2023 Design, Automation & Test in Europe Conference (DATE 2023) – Best Paper Award Candidate

Non-Profiled Side-Channel Assisted Fault Attack: A Case Study on DOMREP

Sayandeep Saha, Prasanna Ravi, Dirmanto Jap and Shivam Bhasin
School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
{sayandeep.saha,prasanna.ravi,djap,sbhasin}@ntu.edu.sg

***Abstract*—Recent work has shown that Side-Channel Attacks (SCA) and Fault Attacks (FA) can be combined, forming an extremely powerful adversarial model, which can bypass even some strongest protections against both FA and SCA. However, such strongest form of combined attack comes with some practical challenges – 1) a profiled setting with multiple fault locations is needed; 2) fault models are restricted to single-bit set-reset/flips; 3) the input needs to be repeated several times. In this paper, we propose a new combined attack strategy called SCA-NFA that works in a non-profiled setting. Assuming knowledge of plaintexts/ciphertexts and exploiting bitsliced implementations of modern ciphers, we further relax the assumptions on the fault model, and the number of fault locations – random multi-bit fault at a single fault location is sufficient for recovering several secret bits. Furthermore, the inputs are allowed to be varied, which is required in several practical use cases. The attack is validated on a recently proposed countermeasure called DOMREP, which individually provides SCA and FA protection of arbitrary order. Practical validation for an open-source masked implementation of GIMLI with DOMREP extension on STM32F407G, using electromagnetic fault and electromagnetic SCA, shows that SCA-NFA succeeds in around 10000 measurements.**

***Index Terms*—Fault Attack, Side-Channel, Masking**

I. Introduction

Fault Attacks (FA) [1] and power/electromagnetic (EM) Side-Channel Attacks (SCA) [2] have shown enormous potency in violating the security and privacy of cryptographic algorithms. In FA, the secret is recovered by inducing a controlled fault during the computation and then by observing faulty system responses (e.g., faulty/correct ciphertexts). SCA attacks, on the other hand, passively observe the power/EM signals from the computing device and exploit their correlation with the intermediate data being processed. Countering these attacks have been an active area of research [1], [3]–[6]. Attack strategies are also being improvised to bypass such countermeasures.

The most common way of countering SCA attacks is to randomize the computation or data. Masking [3], [4] is one of the most widely adopted concepts in this regard, which realizes secret-sharing at the level of circuits. Broadly, each variable in a computation is represented using multiple uniformly random variables known as *shares*. The functions operating on these variables are also split into component functions. Any proper subset of these shares does not reveal information about the actual unshared variable at any stage of computation. As a result, the adversary would need a *higher-order* statistical analysis, with exponentially increasing trace requirements, to perform key recovery. FA countermeasures, on the other hand, utilize some form of redundancy in computation and data to detect/correct the faults. The easiest approach is to compute the same data at least twice and compare the results at the end so that no faulty ciphertext can reach the output. More sophisticated approaches use linear block codes and fine-grained checks, e.g., after each round of a block cipher/permutation [1].

Recent attacks, such as Statistical Ineffective Fault Analysis (SIFA) [7], [8], have shown that most of the existing redundancy-based FA countermeasures can be bypassed. SIFA utilizes events in which injected faults remain ineffective on the target variable (depending on the valuation of the variable) and result in correct ciphertexts. Evidently, the redundancy-based countermeasures detecting the faults does not work. Similarly, a recently proposed attack called FTA [9] can also bypass such countermeasures by only utilizing the (single-bit) information whether a ciphertext is faulty or not. Countering SIFA has been an active area of research since its proposal. It has been shown that a combination of masking and error-correction for each share can counter SIFA attacks [5]. There exists multiple variants of this proposal, one being the DOMREP [10], which utilizes Domain-Oriented Masking (DOM) [4] and duplication code based Error-Correction-Code (ECC) and individually claims security against SCA and FA. Another line of work utilizes carefully crafted masking to make every fault effective, thus removing ineffective faults [6].

