SBCMA: Semi-Blind Combined Middle-Round Attack on Bit-Permutation Ciphers with Application to AEAD Schemes

Xiaolu Hou, Jakub Breier and Shivam Bhasin

***Abstract*—Side-channel and fault injection attacks are well-researched topics within physical security of cryptographic implementations. They can reduce the complexity of the key retrieval to trivial numbers. It was shown that their combination is even more effective, for example in breaking redundancy countermeasures.**

In this paper, we present the first semi-blind combined attack that allows secret key retrieval in unknown plaintext and ciphertext scenario. SBCMA – Semi-Blind Combined Middle-round Attack is aimed at bit-permutation ciphers, such as GIFT-128 which serves as a case study for this work. On average, it can recover GIFT-128 master key with 92.17 encryptions and 91.17 faults. For GIFT-128 based NIST LWC Round 2 candidates, SBCMA requires 13.79 sessions and 94.32 faults on average.

***Index Terms*—Side-Channel Attacks, Fault Injection Attacks, Combined Attacks, Blind Attacks, Symmetric Cryptography**

INTRODUCTION

Hardware attacks constitute a powerful attack vector against cryptographic implementations on embedded devices. Since the inception of side-channel attacks (SCA) in 1996 [1] and fault injection attacks (FIA) in 1997 [2], these two branches have become the major research direction in this field. Current trends such as Internet-of-Things and Edge computing make a push towards transferring the computations from the cloud to the embedded chips. As these systems need to communicate with the outside world, the need of low computational power cryptography becomes more evident, resulting in a branch called *lightweight cryptography*. Multitude of algorithms have been proposed to date, pushing the limits of lowering the resource requirements with every iteration [3]. Bit permutation-based ciphers, such as PRESENT [4] and GIFT [5] have become popular due to their permutation layer, which allows zero-cost in hardware and also very efficient implementation in software [6], [7]. The lightweight cryptography standardization efforts by National Institute of Standards and Technology (NIST)[1] show a great interest in GIFT-based candidates, resulting in 4 out of 32 Round-2 candidates based on this cipher, including one of the ten finalist. Apart from classical cryptanalysis, it is especially important to analyze these proposals w.r.t. hardware attacks due to their intended deployment.

Resistance of PRESENT and GIFT ciphers against SCA and FIA have been explored in series of works in the past (e.g. [8], [9], [10]), but their resistance to physical attacks within AEAD setting is not yet explored. Several AEAD use plaintext or ciphertext masking. It is well known that majority of SCA and FIA operate under a known/chosen plaintext/ciphertext setting, making those attacks impossible when plaintext/ciphertext masking is in place. In this paper, we propose SBCMA on GIFT-128 which leads to full key recovery with a combination of side-channel and fault injection, while having no knowledge of plaintext/ciphertext. Our attack targets bitslice software implementations – these are the most efficient implementations on general-purpose 32-bit devices.

Our contribution.

1) We propose the first *semi-blind* combined physical attack on block ciphers with bit permutations – SBCMA: Semi-Blind Combined Middle-round Attack.
2) We show an application of SBCMA on GIFT-based AEAD schemes proposed for NIST LWC competition. Attack complexities on these schemes are stated in Table I.
3) We provide simulation results for SBCMA with different values of SNR.
4) We provide experimental results of the SBCMA on ARM Cortex-M0 device.
5) We provide a discussion on countermeasures and how SBCMA can be extended to reverse engineering of secret ciphers.

Depending on the attacker ability, we have considered two attack types, SBCMA A and SBCMA B (further detailed in Section III-A).

Attack type		Complexity: Sessions	Complexity: Enc./session	No. of faults
SBCMA A	Known fault mask	13.79	8	94.32
	Chosen fault mask	11.62		76.96
SBCMA B	Known fault mask	20.20		145.58
	Chosen fault mask	14.51		100.08

TABLE I
ATTACK COMPLEXITY (NUMBER OF AEAD SESSIONS/LEAKAGE TRACES, AND NUMBER OF ENCRYPTIONS PER SESSION UNDER ATTACK) AND NUMBER OF FAULTS NEEDED FOR MASTER KEY RECOVERY WITH BCMA ON NIST LWC ROUND 2 CANDIDATES BASED ON GIFT-128.

X. Hou is with Faculty of Informatics and Information Technologies, Slovak University of Technology, Slovakia. E-mail: houxiaolu.email@gmail.com

J. Breier is with Silicon Austria Labs, Graz, Austria. E-mail: jbreier@jbreier.com

S. Bhasin is with Temasek Laboratories, Nanyang Technological University, Singapore. E-mail: sbhasin@ntu.edu.sg

Authors acknowledge the support from the Singapore National Research Foundation ("SOCure" grant NRF2018NCR-NCR002-0001 – www.green-ic.org/socure). This work was partially supported by the European Regional Development Fund (ITMS code: 313021W404).

[1]https://csrc.nist.gov/Projects/lightweight-cryptography

Roadmap. The rest of the paper is organized as follows. In Section I we provide background on physical attacks and discuss related work. In Section II we detail the specifications of GIFT-128 and the specific properties of its bit-permutation operation we target. In Section III we present SBCMA with application to GIFT-128. Section IV shows the simulation and the experimental results. Section V discusses how SBCMA can be applied to LWC candidates. Section VI provides discussion on mitigation and application of SBCMA to reverse engineering. We finally conclude with Section VII.

I. Background

In this section, we outline general background on side-channel attacks, fault injection attacks, and combined attacks. Later, we detail the noted related work and compare it with our proposal.

A. Physical Attacks

a) Side-channel analysis: Side-Channel Analysis (SCA) attacks target implementations of cryptographic primitives in a *passive* manner. They exploit the possibility of observing the physical characteristics of a device during the encryption/decryption process [11]. The attacker obtains so-called *side-channel information* that can be in a form of execution time, power consumption, electromagnetic emanation (EM), cache leakage, etc. This information is then used to reveal information related the secret key used during the computation.

In this work, we focus on SCA attacks that use either power consumption leakage or EM leakage for the analysis. These can be generally divided into three categories:

- Simple Side-Channel Attacks (SSCA) aim at information recovery with a one or few number of side-channel traces. The attacker aims at recognising secret dependent patterns in side-channel trace to determine its value, example conditional multiply in squareand multiply operation of RSA when key bit is 1. In our methodology, we assume the attacker can use SSCA to determine the executed operations of cryptographic algorithm by observing the trace.
- Differential Side-Channel Attacks (DSCA) operate on higher number of side-channel traces compared to SSCA hwere an attacker use statistical means to find dependency between hypothetical leakage based on key hypothesis and the assumed leakage model (like Hamming Weight model) and actual traces. The correct key hypothesis maximizes statistical score. DSCA are more resistant to noise compared to SSCA.
- Side-Channel Assisted Differential Plaintext Attacks (SCADPA [12]) uses side-channel leakage to aid differential cryptanalysis [13]. Generally, the attacker encrypt two known plaintext and tracks its propagation in middle rounds through side-channels. The middle round difference is computed by subtracting side-channel traces from the two plaintext. The input and middle round differences can be used in differential cryptanalysis to recover the key. This attack, initially introduced against bit-permutation based ciphers, was recently extended to SPN ciphers, being capable of targeting intermediate rounds [14].

b) Fault injection attacks: Unlike SCA, Fault Injection Attacks (FIA) are *active* attacks on cryptographic implementations. They utilize various techniques, such as lasers, electromagnetic emanation, voltage glitching, to affect the intermediate values during the computation [15]. After a fault is introduced in the computation, various analysis methods can be then used to retrieve the secret key information. Most of the current block ciphers have been shown vulnerable to FIA, especially to differential fault analysis (DFA) [16], which is a well-researched method in this field.

Generally, the attacker introduces a fault in an intermediate value, and then observes the cipher behavior under this fault. There are different varieties of faults: transient/permanent; data/control flow; stuck-at/random/precise; bit/byte/word faults. Each of these can be exploited in a different manner – for example, DFA exploits data fault propagation through non-linear cipher components [16], statistical ineffective fault attacks exploit faults that do not change the data values [17], persistent fault analysis exploits permanent faults [18], etc. For more details on different fault analysis methods, we refer interested reader to [19].

c) Combined physical attacks: Few proposals also explore the possibility of combining SCA and FIA to propose strong attacks against cipher implementations. One of first proposal of a combined attack was differential behavioral analysis [20], which utilized a combination of SCA and safe-error attack. Assuming stuck-at fault model, it observes if the fault alters the side-channel behavior of the computation to derive the key. In the same year, a combined attack on RSA was proposed in [21], which combines "stuck-at-0" with SCA to overcome masked implementations. Later, several proposals have emerged proposing combination of SCA and FIA under various setting. These include, but not limited to, stuck-at faults with DSCA to break masking comutermeasure [22], fault sensitivity with collision correlation [23], differential fault analysis (DFA) assisted by SCA targeting specific design choices like bit permutations [24], DFA with SCA to break fault hardened implementations [25], [26], and combination of SIFA and SCA [27].

B. Related Work

In this part, we outline the noted related work – combined physical attacks and middle round attacks. High-level comparison of different types of attacks to SBCMA is stated in Table II.

The commonly used attacks like DSCA, SSCA and most fault-based attacks require access to plaintext/ciphertext. Moreover their application to middle rounds is not trivial as the complexity goes beyond brute force when applied only a couple of rounds deeper.

