CAPABARA: A Combined Attack on CAPA

Dilara Toprakhisar*[1], Svetla Nikova[1], and Ventzislav Nikov[2]

[1] COSIC, KU Leuven, Leuven, Belgium,
firstname.lastname@esat.kuleuven.be
[2] NXP Semiconductors, Leuven, Belgium
venci.nikov@gmail.com

Abstract. Physical attacks pose a substantial threat to the secure implementation of cryptographic algorithms. While considerable research efforts are dedicated to protecting against passive physical attacks (*e.g.,* side-channel analysis (SCA)), the landscape of protection against other types of physical attacks remains a challenge. Fault attacks (FA), though attracting growing attention in research, still lack the prevalence of provably secure designs when compared to SCA. The realm of combined attacks, which leverage the capabilities of both SCA and FA adversaries, introduces powerful adversarial models, rendering protection against them challenging. This challenge has consequently led to a relatively unexplored area of research, resulting in a notable gap in understanding and efficiently protecting against combined attacks. The CAPA countermeasure, published at CRYPTO 2018, addresses this challenge with a robust adversarial model that goes beyond conventional SCA and FA adversarial models. Drawing inspiration from the principles of Multiparty Computation (MPC), CAPA claims security against higher-order SCA, higher-order fault attacks, and their combination. In this work, we present a combined attack that breaks CAPA within the constraints of its assumed adversarial model. In response, we propose potential fixes to the design of CAPA that increase the complexity of the proposed attack, although not provably thwarting it. With this presented combined attack, we highlight the difficulty of effectively protecting against combined attacks.

Keywords: Fault attacks · Combined attacks · CAPA.

1 Introduction

Cryptographic algorithms are designed to have certain properties to withstand cryptanalytic attacks. Nevertheless, devices running the implementations of these cryptographic algorithms in practice are susceptible to physical attacks that observe or disrupt their physical characteristics. Side-Channel Analysis (SCA), being a passive physical attack, exploits the observable leakage arising from physical effects, such as power consumption [17], timing [16], or electromagnetic emanation [10]. Masking [4,12,15,19] stands out as a prominent and widely established countermeasure against SCA. The working principle of masking is to

* Corresponding author.

split the secret variable into a number of statistically independent random shares. Consequently, observing all but one share does not reveal information related to the secret variable. Boolean masking [4], being a well-understood countermeasure against SCA, replaces each secret variable $x \in \mathbb{F}_2$ by a vector of s shares $\bar{x} = (x_0, x_1, ..., x_{s-1})$ such that $x = \sum_{i=0}^{s-1} x_i$ over $\mathbb{F}_2$, where each x_i is uniformly random.

Unlike SCA, fault attacks (FA) are active attacks that deliberately disrupt computations through physical means, such as clock/voltage glitching [1], electromagnetic waves [7], and laser injections [14]. Consequently, they extract information from the device's response to the induced errors. Since the seminal work of Boneh *et al.* [3], introducing Differential Fault Attacks (DFA) on RSA, numerous fault analysis methods have emerged, focusing on exploiting incorrect outputs. A commonly employed countermeasure against such attacks involves introducing redundancy in time, area, or information to detect if a fault is injected into the computation. Upon fault detection, these countermeasures either suppress or infect the output, rendering it non-exploitable by the adversaries. Nevertheless, Statistical Ineffective Faults Attacks (SIFA) [9], also referred to as SIFA-1, exploit the dependency between fault propagation and secret values, effectively utilizing correct ciphertexts. This characteristic enables SIFA to circumvent the countermeasures based on straightforward redundancy.

Another prevalent approach combines masking and redundancy to protect against fault attacks. In addition to protecting implementations against SCA through masking, these countermeasures offer protection against SIFA-1-like attacks by introducing randomness to the computation. Nevertheless, Dobraunig *et al.* [8] exploited SIFA-1 (which is then referred to as SIFA-2), circumventing most of the masking combined with redundancy based countermeasures. This underscores the need for more intricate countermeasures, such as fine-grained error detection [5] or error correction, in the landscape of protecting against fault attacks.