Since SIFA protections include both SCA and FA countermeasure (and claims individual security against each), a natural question is whether it can also resist an attack when SCA and FAs are combined together. Recent work has shown that combined attacks are feasible on such SIFA countermeasures using some variants of FTA combined with SCA [11]. However, the adversarial setting in such combined attacks are rather restricted. Firstly, they require a profiled setting for both fault and side-channel (i.e., the attacker possess a device similar to the target on which he can build SCA and fault templates to be used later for attack). Having access to a profiling device is not feasible in many situations. Secondly, FTA attacks utilize information from several fault locations. In many cases only one bit of secret is recovered per fault location. It is tedious and challenging in many practical settings to find out the exact parameters for several different fault locations, especially with low-cost and less precise injection setups. Thirdly, FTA and its SCA-enhanced variants rely on single-bit faults. This

Authors acknowledge the support from the Singapore National Research Foundation ("SOCure" grant NRF2018NCR-NCR002-0001 – www.green-ic.org/socure

978-3-9819263-7-8/DATE23/© 2023 EDAA

is once again a challenge for low-cost injection setups to achieve single-bit precision for several points. Finally, the SCA-enhanced FTA requires the input to be fixed throughout the attack. This is challenging to achieve in several practical situations. For example, in a nonce-respecting Authenticated Encryption (AE) [12], SCA-FTA cannot attack the encryption module as the nonce varies for every encryption.

Observing all these challenges associated with SCA-FTA, in this paper, we propose SCA-NFA (SCA-enhanced Non-profiled Fault Attack), a relaxed combined attack strategy, which: 1) works against SIFA countermeasures (and also without SIFA countermeasures). 2) It can recover several secret bits per fault location. Consequently, very few fault locations are required. 3) The attack can work even with random multi-bit faults, which further enhances its practical applicability with less-precise injection setups. We exploit the bitslicing present in modern ciphers/permutations [12], [13] to enable such multi-bit fault models. 4) Complementary to SCA-FTA, the input may vary randomly in SCA-NFA. However, the adversary needs access to either plaintexts or ciphertexts.

We practically validate the attack on a software implementation of DOMREP countermeasure [10] for GIMLI permutation [13]. The masked implementation of GIMLI is collected from an open-source code base [14], and we only include the error-correction module without tweaking the masking. The attacks are performed with EM-fault injection and EM SCA traces on an STM32F4-Discovery board. It is found that even in the presence of first-order masking and error-correction, the secret can be recovered within 9500-10000 traces on average, with one fault per encryption and first-order SCA[1].

II. Background

A. SIFA and SIFA Countermeasures

SIFA attacks exploit situations when a fault remains ineffective and results in a correct ciphertext. A fault may remain ineffective in several cases – for example, if a bit stuck-at-0 fault is injected at some intermediate state, it remains ineffective when the value assumed by the target bit is 0. Such ineffective faults result in correct ciphertexts. However, while such correct ciphertexts are partially decrypted by guessing some key bits, they result in a statistical bias at the intermediate state only for the correct key guess and, therefore, expose the correct key.

State-of-the-art detection countermeasures fail to detect such ineffective faults. The primary goal of existing SIFA countermeasures is, therefore, to mitigate ineffective faults. One crucial observation here is that most of the SIFA attacks rely upon data-dependent statistically-biased faults. For example, stuck-at-0 is a biased fault model, where the probability of a 0 to 1 transition of a bit with a fault is 0 ($Pr[0 \rightarrow 1] = 0$). In other words, not every possible corruption is equally likely, and whether or not corruption would happen depends on secret intermediates resulting in key recovery. In such cases, the randomization provided by masking schemes works as a good countermeasure against SIFA [5]. However, masking itself can be vulnerable in certain cases, where a fault might bias the S-Box mappings. The S-Box might not assume some of the values in its range (ref. Fig. 1(a)). Injections at intermediate locations of a masked S-Box create such situations [8]. For such cases, a detection/correction operation is needed to prevent ineffective faults for each share of masking. The countermeasure framework in [5] achieves this with ECC for each share after each S-Box layer of a cipher (ref. Fig. 1(b)). Furthermore, ECC operation is repeated multiple times after each S-Box to ensure that it cannot be bypassed with faults. The ECC is realized with duplication code which maintains multiple copies of a variable and performs a majority voting among them to correct the error. The AntiSIFA proposal in [5] instantiates the masking with Threshold Implementation [3], whereas the DOMREP proposal instantiates the masking with DOM [4]. In a different line of work [6], a masking implementation is modified with so-called *Toffoli gates* so that every fault reaches the output and becomes effective (ref. Fig. 1(c)). In this case, detection at the output can be used to prevent SIFA. There exists few other work such as Impeccable circuits [15] and [16] for SIFA protection. However, utilizing fine-grained error-correction/detection remains the central strategy.

B. FTA and SCA-FTA

FTA exploits the fact that fault propagation through logic gates is data-dependent. Referring to Fig. 2, a fault in one of the inputs (a) of a two-input AND gate only propagates to its output when the other input (b) takes a value 1. Consequently, the value assumed by b leaks depending on whether the fault at a propagates. For XOR gates, such propagation is not dependent on the other non-faulted inputs of the gate. Based on this data-dependent fault propagation, FTA forms "fault templates" in a profiling stage and exploits them to expose an intermediate state of a cipher in the online attack phase. In most of the cases, information from multiple fault locations are combined in a template to expose the intermediate states. The greatest advantage of FTA over contemporary FAs is that it does not require access to the ciphertexts, but only needs to know if the ciphertext is faulty or not. FTA enables middle round attacks on block ciphers, and works even in the presence of masking.