A couple of advanced fault attacks were proposed in recent literature. Blind fault attacks (BFA [30]) can operate without precise knowledge or plaintext and ciphertext. However, they need to know if the injected fault has changed the output ciphertext or not. Nevertheless, it needs as high as $\approx$ 160k

Attack	Unknown plaintext	Unknown ciphertext		Targets middle round	Fault model
		Unkn. value	Unkn. change		
SSCA [28]	✗	✗	✗	✗	–
DSCA [28]	✗	✗	✗	✗	–
Blind SCA [29]	✓	✓	✓	✓	–
Blind FIA [30]	✓	✓	✗	✓	Stuck-at
SIFA [17]	✓	✓	✗	✗	Ineffective
Blind SIFA [27]	✓	✓	✓	✗	Ineffective
FTA [31]	✓	✓	✗	✓	Stuck-at
SBCMA (this paper)	✓	✓	✓	✓	Random/chosen

TABLE II
COMPARISON OF DIFFERENT PHYSICAL ATTACKS TO SBCMA.

faults [32] according to the recently proposed improvements, which is $3\times$ lower than 480k faults in the original proposal, both targeting AES-128. Statistical ineffective fault attack [17] can also operate without knowledge of ciphertext and ineffective faults, but it does not apply on middle rounds. The fault model required is a strong stuck-at model. Fault template attacks (FTA) [31] can target middle rounds with the help of detailed profiling of fault behavior. It also works under strong stuck-at fault and attacker needs information if the faults resulted in change of output ciphertext value or not.

Blind SCA is another related area, utilizing leakage measurement at several points of interest during the algorithm computation and comparing the distributions obtained from the leakage [29]. The original attack was later improved in [33] and [34]. These attacks do not assume knowledge of plaintext and ciphertext. As blind SCA are statistical attacks in their nature, the number of required traces is high – around 10k traces in the lowest noise scenario according to [34].

SBCMA as proposed in this paper also targets middle rounds without knowledge of plaintext and ciphertext. In this regard, SBCMA is direct competitor to blind SCA with certain advantages and disadvantages. Blind SCA has high SNR requirements and in particular precise knowledge of points of interest in the side-channel trace must be known. SBCMA is comparatively has a better tolerance to SNR. More specifically, the lowest noise reported in blind SCA is $\sigma = 0.5$ when simulating the operation for a single AES Sbox on 8-bit microcontroller, while SBCMA can tolerate noise up to $\sigma = 0.73$ for 32-bit bitslice variables, keeping the number of executions lower by two orders of magnitude.

II. ATTACK TARGETS

SBCMA aims at the bit-permutation operation with certain properties. We first provide overview of block cipher GIFT which uses such bit-permutation in its diffusion function. Later, we give the details of this particular class of bit-permutation in Section II-B. Table III summarizes attack complexities of SBCMA on GIFT-128.

A. GIFT-128

In this section we follow the terminologies from [5] and describe the specifications of GIFT-128. GIFT-128 consists of 40 rounds, where each round consists of three operations: **SubCells**, **PermBits** and **AddRoundKey**. The cipher state can be expressed as 32 nibbles $S = b_{127}||b_{126}||\dots||b_0 = \omega_{31}||\omega_{30}||\dots||\omega_1||\omega_0$.

Attack type		Round key recovery		Master key recovery	
		Complexity	# faults	Complexity	# faults
SBCMA A	Known mask	89	88	92.17	91.17
	Chosen mask	65	64	71.46	70.46
SBCMA B	Known mask	145.78	144.78	146.194	145.194
	Chosen mask	97	96	99.03	98.03

TABLE III
ATTACK COMPLEXITY (NUMBER OF ENCRYPTIONS/LEAKAGE TRACES) AND NUMBER OF FAULTS NEEDED FOR KEY RECOVERY WITH SBCMA ON GIFT-128.

a) SubCells: operation applies a $4-$bit invertible Sbox to each nibble of the cipher state:

$$\omega_i \leftarrow GS(\omega_i),\ \forall i \in \{0, 1, \dots, 31\}.$$

The specification of the GIFT Sbox is $1a4c6f392db7508e$.

b) PermBits: maps bit i to another bit according to the following formula

$$P(i) = 4\left\lfloor \frac{i}{16} \right\rfloor + 32\left(\left(3\left\lfloor \frac{i \mod 16}{4} \right\rfloor + (i \mod 4)\right) \mod 4\right) + (i \mod 4).$$

c) AddRoundKey: consists of adding the round key and the round constant. A 64-bit round key RK is extracted from the key state and partitioned into two $32-$bit words $RK = U||V = u_{31}\dots u_0||v_{31}\dots v_0$. U and V are then XORed to $\{b_{4i+2}\}$ and $\{b_{4i+1}\}$ of the cipher state respectively:

$$b_{4i+2} \leftarrow b_{4i+2} \oplus u_i,\ b_{4i+1} \leftarrow b_{4i+1} \oplus v_i,\ \forall i \in \{0, \dots, 31\}$$

B. Target Bit-Permutation

Bit-permutation operation has been used as a building block in many SPN cipher designs. We are interested in a particular class of bit-permutation operation, which satisfies the following:

- Each Sbox is a $b-$bit permutation.
- Sboxes in each round can be divided into groups of m in two ways – the Quotient and Remainder groups.
- The bit-permutation operation can be viewed as a group mapping, which takes the outputs from one Quotient group as the input and outputs $m \times b$ bits which will be XORed with the round key and used as inputs for one Remainder group in the next round.
- The input bits of an Sbox in round $i+1$ come from m distinct Sboxes in one Quotient group in round i.

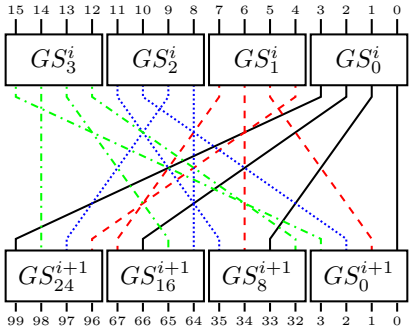


Fig. 1. Grouping of bits and nibbles in GIFT.

- The output bits of an Sbox in round i go to m distinct Sboxes in one Remainder group in round $i+1$.

For both GIFT and PRESENT cipher designs, $m = b = 4$. Let us number the Sboxes in round i of the cipher as $SB_0^i, SB_1^i, \ldots, SB_s^i$, where $s = n/4$ and n is the block size of the cipher. Then, for both PRESENT and GIFT, the Quotient groups and Remainder groups, Qx and Rx, are defined as

- $Qx = \{SB_{4x}, SB_{4x+1}, SB_{4x+2}, SB_{4x+3}\}$;
- $Rx = \{SB_x, SB_{q+x}, SB_{2q+x}, SB_{3q+x}\}$,

where $q = s/4, 0 \leq x \leq q-1$. For example, Figure 1 shows the mapping from Quotient group $Q0$ in round i to Remainder group $R0$ in round $i+1$ for GIFT-128, where GS_j^i denotes the jth Sbox in the ith round.

III. SBCMA – Semi-Blind Combined Middle-Round Attack

In this section we present SBCMA, a combined side-channel and fault attack on the middle-round of bit-permutation based SPN with target bit-permutation as described in Section II-B.

A. Methodology

SBCMA injects faults in a middle round of the cipher and uses side-channel leakage in the following two rounds to recover the round key. The main strength of this attack is **zero knowledge** of inputs and outputs of the cipher. The attacker can recover the secret key just by injecting faults and observing the side-channel leakage.

a) Attacker assumptions.: We assume that the attacker:

- has no control and no knowledge of plaintext and ciphertext but she can repeat the encryption of the same plaintext;
- can inject a fault in an Sbox input in an intermediate round;
- can either choose the fault mask or has the knowledge of the fault mask in case of the random fault model (we note that this fault mask is the input difference of the Sbox under attack);
- can measure the side-channel leakage (power/EM) during the two subsequent rounds after the fault injection.

b) Attack steps.: Here, we assume the order of operations in each round of the target SPN is as follows: Sbox, bit-permutation, round key addition[2]. The side-channel leakage we measure in our attack corresponds to the intermediate value right after bit-permutation and before key addition. We denote this intermediate value as I_i for round i. Consequently, the traces we are interested in is the parts of the traces corresponding to bit-permutation and round key addition. And we denote this part of the trace as L_{I_i} for round i. Keeping the notations from Section II-B. To recover round key at a middle round i, we have the following steps:

1) Fix a Quotient group at round i, say Q, to attack.
2) Run the encryption and measure side-channel leakages L_{I_i} and $L_{I_{i+1}}$.
3) Run the encryption with repeated plaintext and inject a fault in the input of one Sbox from Q in round i. Measure side-channel leakages L'_{I_i} and $L'_{I_{i+1}}$ after the fault injection.
4) Use $L_{I_i} - L'_{I_i}$ to determine the output difference of the faulted Sbox in round i, which also gives the input differences of faulted Sboxes (in the corresponding Remainder group) in round $i+1$.
5) Use $L_{I_{i+1}} - L'_{I_{i+1}}$ to determine the changes in the faulty inputs of Sbox in round $i+2$, which gives the output changes of Sboxes in round $i+1$. For different attacker assumptions (leakage model, device architecture, implementation), the exact information on differentials can vary. We analyze two possible scenarios:
   a. Attacker can observe if input changes or not in each Sbox;
   b. Attacker can observe if input changes or not in each Quotient group.
6) Reduce the input candidates for Sboxes in rounds i and $i+1$ using results from steps 3 and 4.
7) Repeat steps 3-6 for Quotient group Q until the inputs of Sboxes in Q and those in the corresponding Remainder group are recovered.
8) Repeat steps 2-7 for other Quotient groups until the round key for round i is recovered.

We refer to the attacks with conditions a, b separately, as SBCMA A and SBCMA B. We note that in steps 4) and 5), only single trace differences are considered.