In addition to examining the capabilities of SCA and FA adversaries individually, combined attacks that leverage both capabilities simultaneously have attracted attention. However, relying solely on the integration of SCA and FA countermeasures alone is insufficient to protect against such combined attacks. Even advanced countermeasures designed with intricate protection mechanisms may still prove ineffective against these sophisticated attacks [21]. One of the few countermeasures that is designed to protect against combined attacks is the CAPA countermeasure by Reparaz *et al.* [20], published in CRYPTO 2018. CAPA leverages the principles of the Multiparty Computation (MPC) protocol SPDZ [6] that adopts a full threshold setting where all but one party can be corrupted. It performs computations on shared variables and shared information-theoretic MAC tags associated with the secret variables, combining masking and fine-grained redundancy. CAPA claims provable security against higher-order SCA, higher-order (*i.e.,* multiple shot) FA, and their combination. Unlike common SCA and FA adversary models that assume the t-probing model [15] and faulting up to a limited number of gates/registers, CAPA introduces a unique adversary model: *The Tile-Probe-and-Fault Model*. This model assumes that the chip under attack is partitioned into tiles with their own combinational and control logic, and PRNGs, connected to each other by wires. The tile-probe-and-fault model assumes an adversary capable of probing a bounded number of tiles with all their possessed intermediate values, which is an extension of the t-probing model. Simultaneously, the adversary is assumed to have the capability to inject known-value faults to any variable within a bounded number of tiles, which generalizes the bounded gate/register faulting model, or inject random-value faults to an unbounded number of tiles.

The design of CAPA consists of two stages: an evaluation stage to compute the cryptographic algorithms and a preprocessing stage to generate auxiliary data that is used in the evaluation stage, where the two stages can be interleaved. While the evaluation stage adheres to SPDZ principles, the preprocessing stage diverges by utilizing a passively secure shared multiplier to generate auxiliary data. This design choice enhances the efficiency of CAPA compared to SPDZ, where the corresponding offline phase employs somewhat homomorphic encryption for the generation of auxiliary data.

Contributions. In this work, we introduce the first known attack, a combined attack, targeting the CAPA countermeasure within the assumed adversary model, the tile-probe-and-fault model. The proposed attack necessitates an ineffective fault to be injected during the preprocessing stage, resulting in a secret variable in the evaluation stage to be masked by a zero value. Subsequently, this unmasked value is probed to recover the secret. The proposed attack capitalizes on the divergence in the preprocessing stage of CAPA from SPDZ, exploiting an ineffective fault passing to the evaluation stage undetected. This exploitation leads to the breaking of CRYPTO 2018's work within its own adversarial model. In response, we propose fixes for the CAPA countermeasure. Although these fixes do not entirely prevent the proposed attack, they do contribute to an increase in attack complexity.

2 CAPA

Before delving into the proposed attack, we provide an overview of the CAPA countermeasure. We first describe its associated adversarial model, the *tile-probe-and-fault model*, and then its design.

2.1 The Tile-Probe-and-Fault-Model

In this section, we introduce the adversarial model of the CAPA countermeasure along with the security guarantees based on this model. The tile-probe-and-fault model extends the t-probing model as introduced by Ishai *et al.* [15], where an adversary is limited to observe at most t predetermined wires of the Boolean circuit. In the CAPA countermeasure, an integrated circuit is assumed to be separated into tiles, each consisting of wires as well as combinational and sequential logic. This structural arrangement aligns with the MPC setting, where each tile can be viewed as an independent party.

Each tile in the partitioned integrated circuit is denoted with $\mathcal{T}_i$, where wires are running between each pair of tiles as depicted in Figure 1. Each tile encompasses its own combinational logic, control logic, and pseudo-random number generator. The leakage from each tile being local, does not disclose information about other tiles. An adversary with probing and faulting capabilities can obtain information about *all* variables in the probed tile, or inject faults to *all* variables within the targeted tile. The CAPA countermeasure employs Boolean masking, where each secret variable is split into d shares. This implies that the integrated circuit is partitioned into d tiles, where each tile stores and manipulates at most one share of each intermediate variable. The wires running between the tiles carry only the blinded versions of the shares of the secret variables, and faults happening on these wires are confined to affecting only the receiving tiles.