SCA-FTA is an enhancement of FTA which assumes that the adversary can measure the SCA leakage along with fault injection. While previous work on combined attacks [17], [18] conjectured that masking the detection/correction logic or the entire cipher (ref. SCADFA [19]) should prevent such attacks, SCA-FTA shows this to be still feasible by breaking contemporary SIFA countermeasures. The main idea behind the attack is to acquire fault propagation information from the error-detection/correction logic, which behaves differently with and without a fault, even while operating on masked data. The fault templates, in this case, are combined with SCA templates, and are used to expose an intermediate state of a cipher corresponding to a fixed input. The attack has been demonstrated practically on open-source SIFA-protected implementations with single-bit laser-induced faults [11]. Several fault locations are required (only one location is excited per encryption/permutation) for this attack. Another work on com-

[1] https://github.com/sayandeep-iitkgp/SCA-NFA.git

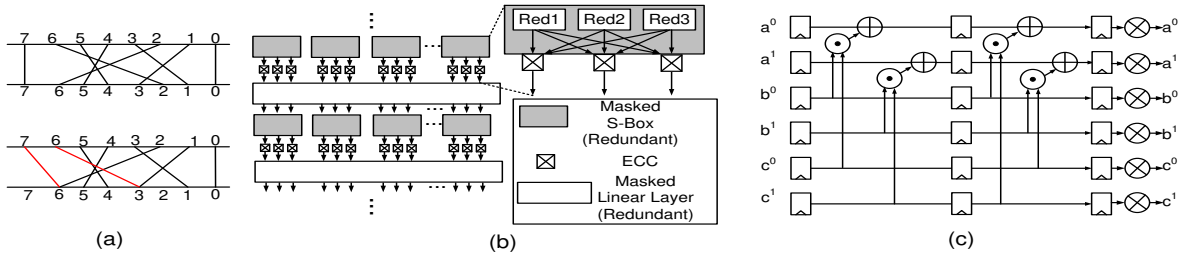

Fig. 1: SIFA and SIFA countermeasures: (a) SIFA faults biasing the S-Box mappings (3-bit χ_3 S-Box); b) SIFA countermeasure with error-correction on each share after the S-Box computation [5]; c) A 2-share Toffoli gate where every fault propagates to the output [6] ($\otimes$ denote the detection operation.). For detection, two redundant copies of the depicted circuit are present implicitly.

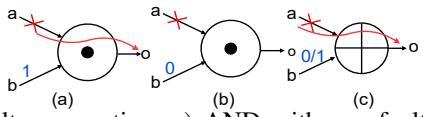

Fig. 2: Fault propagation: a) AND with non-faulted input set to 1. b) AND with non-faulted input set to 0; c) XOR.

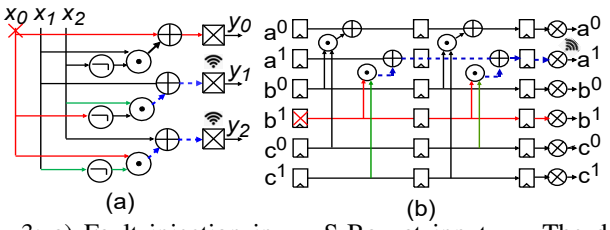

Fig. 3: a) Fault injection in χ_3 S-Box at input x_0. The data-independent fault propagations are shown in red. The data-dependent propagations are shown in blue dotted lines. The green wires control the propagation.; b) Data-dependent fault propagation for a masked Toffoli gate and associate leakage.

bined attack uses leakage of correct ciphertexts [20]. However, the attack is mitigated by SIFA countermeasures.

III. Nonprofiled Combined Attack

A. Information Leakage from Fault Propagation

In this section, we propose a non-profiled combined attack (SCA-NFA), alleviating the restrictions imposed by SCA-FTA. Before describing the attack, we first discuss the leakage sources we exploit. Fig. 3(a) explains the nature of the leakage through an example. In this case, the fault injected at the x_0 input of a 3-bit unmasked χ_3 S-Box [11] propagates through many of its outputs. However, the propagation through the output y_0 happens with probability 1 whenever there is an active fault at x_0 (i.e., the fault changes the value of x_0). In contrast, propagation through y_1 and y_2 are conditional on the inputs x_2 and x_1, respectively due to the data-dependent propagation through AND gates (Fig. 2). y_1 gets corrupted only when $x_2 = 1$, and y_2 gets corrupted only when $x_1 = 0$.

