On Side-Channel Vulnerabilities of Bit Permutations: Key Recovery and Reverse Engineering

Jakub Breier[1], Dirmanto Jap[1], Xiaolu Hou[2] and Shivam Bhasin[1]

[1]Physical Analysis and Cryptographic Engineering
Temasek Laboratories
[2]School of Computer Science and Engineering
Nanyang Technological University, Singapore
Email: {jbreier, djap, sbhasin}@ntu.edu.sg
ho0001lu@e.ntu.edu.sg

***Abstract*—Lightweight block ciphers rely on simple operations to allow compact implementation. Thanks to its efficiency, bit permutation has emerged as an optimal choice for state-wise diffusion. It can be implemented by simple wiring or shifts. However, as recently shown by Spectre and Meltdown attacks, efficiency and security often go against each other. In this work, we show how bit permutations introduce a side-channel vulnerability that can be exploited to extract the secret key from the cipher. Such vulnerabilities are specific to bit permutations and do not occur in other state-wise diffusion alternatives. We propose Side-Channel Assisted Differential-Plaintext Attack (SCADPA) which targets this vulnerability in bit permutation operation. SCADPA is experimentally demonstrated on `PRESENT`-80 on an 8-bit microcontroller, with the best case key recovery in 17 encryptions. The attack is then extended to latest bit-permutation based cipher `GIFT`, allowing full key recovery in 36 encryptions. We also propose and experimentally verify an automatic threshold method which can be easily applied to SCADPA, allowing automation of the attack. Moreover, SCADPA on bit permutations has other applications. Application for reverse engineering secret sboxes in PRESENT-like proprietary ciphers is shown. We also highlight a special case, where fixing one vulnerability opens another one. This is shown by applying SCADPA on some assembly level fault attack countermeasures, rendering it less secure than unprotected implementations. Lastly, we also provide several different attack scenarios, such as targeting different encryption modes.**

I. Introduction

The area of smart devices brings forward the question about energy efficiency. In the past, where desktop computers were the norm, chip manufacturers could afford to release devices with high computational power at the expense of the power consumption. However, this trend is changing rapidly, thanks to shrinking size of mobile devices and emergence of Internet of Things. Often, the raw computational power is lowered, while keeping the battery last longer. It is therefore crucial for the algorithm designers to develop smaller and more efficient designs that would fit such resource-constrained devices. This affects the area of cryptography as well, where we can see more efficient algorithms emerge every year. Recently, *NIST* has launched a call for proposal to standardize lightweight crypto algorithms [1].

This work focuses on lightweight block ciphers which use bit permutation based diffusion layer to achieve efficient implementations. Common examples of such block ciphers are `PRESENT` [2], which is an *ISO* standard, `GIFT` [3], etc. Bit permutation is an interesting design choice as it has negligible area footprint in hardware and can be implemented with only wires [2]. Other ciphers, such as `GIFT`, with careful permutation choices can be optimized for both hardware and software implementations.

It was recently shown that through Spectre and Meltdown [4] attacks on modern processor architectures, that architectural optimizations for efficiency or performance can result in a security disaster. In this paper, we bring forward a specific vulnerability of bit permutation based diffusion function, which can be simply exploited using side-channel. The exploit arises from the simple structure of bit permutation and is not easily found in diffusion functions of standard ciphers (like `MixColumns` in `AES`). The demonstrated vulnerability is more serious in low-cost platforms like 8-bit microcontrollers due to serialized execution of the algorithm. Such design vulnerabilities further make the need of countermeasures critical, however, lightweight and countermeasures do not often go hand-in-hand. To resist theoretical attacks, cipher designers add extra rounds to avoid vulnerabilities due to simple diffusion layer. Since the proposed attack exploits all the information in the first round, extra rounds will not add any security.

A bit permutation layer diffuses the output bits of an Sbox (non-linear Substitution layer) to multiple Sboxes. By observing the numbers of affected Sboxes in a given round, the (key-dependent) Sbox output in the previous round can be determined, thus revealing information about the secret key. In this paper, we present *Side-Channel Assisted Differential-Plaintext Attack* (SCADPA) which exploits bit permutation construction for secret key retrieval through observed side-

channel leakage. Here, SCADPA is chosen plaintext attack, where the plaintexts are chosen to effectively exploit the bit-permutation leakage. It observes the difference of propagation through side-channel, thus revealing the differential of Sbox output. With the knowledge of the plaintext, this differential can be solved to reveal the corresponding key candidates.

Unlike usual side-channel attacks (SCA [5]), SCADPA is not a statistical attack but rather a differential attack. For a carefully chosen set of plaintexts, it observes differential on an internal sensitive value. As we show later, these differentials for bit permutation ciphers can be obtained by simply subtracting the power measurements. Once the internal differentials are known, the attack is similar to classical differential cryptanalysis for secret key retrieval. While cryptanalysis can be applied on a full cipher and handle high complexity, we manage to restrict the attack to a single round, thus keeping the complexity negligible. In some cases, multiple plaintext pairs might be needed to determine a unique key candidate. With the demonstrated experiments on PRESENT-80, SCADPA can reveal the key in 17 encryptions for the best case and 65 in the worst case. The methodology is further capable of reverse engineering secret Sboxes in PRESENT-like ciphers with $17-46$ encryptions. Further, we extend our attack on GIFT-64-128 and we show that a fault injection countermeasure based on redundancy can be rendered less secure than an unprotected design by SCADPA.

A. Related Works

Chosen plaintext side-channel attacks have previously been proposed in different context. A side-channel based collision attack [6] was proposed under chosen plaintext setting. It detects collision in some internal value of initial rounds of the cipher to retrieve the key. This attack was extended to break secret AES-like ciphers [7]. Some proposed attacks also used chosen plaintext setting to amplify power [8] or timing [9] side-channel leakage. Apart from block ciphers, chosen plaintexts attack were also applied to break public key cryptography [10] and hash functions (HMAC [11]). Recently, a chosen plaintext attack on DES third round was proposed in [12], exploiting the Feistel structure in the design. The key motivation of the attack was that most countermeasures protect only corner rounds for area-security trade-off, enabling attack on internal rounds. To the best of our knowledge, SCADPA is the first attack proposition to exploit the diffusion function in a block cipher.

Reverse Engineering via Side-Channel. Side-Channel Analysis for Reversed Engineering (SCARE) was first introduced in by Novak et al. [13], who proposed a recovery of one of the two secret Sboxes of algorithm A3/A8 used in GSM. Rivain and Roche [14] showed how to recover an equivalent representation of SPN cipher with a fixed first word of a round key. The most recent article up to date, published by Clavier et al. [15] shows a recovery of the secret cipher blocks of AES-like ciphers by both side-channel analysis and ineffective fault analysis.

B. Our Contribution

The contributions of this paper are as follows[1]:

- We identify a specific vulnerability in bit-permutation based diffusion functions exploitable through side-channel, and we propose a specific attack called SCADPA, which exploits the identified vulnerability.
- We present a practical demonstration of SCADPA on a low cost platform, using PRESENT-80 lightweight cipher as a target algorithm.
- We provide a comprehensive analysis of SCADPA in numerous test scenarios including application to 8-bit and 32-bit architectures.
- We further evaluate SCADPA on GIFT, a recent bit-permutation cipher designed for optimal performance across platforms.
- We extend SCADPA methodology to reverse engineer secret Sbox in PRESENT-like ciphers.
- We demonstrate that redundancy-based countermeasures against fault injection attacks, such as Internal Redundancy Countermeasure [17] greatly increase the success rate of SCADPA as compared to a standard unprotected counterpart.