B. Application of SBCMA to GIFT-128

In this part, we first detail the attack to recover the first 8 bits of a round key for GIFT-128 at any middle round i. Then, we give estimates for attack complexity to recover the master key for different attacker capabilities. We recall that the intermediate value that we would like to measure the leakage is the state of the cipher right after bit-permutation and before key addition. We use L_{I_i} to denote the leakage for this intermediate value in round i. The attack steps are as follows:

1) Run the encryption and measure the side-channel leakage L_{I_i} and $L_{I_{i+1}}$.
2) Run the encryption with repeated plaintext and inject a fault in the input of Sbox GS_0^i at round i. The fault mask is an input difference of Sbox GS_0^i.
3) Measure the side-channel leakages L'_{I_i} and $L'_{I_{i+1}}$ after the fault injection.

[2]For ciphers with round key addition before the Sbox computation, the attack recovers round key $i+1$.

Algorithm 1: SBCMA attack steps for recovery of the first 8 bits of round key i.

Data: i : target round for round key recovery; $Q0$: the first Quotient group in round i; $R0$: Remainder group in round $i+1$ corresponding to $Q0$

while *inputs for* $Q0$ *and* $R0$ *are not recovered* **do**
- // $R0$ consists of: GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}, GS_{24}^{i+1}
- **for** *Sbox* GS *in* $Q0$ **do**
  - run encryption, measure L_{I_i} and $L_{I_{i+1}}$;
  - run encryption with repeated plaintext;
  - inject one fault in the input of GS;
  - // fault mask is input difference for GS
  - measure L'_{I_i} and $L'_{I_{i+1}}$;
  - calculate $L_{I_i} - L'_{I_i}$;
  - // this gives output difference for GS and input differences for GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}, GS_{24}^{i+1}
  - calculate $L_{I_{i+1}} - L'_{I_{i+1}}$;
  - // this gives information on output changes for GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}, GS_{24}^{i+1}
  - reduce input candidates for GS and $R0$;
  - // using the input and output differences information gained in the previous steps

recover inputs for $Q0$ and $R0$;

recover the first 8 bits of round key i;

4) Use $L_{I_i} - L'_{I_i}$ to determine if there are changes in the inputs of the four faulted Sboxes in round $i+1$, $GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}, GS_{24}^{i+1}$, which gives the output difference of Sbox GS_0^i and also the input differences of Sboxes $GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}, GS_{24}^{i+1}$.
5) Use $L_{I_{i+1}} - L'_{I_{i+1}}$ to gain information on the output differences of $GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}$ and GS_{24}^{i+1}.
6) With the output difference from step 4 and the input difference from step 2, reduce input candidates for GS_0^i.
7) With the input difference from step 4 and the output difference from step 5, reduce input candidates for $GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}, GS_{24}^{i+1}$.
8) Repeat steps 1-7 for the other Sboxes in the same Quotient group as GS_0^i, namely GS_1^i, GS_2^i, GS_3^i.
9) Repeat steps 1-8 until the inputs of Sboxes $GS_0^i, GS_1^i, GS_2^i, GS_3^i, GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}, GS_{24}^{i+1}$ are recovered to get the first 8 bits of ith round key.

To illustrate step 4, for example, if the side-channel leakage indicates there is a change in the inputs of GS_0^{i+1} and GS_{16}^{i+1} and there is no change in the inputs of $GS_8^{i+1}, GS_{24}^{i+1}$, then we can conclude that the output difference of GS_0^i is $0x5$ (0101) and the input differences of GS_0^{i+1} and GS_{16}^{i+1} are $0x1$ (0001) and $0x4$ (0100) respectively. In Section IV-B1, we simulate the side-channel leakages when fault is injected in input of GS_0^i. Lines in Figure 4 indicate the maximum absolute difference for the part of $L_{I_i} - L'_{I_i}$ that corresponds to the leakage for first bits of inputs of $GS_0^{i+1}, GS_{16}^{i+1}, GS_8^{i+1}$, and GS_{24}^{i+1}[3]. By the design of the permutation, when a fault is injected in input of GS_0^i, this fault does not propagate to first bits of $GS_{16}^{i+1}, GS_8^{i+1}$, or GS_{24}^{i+1} inputs. It can only influence the first bit of GS_0^{i+1} input through the first bit of GS_0^i output. Black (resp. gray) line simulates the case when the fault propagates (resp. does not propagate) to the first bit of GS_0^i output. We can see that when the SNR is big enough (≥ 0.47), we can successfully distinguish the two cases and conclude if there is a change in the first bit of GS_0^i output, and equivalently, if there is a change in the first bit of GS_0^{i+1} input. Experimental results for this setting are shown in Section IV-C1.

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

TABLE IV
GIFT SBOX OUTPUT DIFFERENCE TABLE - COLUMNS CORRESPOND TO INPUT DIFFERENCES, ROWS CORRESPOND TO VALUES OF INPUTS, AND ENTRIES ARE OUTPUT DIFFERENCES.

Step 5 can be achieved in a similar way, for example, the change/no-change in Sboxes $GS_0^{i+2}, GS_8^{i+2}, GS_{16}^{i+2}, GS_{24}^{i+2}$ correspond to change/no-change in the 1st, 2nd, 3rd, and 4th bits of GS_0^{i+1} output, respectively. In Section IV-B2, we simulate the side-channel leakages when fault is injected in input of GS_0^i. For the simulation, we consider SBCMA B scenario, where we assume the attacker can only observe the output changes of each Quotient group in round $i+1$. Lines in Figure 5 indicate the maximum absolute difference for the part of $L_{I_{i+1}} - L'_{I_{i+1}}$ that corresponds to the leakage for first bits of outputs for Sboxes in Quotient group 0 in round $i+1$. Since the fault was injected in GS_0^i, only GS_0^{i+1} and GS_8^{i+1} outputs would be influenced from this Quotient group. Black (resp. gray) line simulates the case when the fault propagates (resp. does not propagate) to the first bit of GS_0^{i+1} or GS_8^{i+1} output. We can see that when the SNR is big enough (≥ 0.47), we can successfully distinguish the two cases and conclude if there is a change in the first bit of GS_0^{i+1} or GS_8^{i+1} output. Experimental results are also provided for this setting in Section IV-C2.

The fault propagation is further illustrated in Figure 2.

1) Recovery of Sbox Inputs for Round i: First, we note that the information we have for recovering the inputs of Sboxes $GS_0^i, GS_1^i, GS_2^i, GS_3^i$ are the input and the output differences, obtained from known fault masks and side channel leakage (step 4). Table IV gives the Output Difference Table for GIFT Sbox, where the columns correspond to input differences, rows

[3]The implementation enables such leakage information. Details are explained in Section IV-B1.

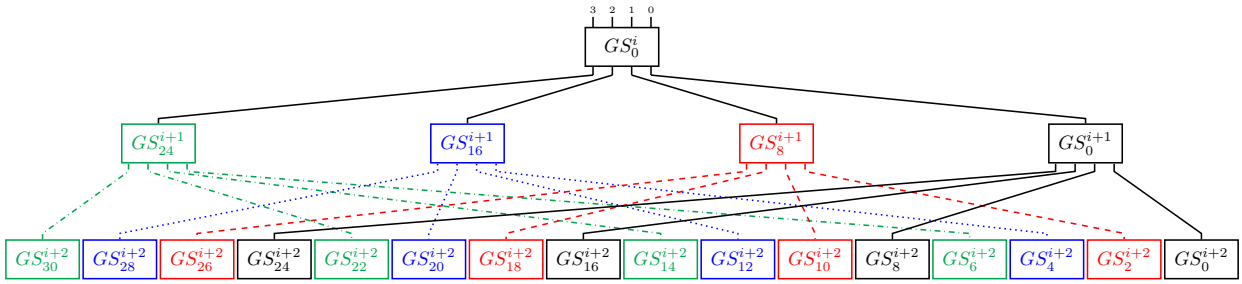


Fig. 2. Fault propagation after 2 rounds of GIFT-128. By observing the input change of Sbox in round $i+1$ (resp. $i+2$), attacker can gain knowledge of Sbox output difference in round i (resp. $i+1$).

No. of pairs m	Total cases	Successful cases	Success Rate	Probability exactly m pairs are needed
2	105	78	74.29	74.29
3	455	420	92.31	18.02
4	1365	1340	98.17	5.86
5	3003	2997	99.80	1.63
6	5005	5005	100.00	0.2

TABLE V
DISTINGUISHING GIFT SBOX INPUT FROM THE KNOWN FAULT MASK AND OUTPUT DIFFERENCE PAIRS. THE FIRST COLUMN IS THE NUMBER OF KNOWN FAULT MASK AND OUTPUT DIFFERENCE PAIRS. SECOND COLUMN IS THE TOTAL NUMBER OF POSSIBLE CASES FOR m PAIRS. THIRD COLUMN IS THE NUMBER OF PAIRS WHICH CAN UNIQUELY IDENTIFY EACH INPUT OF GIFT-SBOX. FOURTH COLUMN IS THE PERCENTAGE OF SUCH PAIRS. FIFTH COLUMN IS THE PROBABILITY THAT EXACTLY m PAIRS OF KNOWN FAULT MASK AND OUTPUT DIFFERENCE ARE NEEDED TO RECOVER THE GIFT SBOX INPUT.

correspond to input values and entries are output differences. More specifically, an entry x at cell i, j is the output difference of value i with input difference j, i.e. $GS(i) \oplus GS(i \oplus j) = x$. To recover the Sbox input, we need to find a sub-table with the same number of rows and a subset of the columns such that all the rows are distinct in this sub-table. The columns in this sub-table give us the combination of fault masks that can help us recover the Sbox input.

Chosen fault mask. By an exhaustive search, we found that the smallest number of columns needed is **2** and there are 78 such choices. For example, if we look at the columns 1 and 2 of Table IV, all the rows are distinct. Thus, by observing the two output differences corresponding to fault masks 1 and 2, the attacker can uniquely identify the input of the Sbox. So, the number of chosen faults needed to recover the Quotient group inputs in round i, $GS_0^i, GS_1^i, GS_2^i, GS_3^i$, is **8**.