It is feasible to capture this data-dependent fault propagation in many ways. We focus on the cases where an FA countermeasure is associated. The simplest form of FA countermeasures involves a check operation at the end of the computation, which is usually implemented with XORs between the outputs of two independent executions. If one of the executions is faulted and has data-dependent fault propagation, the XOR output would vary depending on the propagation. Referring to the S-Box example, if we consider an error-detection circuit at each output, then the error will be detected at y_1 only when $x_2 = 1$. While this XOR outcome is never released at the output, it can be captured through SCA leakage exposing x_2.

A very similar situation arises if the detection logic is replaced by correction. Without loss of generality, let us consider a correction logic based on duplication and majority voting, as utilized in multiple SIFA countermeasures [5], [10]. The correction logic for single-bit errors works as follows:

$$y_i = y_{i0}y_{i1} + y_{i0}y_{i2} + y_{i1}y_{i2} \tag{1}$$

Here, (y_{i0}, y_{i1}, y_{i2}) denote three redundant copies of the same computation. "+" denote the XOR, and $y_{ij}y_{ij'}$ denote AND between y_{ij} and $y_{ij'}$. Now, depending on whether there is a fault in any one of (y_{i0}, y_{i1}, y_{i2}), the logic in Eq. (1) would behave differently. We consider a situation when the inputs to this correction logic are correct, i.e. either $(0, 0, 0)$ or $(1, 1, 1)$. The outputs of the three AND operations in Eq. (1) are either 0 or 1 in this case (i.e., the outputs are either $(0, 0, 0)$ or $(1, 1, 1)$ after the AND layer). However, in case one of the input bit is corrupted (e.g. the input is $(1, 0, 0)$ or $(0, 1, 1)$, the output of the first and second AND operation ($y_{i0}y_{i1}$ and $y_{i0}y_{i2}$) become 0 irrespective of the inputs. This helps in distinguishing whether the input to the correction logic is corrupted or not. In statistical attacks, where the inputs to the cipher and, therefore, to the correction logic vary, this observation creates a distinguishable signature on the SCA leakage.

Next, we examine the aforementioned result for masked implementations. Without loss of generality, we focus on Domain Oriented Masking (DOM) here as it has been utilized in DOMREP [10]. A DOM AND gate is expressed as follows:

$$\begin{aligned} q^0 &= x_0^0 x_1^0 + (x_0^0 x_1^1 + z) \\ q^1 &= x_0^1 x_1^1 + (x_0^1 x_1^0 + z) \end{aligned} \tag{2}$$

Here we compute an AND operation $y = x_0 x_1$ with this DOM AND. x_0 is randomly shared as x_0^0 and x_0^1, such that $x_0 = x_0^0 + x_0^1$. Shares of x_1 are (x_1^0, x_1^1) and of y are (q^0, q^1). z is a fresh random bit. The masked XORs in DOM are XOR between shares due to the linearity property of XOR.

Let us consider a fault injected at the x_0^0 input of this DOM AND. Referring to Eq. (2), the fault will only affect the first output bit q^0. Also, the fault propagation is conditioned on $(x_1^0 + x_1^1)$, i.e. the fault propagates only if $(x_1^0 + x_1^1) = x_1 = 1$. In other words, the fault combines the two random shares of x_1, which creates a data-dependent leakage and breaks the

masking security. Another important observation, in this case, is that fault propagates through only one share of the output. If we assume ECC/detection circuits at the end of each share (which is suggested in the SIFA countermeasures [10]), then utilizing the leakage from the ECC circuit of q^0 will give us information about an unmasked bit. In terms of masking security, this is equivalent to performing a first-order SCA attack (with faults) on a first-order masked implementation. Moreover, the information leakage we deal with generates first-order leakage irrespective of the masking order if DOM (and some other recent masking schemes such as PINI [21]) is used. In a nutshell, *with a single fault per execution and first-order SCA, one can break masked implementations of arbitrary orders with such information leakage* [11]. One may observe that the leakage remains valid also for the Toffoli-gate based building blocks used in [6] as SIFA countermeasure (ref. Fig. 3(b)). This structure represents a 2-share masked version of $y = a + bc$ and ensures that every fault injected at the input or intermediates indeed reaches the output. However, the fault propagated to a^1 for an injection at input b^1 depends on $(c^1 + c^0) = c$, even though there is a propagation to b^1 with probability 1.0 (Fig. 3(b)). Consequently, the proposed combined information leakage remains valid. So far, in this section, we have only discussed SIFA countermeasures. However, the leakage described here would trivially happen for most standard (non-SIFA) FA countermeasures (with/without masking). In such cases, it is always possible to generate a data-dependent (ineffective) fault, which will generate an exploitable SCA leakage while interacting with the mandatory error-detection logic at the end of the computation. The leakage should occur even for other choices for ECC [15].