C. Organization

The rest of the paper is organized as follows. Section II recalls basics of PRESENT-80 block cipher, followed by Section III which describes the key methodology of SCADPA. Experimental validation of SCADPA on an 8-bit microcontroller is presented in Section IV. Section V describes the attack on block cipher GIFT. Extension of the attack to recovery of secret Sboxes is provided in Section VI. Section VII shows how SCADPA can be extended to 32-bit architectures, with minor modifications. Some discussions are provided in Section VIII, exploring attacks on countermeasures, different block cipher operation modes, vulnerability of other diffusion functions etc. Final conclusions are drawn in Section IX.

II. Background

In this part, we first provide a high-level overview of PRESENT cipher in Section II-A. In Section II-B, we focus on permutation layer of PRESENT, while detailing the properties that are exploited by SCADPA.

A. PRESENT Cipher

PRESENT is a lightweight block cipher based on Substitution-Permutation Network (SPN) [2]. Therefore, it

[1]This paper is an extended version of [16]. While the original paper outlines the basic method to use Side-Channel Assisted Differential Plaintext Attacks (SCADPA), this paper provides new methods, targets, results, and use cases. More specifically, we add the following:

- extension of SCADPA to lightweight cipher GIFT,
- new method to reverse engineer Sboxes, with results on PRESENT and GIFT,
- extension of SCADPA to 32-bit architectures,
- discussion on redundancy based fault countermeasures, with application on Internal Redundancy Countermeasure [17].

consists of three operations: addRoundKey is a bit-wise xor of the state with the round key; sBoxLayer is a nibble-wise nonlinear substitution; pLayer is a bit permutation. The structure of one round of the cipher is depicted in Figure 1. PRESENT consists of 31 rounds, followed by a post-whitening addRoundKey at the end. The variant used in our experiments, PRESENT-80, has a secret key of size 80 bits and a block size of 64 bits. Table I shows PRESENT Sbox that is executed on all 16 nibbles of the PRESENT-80 state during the sBoxLayer. The Sbox function is further denoted as $S(.)$.

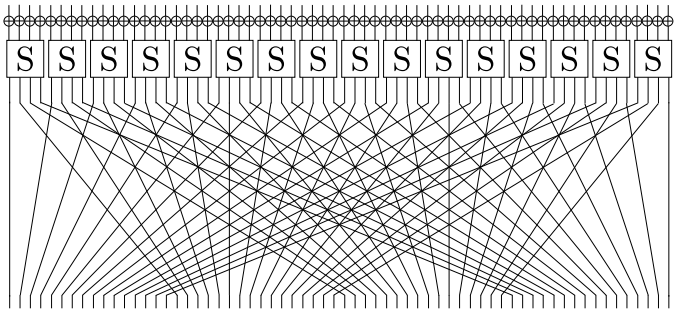

Fig. 1: Structure of one round of PRESENT-80 cipher.

TABLE I: PRESENT S-box.

x	0	1	2	3	4	5	6	7
$S(x)$	C	5	6	B	9	0	A	D
x	8	9	A	B	C	D	E	F
$S(x)$	3	E	F	8	4	7	1	2

B. Bit Permutation Properties of PRESENT-80

There are three main properties of the pLayer, resulting from the optimal diffusion requirement, that are exploited in the following:

1) Output of one nibble is distributed into four distinct nibbles.
2) Input to one nibble consists of outputs from four distinct nibbles.
3) In pLayer, the cipher state can be split into four different groups of four nibbles, where one input group affects exactly one output group.

Examining these properties, it can be seen that by changing chosen four nibbles in the plaintext, it is possible to affect the whole cipher state after the first permutation.

Figure 2 shows this behavior by changing the first and the eighth nibble of the plaintext. The underlying implementation computes sBoxLayer nibble-wise while addRoundKey is computed byte-wise owing to the ALU support for bit-wise xor. This is to achieve the best speed-memory trade-off. Similarly, in 32-bit architectures, sBoxLayer would be implemented as a 4-bit or 8-bit look-up table and addRoundKey with pLayer would be done on 32-bit words, to achieve best speed-memory trade-off.

By observing the changing nibbles in round 2, the change in Sbox output at round 1 can be determined. This value directly

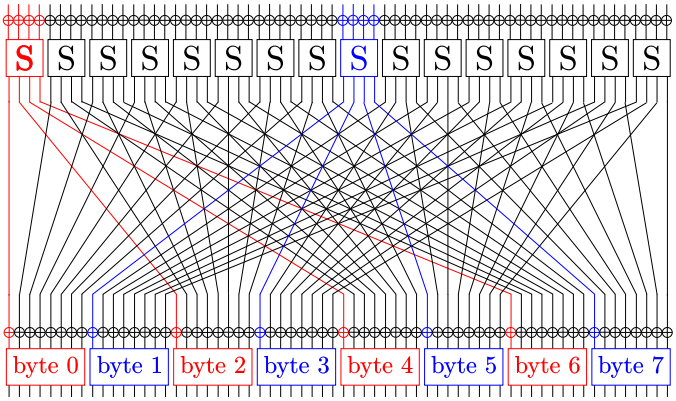

Fig. 2: Bytes in round 2 that could be potentially affected by changing the first and the eighth nibble of the plaintext.

depends on the secret key which can be exploited for key retrieval. In the following, we use side-channel measurements to observe the changed nibbles.

III. SCADPA Methodology

In this part we first explain how the SCADPA method works in Section III-A, followed by steps to choose optimal plaintexts for the attack in Section III-B. Next, we provide an attack example in Section III-C. Finally, we state the attack complexity in Section III-D. Although this section is based on PRESENT-80, the techniques are generally applicable on similar ciphers.

A. SCADPA Procedure

Using the information from the previous part, we propose SCADPA. SCADPA exploits the permutation properties to observe changed nibbles and retrieve differential at Sbox output in round 1. The differential can be solved for key retrieval by using a standard differential attack on non-linear layer.

The attack steps can be summarized as follows:

Step 1: Encrypt a chosen plaintext p, by using an unknown secret key k, denoted as $E_k(p)$.
Step 2: Capture the power/EM leakage of the Device Under Test (DUT) during the second encryption round to get the trace t.
Step 3: Choose another plaintext $p' \neq p$ that differs exactly in one nibble at i^{th} position. The nibble at position i in plaintext p is denoted p_i and $x_i = p_i \oplus k_i$. Similarly, $x'_i = p'_i \oplus k_i$.
Step 4: Capture the leakage for $E_k(p')$ to get t'.
Step 5: Calculate $\Delta t = t - t'$.
Step 6: By examining Δt, get the Sbox output change $\Delta S_i = S(x_i) \oplus S(x'_i)$ of round 1 in the position where the plaintext had changed.
Step 7: Calculate all possible candidates for key nibble k_i such that with input pair p_i and p'_i, the change determined in Step 6 would appear.

Step 8: Repeat steps 3-7 with another p'_i, taking intersection of all the calculated key candidates, until there is just one candidate for k_i.

Step 9: Repeat steps 3-8 for all $i \in [0, 15]$ to recover the whole round key.

Step 10: Compute the remaining 16 bits of the secret key by exhaustive search or repeating SCADPA on the next round.

B. SCADPA *Acceleration by Optimal Plaintext Choice*

In order to reduce the key complexity and retrieve the secret key with fewer number of encryptions, it is possible to change the value of multiple nibbles in the plaintext. Based on the permutation layer, multiple nibbles can be changed without affecting same locations in the next round and hence, can be analyzed independently. The optimal methodology executes the following steps:

Step 1: Keep the record of nibbles of key that have not been recovered ($I = \{0, \ldots, 15\}$)

Step 2: Start by choosing one nibble ($n_i \in I$) and calculate all possible affected nibbles at the beginning of the next round (N_i).

Step 3: Choose another nibble ($n_j \in I$, $n_j \neq n_i$) and check if the affected nibbles interfere (check if $N_i \cap N_j = \emptyset$). If true, keep n_j, else, move to other nibble . Repeat, until no nibble remaining, then update $I \setminus \{n_i, n_j, ...\}$.