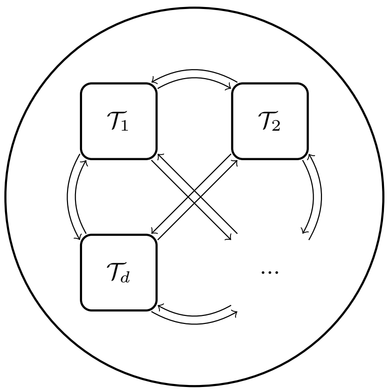

Fig. 1: Tile architecture of an integrated circuit

Probing capabilities. The CAPA countermeasure assumes an adversary with d_p-probing capabilities, where they are given information about all intermediate variables within the specified d_p tiles from the beginning to the end of the computation with a probability of one. This contrasts with the commonly assumed t-probing model, where an adversary can access only t intermediate values. CAPA's assumption of a more potent probing adversary aligns with real-world scenarios, accommodating, for instance, EM probes that enable the observation of multiple intermediate values in the target area.

Faulting capabilities. CAPA assumes two types of fault capabilities under the tile-probe-and-fault model: d_f-faulting and ϵ-faulting. In d_f-faulting, an adversary can inject chosen-value faults to any number of precisely chosen intermediate variables within the specified d_f tiles. On the other hand, ϵ-faulting allows an adversary to inject random-value faults following some distribution to any variable within any tile, for example, flipping each bit with a certain probability. However, the adversary cannot set all variables to a chosen fixed value.

Adversarial Models. CAPA assumes two adversarial models: $\mathcal{A}_1$ with both d_p-probing and d_f-faulting capabilities, and $\mathcal{A}_2$ with both d_p-probing and ϵ-faulting capabilities. Let $\mathcal{P}_1$ denote the set of up to d_p tiles that can be probed, and $\mathcal{F}_1$ denote the set of up to d_f tiles that can be faulted by $\mathcal{A}_1$, ensuring that at least one share/tile remains unaccessed with the following constraint:

$$(\mathcal{F}_1 \cup \mathcal{P}_1) \subseteq \cup_{j=1}^{d-1} \mathcal{T}_{i_j}.$$

Furthermore, a chosen-value fault injected by the adversary $\mathcal{A}_1$ cannot be preceded by a probe within the same clock cycle; it can only depend on the probes from previous clock cycles.

Let $\mathcal{P}_2$ denote the set of up to d_p tiles that can be probed and $\mathcal{F}_2$ denote the set of tiles that can be faulted by $\mathcal{A}_2$. Then, the number of probed tiles remains limited to $d-1$:

$$\mathcal{P}_2 \subseteq \cup_{j=1}^{d-1} \mathcal{T}_{i_j}.$$

However, for $\mathcal{A}_2$ injecting random-value faults, the limitation on the number of tiles that can be faulted no longer applies, as the injected faults do not set the unshared values to chosen fixed values:

$$\mathcal{F}_2 \subseteq \mathcal{T}.$$

It is important to note that the tiles probed or faulted by both adversaries are predetermined, and cannot be adapted during the computation.

2.2 The CAPA Design

The CAPA design consists of two stages: the preprocessing step and the evaluation step. During the preprocessing step, auxiliary data is generated, which is subsequently used in the evaluation step to blind the sensitive variables.

All computations in the design are performed over $\mathbb{F}_{2^k}$. The preprocessing stage variables are denoted by a, b, c, and the evaluation stage variables (*i.e.,* secret variables) are denoted by lowercase letters x, y, z. The bold versions of these letters indicate that the variables are shared (*e.g.,* $\boldsymbol{a} = (a_0, ..., a_{d-1})$ where $\sum_{i=0}^{d-1} a_i = a$). The MAC key is denoted by α, which is also shared among the tiles such that $\boldsymbol{\alpha} = (\alpha_0, ..., \alpha_{d-1})$. Then, a value $a \in \mathbb{F}_{2^k}$ is represented as a pair $\langle \boldsymbol{a} \rangle = (\mathbf{a}, \boldsymbol{\tau_a})$ consisting of the data and the associated tag shares in the masked domain, where $\tau_a = \alpha a$. In the CAPA design, there can be multiple independent MAC keys, associating multiple tags with each variable. However, for the sake of simplicity, we assume that the design employs only a single MAC key in this work. The Kronecker delta function is denoted by $\delta_{i,j}$, returning 1 if $i = j$, and 0 otherwise.