B. SCA-NFA Attack

One way of exploiting the leakage described in the last subsection was proposed in SCA-FTA in a profiled setting. The plaintext was held fixed. Each of the inputs of an S-Box were utilized for injection (one location per encryption), and typically 1-2 bits of information (from the S-Box input) per fault location were extracted. For example, referring to Fig. 3(a), information about x_2 and x_1 can be recovered for this unmasked implementation in SCA-FTA. However, for masked implementations, since only first-order leakage is measured, only one bit (either x_2 or x_1) can be recovered for fault-injection in one of the shares of x_0. This indicates that several fault locations, that too with single-bit precision, are required for extracting a full intermediate state.

1) Reducing Fault Locations and Varying Input: SCA-NFA exploits the fact that the leaked intermediate bit is dependent on multiple secret bits, and instead of recovering the leaked intermediate, it exploits the leakage to recover these dependent bits. For example, if the leaked intermediate is the input to the last round S-Box of a block cipher, then it depends on multiple bits of the last round key during decryption. However, in order to exploit this dependency for key recovery, one must have access to plaintexts/ciphertexts and need to vary them. Varying plaintext allows to alleviate one of the major restrictions imposed by SCA-FTA. Moreover, it also allows the use of correlation and Difference-of-Mean (DM) distinguishers [2], which are commonly used in non-profiled SCA attacks. The non-profiled attack algorithm is presented in Algorithm 1. Here Dist denote some correlation/DM distinguisher. The PartDecrypt function calculates the leaked intermediate value (correspond to a fault location) for each (multi-bit) key guess kg. Since one fault location can recover multiple secret bits, the need for templating on multiple fault locations is also removed.

Algorithm 1 SCA-NFA Attack

Input: Plaintext/Ciphertext set $\mathcal{D}$, Corresponding leakage $\mathcal{L}$
Output: Secret k
1: $\mathcal{K} \leftarrow$ Guesskey() ▷ Guess the associated secrets
2: **for** $kg \in \mathcal{K}$ **do**
3: $\mathcal{S} \leftarrow$ PartDecrypt$(\mathcal{D}, kg)$ ▷ Partially decrypt
4: $G \leftarrow$ Dist$(\mathcal{S}, \mathcal{L})$ ▷ Dist: correlation/DM
5: **end for**
6: **return** $k \leftarrow \underset{kg \in \mathcal{K}}{\arg\max}\ G$

2) Relaxing the Fault Model: In general the fault propagation paths for single-bit faults are easier to interpret than their multi-bit counterparts. The information leakage in our attack has also been described with single-bit faults. However, due to typical structures of modern block-cipher/permutation implementations this single-bit restriction can be relaxed. Modern block ciphers/permutations are implemented in a bitsliced manner [12], [13]. Bitslicing helps in computing the S-Boxes in parallel for software implementations. The main idea is to pack bits in registers in a way so that the same set of operations are performed over all of the bits in a register. Taking advantage of the 32/64-bit data width of the registers, this can simultaneously process multiple S-Boxes in a cipher round. For example, on a 32-bit implementation, we can process 32 S-Boxes in parallel.

In SCA-NFA, we exploit the fact that the bits in a register are independent of each other, at least for a round. For example, all the least-significant bits of 32 different S-Boxes can be packed in one register. Therefore, faulting one of these registers, even with a 32-bit random fault model, creates 32 parallel (single-bit) fault propagation paths for 32 different S-Boxes. While the leakage from detection/correction would combine information for all these propagations, one can still exploit one of these propagation paths with correlation/DM distinguisher while considering others as algorithmic noise. Note that we only exploit leakage corresponding to the correction/detection logic of one share (first-order attack), which usually remains in one register (and its copies) in bitsliced implementations. To illustrate why the leakage would still remain informative, let us consider the Hamming Weight (HW) leakage model and an n-bit register $R = (r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_{n-1})$ associated with the error-detection/correction phase. Each r_i bit here carries information about one internal bit due to fault propagation. For simplicity, let us also assume that $r_i = 1$ if there is a fault propagation (it can be 0 or some pattern depending on the ECC). Now, a leakage trace having fault propagation to r_i can be represented as $\mathbf{t} = HW(r_0, r_1, \cdots r_{i-1}, r_{i+1}, \cdots r_{n-1}) + HW(r_i) + \mathcal{N}(\mu, \sigma)$. Here, $\mathcal{N}(\mu, \sigma)$ denote the Gaussian noise with mean μ and variance σ^2. One should note that depending on $HW(r_i) = 1$ or 0, the leakage will vary. The other HW component acts as algorithmic noise. If we consider the DM distinguisher,