Random fault with known fault mask. In Table V we summarize the results for different number of input (known fault mask) and output (step 4) difference pairs. The first column indicates the number, m, of input/output difference pairs. The second column is the total number of possible combinations of m input/output difference pairs. As we have in total 15 possible input differences, the total number of combinations of input/output difference pairs is 15 choose m. The third column gives us the number of successful cases – combinations that can help an attacker uniquely identify the input value of the Sbox. This was calculated by the observation mentioned above – we construct sub-tables of Table IV and check if the rows are all distinct. Fourth column lists the percentage of successful cases among the total cases – obtained by the value in column three divided by the value in column two. The fifth column shows the probability that the attacker needs exactly m pairs of known fault masks and output differences to uniquely identify the Sbox input. To give an example on how these values are obtained, we detail the calculations for $m = 3$. First, we notice that column four actually gives us the probability that $\leq m$ pairs needed for input recovery. The probability that "exactly 3 pairs are needed" is equal to the probability that "at most 3 pairs are needed" *minus* the probability that "at most 2 pairs are needed" i.e.

$$Pr[m = 3] = Pr[m \leq 3] - Pr[m \leq 2] = 0.9231 - 0.7429 = 0.1802$$

We also note that as $Pr[m \leq 6] = 1$, the probability that 7 or more pairs are needed is 0. So, on average

$$\sum_{m=2}^{6} m \times \text{Prob(exactly } m \text{ pairs are needed)} = \mathbf{2.34}$$

faults are needed to recover the input of one Sbox. On average, number of random faults needed to recover the Quotient group inputs in round i is **9.36**.

2) Recovery of Sbox Inputs for Round $i+1$: To recover the inputs of Sboxes $GS_0^{i+1}, GS_8^{i+1}, GS_{16}^{i+1}, GS_{24}^{i+1}$, we use the individual input differences, which are obtained by side-channel leakage in Step 4) (Section III-B). We note that with the attack setting, not all the input differences are achievable. In particular, we assume the attacker can attack one Sbox input in round i at once. For example, when one of $GS_0^i, GS_1^i, GS_2^i, GS_3^i$ is under fault attack, the possible input masks for GS_0^{i+1} are $0x1, 0x2, 0x4, 0x8$, respectively (see Figure 1).

The output difference information depends on the attacker abilities as mentioned in Section III-A. In the following, we analyze the attack complexities for both SBCMA A and SBCMA B scenarios. Recall scenario A means the attacker can observe if input changes or not in each Sbox in round $i+2$; and scenario B means the attacker can observe if input changes or not in each Quotient group in round $i+2$.

a) SBCMA A: In case the attacker can observe the input change in each Sbox in round $i+2$, she can get the Sbox output difference for each Sbox in round $i+1$ (see Figure 2).

Chosen fault mask. As discussed in Section III-B1, we need on average 2 faults for each Sbox input recovery in round i and there are 78 choices for the fault masks. When considering the whole Quotient group Q0 in round i, there are 78^4 possible combinations of those fault masks. Each fault mask propagates to input masks for each Sbox from Remainder group R0 in round $i+1$. By exhaustive search we found that 1377 of them can help us recover the inputs of R0 in round $i+1$. For example, the combination of fault masks $0x7$ and $0xa$; $0x7$ and $0x8$; $0x2$ and $0x7$; $0x0$ and $0x7$ in the input of $GS_0^i, GS_1^i, GS_2^i, GS_3^i$, respectively, can recover the inputs of

all 8 Sboxes (4 in Q0 from round i, 4 in R0 from round $i+1$). Thus, for chosen fault masks we need **8** faults to recover 8 bits of the round key and **64** faults for recovery of the full round key.

Random fault with known fault mask. With random fault, we can assume the four input masks, $0x1, 0x2, 0x4, 0x8$, of GS_0^{i+1} appear randomly with the same probability. By exhaustive search we find that in half of the cases, two out of the four input masks can recover GS_0^{i+1} input. In 75% of the cases, three input masks can recover the Sbox input. And if all four input masks are observed, the input can always be determined. Thus, on average, we need $2\times0.5+3\times0.25+4\times0.25 = 2.75$ input masks, which means 2.75 different faults. Same results hold for $GS_8^{i+1}, GS_{16}^{i+1}$ and GS_{24}^{i+1}. In total, we need 11 faults to recover Remainder group R0 input in round $i+1$. As discussed in Section III-B1, we need on average 9.36 faults to recover the Quotient group Q0 input in round i. We can conclude that **11** faults can recover inputs for all 8 Sboxes and consequently 8 bits of the key. Hence, for random fault we need **88** faults for the recovery of the ith round key.

b) SBCMA B: In this scenario, we assume the attacker can only determine if there is a change in each Quotient group of round $i+2$. Figure 3 illustrates this situation with a fault injected in Sbox GS_0^i – the input differences for two adjacent Sboxes with a same color cannot be differentiated.

Let us consider a mapping, denoted as *Combined-Sbox*, that takes 8 bits as input and outputs 4 bits defined as follows:

$$\textit{Combined-Sbox}(\boldsymbol{x}) = GS(\boldsymbol{x}_1) \oplus GS(\boldsymbol{x}_2),$$

where $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the first and the second nibble of $\boldsymbol{x}$ respectively. GS denotes GIFT-Sbox. Then, for input $\boldsymbol{x}$ and input difference $\Delta\boldsymbol{x}$, the output difference is given by $GS(\boldsymbol{x}_1 \oplus \Delta\boldsymbol{x}_1) \oplus GS(\boldsymbol{x}_2 \oplus \Delta\boldsymbol{x}_2)$, where $\Delta\boldsymbol{x}_1$ and $\Delta\boldsymbol{x}_2$ are the first and the second nibble of $\Delta\boldsymbol{x}$, respectively.

With this notion, we can consider the pair of Sboxes GS_0^{i+1}, GS_8^{i+1} as one Combined-Sbox. As discussed above, we know the input difference of this Combined-Sbox from Step 4. We note that there are only 12 possible input differences in our attack scenario: when GS_0^i (resp. $GS_8^i, GS_{16}^i, GS_{24}^i$) is attacked, the possible input differences for this Combined-Sbox are $0x01, 0x20, 0x21$ (resp. $0x02, 0x40, 0x42$, $0x04, 0x80, 0x84$, $0x08, 0x10, 0x18$). Furthermore, the output difference as defined above can be obtained in Step 5.

We have constructed an Output Difference Table for the Combined-Sbox, where columns correspond to 12 input differences, rows correspond to input values and entries are output differences. With each observed input and output difference pairs, we can reduce the candidate for the Combined-Sbox input[4].

Chosen fault mask. With each combination of fault masks at round i we can translate it to input masks of two Combined-Sboxes (GS_0^{i+1}, GS_8^{i+1} and $GS_{16}^{i+1}, GS_{24}^{i+1}$) at round $i+1$. As shown in Section III-B1, we need on average 8 fault masks to recover the Quotient group Q0 input at round i. Starting from 8 fault masks, we did exhaustive search with GIFT Sbox Output Difference Table (Table IV), which tells us if the fault mask can recover input of Sbox in round i, and Output Difference Table for the Combined-Sbox, which tells us if the propagated fault can help us recover inputs of Combined-Sboxes in round $i+1$. We found that **12** fault masks at round i are enough to recover inputs of all 8 Sboxes. Hence, we need **96** faults to recover the ith round key.

Random fault with known fault mask. By a similar analysis of Table IV that resulted in Table V, we analyzed the Output Different Table for the Combined-Sbox and the results are presented in Table V in Appendix A. On average, the number of faults needed to recover a Combined-Sbox input is 9.049. The same result holds for GS_{16}^{i+1} and GS_{24}^{i+1} (viewed as a Combined-Sbox). For recovery of Q0 input in round i we need on average 9.36 random faults (Section III-B1), thus we need on average **18.098** random faults for recovery of 8 bits of round key. Consequently, we need **144.784** faults to recover the full ith round key.

3) Complexity for Recovery of Master Key: We note that after recovery of the ith round key, the attacker knows the cipher state after round i **PermBits**. By a similar attack as in Section III-B1, we can inject faults at each Sbox input of round $i+2$ and measure the side-channel leakage of round $i+2$ after the fault injection to get inputs of Sboxes in round $i+2$. This way, we recover the $(i+1)$th round key and together with the ith round key we get the master key. Following similar argument as in Section III-B1, the attacker needs another 64 (resp. 74.88) faults to get the master key in case of chosen fault mask (resp. random fault with known fault mask).

On the other hand, with the knowledge of output differences for each Sbox in round $i+1$, we can get the input differences for all Sboxes in round $i+2$. We assume the attacker has stored the traces for each encryption carried out during the attack. In case there is a faulty input in only one Sbox for any of the Quotient groups in round $i+2$, by taking the difference of traces corresponding to round $i+2$ we can figure out the output difference for this particular Sbox. For example, if we inject a fault in GS_0^i and the input of GS_0^{i+2} is changed but the input of GS_2^{i+2} does not change, then we have only one Sbox input change in the Quotient group Q0 ($GS_0^{i+2}, GS_1^{i+2}, GS_2^{i+2}, GS_3^{i+2}$) of round $i+2$. This saves us one fault for the Sbox GS_0^{i+2}. In Appendix B, we provide detailed analysis for the number of faults saved to recover the master key. The total number of faults needed for each attack scenario is as follows: SBCMA A chosen fault: $64 + 6.46 = \mathbf{70.46}$; SBCMA A random fault with known fault mask $88 + 3.17 = \mathbf{91.17}$; SBCMA B chosen fault: $96 + 2.03 = \mathbf{98.03}$; SBCMA B random fault with known fault mask $144.784 + 0.41 = \mathbf{145.194}$

IV. Experiment

In this section, we target publicly available bit slice implementation[5] for GIFT-128 by SBCMA B.

a) Difference Recognition by Side Channels: As it was shown before, in bit permutation based ciphers it is possible to recognize differences in Sbox outputs of round i by observing

[4]As the table has 3,072 entries, we omit the table in this document.