We first describe the multiplication as a building block in the evaluation stage, which is the focal point of interest in the proposed attack. Subsequently, we delve into the preprocessing components.

Evaluation Stage

Each tile $\mathcal{T}_i$ stores the i^{th} share of each sensitive and auxiliary variable, along with the i^{th} share of the associated MAC tags and the MAC key. Linear operations, such as addition, do not necessitate communication between tiles, unlike nonlinear operations like multiplication which also require auxiliary data.

Multiplication. To compute the multiplication of $(\boldsymbol{x}, \boldsymbol{\tau_x})$ and $(\boldsymbol{y}, \boldsymbol{\tau_y})$, a Beaver triple $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ satisfying $c = ab$, generated during the preprocessing stage, is utilized. The multiplication then proceeds as follows:

Step 1: blinding. Each tile $\mathcal{T}_i$ locally randomizes its share of the secrets $\boldsymbol{x}$ and $\boldsymbol{y}$: $\varepsilon_i = x_i + a_i$ and $\eta_i = y_i + b_i$. Likewise, the associated MAC tag shares are computed: $\tau_{\varepsilon_i} = \tau_{x_i} + \tau_{a_i}$ and $\tau_{\eta_i} = \tau_{y_i} + \tau_{b_i}$.

Step 2: partial unmasking. Each tile $\mathcal{T}_i$ broadcasts ε_i and η_i computed in Step 1 to the other tiles. Then, each tile locally computes and stores the values $\varepsilon = \sum_{i=0}^{d-1} \varepsilon_i$ and $\eta = \sum_{i=0}^{d-1} \eta_i$. ε and η (*i.e.,* blinded versions of the secrets) are partially unmasked as their MAC tags $\boldsymbol{\tau_\varepsilon}$ and $\boldsymbol{\tau_\eta}$ remain shared among the tiles.

Step 3: checking the MAC tags of the partially unmasked values. The tiles check whether the tags $\boldsymbol{\tau_\varepsilon}$ and $\boldsymbol{\tau_\eta}$ are consistent with the unmasked public values ε and η using a method described below.

Step 4: Beaver computation. Each tile locally computes the following:

$$\begin{aligned} z_i &= c_i + \varepsilon b_i + \eta a_i + \varepsilon\eta\delta_{i,1}, \\ \tau_{z_i} &= \tau_{c_i} + \varepsilon\tau_{b_i} + \eta\tau_{a_i} + \alpha_i\varepsilon\eta. \end{aligned} \tag{1}$$

One can verify that the shared output $(\boldsymbol{z}, \boldsymbol{\tau_z})$ of the above protocol corresponds to $z = xy$ given that no faults are present.

Checking the MAC tags of partially unmasked values. The CAPA multiplication continues its computation only if both of the partially unmasked values are consistent with their associated MAC tags. Consider a partially unmasked value $\varepsilon = x + a$, then, each tile has its own share of the corresponding MAC tag which is computed in the first step of the multiplication as $\tau_{\varepsilon_i} = \tau_{x_i} + \tau_{a_i}$. Then, to verify the correctness of the MAC tag, each tile locally computes and broadcasts the value $\alpha_i\varepsilon + \tau_{\varepsilon_i}$ to the other tiles. Ensuring the correctness of the tag, where $\sum_{i=0}^{d-1} \tau_{\varepsilon_i} = \alpha\varepsilon$ holds, involves each tile computing $\sum_{i=0}^{d-1}(\alpha_i\varepsilon + \tau_{\varepsilon_i})$ and proceeding only if the result is zero.

Preprocessing Stage

The Beaver triples $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$, satisfying $c = ab$, used in the multiplication, are generated in the preprocessing stage of the CAPA design.