the algorithmic noise component gets averaged out during the calculation of the difference of the means. These observations clearly establish that multi-bit faults can be exploited for combined attacks. In principle, SCA-FTA can also exploit such multi-bit faults. However, due to the fixed input, the removal of algorithmic noise is complex and requires a huge number of profiling and attack traces.

IV. Experimental Evaluation: Attacking DOMREP

The proposed SCA-NFA attack applies to several existing SIFA countermeasures (such as the one based on Toffoli-gates [6]) and also for non-SIFA protections. In this section, we focus on one SIFA countermeasure called DOMREP [10], which uses DOM gates and per-share ECC to implement the S-Boxes. The original DOMREP proposal was developed on the permutation of GIMLI AEAD [13], which motivates us to do the same. GIMLI is a sponge-duplex based AEAD construction, utilizing an iterative permutation of 384-bits. The permutation maintains a 3×4 state matrix S given as follows:

$$S = \begin{pmatrix} A_0 & A_1 & A_2 & A_3 \\ B_0 & B_1 & B_2 & B_3 \\ C_0 & C_1 & C_2 & C_3 \end{pmatrix} \quad (3)$$

Here each A_i, B_i and C_i is a 32-bit value. The state-update happens through Algorithm 2. In the DOMREP implementation, each of these 32-bit units are shared in two shares, and each share is duplicated three times to ensure first-order SCA and single-bit fault protection. Overall, the state size becomes $384 \times 2 \times 3 = 2304$ bits. Also, ECC is performed for each share at the end of each round.

Algorithm 2 GIMLI Permutation

for $r = 24$ to 1 **do**
  for $j = 0$ to 3 **do**
    $t_a \leftarrow A_j \lll 24$; $t_b \leftarrow B_j \lll 9$; $t_c \leftarrow C_j$ ▷ Fault location in red
    $A_j \leftarrow t_c + t_b + ((t_a t_b) \lll 3)$ ▷ Conditional propagation in blue
    $B_j \leftarrow t_a + t_b + ((t_a | t_c) \lll 1)$ ▷ Mandatory propagation in red
    $C_j \leftarrow t_a + (t_c \lll 1) + ((t_b t_c) \lll 3)$
  end for
  if $r \mod 4 == 0$ **then** $A_0||A_1||A_2||A_3 \leftarrow A_1||A_0||A_3||A_2$
  else if $r \mod 4 == 2$ **then** $A_0||A_1||A_2||A_3 \leftarrow A_2||A_3||A_0||A_1$
  end if
  if $r \mod 4 == 0$ **then** $A_0 \leftarrow A_0 + \texttt{0x9e377900} + r$
  end if
end for

We modify an open-source masked software implementation of GIMLI [14], by augmenting the ECC described in Eq. (1). The change in the code is minimal – we only need to copy each share three times with the same randomness and implement the ECC as described in Eq. (1). We do not tweak the masking.

A. The Attack

The attack is performed on the permutation at the initialization phase of the GIMLI AEAD, where the secret key is used along with the nonce. Each A_i in the GIMLI state is filled with the nonce, and the rest of the 256 bits are used as secret key. The nonce is accessible by the adversary and utilized by us as plaintext (varying randomly). The output of this permutation is not directly accessible. Also, since we implement DOMREP, every output would be corrected, and no information can be obtained solely from fault. But a combined attack strategy can exploit the leakage from the ECC of first few rounds.

The fault in our attack is injected at the beginning of round 22. Without loss of generality, we select the injection location as one share of t_a, and the fault is a 32-bit random fault. The SCA is first-order on the ECC leakage at the end of round 22. In Algorithm 2, one may observe that the faulted t_a is AND-ed with another value t_b. According to the principle of fault propagation, therefore, the fault at any bit of t_a will only propagate to the output of this AND if the corresponding bit in t_b is 1. This property holds irrespective of the masking, and bits of t_b are leaked through the ECC. There are total 32 bits in the faulted register, and for each of these bits, the aforementioned fault propagation principle is applied independently. Therefore, focusing on a single-bit for key recovery is sufficient. Without loss of generality, we choose the bit 7 of t_b for our hypothesis building. As shown in [22], this bit depends on total 11 key bits, among which 5 can be recovered uniquely. Also, a linear relation among the other 6 bits can be recovered. Algorithm 1 is applied with randomly chosen nonce and DM distinguisher, with the nonce partially encrypted till the leaked bit with the guessed key bits. The correlation distinguisher also needs a similar ciphertext count.