Step 4: Choose plaintext set that changes only on these nibble positions ($\{n_i, n_j, ...\}$).

Step 5: Repeat step 2-4, until $I = \emptyset$

Another option is to choose the pair difference in the plaintext that could minimize the key candidate. This is dependent on the Sbox used. In the case of `PRESENT`, as shown later in Figure 3, for one plaintext pair, there could be either 2 or 4 key candidates. However, even with two plaintext pairs, there is still slight chance on getting more than 1 key candidate.

A natural question would be – 'how many nibbles of `PRESENT` can we attack at once by using SCADPA?' As can be seen in Figure 2, one nibble can affect up to half of the state at the next round `addRoundKey`. Nibbles $0-7$ ("group 1") affect bytes $0, 2, 4, 6$, while nibbles $8-15$ ("group 2") affect the remaining bytes $1, 3, 5, 7$. Therefore, by combining one nibble from each group, we can retrieve two nibbles at the same time while avoiding the interference. This knowledge can help us to reduce the number of encryptions to half. Parallelization of attack to two nibbles is limited by the byte-wise granularity of `addRoundKey`. The following `sBoxLayer` with nibble-wise granularity would allow up to 4 nibbles in parallel, however at the cost of needed profiling.

C. Attack Example

We illustrate our method by a simple example that shows how to recover one nibble of the round key i.e. $i = 1$. We fix p_i = `0x0` and p'_i = `0xA`. After observing the Δt, we figure out that ΔS_i = `0x2`. By knowing that Δp_i = `0xA`, there can be two candidates for k_i: `0xF` and `0x5`. We make another experiment, now with p'_i = `0x3`. By capturing another trace, we determine ΔS_i = `0x6`, giving us four different candidates for k_i: `0xC`, `0xD`, `0xE`, and `0xF`. The only intersecting candidate is `0xF`, therefore we know that the key nibble has to be this value. The retrieval of ΔS_i from real power traces is explained in the following section.

D. Attack Complexity

In this section, we compute the attack complexity for single key nibble retrieval and full round key retrieval.

The single nibble retrieval complexity depends on the underlying Sbox function, being the only non-linear function in the derived differential equation. Table II shows the extended Difference Distribution Table (DDT) for `PRESENT` Sbox. Input difference to the Sbox is denoted as δ, while the output difference is denoted as Δ. From this table, it can be observed that one needs two input differences to uniquely identify the Sbox input. Therefore, to fully recover the first round key with a chosen plaintext model and one nibble recovery at a time, the attack requires one reference trace and 32 difference traces, resulting to 33 encryptions in total. However, the method from Section III-B provides more optimized way to mount the attack, with two nibbles at a time, resulting to 17 encryptions in total.

Now, we can consider a known plaintext model which is more relaxed requirement. As can be seen in Table III, using a single plaintext pair $\Delta p'_i$ for determining ΔS, it will yield 2 key candidates in $\approx 60\%$ of cases and 4 candidates for the rest. By adding additional plaintext pair, the probability of identifying a unique key candidate k_i are 94.3%, 99.5% and 100% for 2, 3 and 4 plaintext pair respectively. Since, the same reference plaintext can be used, with 5 encryptions a unique key candidate can be identified in worst case. The numbers would change for an Sbox other than `PRESENT`, but will stay comparable.

Figure 3 shows the complexities for different scenarios. As mentioned previously, the best case requires 17 encryptions. In case we can choose the optimal plaintexts but for some reason can only recover one nibble at a time, `SCADPA` will require 33 encryptions. The worst case only recovers one nibble at a time while using the conservative reduction of candidates and not exploiting any specific Sbox properties. In this case, it will require 4 values to reduce the key candidate to 1 per nibble. Hence, it will require 65 encryptions.

Please note that it is also possible to make a search on remaining candidates. For example, if we have 2 candidates for each nibble, we can determine the value with a brute-force search with complexity of 2^{16}. For such case, it would only require 9 encryptions in case we target two nibbles at a time. Similarly, operating at nibble wise granularity at the following `sBoxLayer` can recover the key in 9 encryptions as well.

IV. Experimental Results

In this part, we show and discuss experimental results obtained by performing SCADPA on a microcontroller implementation of `PRESENT`-80 cipher [2]. Sections IV-A and IV-B provide an overview of the experimental setup and results,

TABLE II: Extended difference distribution table for PRESENT Sbox. Columns represent input difference, rows represent output difference and entries are Sbox inputs.

Δ \ δ	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
1			9a		36		078f				5e		1c		24bd
2						8e	34		09	5f		1d	67ab	2c	
3	cdef	46	12					3b		0a			58	79	
4			47		8d				35ac		0b		2f		169e
5		cdef		0145						2389		67ab			
6		9b	cdef	37		06	25		18					4a	
7	67ab		03	8c				5d				2e	49	1f	
8						17	ad		6f	4e	2389	0c		5b	
9	0145			9d	be			2a			7c	3f		68	
a		02	56	bf	9c					7d	1a	48	3e		
b			8b		27	35ac		169e			4f		0d		
c		8a		26	0145	9f	bc		7e					3d	
d	2389	57			af			4c		1b	6d			0e	
e		13		ae					24bd	6c		59			078f
f						24bd	169e	078f							35ac

TABLE III: Probability of determining key candidates.

# of key candidates	# of plaintext pairs			
	1	2	3	4
1	0	0.94315	0.99557	1
2	0.59948	0	0	0
3	0	0	0	0
4	0.40052	0.05685	0.00443	0

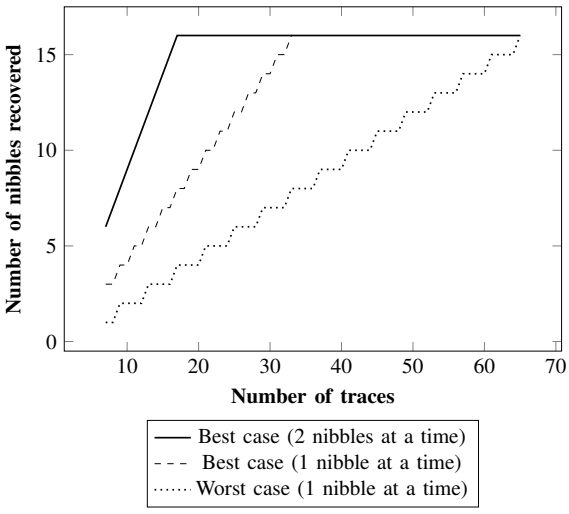

Fig. 3: Number of traces required for recovering key nibbles of PRESENT.

respectively. Section IV-C shows a way of automating the recognition of the Sbox output differences.

A. Setup

As a DUT, we used a standard 8-bit microcontroller from Atmel, ATmega328P, mounted on Arduino UNO development board. We have measured an electromagnetic emanation with a Langer RF-U 5-2 probe. Signal was captured with LeCroy WaveRunner 610 Zi oscilloscope.

We have used an implementation of PRESENT-80 cipher, where sBoxLayer is computed nibble-wise and the rest of the operations are done byte-wise. Sampling rate was set at 500 MS/s, while the addRoundKey takes ≈7000 samples. A difference introduced in the first round could be observed during the second round. We chose to observe the difference at the second addRoundKey as in our case the start of round was easily identifiable by visual inspection of the trace. Choice of observing the difference on addRoundKey has an advantage and a disadvantage. The advantage being that precise profiling was not needed as compared to locating time instants for sBoxLayer. The disadvantage is that since addRoundKey is done byte-wise, the observed differences are limited to byte level. Observed differences on the following sBoxlayer can be nibble precise, given appropriate profiling.

B. Results

To support our method, we have conducted experiments showing the possibility of distinguishing ΔS.