[5]https://www.isical.ac.in/~lightweight/COFB/resource.html

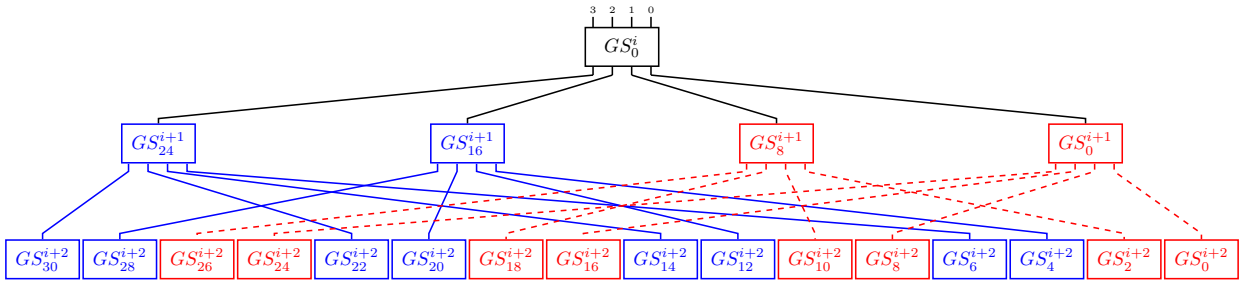


Fig. 3. Fault propagation after 2 rounds of GIFT-128, illustrating the situation when only combinations of 2 Sboxes can be distinguished.

data operations in round $i+1$ [12], [35]. This is due to the characteristics of bit permutations with optimal diffusion, where the x output bits of one Sbox are distributed to x different Sboxes in the subsequent round. While the above-mentioned works focused on propagating differentials from plaintext and observing the differences in the second round, later it was shown that middle rounds can be attacked in a similar fashion with so-called See-In-The-Middle attack (SITM) [14]. SITM also extended the methodology to SPN ciphers which were protected by shuffling.

In the following sections, we simulate how this characteristic can be observed by the side-channel leakage from a bitslice implementation of GIFT.

A. Bit Slice Implementation of GIFT-128

In software, it is usually recommended to use bitslice implementations to fully utilize the bandwidth provided by the architecture. In this part, we detail the attack on bitslice implementation of GIFT for 32-bit architectures.

The state of an intermediate value is represented by four 32−bit variables S_0, S_1, S_2, S_3. The **SubCell** operation is then carried out as follows: $\forall j \in \{0, 4, \ldots, \frac{n}{4} - 1\}$

$$(s_{3,j}||s_{2,j}||s_{1,j}||s_{0,j}) \leftarrow GS(s_{3,j}||s_{2,j}||s_{1,j}||s_{0,j})$$

where $s_{i,j}$ denotes the jth entry of S_i. And **PermBits** follows the permutation described in Table 17 in [5] such that

$$s_{i,P_i(j)} \leftarrow s_{i,j}, \forall i \in \{0,1,2,3\}, j \in \{0, 4, \ldots, \frac{n}{4} - 1\}.$$

We focus on the bislice implementation provided in [36]. In this implementation, **PermBits** is realized by the `rowperm` function, detailed in Listing 1, which permutes one nibble of the input S in one loop (line 5 - 8).

```
0   rowperm(uint32_t S, int B0_pos, int B1_pos,
1                       int B2_pos, int B3_pos){
2       uint32_t T=0;
3       int b;
4       for(b=0; b<8; b++){
            //permute bits at positions 4b
5           T |= ((S>>(4*b+0))&0x1)<<(b + 8*B0_pos);
            //permute bits at position 4b+1
6           T |= ((S>>(4*b+1))&0x1)<<(b + 8*B1_pos);
            //permute bits at position 4b+2
7           T |= ((S>>(4*b+2))&0x1)<<(b + 8*B2_pos);
            //permute bits at position 4b+3
8           T |= ((S>>(4*b+3))&0x1)<<(b + 8*B3_pos);
9       }
10      return T;
11  }

    // call rowperm with state variables S[0-3]
12  S[0] = rowperm(S[0],0,3,2,1);
13  S[1] = rowperm(S[1],1,0,3,2);
14  S[2] = rowperm(S[2],2,1,0,3);
15  S[3] = rowperm(S[3],3,2,1,0);
\vspace{-0.6cm}
```

Listing 1. Implementation of permutation in bitslice GIFT.

B. Leakage Simulation

The side-channel leakage we would like to exploit happens at the execution of each iteration of the for loop (line 4) in the target rounds i, $i+1$ for recovery of the ith round key. Keeping the notations from Section II-B. At each round, with input $S[j]$ to `rowperm`, the bth iteration of the for loop applies **PermBits** to the jth bit of each Sbox output in Quotient group b in that round.

We can assume the side-channel leakage in the bth iteration is closely related to the four jth bits of the Sboxes' outputs in Quotient group b. We would like to exploit this relation. In the following, we analyze this leakage for different Signal-to-Noise Ratio (SNR) values.

We implemented `rowperm` in ARM Cortex-M4 assembly and leakaqe traces are generated as follows:

- We have adopted the Hamming Weight (HW) model. Namely, we model the leakage of an assembly instruction as the sum of hamming weights of the value in each register involved in the instruction (when loaded in the pre-charged data bus).
- The noise component at each time sample are mutually independent and each of them follows the same Gaussian distribution with mean 0 and variance σ^2. For different SNR, we vary the noise variance σ^2.

We remark that we have also simulated the leakage with the regression model [37]. We found that deviation from HW model due to non-equal contribution of each bit does not influence the attack results – there is no improvement nor degradation of the difference recognition.

1) Recovery of Sbox Inputs for Round i: In this part, we simulate the attack as described in Step 4 of Section III-B. If the fault is injected at the input of GS_0^i, we know there is no change in the outputs of the other three Sboxes in Quotient group Q0 of round i. Consequently, the side-channel leakage of 0th iteration of `rowperm` at round i with inputs $S[0]$; $S[1]$; $S[2]$ and $S[3]$ indicates if there is a change in the 1st; 2nd; 3rd and 4th bit of the GS_0^i output, respectively.

To simulate the leakage of the 1st bit of the GS_0^i output, we use the following steps:

1) Set the SNR value and calculate the corresponding noise variance σ^2.
2) Generate three random 32−bit values X, Y and Z such that $X = Z$ and

$$Y[0] = Z[0] \oplus 1, Y[i] = Z[i] \text{ for } i = 1, 2, \ldots, 31.$$

3) Generate one side-channel trace T_X for the 0th iteration of `rowperm(X, 0, 3, 2, 1)` with the Hamming Weight model and noise variance σ^2.
4) Generate one side-channel trace T_Y for the 0th iteration of `rowperm(Y, 0, 3, 2, 1)` with the Hamming Weight model and noise variance σ^2.
5) Generate one side-channel trace T_Z for the 0th iteration of `rowperm(Z, 0, 3, 2, 1)` with the Hamming Weight model and noise variance σ^2.
6) Find $\max |T_X - T_Z|$ and $\max |T_Y - T_Z|$ for the part of the traces that corresponds to line 5 in Listing 1[6].
7) Repeat the above for different SNR values between $0.01-20$.

In this simulation, Z simulates an intermediate value `S[0]` as the input of `rowperm` in round i without fault injection. X simulates value of Z for the scenario when the fault does not propagate to the first bit of the GS_0^i output. Y simulates the case when the fault propagates to the first bit of the GS_0^i output. Note that since only one Sbox is under fault attack, the other bits in Z do not change.

If we can distinguish these two scenarios by observing the peaks of $|T_X - T_Z|$ and $|T_Y - T_Z|$ then we can carry out the attack. In Figure 4 we show the simulated results for plot of the two different peaks versus the SNR. The black line corresponds to the peaks for $|T_Y - T_Z|$ and the grey line corresponds to the peak for $|T_X - T_Z|$. From the figure we can see that with SNR $>= 0.47$ we can successfully conclude if there was a change in the first bit of the intermediate value Z(`S[0]`) and hence if there is a change in the 1st bit of GS_0^i output. We note that the attack for the remaining three bits of the GS_0^i output work in a similar way.

2) SBCMA B – Recovery of Sbox Inputs for Round $i+1$: In this part, we simulate the attack as described in Step 5 of Section III-B. Parameters of our simulation setup, such as leakage model, and GIFT implementation allowed the usage of SBCMA B. In this part we only focus on this type of analysis method.

To recover Sbox inputs for round $i+1$, we use the traces in round $i+1$ to obtain information on output differences for the Sboxes. Recall from Section III-B2 that we try to infer the output difference of *Combined-Sbox*es GS_0^{i+1}, GS_8^{i+1} and $GS_{16}^{i+1}, GS_{24}^{i+1}$. At round $i+1$, the side-channel leakage of 0th iteration of `rowperm` with input $S[0]$; $S[1]$; $S[2]$ and $S[3]$ indicates if there is a change in the 1st; 2nd; 3rd and 4th bit of the *Combined-Sbox* GS_0^{i+1}, GS_8^{i+1} output, respectively. And the side-channel leakage of the 1st iteration of `rowperm` with input $S[0]$; $S[1]$; $S[2]$ and $S[3]$ indicates if there is a change in 1st; 2nd; 3rd and 4th bit of the *Combined-Sbox* $GS_{16}^{i+1}, GS_{24}^{i+1}$ output respectively. To simulate the leakage of 1st bit of *Combined-Sbox* GS_0^{i+1}, GS_8^{i+1} output, we follow the same steps as in Section IV-B1 with Steps 2 and 5 changed to the following:

2. Generate three random 32−bit values X, Y and Z such that[7]

$$X[i] = Z[i] \text{ for } i \neq 4, 6,$$
$$Y[0]||Y[2] \neq Z[0]||Z[2], Y[i] = X[i] \text{ for } i \neq 0, 2,$$

with no restrictions on the 4th and 6th bits of X, Y.