One interesting fact of 32-bit fault injection is that several other key bits can be recovered from the same fault and leakage measurement campaign. For example, in the aforementioned attack we utilized the 7th bit of a leaked state, but the other 31 bits can also be used independently to recover other key bits. More precisely, for a 32-bit injection, total 32 bits of the state are leaked, and exploited for key recovery, which exposes around $32 \times 6 = 192$ bits. However, there are overlaps among these recovered bits, and roughly 4 injection locations recover the entire secret. This is a huge gain over SCA-FTA, which requires one fault location for each key bit.

B. Experimental Evaluation

We evaluate the attack on an STM32F4-Discovery board with EM-based injection and EM-based SCA measurements. The implementation runs at 168 MHz. The injection pulse is generated from a pulse generator upon receiving a trigger signal from the target and propagated through an injection probe. Another EM probe is used for SCA measurements (controlled through a second trigger) with an oscilloscope with a 2.5 GHz sampling frequency. The entire setup is controlled from the oscilloscope through Python scripts. The schematic of the attack setup, and the probe positions on the chip, are shown in Fig. 5. In order to find a proper fault location, we profile the device for different spatial locations (controlled through an XYZ table), voltage, pulse-width, and delay values. It is observed that a voltage of 200 V, a pulse-width of 6 ns, and a delay of 120 – 125 ns are ideal for the attack in our case, providing 99% repeatable faults. To save storage, we only save the leakage corresponding to a particular ECC module. The sample leakage traces are depicted in Fig. 6(a) having 2500 points. To find out the Point-of-Interest (PoI), we compute the Signal-to-Noise Ratio (SNR) of the traces depicted in Fig. 6(b). The PoI is found at point 913 of the trace.

The attack experiments were performed in three phases – first, we simulate the attack with random 32-bit faults and

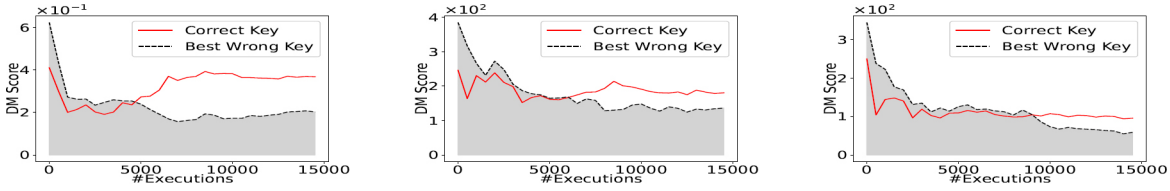

Fig. 4: Attack result: a) Simulated fault and leakage; b) Simulated fault and real leakage; c) Real fault and leakage. The plots are shown for 5-bit key recovery (averaged over 100 experiments). The same traces are utilized for recovering several other bits.

HW leakage (ref. Fig. 4(a)). The attack happens within 5000-5500 traces on average for 100 different experiments. Next, we simulate the fault and measure actual EM leakage from the device (ref. Fig. 4(b)), which takes 6500-7000 traces for key recovery. In the final phase, we inject real faults and measure actual leakage, which requires 9500-10000 traces for key recovery (ref. Fig. 4(c)).

A combined attack must ensure that the fault injection does not affect the SCA measurements significantly, making the SNR negligible. Generally, it can be ensured if the injection clock cycle is distant from the SCA measurement clock cycles. In our case (and for most software implementations), it is achieved easily as the injection happens at the beginning of a round, and measurements are taken at the end, which are several clock cycles apart. It is also evident from the SNR plots of real vs. simulated faults in Fig. 4(b), which shows that the differences between two SNRs are not very significant. We note that the SNR is low in general for the attacks as there is a lot of algorithmic noise. However, that does not hinder key recovery.

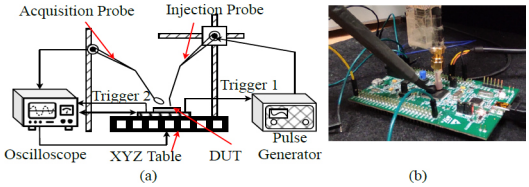

Fig. 5: a) Combined attack setup; b) Probe positions

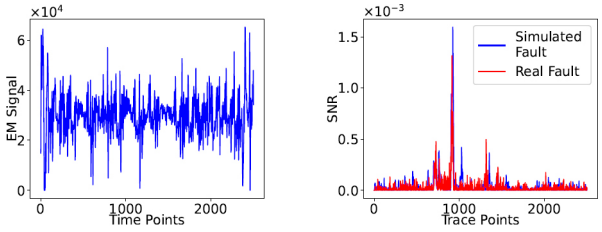

Fig. 6: a) Sample trace; b) Trace SNR: real vs. simulated faults.