Fig 4 shows differences in power consumption captured in the second addRoundKey, by calculating Δt. The implementation we used computes the addRoundKey in a reverse order, therefore the difference peaks follow this order. In order to improve signal-to-noise ratio and produce clear plots, both t and t' were averaged from multiple executions. Nevertheless, it was possible to see the difference and raw traces. By observing this difference, the output difference of sBoxLayer in round 1, i.e. ΔS_i, can be clearly distinguished. Once ΔS_i and Δp_i are known, it can be used to solve the value of secret key k_i, as shown in the previous section.

C. Automatic Recognition of the Difference

If we look at Figure 4, it is obvious that determination of differences can be automated. For this purpose, one can use a simple algorithm that sets the threshold, enabling the separation of interesting areas from the random noise. Following steps can be done in order to make the automatic difference recognition:

Step 1: Calculate the mean (μ) and the standard deviation (σ) from the difference trace Δt, however both traces t

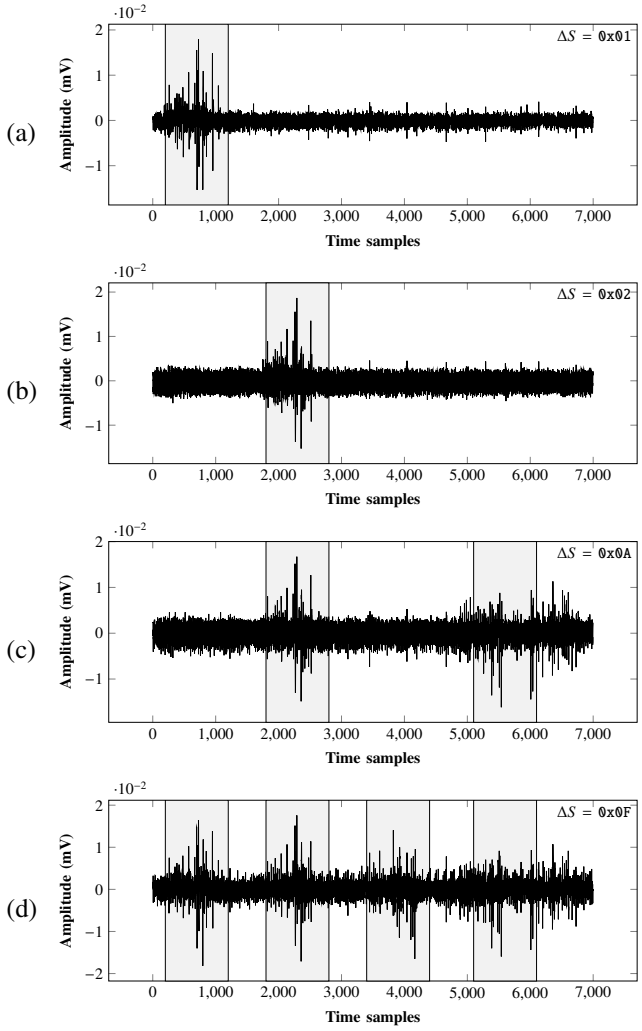

Fig. 4: Difference traces Δt showing `addRoundKey` of round 2, revealing the output difference of the Sbox at round 1. Bytes are processed in a reversed order, therefore the pattern showing the difference also has to be reversed. Difference between the Sbox outputs for the plots (highlighted by the gray background) is as follows: (a) `0x01`, (b) `0x02`, (c) `0x0A`, and (d) `0x0F`.

and t' come from the same plaintext, i.e. there is no difference in the Sbox output.

Step 2: Set the basic positive and negative threshold to be $r_+ = \mu + \sigma$ and $r_- = \mu - \sigma$, respectively.

Step 3: Choose n to be a multiplier of r_+ and r_- in a following way: start from $n = 1$ and increment the value by 1 until all the points of the trace fall between the positive and negative threshold.

Step 4: Use these values for all the measured traces to indicate points where the difference traces cross the thresholds.

Step 5: By the density distribution of the crossing points, together with the timing, determine ΔS.

Thresholds for ΔS =`0x01` are stated in Figure 5.

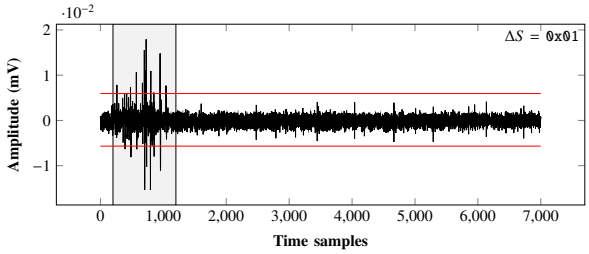

Fig. 5: Threshold that enables determination of the ΔS. Values used for thresholding were: $\mu = 3.452 \times 10^{-5}, \sigma = 1.453 \times 10^{-3}, n = 4$.

V. SCADPA Applied on Lightweight Cipher GIFT

`GIFT` [3] is a lightweight block cipher, designed to improve the efficiency and correct the weaknesses of `PRESENT`. Similar to `PRESENT`, `GIFT` is based on Substitution-Permutation Network (SPN), where each round is composed of substitution layer (`SubCells`), permutation layer (`PermBits`) and key addition (`AddRoundKey`). `GIFT` has 128-bit key and supports either 64-bit or 128-bit plaintext. The design of 4-bit Sbox (`SubCells`) and the permutation layer of `GIFT` is different from `PRESENT`, however, the idea of SCADPA can still be applied.

TABLE IV: `GIFT` SubCells.

x	0	1	2	3	4	5	6	7
$S(x)$	1	A	4	C	6	F	3	9
x	8	9	A	B	C	D	E	F
$S(x)$	2	D	B	7	5	0	8	E

Given the case of `GIFT`-64-128 (64-bit), the first 32 bits of the input to the permutation layer will go to bytes 0, 2, 4, and 6, whereas the other half will go to the remaining bytes. In general, the diffusion of each nibble after the `SubCells` can be formulated as shown in Table V. In summary, each bit in the nibble will go to different bytes in the subsequent round and the order of the targeted byte is as denoted in the table. Here, $[i_0, ..., i_{j-1}]_{\gg n}$ denotes right circular shift by n elements in the array. In this case, SCADPA can be applied using the same rationale as for the case of `PRESENT`-80, described earlier, since the affected bytes follow the same diffusion pattern.

One of the challenges is due to the key addition performed in `GIFT`. Here, the key is separated into eight 16-bit words. Two words are used in each round, that is, for `GIFT`-64-128, there will only be 32-bit key for each round. For each round, the key is extracted as follows: take the first two words, k_0 and k_1 (referred to as U and V, respectively). At the next round, the key is rearranged as follows:

$$k_7||k_6||\dots||k_1||k_0 \leftarrow k_1 \ggg 2||k_0 \ggg 12||\dots||k_3||k_2.$$

Here, $\ggg i$ is an i bits right rotation within a 16-bit word. The round keys U and V are then `xor`-ed with the state as follows: $b_{4i+1} \leftarrow b_{4i+1} \oplus u_i$ and $b_{4i} \leftarrow b_{4i} \oplus v_i$, $\forall i \in \{0, ..., 15\}$. A single bit (set to 1) and 6-bit round constant are also added to

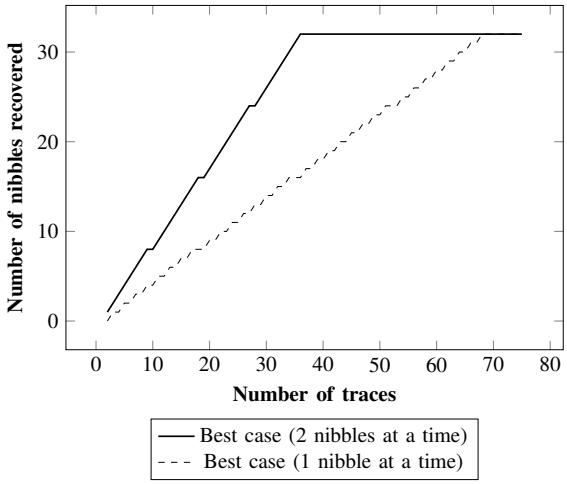

Fig. 6: The number of key nibbles retrieved by given number of traces attacking GIFT-64-128 targeting multiple rounds.

bits 63/127 (most significant bit, depending on the version of algorithm), 23, 19, 15, 11, 7, 3. Thus, for the attack, in each round, only 32 bits of the key can be recovered.