5. Find $\max_i |T_X[i] - T_Z[i]|$ and $\max_i |T_Y[i] - T_Z[i]|$ for the part of traces that correspond to lines 5 and 7 in Listing 1.

In this simulation, Z simulates an intermediate value `S[0]` as the input of `rowperm` in round $i+1$ without fault injection. X simulates the case when the fault does not propagates to the 1st bit of GS_0^{i+1} or GS_8^{i+1} output. Y simulates the case when the fault propagates to the 1st bit of *Combined-Sbox* GS_0^{i+1}, GS_8^{i+1} output – the 1st bit of GS_0^{i+1} output or/and the 1st bit of the GS_8^{i+1} output. Furthermore, "no restrictions on the 4th, 6th bits of X, Y" means the 1st bits of $GS_{16}^{i+1}, GS_{24}^{i+1}$ outputs may or may not be faulty.

To successfully carry out the attack, the attacker needs to distinguish the two different scenarios by observing the highest peak in the trace differences obtained in Step 6. In Figure 5 we show the simulated results of the two different peaks versus the SNR. The black line corresponds to the peaks for $T_Y - T_Z$ and the grey line corresponds to the peaks for $T_X - T_Z$ generated in Step 10. The distinguishing margin increases with SNR as expected and allows to successfully conclude if there was a change in the first bit of the intermediate value Z (`S[0]`) and hence if there is a change in the 1st bit of the *Combined-Sbox* GS_0^{i+1}, GS_8^{i+1} output.

We note that the attack for the remaining three bits of the *Combined-Sbox* GS_0^{i+1}, GS_8^{i+1} output work in a similar way. Same for the *Combined-Sbox* $GS_{16}^{i+1}, GS_{24}^{i+1}$ output.

C. Experimental Evaluation

In this part, we show practical results of applying SBCMA on a bitslice implementation of GIFT-128. The experiments were conducted by using a ChipWhisperer evaluation platform as a measurement device, and ARM Cortex-M0 microcontroller as a device under test.

Similar to Section IV-B, we focus on validating the possibility of exploiting side-channel leakages for achieving steps 4 and 5 as described in Section III-B.

1) Recovery of Sbox Inputs for Round i: : In this part, we show experimental results for Step 4 of Section III-B. We have conducted 10 experiments. For each experiment, we generate three random 32-bit values X, Y and Z such that $X = Z$ and

$$Y[0] = Z[0] \oplus 1, Y[i] = Z[i] \text{ for } i = 1, 2, \ldots, 31.$$

Z corresponds to the intermediate value `S[0]` as the input of `rowperm` in round i without fault injection. X corresponds to value of Z for the scenario when the fault does not propagate to the first bit of the GS_0^i output. Y represents the case when the fault propagates to the first bit of the GS_0^i output. Note

[6]Here, we assume the attacker can distinguish which part of the trace corresponds to which operation. This can be achieved for most side-channel attacks, see e.g. [28].

[7]|| indicates concatenation.

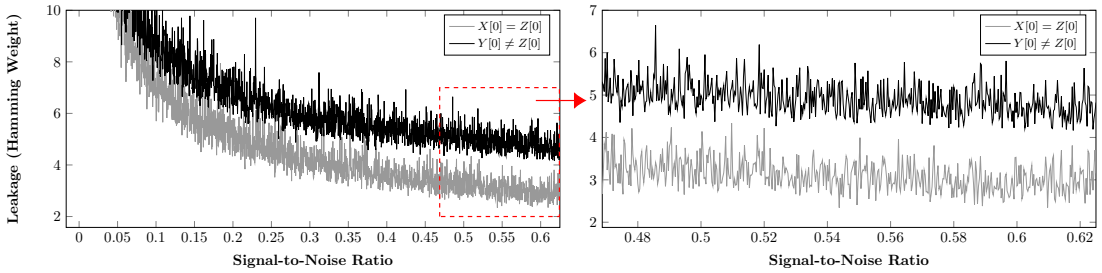


Fig. 4. Plot showing leakage against the signal-to-noise ratio for SBCMA– recovery of Sbox input in round i. $X[i] = Y[i] = Z[i]$ for $i \neq 0$.

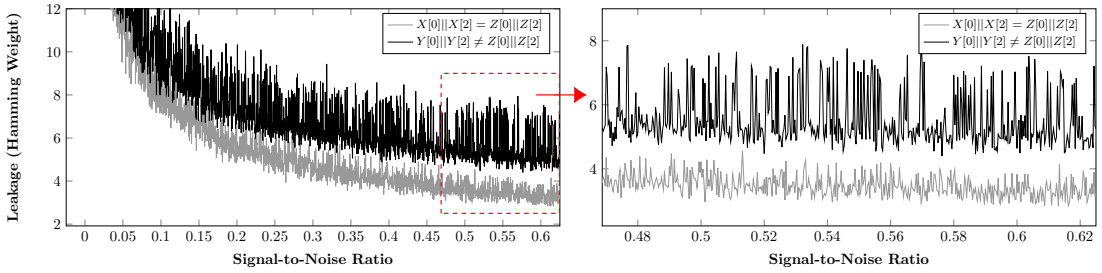
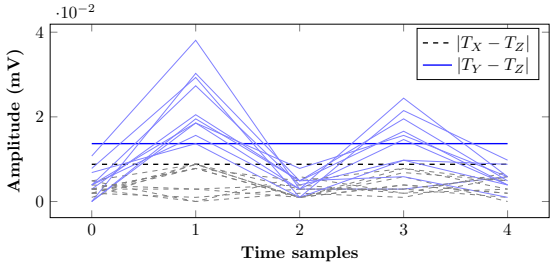
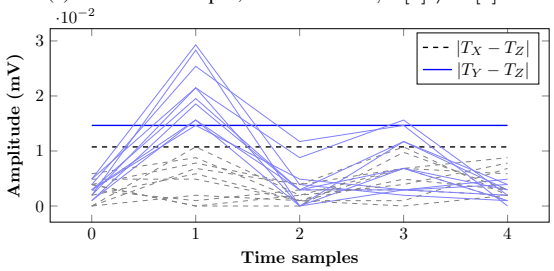


Fig. 5. Plot showing leakage against the signal-to-noise ratio for SBCMA B. $X[i] = Y[i] = Z[i]$ for $i \neq 0, 2, 4, 6$ and $X[i], Y[i]$ are random for $i = 4, 6$



(a) SBCMA B Step 4, where $X = Z, Y[0] \neq Z[0]$.



(b) SBCMA B Step 5, where $X[i] = Z[i]$ for $i \neq 4, 6$, $Y[0]||Y[2] \neq Z[0]||Z[2], Y[i] = X[i]$ for $i \neq 0, 2$.

Fig. 6. Observation of differences for SBCMA B from a real set of traces. Measurement was done on a ChipWhisperer Lite platform with ARM Cortex-M0 as a DUT.

that since only one Sbox is under fault attack, the other bits in Z do not change.

The side-channel leakages T_X, T_Y, T_Z are measured for line 5 of Listing 1 where the difference can be clearly recognized. The results are plotted in Figure 6 (a). Blue traces are calculated using $|T_Y - T_Z|$ and each blue trace has a corresponding black dotted trace which is equal to $|T_X - T_Z|$. The horizontal guiding lines show a distinguishing area between these two sets. The blue guiding line takes the lowest peak of black traces (which is the minimum of $\max|T_Y - T_Z|$ among the experiments). The black guiding line takes the highest peak from the black dotted traces (which is the maximum of $\max|T_X - T_Z|$ among the experiments). There was no averaging or other post-processing done on the traces. The results show that we can successfully distinguish if there is a change in the first bit of GS_0^i output.

2) SBCMA B – Recovery of Sbox Inputs for Round $i+1$: Next, we experimentally verify the attack as described in Step 5 of Section III-B. We have conducted 10 experiments. For each experiment, we generate three random 32−bit values X, Y and Z such that

$$X[i] = Z[i] \text{ for } i \neq 4, 6,$$
$$Y[0]||Y[2] \neq Z[0]||Z[2], Y[i] = X[i] \text{ for } i \neq 0, 2,$$

with no restrictions on the 4th and 6th bits of X, Y. Here $||$ indicates concatenation.

Z corresponds to an intermediate value `S[0]` as the input of `rowperm` in round $i+1$ without fault injection. X corresponds to the case when the fault does not propagates to the 1st bit of GS_0^{i+1} or GS_8^{i+1} output. Y represents the case when the fault propagates to the 1st bit of *Combined-Sbox* GS_0^{i+1}, GS_8^{i+1} output – the 1st bit of GS_0^{i+1} output or/and the 1st bit of the GS_8^{i+1} output. Furthermore, "no restrictions on the 4th, 6th bits of X, Y" means the 1st bits of $GS_{16}^{i+1}, GS_{24}^{i+1}$ outputs may or may not be faulty.

