Todo, Y., Morii, M.: Bit-based division property and application to Simon family. In: FSE. Lecture Notes in Computer Science, vol. 9783, pp. 357–377. Springer (2016)
44. Vafaei, N., Zarei, S., Bagheri, N., Eichlseder, M., Primas, R., Soleimany, H.: Statistical effective fault attacks: The other side of the coin. IACR Cryptol. ePrint Arch. p. 642 (2022)
45. Wagner, D.A.: The boomerang attack. In: FSE. Lecture Notes in Computer Science, vol. 1636, pp. 156–170. Springer (1999)
46. Wang, G., Wang, S.: Differential Fault Analysis on PRESENT Key Schedule. In: Liu, M., Wang, Y., Guo, P. (eds.) 2010 International Conference on Computational Intelligence and Security, CIS 2010, Nanning, Guangxi Zhuang Autonomous Region, China, December 11-14, 2010. pp. 362–366. IEEE Computer Society (2010)
47. Wang, Q., Grassi, L., Rechberger, C.: Zero-Sum Partitions of PHOTON Permutations, Lecture Notes in Computer Science, vol. 10808, pp. 279–299. Springer (2018)
48. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Advances in Cryptology – ASIACRYPT. Lecture Notes in Computer Science, vol. 10031, pp. 648–678 (2016)

49. Zhang, F., Guo, S., Zhao, X., Wang, T., Yang, J., Standaert, F., Gu, D.: A framework for the analysis and evaluation of algebraic fault attacks on lightweight block ciphers. IEEE Trans. Inf. Forensics Secur. **11**(5), 1039–1054 (2016)
50. Zhang, F., Lou, X., Zhao, X., Bhasin, S., He, W., Ding, R., Qureshi, S., Ren, K.: Persistent fault analysis on block ciphers. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 150–172 (2018)
51. Zhang, F., Zhang, Y., Jiang, H., Zhu, X., Bhasin, S., Zhao, X., Liu, Z., Gu, D., Ren, K.: Persistent fault attack in practice. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(2), 172–195 (2020)
52. Zhao, X., Wang, T., Guo, S.: Fault-propagation pattern based DFA on SPN structure block ciphers using bitwise permutation, with application to PRESENT and printcipher (2011)

A Results on GIFT-128

The balanced bits from 5-round property is given below. Using a random nibble fault we get a total 80 balanced bits after 5 rounds. However the balanced bits are divided into two sets. If the active nibble belongs to $\{N_0, ..., N_{15}\}$ we get the following balanced bits

- $\mathcal{A}$ = 0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 16, 17, 18, 20, 21, 22, 24, 25, 28, 29, 32, 33, 34, 36, 37, 38, 40, 41, 44, 45, 48, 49, 50, 52, 53, 54, 56, 57, 60, 61, 64, 65, 66, 68, 69, 70, 72, 73, 76, 77, 80, 81, 82, 84, 85, 86, 88, 89, 92, 93, 96, 97, 98, 100, 101, 102, 104, 105, 108, 109, 112, 113, 114, 116, 117, 118, 120, 121, 124, 125.

If the active nibble belongs to $\{N_{16}, ..., N_{31}\}$ we get the following balanced bits

- $\mathcal{B}$ = 0, 1, 4, 5, 8, 9, 10, 12, 13, 14, 16, 17, 20, 21, 24, 25, 26, 28, 29, 30, 32, 33, 36, 37, 40, 41, 42, 44, 45, 46, 48, 49, 52, 53, 56, 57, 58, 60, 61, 62, 64, 65, 68, 69, 72, 73, 74, 76, 77, 78, 80, 81, 84, 85, 88, 89, 90, 92, 93, 94, 96, 97, 100, 101, 104, 105, 106, 108, 109, 110, 112, 113, 116, 117, 120, 121, 122, 124, 125, 126.

While the intersection of these two sets contains the following 64 balanced bits

- $\mathcal{A} \cap \mathcal{B}$ = {0, 1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 25, 28, 29, 32, 33, 36, 37, 40, 41, 44, 45, 48, 49, 52, 53, 56, 57, 60, 61, 64, 65, 68, 69, 72, 73, 76, 77, 80, 81, 84, 85, 88, 89, 92, 93, 96, 97, 100, 101, 104, 105, 108, 109, 112, 113, 116, 117, 120, 121, 124, 125 }.

Thus we have 64 balanced bits after 5 round in fault invariant setting, i.e., any random active nibble at the input results to 64 output bits. Moreover, input of each sbox in the 6-th round contains two balanced bits, namely the first and second input bit.