V. Discussion and Conclusion

In this paper, we propose and practically validate a new combined attack strategy called SCA-NFA, which breaks some of the strongest forms of countermeasures in a non-profiled setting, even with multi-bit faults and no restrictions imposed upon the inputs. Although, we need access to plaintexts/ciphertexts, we conjecture that this requirement can also be relaxed by combining the attack strategy with approaches like Blind SIFA [20] in future. On the constructive side, developing countermeasures can be an interesting future work. One intuition is to put the error-detection operations inside the DOM gates (after each constituent AND operation) so that faults can be detected or corrected before they combine the shares through propagation [5], [11]. This approach, however, increases the number of XORs in the computation and would also require further investigation for other unforeseen attack points.

References

[1] S. Patranabis and D. Mukhopadhyay, *Fault tolerant architectures for cryptography and hardware security*. Springer, 2018.

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

[3] S. Nikova and et. al., "Threshold implementations against side-channel attacks and glitches," in *ICICS*. Springer, 2006, pp. 529–545.

[4] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," IACR Cryptology ePrint Archive, 2016.

[5] S. Saha and et al., "A framework to counter statistical ineffective fault analysis of block ciphers using domain transformation and error correction," *IEEE TIFS*, vol. 15, pp. 1905–1919, 2020.

[6] J. Daemen and et. al., "Protecting against statistical ineffective fault attacks," *TCHES*, vol. 2020, no. 3, pp. 508–543, 2020.

[7] C. Dobraunig and et. al., "SIFA: exploiting ineffective fault inductions on symmetric cryptography," *TCHES*, vol. 2018, no. 3, pp. 547–572, 2018.

[8] C. Dobraunig and et al., "Statistical ineffective fault attacks on masked AES with fault countermeasures," in *ASIACRYPT*. Springer, 2018.

[9] S. Saha and et. al., "Fault template attacks on block ciphers exploiting fault propagation," in *EUROCRYPT*. Springer, 2020, pp. 612–643.

[10] M. Gruber and et. al., "DOMREP–an orthogonal countermeasure for arbitrary order side-channel and fault attack protection," *IEEE TIFS*, vol. 16, pp. 4321–4335, 2021.

[11] S. Saha and et. al., "Divided we stand, united we fall: Security analysis of some SCA+ SIFA countermeasures against SCA-enhanced fault template attacks," in *ASIACRYPT*. Springer, 2021, pp. 62–94.

[12] "NIST call for lightweight cryptography," https://csrc.nist.gov/Projects/lightweight-cryptography, 2015.

[13] D. J. Bernstein and et. al., "GIMLI - NIST computer security resource center," https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/gimli-spec.pdf, 2015.

[14] "Masked GIMLI," https://github.com/rweather/lightweight-crypto/tree/master/src/individual/Gimli_masked, 2020.

[15] A. R. Shahmirzadi, S. Rasoolzadeh, and A. Moradi, "Impeccable circuits II," in *DAC*. IEEE, 2020, pp. 1–6.

[16] J. Breier and et. al., "A countermeasure against statistical ineffective fault analysis," *IEEE TCAS II*, vol. 67, no. 12, pp. 3322–3326, 2020.

[17] T. Roche and et. al., "Combined fault and side-channel attack on protected implementations of aes," in *CARDIS*. Springer, 2011, pp. 65–83.

[18] S. Saha and et. al., "Breaking redundancy-based countermeasures with random faults and power side channel," in *FDTC*, 2018, pp. 15–22.

[19] S. Patranabis and et. al., "SCADFA: Combined SCA+DFA attacks on block ciphers with practical validations," *IEEE Trans. Computers*, vol. 68, no. 10, pp. 1498–1510, 2019.

[20] M. Azouaoui, K. Papagiannopoulos, and D. Zürner, "Blind side-channel SIFA," in *DATE*. IEEE, 2021, pp. 555–560.

[21] G. Cassiers and F.-X. Standaert, "Trivially and efficiently composing masked gadgets with probe isolating non-interference," *IEEE TIFS*, vol. 15, pp. 2542–2555, 2020.

[22] M. Gruber, M. Probst, and M. Tempelmeier, "Statistical ineffective fault analysis of GIMLI," in *HOST*. IEEE, 2020, pp. 252–261.