The side-channel leakages T_X, T_Y, T_Z are measured for line 5 of Listing 1 where the difference can be clearly recognized. The results are plotted in Figure 6 (b). Blue traces are calculated using $|T_Y - T_Z|$ and each blue trace has a corresponding black dotted trace which is equal to $|T_X - T_Z|$. The horizontal guiding lines show a distinguishing area between these two sets. The blue guiding line takes the lowest peak of black traces (which is the minimum of $\max|T_Y - T_Z|$ among the experiments). The black guiding line takes the highest peak from the black dotted traces (which is the maximum of $\max|T_X - T_Z|$ among the experiments). The results show that

we can successfully distinguish if there was a change in the first bit of the intermediate value `S[0]` and hence if there is a change in the 1st bit of the *Combined-Sbox* GS_0^{i+1}, GS_8^{i+1} output.

We note that there was no averaging or other post-processing done on the traces. From the experimental result it can be concluded that the difference recognition can be done even on a relatively inexpensive measurement setup.

V. Application to Lightweight AEAD Schemes

In this section we detail how SBCMA can be used to attack the NIST LWC competition finalist GIFT-COFB. We note that similar attack method also applies to round-2 candidates ESTATE, HYENA and SUNDAE-GIFT with the same attack complexity.

AEAD algorithm takes four inputs: plaintext, associated data, nonce, and secret key; and outputs the ciphertext and a tag. Such algorithm provides confidentiality of the plaintext and integrity of the ciphertext.

We have the following assumptions:

1) The length of the messge is long. Paricularly, there are more than 8 blocks of message for each encryption.
2) Attacker can repeat the same encryption.
3) Attacker has no knowledge of plaintext, ciphertext, nonce, associated data or tag.

Application of SBCMA to GIFT-COFB. COFB is an AEAD scheme based on an underlying cryptographic primitive, which is an n-bit block cipher, E_K. The key of the scheme is the key of the block cipher, i.e. K. We consider the case when the underlying block cipher is $128-$bit GIFT, which is the recommended setting in the scheme design. The schematic of the GIFT-COFB encryption can be found in Figure 2.2 in [36], where N is the nonce, $A[i], M[i], C[i]$ denote the ith block of associated data, the message and the ciphertext, respectively. $X[j], Y[j]$ denote intermediate values. G is a feedback function defined as follows: let $Y \in \{0,1\}^n$ and $Y[1], Y[2]$ be the first and the second $n/2$ bits of Y. Then, $G(Y) = (Y[2], Y[1] <<< 1)$, where $<<< 1$ denotes left rotation by 1 bit.

To apply SBCMA we can utilize all the encryption blocks in the scheme. To attack the ith round key, for each group of Sboxes, we recover the inputs using one encryption of a block. Since there are more than 8 blocks, we can recover the groups with different blocks, thus revealing different parts of the ith round key. The detailed steps are as follows:

1) Apply SBCMA to Quotient group $\mathrm{Q}(j-1)$ in round i of the encryption for a message block j, $j = 1, 2, \ldots, 8$. Successful attack on each encryption block recovers 8 bits of ith round key.
2) After Step 1, we recover the $16-$bit output for Remainder group $\mathrm{R}(j-1)$ in round $i+1$ for encryption block j. Those 16 bits correspond to inputs to 16 different Sboxes in round $i+2$. There are 8 bits which are XOR-ed with $i+1$th round key bits. Then, for each encryption block j, we inject faults in those 8 Sbox inputs for round $i+2$ to recover the inputs, which gives the 8 bits of the key.

For SBCMA A with random fault and known fault mask (resp. chosen fault mask), Step 1 requires 11 (resp. 8) faults on each block and by a similar calculation as in Appendix B, Step 2 needs an extra 0.79 (resp. 1.62) faults on each block. In total, to recover the master key we need $\mathbf{94.32}$ (resp. $\mathbf{76.96}$) faults and $\mathbf{13.79}$ (resp. $\mathbf{11.62}$) sessions for random fault (resp. chosen fault mask).

For SBCMA B with random fault and known fault mask (resp. chosen fault mask), Step 1 requires 18.098 (resp. 12) faults on each block and by a similar calculation as in Appendix B, Step 2 needs an extra 0.1 (resp. 0.51) faults. In total, to recover master key we need $\mathbf{145.58}$ (resp. $\mathbf{100.08}$) faults and $\mathbf{20.20}$ (resp. $\mathbf{14.51}$) sessions for random fault (resp. chosen fault mask).

VI. Discussion

a) Mitigation techniques: There are generally two ways to mitigate the SBCMA attack: application of SCA countermeasures; and implementation that processes bits in a way that it is not possible to distinguish the changes from the leakage trace.

When it comes to SCA countermeasures, masking is an obvious choice to protect the implementation, as it is provably secure [38]. If the masks are completely independent and uniformly distributed, the attack will not work. However, if a bias in masks is present, the attack could be still possible with increased effort, as stated in [35]. Similarly, if there is a hiding-based countermeasure, any imbalance can reveal the differentials in the leakage traces. Shuffling countermeasures were shown to be vulnerable against SCA that aim at distinguishing middle rounds differentials [14]. As SBCMA falls in this category, the same technique can be applied. We leave this investigation for future work.

Another direction is implementations that do not allow measurement of differentials according to steps in Section III-A. Naïve and standard bitslice implementations leak information the same way as shown in Section IV-B1. However, if the grouping of bits is done in a different fashion, such as in Fixslicing GIFT implementation [7], recognition of difference can be made impossible. In Fixslicing implementation, five rounds are grouped together for efficiency, requiring different ordering of operations. While the difference recovery for round $i+1$ according to SBCMA methodology still works, the differences in round $i+2$ remain hidden, effectively thwarting the attack.

b) Reverse engineering of secret Sboxes: Let us consider a cipher with a secret Sbox component and a bit permutation operation satisfying the conditions in Section II-B. We describe how SBCMA with chosen fault model can be extended to recover the secret Sbox component as well as the round key i, for any round i. First, we inject 5 chosen faults, with values $1, 2, 3, 4, 5$ in each Sbox input from round i. Using side-channel leakage, we record the input and the output masks for each Sbox from rounds i and $i+1$.

We further inject 10 different faults $(6-f)$ in GS_0^i input, thus we have 15 pairs of different input (Δ_j) and output (δ_j) masks $(j = 1, 2, \ldots, 15)$ for GS_0^i. Thus, for each hypothesis

of the input value x and the corresponding output value $y = GS_0^i(x)$, we can construct a hypothesis for the secret Sbox $GS(x \oplus \Delta_j) = y \oplus \delta_j$. We assume that x is not mapped to itself. This gives us $16 \times 15 = 240$ different Sbox hypotheses.

For each hypothesis of an Sbox, we can carry out SBCMA with the input and the output mask information we have collected for each Sbox in round $i, i+1$. We assume that for the secret Sbox, output masks corresponding to $1, 2, 3, 4, 5$ are enough to recover the input. This is a reasonable assumption as it holds for all the existing Sbox designs. Then, we can recover one round key hypothesis for round i. In case one round key is enough to recover the master key, with one pair of correct plaintext and ciphertext we can get the correct key guess out of the 240 hypotheses.

c) Practical considerations: As stated in the previous parts, on average, the attacker needs to achieve 91.17 faulty encryptions to recover the master key of GIFT-128. This is under the assumption that the attacker has carried out the profiling phase and is capable of injecting faults at every iteration. If this is not the case, the number of attempts will be higher. For example, results from [39] indicate 39.18% success rate when injecting faults into 4-bit registers on an FPGA (same size as GIFT-128 Sbox). Based on that, the number of encryptions to achieve 91.17 faults would be 232.7.

VII. Conclusion

In this paper, we have presented SBCMA – a combined attack on bit-permutation operation in symmetric block ciphers which does not require any knowledge of plaintext or ciphertext. We showed how this attack can be applied to GIFT-128 as well as to LWC candidates that use GIFT-128 as their main building block. The simulation results show the feasibility of the attack.

For future work, it would be interesting to analyze how the attacks can be extended to implementations with countermeasures. Another potential improvement would be to relax the requirement on fault model, to include random faults without the knowledge of the fault mask.

References

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*. Springer, 1996, pp. 104–113.

[2] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1997, pp. 37–51.

[3] K. McKay, L. Bassham, M. Sönmez Turan, and N. Mouha, "Report on lightweight cryptography," National Institute of Standards and Technology, Tech. Rep., 2016.

[4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 450–466.

[5] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: a small present," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 321–345.

[6] T. B. Reis, D. F. Aranha, and J. López, "Present runs fast," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 644–664.

[7] A. Adomnicai, Z. Najm, and T. Peyrin, "Fixslicing: a new gift representation: fast constant-time implementations of gift and gift-cofb on arm cortex-m," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 402–427, 2020.

[8] F. De Santis, O. M. Guillen, E. Sakic, and G. Sigl, "Ciphertext-only fault attacks on present," in *International Workshop on Lightweight Cryptography for Security and Privacy*. Springer, 2014, pp. 85–108.

[9] N. F. Ghalaty, B. Yuce, and P. Schaumont, "Differential fault intensity analysis on present and led block ciphers," in *COSADE*. Springer, 2015, pp. 174–188.

[10] S. Patranabis, N. Datta, D. Jap, J. Breier, S. Bhasin, and D. Mukhopadhyay, "Scadfa: Combined sca+ dfa attacks on block ciphers with practical validations," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1498–1510, 2019.

[11] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," in *European Symposium on Research in Computer Security*. Springer, 1998, pp. 97–110.

[12] J. Breier, D. Jap, and S. Bhasin, "SCADPA: Side-channel assisted differential-plaintext attack on bit permutation based ciphers," in *2018 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2018, pp. 1129–1134.

[13] E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," in *CRYPTO*, vol. 90. Springer, 1991, pp. 2–21.

[14] S. Bhasin, J. Breier, X. Hou, D. Jap, R. Poussier, and S. M. Sim, "Sitm: See-in-the-middle side-channel assisted middle round differential cryptanalysis on spn block ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 95–122, 2020.