TABLE V: GIFT: affected bytes after bit permutation for each nibble.

Bits in targeted nibble i	Bytes affected in next round
$[b_0, b_1, b_2, b_3]_{0 \le i \le 7}$	$[0, 2, 4, 6]_{\gg (i \bmod 4)}$
$[b_0, b_1, b_2, b_3]_{8 \le i \le 15}$	$[1, 3, 5, 7]_{\gg (i \bmod 4)}$

In order to recover the rest of the key, the attack can then be performed on multiple consecutive rounds. Since the first round key has been recovered, to achieve the differential at the next round, the attacker can just "peel off" the first round and continue with SCADPA in the same way on the subsequent round. To recover the whole 128-bit key, this has to be repeated 4×, or 3× with brute-forcing the last 32 key bits. In Figure 6, we show the result of attacking GIFT-64-128. As can be seen in the plot, 68 traces are required to recover all the 128 key bits in case only one nibble is changed each time. In the best case, when two nibbles are targeted at a time, the attack requires 36 traces (4 reference traces and then 32 differential traces). This is due to the reason that for GIFT, on top of the longer key length, with each nibble we can only recover 2 bits, compared to 4 bits for PRESENT.

VI. Reverse Engineering of Secret Ciphers

As mentioned in the literature [15], one can still find solutions that are using secret cipher components despite the Kerckhoffs' principle. Often industry and agencies would use secret cipher for some specific applications. However, confidence in security of a cipher is only developed by detailled and lengthy analysis. This, for example, is the case of AES which has been extensively studied for over two decades without discovering a serious flaw. A common practice to design secret ciphers is to take a well studied cipher and replace an operation with a secret operation of equivalent security properties. Replacing Sbox of a well studied cipher like AES or PRESENT, with a secret Sbox of same cryptographic strength as the original, results in a new secret cipher.

For a PRESENT-like cipher, each round function consists of three operations in the following order: key addition, Sbox and bit permutation. In this section, we extend the SCADPA approach to recover secret components of PRESENT-like ciphers. The assumptions for the test scenario is as follows:

- Exactly one of the substitution layer or the permutation layer is replaced by a secret component, but not both.
- The attacker can feed chosen plaintext and observe the output ciphertext, without the knowledge of the secret key.
- All Sboxes are identical, which is important to keep the design lightweight.

In Section VI-A, we demonstrate how Sbox can be recovered using SCADPA with known bit permutation. Section VI-B then describes how to recover bit permutation without the assumption whether Sbox is known or not, by using a fault injection.

A. Recovery of Secret Sbox

Let us consider a PRESENT-like cipher using a secret Sbox, while the permutation layer is unchanged (or known). We describe a method that can reverse engineer the secret Sbox as well as recover the first round key with at most 46 measurements by using SCADPA – 16 for the first key nibble and 30 for the rest (this can be optimized to 17 according to Section III-B). Following the above notations, let S denote the secret Sbox operation and let k_0 denote the first nibble of the first round key. The 16 measurements which use all the possible values of p_0 give us an array of Sbox output differences, say `dif`, such that

$$\texttt{dif}[i] = S(k_0) \oplus S(k_0 \oplus i).$$

Note that in the case $S(0) = 0$ and $k_0 = 0$, the array `dif` is the unique solution for the secret Sbox. All the possible solutions for the Sbox and corresponding k_0 can be calculated using Algorithm 1. Line 1 considers different values of k_0 and line 2 iterates through different values of $S(k_0)$. With each fixed pair of k_0 and $S(k_0)$, an Sbox solution is uniquely determined using `dif` (line 4). For each Sbox solution, there is a unique value for the first round key using the measurement results and SCADPA. This gives the secret Sbox as well as the secret key used in the original encryption algorithm.

In the following, we describe the steps to reverse engineer the Sbox in detail:

Step 1: Run SCADPA for 16 different values of the first plaintext nibble p_0.

Step 2: Run SCADPA in a normal way for the rest of the nibbles $p_i; 1 \le i \le 15$, i.e. with 2-3 different values of p_i.

Step 3: After recovering the differences, obtain the `dif` array and run Algorithm 1.

Algorithm 1: Calculate all Sbox candidates and corresponding k_0

```
Input : dif: array of output differences
Output: ARRAY A of (Sbox,key): array of tuples of
        candidate Sboxes and corresponding k0 value
for int k: 0 to 15 do
    for int m: 0 to 15 do
        for int i: 0 to 15 do
            newSbox[i ⊕ k] = dif[i] ⊕ m;
        A.add(newSbox,k);
return A;
```

Step 4: For each pair of Sbox and first nibble of the first round key returned by Algorithm 1, using the information from Step 2, we can construct a cipher. In total we have 256 potential cipher solutions. In this way, each candidate for Sbox will also yield a unique candidate for the first round key.

Step 5: Run each of the ciphers with the given plaintext and compare the ciphertext with the one obtained from the original algorithm. If the ciphertexts are equal, the Sbox value and the first round key value can be determined.

When it comes to reverse engineering of secret Sbox used in `GIFT`, the measurement part works in the same way as explained before. However, there is an advantage for the attacker – `GIFT` does not use pre-whitening key, therefore, the input to `SubCells` layer in the first round is known to the attacker. This reduces the search space from 256 to 16 in Step 3.

Obviously, in case the permutation layer is not known to the attacker, she cannot use SCADPA. However, in the following part, we describe a way to reverse engineer this part of the cipher with a usage of fault injection. Since reverse engineering the permutation does not require knowledge of the Sbox, the attacker can first recover permutation using the following attack and then apply SCADPA for Sbox recovery.

B. Recovery of Secret Bit Permutation

Another operation that can be secret is the bit permutation. To successfully recover this layer, SCADPA alone is not sufficient since it can only identify the permutation with byte precision. This is because SCADPA is limited to initial rounds of the cipher, whose output is not observable by the attacker. A difference inserted at plaintext will be untraceable when traversing all the rounds of the cipher. Therefore, to recover the bit permutation, the difference must be inserted at later rounds. This can be achieved by specialized equipment often used in fault injection attacks. The attacker introduces differences in the last round of the cipher and observes the changes in the ciphertext. Therefore, it is similar to differential fault analysis, but only targeting the last permutation layer of the cipher.

We assume the bit permutation is of optimal diffusion. As it turns out, this property can be helpful in designing an efficient reverse engineering method. More specifically, the structure of the four Sbox groups is the same, meaning that as long as the secret design uses optimal diffusion, it is sufficient for the attacker to get information about one group to know the whole permutation. Along with the initial assumptions, we assume a fault model where the attacker can introduce bit flip faults in the last round at the Sbox output before the permutation layer.

By flipping bits before the permutation layer, the attacker can identify fault propagation in the ciphertext, revealing the bit position before the permutation. She just repeats this 16×, for the whole Sbox group and then uses this information to identify the whole permutation layer. Note that in case the permutation is not regular, the attacker has to flip all the bits in the state, resulting to 64 faults.

VII. Extension of SCADPA to 32-bit Architectures

When considering the attack on 32-bit architectures, one has to distinguish between different possibilities of implementing the round function. In case the operations are computed on 4/8-bit blocks, the attack can be carried out in the same way as in Section III-A. However, in case `addRoundKey` is computed as xor of 32-bit blocks, the attacker gains less information about the processed data and therefore, an updated attack strategy has to be used. This behavior is depicted in Figure 7, where one can easily observe that two output bits of each Sbox go to one word while the other two bits go to another word. In this part, we will describe the method of use for such case.