B Sbox Division Trail Algorithm

C PRESENT Partial Subkeys Recovery of the Last Two Rounds

Here we discuss partial recovery of the last two round keys. To recover the partial subkeys of the last two rounds we use our random nibble fault

Algorithm 2 SboxDivisionTrail ($k = (k_0, k_1, \cdots, k_{n-1})$ [48]

```
1: 𝕊_k = {a | a ⪰ k}
2: F(X) = {π_a(x) | a ∈ 𝕊_k}
3: 𝕂 = φ
4: for u ∈ F_2^n do
5:     if π_u(x) ∩ F(X) ≠ φ then
6:         Flag = True
7:         R = φ
8:         for v ∈ 𝕂 do
9:             if v ⪰ u then
10:                Flag = False
11:            else if u ⪰ v then
12:                R = R ∪ {v}
13:        if Flag = True then
14:            𝕂 = 𝕂 \ R
15:            𝕂 = 𝕂 ∪ {u}
   return 𝕂
```

distinguisher for 4 and 5 rounds. For this we construct an active nibble at the input of R^{25} by injecting 15 faults. From Table 3, we can observe that all the 64 bits at R^{28} and the first 4 bits of R^{29} become balanced. Now take all possible key values of the nibbles N_j^{30} for $j \in \{0, 4, 8, 12\}$ and partially decrypt the corresponding nibbles of the ciphertext. Due to the propagation of the division property, we get balanced bits at N_0^{29} (from Table 3). We take the xorsum of the partially decrypted ciphertexts and check for which key nibble values the xorsum at N_0^{29} becomes 0. We take those as possible key choice for the nibbles $\{0, 4, 8, 12\}$ in R^{30} and proceed in the next step. Upto this point we get 2^{12} key choices for the nibbles as we have 4 bit filters at R^{29}. In the next step we take all the key values of the nibbles N_j^{29} for $j \in \{0, \cdots, 3\}$ and partially decrypt one more round. In the output of R^{28} we check whether the xorsum of the decrypted ciphertexts becomes 0 or not. As 16 bits at R^{28} becomes balanced hence the key choices of the nibbles N_j^{30} for $j \in \{0, 4, 8, 12\}$ and N_j^{29} for $j \in \{0, \cdots, 3\}$ reduces to 2^{12}. Thus total 20 bit subkey of the last two rounds can be recovered using this attack. The algorithm for the last round key recovery of this attack is given in Appendix C.1 while the pictorial view of the full attack is given in Figure 6 (red-colored bits are the balanced bits).

Complexity of the attack: The data complexity for this case is 2^{16} as we have used 16 plaintext-ciphertext pairs to recover the partial subkeys. In this case also instead of decrypting the whole ciphertext at once we paritally decrypt the 0-th $\mathcal{QR}$ group i.e. the nibbles N_j^{30} for $j \in \{0, 4, 8, 12\}$ in the last round and N_j^{29} for $j \in \{0, 1, 2, 3\}$ in the second last. In the last round we have 16 bit keys and 4 bit filters for this $\mathcal{QR}$ group. Hence the number of keys that passes through the filter is 2^{12}. For each of the key in the last round we take all possible values in

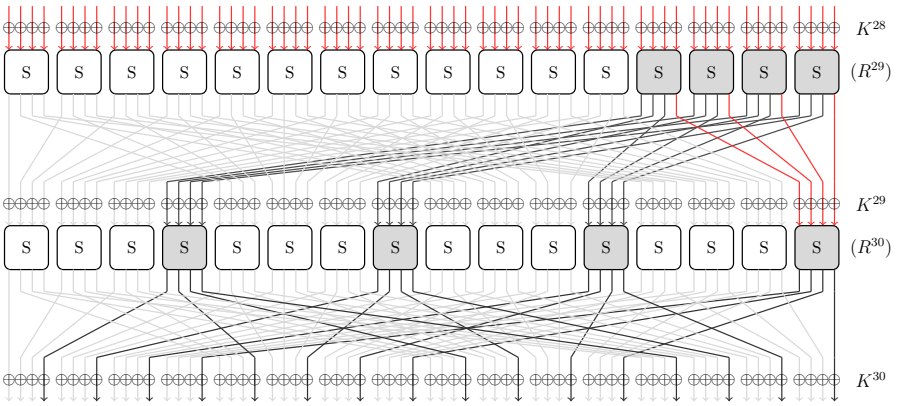

Fig. 6: Partial key recovery attack on PRESENT

the second last round and check the zero-sum. Hence the complexity for this group is $2^{12} \times 2^{16} = 2^{28}$ sbox inversions and the number of keys that passes through the filer is 2^{12} for 0-th $\mathcal{QR}$ group.

C.1 Last Round Key Recovery of PRESENT

Algorithm 3 RoundKeyRecoveryPRESENT

Input: A list of ciphertexts $\mathcal{L}_c$
Output: A list of keys $\mathcal{L}_k$
Initialize 16 empty lists $\mathcal{L}_k^i$ for $i \in \{0, \cdots, 15\}$ ▷ for each nibble
Prepare sixteen lists $\mathcal{L}_c^i$ for $i \in \{0, \cdots, 15\}$ of 4 bit values from $\mathcal{L}_c$
for $i = 0$ to 15 **do** ▷ for each nibble
    for $k^{30} = 0$ to 15 **do**
        $S = 0$
        for $c \in \mathcal{L}_c^i$ **do**
            $S = S \oplus (R^{30})^{-1}(c, k^{30})$
        if S & `0xf= 0x0` **then** ▷ check the input bits of each nibble
            $\mathcal{L}_k^i = \mathcal{L}_k^i \cup \{k^{30}\}$
Construct full round key for R^{30} from $\mathcal{L}_k^i$ for $i \in \{0, \cdots, 15\}$ and store in to $\mathcal{L}_k$
return $\mathcal{L}_k$