[15] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[16] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *CRYPTO*. Springer, 1997, pp. 513–525.

[17] C. Dobraunig, M. Eichlseder, T. Korak, S. Mangard, F. Mendel, and R. Primas, "Sifa: exploiting ineffective fault inductions on symmetric cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 547–572, 2018.

[18] F. Zhang, X. Lou, X. Zhao, S. Bhasin, W. He, R. Ding, S. Qureshi, and K. Ren, "Persistent fault analysis on block ciphers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 150–172, 2018.

[19] M. Joye and M. Tunstall, *Fault analysis in cryptography*. Springer, 2012, vol. 147.

[20] B. Robisson and P. Manet, "Differential behavioral analysis," in *Cryptographic Hardware and Embedded Systems*, 2007, pp. 413–426.

[21] F. Amiel, K. Villegas, B. Feix, and L. Marcel, "Passive and active combined attacks: Combining fault attacks and side channel analysis," in *Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2007, pp. 92–102.

[22] C. Clavier, B. Feix, G. Gagnerot, and M. Roussellet, "Passive and active combined attacks on aes combining fault attacks and side channel analysis," in *Fault Diagnosis and Tolerance in Cryptography, 2010 Workshop on*. IEEE, 2010, pp. 10–19.

[23] A. Moradi, O. Mischke, C. Paar, Y. Li, K. Ohta, and K. Sakiyama, "On the power of fault sensitivity analysis and collision side-channel attacks in a combined setting," *Cryptographic Hardware and Embedded Systems–CHES 2011*, p. 292, 2011.

[24] S. Patranabis, J. Breier, D. Mukhopadhyay, and S. Bhasin, "One plus one is more than two: a practical combination of power and fault analysis attacks on present and present-like block ciphers," in *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2017, pp. 25–32.

[25] S. Saha, D. Jap, J. Breier, S. Bhasin, D. Mukhopadhyay, and P. Dasgupta, "Breaking redundancy-based countermeasures with random faults and power side channel," in *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2018, pp. 15–22.

[26] A. Papadimitriou, K. Nomikos, M. Psarakis, E. Aerabi, and D. Hely, "You can detect but you cannot hide: Fault assisted side channel analysis on protected software-based block ciphers," in *2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*. IEEE, 2020, pp. 1–6.

[27] M. Azouaoui, K. Papagiannopoulos, and D. Zürner, "Blind side-channel sifa," in *2021 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2021, pp. 555–560.

[28] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2008, vol. 31.

[29] Y. Linge, C. Dumas, and S. Lambert-Lacroix, "Using the joint distributions of a cryptographic function in side channel analysis," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 199–213.

[30] R. Korkikian, S. Pelissier, and D. Naccache, "Blind fault attack against spn ciphers," in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE, 2014, pp. 94–103.

[31] S. Saha, A. Bag, D. B. Roy, S. Patranabis, and D. Mukhopadhyay, "Fault template attacks on block ciphers exploiting fault propagation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2020, pp. 612–643.

[32] Y. Li, M. Chen, Z. Liu, and J. Wang, "Reduction in the number of fault injections for blind fault attack on spn block ciphers," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 2, pp. 1–20, 2016.

[33] C. Clavier and L. Reynaud, "Improved blind side-channel analysis by exploitation of joint distributions of leakages," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 24–44.

[34] C. Clavier, L. Reynaud, and A. Wurcker, "Quadrivariate improved blind side-channel analysis on boolean masked aes," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2018, pp. 153–167.

[35] J. Breier, D. Jap, X. Hou, and S. Bhasin, "On side channel vulnerabilities of bit permutations in cryptographic algorithms," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1072–1085, 2019.

[36] S. Banik, A. Chakraborti, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, and Y. Todo, "GIFT-COFB," *Submission to NIST Lightweight Cryptography Competition (round 2)*, vol. 1, 2019.

[37] W. Schindler, K. Lemke, and C. Paar, "A stochastic model for differential side channel cryptanalysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 30–46.

[38] E. Prouff and M. Rivain, "Masking against side-channel attacks: A formal security proof," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 142–159.

[39] J. Breier, W. He, S. Bhasin, D. Jap, S. Chef, H. G. Ong, and C. L. Gan, "Extensive laser fault injection profiling of 65 nm fpga," *Journal of Hardware and Systems Security*, vol. 1, no. 3, pp. 237–251, 2017.

APPENDIX

A. Table for Section III-B2 SBCMA B

By a similar analysis of Table IV that resulted in Table V (Section III-B1), we analyzed the Output Different Table for the Combined-Sbox and the results are presented in Table VI.

m: No. of pairs	Total cases	Successful cases	Success Rate	Probability exactly m pairs needed
4	495	9	1.8	1.8
5	792	54	6.8	5.0
6	924	141	15.3	8.5
7	792	210	26.5	11.2
8	495	196	39.6	13.1
9	220	118	53.6	14.0
10	66	45	68.2	14.6
11	12	10	83.3	15.1
12	1	1	100	16.7

TABLE VI
DISTINGUISHING COMBINED-SBOX INPUT FROM THE DIFFERENCE PAIRS. THE FIRST COLUMN IS THE NUMBER OF INPUT/OUTPUT DIFFERENCE PAIRS. SECOND COLUMN IS THE TOTAL NUMBER OF POSSIBLE CASES FOR m PAIRS. THIRD COLUMN IS THE NUMBER OF PAIRS WHICH CAN UNIQUELY IDENTIFY EACH INPUT OF COMBINED-SBOX. FOURTH COLUMN IS THE PERCENTAGE OF SUCH PAIRS. FIFTH COLUMN IS THE PROBABILITY THAT EXACTLY m PAIRS OF INPUT/OUTPUT DIFFERENCES ARE NEEDED TO RECOVER DOUBLE-SBOX INPUT.

B. Number of faults calculation for SBCMA master key recovery

In Section III-B3 we argued that in case there is faulty input in only one Sbox for any of the Quotient groups in round $i+2$, by taking the difference of traces corresponding to round $i+2$ we can figure out the output difference for this particular Sbox. This can help us reduce the extra number of faults needed to recover the master key. In this section, we calculate the expected number of extra faults needed.

Attack type	Fault mask	y	p	p_r	Recovered Sbox	No. of extra faults
SBCMA A	Known	11	0.96	0.96	30.65	3.17
	Chosen	8	0.90	0.90	28.77	6.46
SBCMA B	Known	18.1	0.99	0.99	31.82	0.41
	Chosen	12	0.97	0.97	30.99	2.03

TABLE VII
y - NO OF FAULTS INJECTED IN EACH QUOTIENT GROUP IN ROUND i; p - PROBABILITY ONE FAULT MASK CAN BE SAVED FOR ONE SBOX; p_r - PROBABILITY THAT ONE SBOX INPUT CAN BE RECOVERED FROM INFORMATION OBTAINED BY ATTACKER ROUND KEY i

When one fault is injected in Sbox input from round i, there are 16 Sbox inputs in round $i+2$ which may have one bit change. Let $\boldsymbol{d} = d_{15}d_{14}\dots d_0$ denote the 16 bits. We assume the $2^{16}-1$ possible values for $\boldsymbol{d}$ appear randomly with the same probability.

Let y denote the number of faults injected in each Quotient group of round i. First, let use focus on Sbox GS_0^{i+1}. The analysis for other Sboxes in round $i+2$ are similar.

a) Fault Masks: Suppose there is fault injection in Quotient group Q0 in round i. In this case there are 2 Sbox inputs from Q0 round $i+2$ that might have one bit change. For GS_0^{i+1} to be the only one with faulty input in Q0 of round $i+2$, we need input of GS_0^{i+1} changes and that of GS_2^{i+1} does not change. Such a scenario means a fault mask $0x1$ in GS_0^{i+2} can be saved for attacking GS_0^{i+2}. The probability for this to happen with one fault injection in Q0 of round i is $\frac{2^{14}}{2^{16}-1}$. With y faults injected, the probability that fault mask $0x1$ in GS_0^{i+2} can be saved, denoted by p_1, is given by

$$p_1 = 1 - \left(1 - \frac{2^{14}}{2^{16}-1}\right)^y.$$

Similarly, for fault injection in Quotient group Qj ($j = 1, 2, 3$) in round i. We get the probability that fault mask $0x2$, $0x4$, $0x8$ in GS_0^{i+2} can be saved, denoted by p_2, p_3, p_4 respectively are equal to p_1. For simplicity, let us denote this probability by p.

b) Recovery of input.: To recover the input of GS_0^{i+2} with four possible input masks $1, 2, 4, 8$, we carried out a similar analysis as in Section III-B1. The probability that $2, 3, 4$ input masks are enough to recover the input is $0.5, 0.75, 1$ respectively. Thus the probability that input of GS_0^{i+2} can be recovered with information from attacking round key i, denoted by p_r, is

$$\begin{aligned} p_r &= \sum_{m=2}^{4} Prob(\text{exactly } m \text{ input masks happened}) \\ &\quad * Prob(m \text{ input masks can recover the input}) \\ &= 0.5p^2(1-p)^2\binom{4}{2} + 0.75p^3(1-p)\binom{4}{3} + p^4.1 \end{aligned}$$

The same probability p_r holds for other Sboxes in round $i+2$. Thus we can conclude that $32p_r$ Sbox inputs can be recovered without extra fault injection. For chosen fault mask (resp. random fault with known fault mask), we need 2 (resp. 2.34) faults to recover Sbox input (Section III-B1). Hence the number of extra faults needed to recover master key $(32 - 32p_r) * 2$ (resp. $(32 - 32p_r) * 2.34$) for chosen fault (random fault with known fault mask) model.