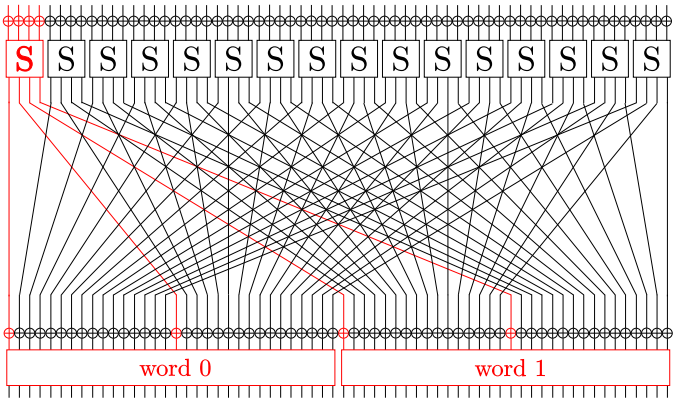

Fig. 7: `PRESENT` implemented on 32-bit architecture, where `addRoundKey` is computed on 32-bit words. One can easily see that from one Sbox, the output bits are split equally between the two words.

When the attacker obtains the difference of the traces, Δt, she can distinguish three possible cases when observing the `addRoundKey`:

- only `word0` has changed, indicated by $Y|N$,
- only `word1` has changed, indicated by $N|Y$,
- both words have changed, indicated by $Y|Y$.

Step 6 in SCADPA procedure (Section III-A) will be changed as follows:

Step 6: By examining Δ_t, get the change of Sbox output between $S(x_i)$ and $S(x_i')$, which is denoted by $Y(N)|Y(N)$ as indicated above.

For PRESENT Sbox, the information the attacker can get is stated in Table VI, which represents a DDT with output difference following the three cases described above. Now, she can construct an optimal attack strategy that will require the lowest number of traces, depending on the chosen plaintext. She will iterate through possible combinations of input differences until she finds a combination that yields a unique solution for the Sbox input. To be more specific, for PRESENT Sbox, the minimal number of input differences needed is 4. One such combination is stated in Table VII, for differences `0x3, 0x7, 0xd, 0xe`. For example, the attacker first measures the trace for $p = 0$, which will serve as a reference trace. Then to attack ith nibble of the first round key she measures the traces for p_i =`0x3, 0x7, 0xd, 0xe`, respectively. The four differences between the four traces and the reference trace will uniquely determine the input of Sbox for the first encryption, i.e. $p_i \oplus k_i$. Considering that attacker needs one reference trace and 4 other traces per nibble, the total number of chosen plaintexts to fully recover the round key is 65.

A. More Precise Attacker Model

Now, let us consider a more powerful attacker where she is able to distinguish not only whether there is a change in each of the 32-bit words, but also what is the Hamming weight of such change. This knowledge can help her distinguish how many bits of the Sbox output were changed and therefore, she needs lower number of encryptions to recover the key. Let us denote this difference as $\Delta_{HW_j} = HammingWeight(word_j \oplus word'_j)$, where $j \in 0, 1$ is index of the word. Thus for two different plaintext nibbles p_i, p'_i, Δ_{HW_0} and Δ_{HW_1} are the Hamming weights of the first and last two bits of $S(k_i \oplus p_i) \oplus S(k_i \oplus p'_i)$ respectively. Step 6 in SCADPA procedure (Section III-A) will be changed as follows:

Step 6: By examining Δ_t, get the change of Sbox output between $S(x_i)$ and $S(x'_i)$: Δ_{HW_0} and Δ_{HW_1}.

For PRESENT Sbox, Table VIII shows the DDT for this case – columns denoting the input difference and rows denoting Δ_{HW_i} for each word. One can easily see that for such case, the attacker only needs two difference traces on top of the reference trace to uniquely identify the Sbox input. Therefore, she needs 33 encryptions in total, same as the single nibble attack described for 8-bit architecture.

B. Reverse Engineering

Now let us consider the reverse engineering problem described in Section VI-A for 32-bit architecture. In this case, the array `dif` cannot be obtained directly. However, the attacker can still calculate different candidates for `dif` using the same amount of SCADPA measurements as detailed in Step 1 and 2 of Section VI-A. In this section, we illustrate how to achieve this for each of the two attacker capabilities discussed above.

In case the attacker can observe the change of `word0` and `word1` as described in the beginning of this section, instead of `dif`, the attacker can obtain an array $\texttt{dif}_{YN}$ of length 15, such that

- $\texttt{dif}_{YN}[i]$ =Y|N if the first two bits of $S(k_0) \oplus S(k_0 \oplus i)$ are non-zero and the second two bits of $S(k_0) \oplus S(k_0 \oplus i)$ are zero;
- $\texttt{dif}_{YN}[i]$ =N|Y if the first two bits of $S(k_0) \oplus S(k_0 \oplus i)$ are zero and the second two bits of $S(k_0) \oplus S(k_0 \oplus i)$ are non-zero;
- $\texttt{dif}_{YN}[i]$ =Y|Y if the first two bits of $S(k_0) \oplus S(k_0 \oplus i)$ are non-zero and the second two bits of $S(k_0) \oplus S(k_0 \oplus i)$ are non-zero.

Due to the fact that Sbox is a bijective function on $\mathbb{F}_2^4$, for any Sbox S, the values $S(k_0) \oplus S(k_0 \oplus i)$ for $i = 1, 2, \ldots, 15$ are always a permutation of the 15 values of $1, 2, 3, \ldots, 15$. Thus, in the array $\texttt{dif}_{YN}$, we will have exactly 3 of $Y|N$, 3 of $N|Y$ and 9 of $Y|Y$. Furthermore, for any i, we have the following observations:

- if $\texttt{dif}_{YN} = Y|N$, there are 3 possibilities for $S(k_0) \oplus S(k_0 \oplus i)$: $1000, 0100, 1100$;
- if $\texttt{dif}_{YN} = N|Y$, there are 3 possibilities for $S(k_0) \oplus S(k_0 \oplus i)$: $0010, 0001, 0011$;
- if $\texttt{dif}_{YN} = Y|N$, there are 9 possibilities for $S(k_0) \oplus S(k_0 \oplus i)$: $0101, 0110, 0111, 1001, 1010, 1011, 1101, 1110, 1111$.

Hence, for the array $\texttt{dif}_{YN}$, the attacker has $3^3 \times 3^3 \times 9^9 = 3^{24}$ possible solutions for `dif`. Then she can continue with steps 3-5 as in Section VI-A. In this way, the number of ciphers she needs to check is $3^{24} \times 2^8 \approx 2^{46}$. This still stays with in the brute force complexity.

Now, we consider the case the attacker can observe Hamming weight of the differences as described in Section VII-A. Then, from the SCADPA measurements she can obtain an array $\texttt{dif}_{HW}$ of length 15, such that $\texttt{dif}_{HW}[i] = \Delta_{HW_{i0}} | \delta_{HW_{i1}}$, where $\Delta_{HW_{i0}}$ and $\Delta_{HW_{i1}}$ are the Hamming weights of the first and last two bits of $S(k_i) \oplus S(k_i \oplus i)$ respectively. By a similar argument as above, there are respectively $4, 2, 2, 2, 2, 1, 1, 1$ permutations of $11, 12, 21, 10, 01, 20, 02, 22$ in the array $\texttt{dif}_{HW}$. And for each i, the possible values of $S(k_0) \oplus S(k_0 \oplus i)$ are respectively $4, 2, 2, 2, 2, 1, 1, 1$ for $11, 12, 21, 10, 01, 20, 02, 22$. Hence, in this case, the number of ciphers the attacker needs to check is $2^{16} \times 2^8 = 2^{24}$.

VIII. Discussion

In this part we will first discuss how side-channel countermeasures affect the successful application of SCADPA in Section VIII-A. Section VIII-B then explains how a fault attack countermeasure based on redundancy can increase implementation vulnerability w.r.t. SCADPA. Next, we show possibilities of attacking different block cipher modes of operation in Section VIII-C. Finally, in Section VIII-D we state alternatives for permutation layer as well as countermeasures that prevent successful application of SCADPA.

A. Side-Channel Countermeasures

As SCADPA exploits leakage of bit permutation through side-channel, any countermeasure which randomizes side-channel information and/or decreases signal-to-noise ratio can protect against such attacks. This includes both hiding [18] and masking countermeasures [19]. However, any bias in the

TABLE VI: Extended difference distribution table for PRESENT Sbox, where the columns follow the input difference, rows follow the change of the two output words and entries represent the Sbox inputs – w0 denotes word0, w1 denotes word1.

δ / w0|w1	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$N\|Y$	cdef	46	129a		36	8e	03478f	3b	09	05af	5e	1d	15678abc	279c	24bd
$Y\|N$		8a	47	26	01458d	179f	abcd		3567acef	4e	02389b	0c	2f	35bd	169e
$Y\|Y$	012345 6789ab	012357 9bcdef	03568 bcdef	0134578 9abcdef	279a bcef	02345 6abcd	12569e	0124567 89acdef	1248bd	12367 89bcd	1467 acdf	234567 89abef	0349de	0146 8aef	0357 8acf

TABLE VII: Example of output differences for input difference 0x3, 0x7, 0xd, 0xe. White color indicates $Y|N$, red color $N|Y$, and black color indicates both words changed – $Y|Y$. Each row is unique, and therefore, enables recognition of the Sbox input.

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

implementation of the countermeasure can still render the attack possible. For instance, an unbalanced implementation of hiding will still allow to observe the difference. Similarly for masking, ΔS would depend upon $p_i \oplus m_i \oplus p'_i \oplus m'_i$. If masks m_i, m'_i are totally independent and uniformly distributed, the attack is not possible, however with biases in the mask, the attack can still be carried out with increased effort. Shuffling of the order of the Sboxes and/or key additions would make the attack harder since only the hamming weight could be directly observed, instead of the difference value.

Nevertheless, countermeasures can incur significant overheads. Thus, for lightweight cryptography specially oriented for low-cost platforms, the functions must be carefully chosen to avoid such vulnerabilities.

B. Internal Redundancy Countermeasure

Recently, a software countermeasure against fault attacks, called Internal Redundancy Countermeasure (IRC) [17], was proposed. IRC utilizes a redundancy within an instruction to protect against simple data faults and instructions skips. To do this, data blocks, together with reference blocks are duplicated and concatenated within one instruction, as can be seen in Figure 8. While the data blocks contain user data that is being encrypted, the reference blocks are protecting against instruction skips, and are not interesting for the attacker since their value is known. The proposal was aimed at 32-bit

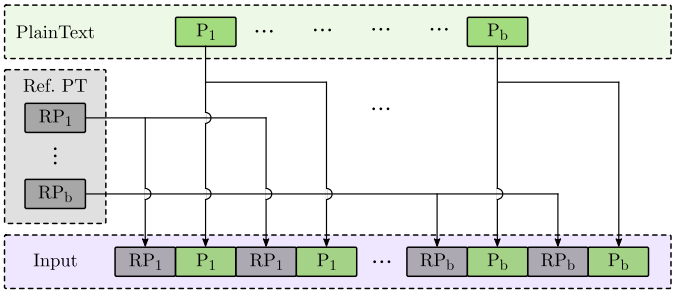

Fig. 8: Block concatenation for Internal Redundancy Countermeasure (IRC).

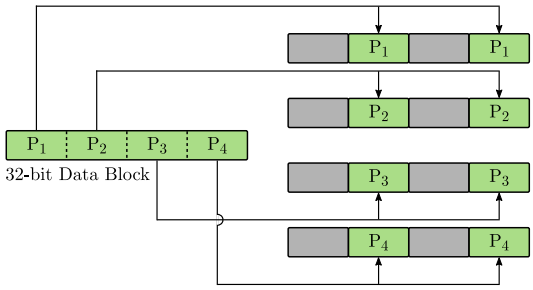

Fig. 9: Construction of four redundant blocks out of one 32-bit block according to IRC.

architectures, leading to four blocks of 8-bits per instruction. Therefore, one 32-bit data block is split into four 32-bit redundant blocks, as stated in Figure 9.

Such countermeasure therefore effectively transforms a 32-bit implementation into an 8-bit one in the view of SCADPA methodology. Splitting the data to smaller chunks means they can be analyzed separately in the same way as in the case of 8-bit implementation stated in Section III-A. In case of PRESENT, it means that the difference in addRoundKey will be distinguishable according to Figure 2.

This behavior can be seen in Figure 10, which shows the difference between the unprotected implementation and IRC protected implementation of PRESENT on 32-bit architectures. According to results from Section VII, this requires 65 traces in total for unprotected 32-bit implementation. However, only 17 traces will be required in case of the IRC protected implementation. This is a common example where fixing one vulnerability open doors for other vulnerabilities.

C. Attack on Different Modes of Operation

Few block cipher operation mode are well oriented towards plaintext selection of SCADPA. When it comes to Counter

TABLE VIII: Extended difference distribution table for PRESENT Sbox, where the columns follow the input difference and rows follow the Hamming weight of the two output words.

δ / $\delta_{HW_0}\|\delta_{HW_1}$	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0|1			9a		36	8e	03478f		09	5f	5e	1d	167abc	2c	24bd
1|0			47		8d	17	ad		356acf	4e	02389b	0c	2f	5b	169e
1|1	0145	029bcdef	56cdef	0134579bdf	9bce	06	25	2a	18	23789d	17ac	34678abf	3e	468a	
0|2	cdef	46	12					3b		0a			58	79	
2|0		8a		26	0145	9f	bc		7e					3d	
1|2	67ab		038b	8c	27	35ac		1569de			4f	2e	049d	1f	
2|1	2389	1357		ae	af			4c	24bd	16bc	6d	59		0e	078f
2|2						24bd	169e	078f							35ac

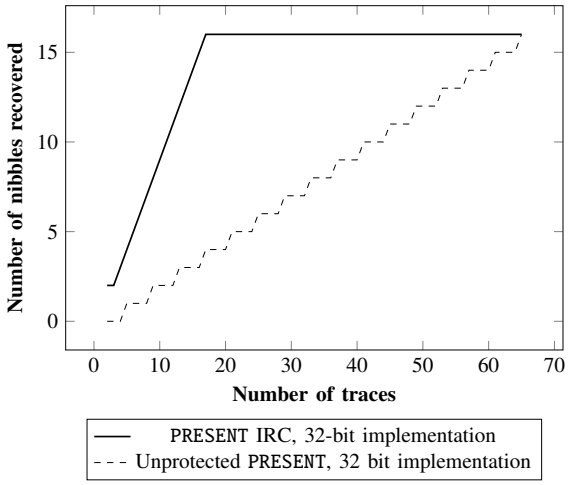

Fig. 10: Number of nibbles recovered for unprotected implementation and for the one protected with IRC for different number of traces.

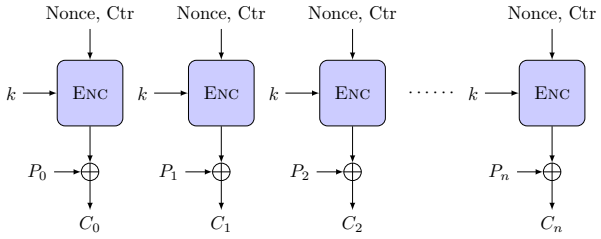

Fig. 11: Counter (CTR) mode of operation (encryption) [20].

(CTR) mode (Figure 11), the input to the encryption algorithm consists of a nonce and a counter. While the nonce is a random number, counter normally increases by 1 after each block. While the nonce stays fixed, the incrementing counter satisfies the chosen plaintext criteria of SCADPA as discussed in Section III. The attack allows to recover few nibbles directly affected by the counter in the first round. For the remaining nibbles, chosen nonce or attack on second round can be carried out in a similar way.

On the other hand, when targeting Cipher Block Chaining (CBC) mode, one has to aim at the decryption module (Figure 12). This comes from the property of CBC where the plaintext is first xor-ed with the IV (first block) or with ciphertext from the previous block. In this case, chosen-

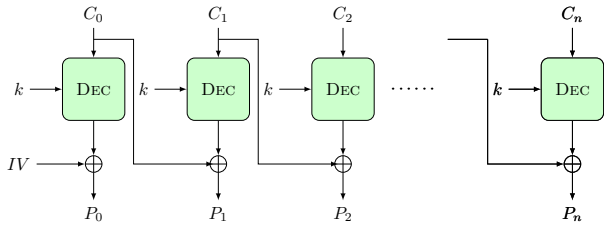

Fig. 12: Cipher Block Chaining (CBC) mode of operation (decryption) [20].

plaintext attack changes to chosen-ciphertext, without the knowledge of plaintext. The same hold for Propagating Cipher Block Chaining (PCBC) mode.

D. Alternatives for Diffusion Layer

Certain diffusion function do not offer vulnerabilities that are exploited by SCADPA in bit permutation. AES [21], the *NIST* standard for symmetric key cryptography, is a relevant example. AES encrypts 128-bit data block with a 128/192/256 bits secret key in 10/12/14 rounds. The data is organised in a 4×4 matrix of bytes called state and the round function is applied upon it. A round comprises of four operations i.e. SubBytes (SB), ShiftRows (SR), MixColumns (MC) and AddRoundKeys (ARK). SB is a 8×8 non-linear table look up and ARK is the round key addition. We concentrate on diffusion functions i.e. SR and MC. SR applies a cyclic shift on the rows (1, 2, 3, 4) with offsets (0, 1, 2, 3). MC operates on four bytes of each column of the state. The four bytes are combined using an invertible linear transformation. When a difference is inserted at the input, irrespective of its value, the difference is always propagated on all four bytes, preventing ΔS leakage. This is illustrated in Figure 13.

Recent trends show that it is possible to design a lightweight cipher with similar diffusion function and not only rely on bit permutations. SKINNY [22], PRINCE [23] are common examples. Other ultra-lightweight ciphers like SIMON and SPECK [24], use several bit shifts applied to a partial intermediate state to provide the diffusion. Furthermore, only a binary operation is used to provide non-linearity. Both operations combined would prevent a successful application of SCADPA.

IX. Conclusions

In this paper, we identify a vulnerability in bit permutation based lightweight ciphers (PRESENT, GIFT, etc) and develop a

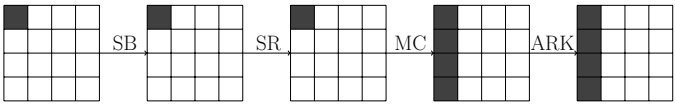

Fig. 13: Difference Diffusion in AES.

side-channel assisted methodology called SCADPA to exploit it. With a practical case study on low-cost microcontroller running PRESENT-80, we were able to practically recover the secret key with as low as 17 encryptions and an exhaustive search with complexity of 2^{16}. In case of GIFT, the number of encryptions in the best case was 36. We extend the methodology to enable the recovery of secret Sboxes in PRESENT-like ciphers. We further discuss how the application of redundancy-based fault attack countermeasures increases success rates of SCADPA. Several attacker models are presented, with different complexities of retrieving the key. To avoid the presented attacks, usage of more complex yet low-cost diffusion function is encouraged.

In the future work, it would be interesting to look at possibilities of exploiting different side-channel countermeasures. Especially, if randomness in masking is biased or the leakage characteristics of hiding are not uniform.

References

[1] K. A. McKay, L. Bassham, M. S. Turan, and N. Mouha, "Report on lightweight cryptography," *NIST DRAFT NISTIR*, vol. 8114, 2016.

[2] 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 *CHES*, vol. 4727. Springer, 2007, pp. 450–466.

[3] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A Small Present," *Cryptographic Hardware and Embedded Systems-CHES*, pp. 25–28, 2017.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *ArXiv e-prints*, Jan. 2018.

[5] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in cryptology - CRYPTO'99*. Springer, 1999, pp. 789–789.

[6] K. Schramm, T. Wollinger, and C. Paar, "A new class of collision attacks and its application to DES," in *FSE*, vol. 2887. Springer, 2003, pp. 206–222.

[7] C. Clavier, Q. Isorez, and A. Wurcker, "Complete SCARE of AES-Like Block Ciphers by Chosen Plaintext Collision Power Analysis," in *INDOCRYPT*, vol. 8250. Springer, 2013, pp. 116–135.

[8] N. Veyrat-Charvillon and F.-X. Standaert, "Adaptive chosen-message side-channel attacks." in *ACNS*, vol. 6123. Springer, 2010, pp. 186–199.

[9] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," *Computer Security - ESORICS 98*, pp. 97–110, 1998.

[10] B. den Boer, K. Lemke, and G. Wicke, "A DPA Attack against the Modular Reduction within a CRT Implementation of RSA," in *CHES*, vol. 2523. Springer, 2002, pp. 228–243.

[11] L. Guo, L. Wang, D. Liu, W. Shan, Z. Zhang, Q. Li, and J. Yu, "A chosen-plaintext differential power analysis attack on HMAC-SM3," in *Computational Intelligence and Security (CIS), 2015 11th International Conference on*. IEEE, 2015, pp. 350–353.

[12] O. Reparaz and B. Gierlichs, "A first-order chosen-plaintext dpa attack on the third round of des," 2017, https://eprint.iacr.org/2017/1257.

[13] R. Novak, "Side-channel attack on substitution blocks," in *ACNS*, vol. 2846. Springer, 2003, pp. 307–318.

[14] M. Rivain and T. Roche, "SCARE of secret ciphers with SPN structures," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2013, pp. 526–544.

[15] C. Clavier, Q. Isorez, D. Marion, and A. Wurcker, "Complete reverse-engineering of aes-like block ciphers by scare and fire attacks," *Cryptography and Communications*, vol. 7, no. 1, pp. 121–162, 2015.

[16] J. Breier, D. Jap, and S. Bhasin, "SCADPA: Side-Channel Assisted Differential-Plaintext Attack on Bit Permutation Based Ciphers (To be published)," in *Design, Automation and Test in Europe (DATE), 2018*. IEEE, 2018, pp. 1–6.

[17] B. Lac, A. Canteaut, J. J. Fournier, and R. Sirdey, "Thwarting fault attacks using the internal redundancy countermeasure (irc)," Cryptology ePrint Archive, Report 2017/910, 2017, https://eprint.iacr.org/2017/910.

[18] P. Hoogvorst, J.-L. Danger, and G. Duc, "Software Implementation of Dual-Rail Representation," in *COSADE*, 2011, Darmstadt, Germany.

[19] 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.

[20] J. Jean, "TikZ for Cryptographers," https://www.iacr.org/authors/tikz/, 2016.

[21] N. F. Pub, "197: Advanced encryption standard (AES)," *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[22] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY Family of Block Ciphers and its Low-Latency Variant MANTIS," Cryptology ePrint Archive, Report 2016/660, 2016, http://eprint.iacr.org/2016/660.

[23] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger *et al.*, "PRINCE–a low-latency block cipher for pervasive computing applications," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2012, pp. 208–225.

[24] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, "The SIMON and SPECK lightweight block ciphers," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.