Auxiliary data generation. The Beaver triple $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ is generated through a process where each tile $\mathcal{T}_i$ independently draws random shares a_i and b_i such that $\boldsymbol{a} = (a_0, ..., a_{d-1})$ and $\boldsymbol{b} = (b_0, ..., b_{d-1})$, where $a = \sum_{i=0}^{d-1} a_i$ and $b = \sum_{i=0}^{d-1} b_i$. Subsequently, a passively secure multiplier (*e.g.,* [18], [2], [11], [13], [19]) is employed to compute $\boldsymbol{c} = (c_0, ..., c_{d-1})$, where $c = \sum_{i=0}^{d-1} c_i$ such that $c = ab$. Simultaneously, shared MAC tags $\boldsymbol{\tau_a}, \boldsymbol{\tau_b}, \boldsymbol{\tau_c}$ are computed using a passively secure multiplier and the shared MAC key $\boldsymbol{\alpha}$.

Relation verification of auxiliary data. The faults injected during the evaluation stage in CAPA are detected through the MAC tag check performed in Step 3 of the CAPA multiplication. In contrast, similar to SPDZ, to detect the faults injected during the preprocessing stage, a relation verification is executed for each Beaver triple transitioning from the preprocessing stage to the evaluation stage. This verification ensures the correctness of the triple, specifically validating that $c = ab$. The verification of the correctness of $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ is executed by employing another Beaver triple $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$:

1. A random $r_1 \in \mathbb{F}_{2^k}$ is drawn.
2. Utilizing the second triple $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$, the multiplication of $r_1 \langle \boldsymbol{a} \rangle$ and $\langle \boldsymbol{b} \rangle$, and the associated MAC tag are computed. This involves a constant multiplication with r_1, which is performed locally as it is a linear operation, and an actively secure multiplication executed as described in Equation 1. Then, the outcome $\langle \tilde{\boldsymbol{c}} \rangle$ is a shared representation of $\tilde{c} = r_1 ab$.
3. Each tile $\mathcal{T}_i$ locally computes and broadcasts the differences of the shares of $r_1 c$ and $\tilde{c}$, and their corresponding tags: $\Theta_i = r_1 c_i + \tilde{c}_i$ and $\tau_{\Theta_i} = r_1 \tau_{c_i} + \tau_{\tilde{c}_i}$. Then, $\Theta = \sum_{i=0}^{d-1} \Theta_i$ and $\tau_\Theta = \sum_{i=0}^{d-1} \tau_{\Theta_i}$ are unmasked.
4. If at least one of the unmasked differences, Θ and τ_Θ, is nonzero, $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ is rejected as a valid triple.
5. Else, another $r_2 \in \mathbb{F}_{2^k}$ is drawn such that $r_1 \neq r_2$, and the described procedure is repeated for a second time.

While this stage corresponds to the offline phase of the MPC protocol, SPDZ, there are notable distinctions rendering the CAPA preprocessing stage more lightweight. In CAPA, Beaver triples are generated on the fly, as needed, resulting in reduced storage requirements. Moreover, these triples are generated using a passively secure multiplier, in contrast to SPDZ's use of a somewhat homomorphic encryption scheme. This choice is motivated by somewhat homomorphic encryption exceeding any efficiency requirements for deployment in hardware devices. Unlike SPDZ, where each party proves the correctness of the generated randoms (a_i, b_i) that are shares of the Beaver triple elements $(\boldsymbol{a}, \boldsymbol{b})$, CAPA does not error check the generated Beaver triple elements which are then fed into a passively secure multiplier. The proposed attack leverages this distinction in CAPA's design from SPDZ, specifically the absence of error checks of the generated Beaver triples. CAPA solely verifies $c = ab$, overlooking ineffective faults.

3 The Combined Attack Description

In this section, we describe the proposed combined attack targeting CAPA. The attack assumes an adversarial model that stays within the adversarial model of CAPA, namely, the tile-probe-and-fault model. The adversary in the proposed attack is assumed to have both faulting and probing capabilities. Importantly, the attack does not rely on any strong assumptions inherent in the tile-and-probe-fault model, such as probing or faulting all variables within a tile. Specifically, the attack demonstrates effectiveness by faulting a single variable and probing another one, which retains the feasibility even under conventional adversarial models like t-probing and a bounded number of gate/register faulting. The attack necessitates only a single fault injected during the preprocessing step, which may take the form of a random fault (*i.e.,* randomly chosen set, reset, or bit flip fault). This fault is complemented by a probing event during the evaluation stage, where the probed value is subsequently unmasked due to the injected fault becoming ineffective.

For readability, we consider the attack scenario where the CAPA countermeasure is instantiated with $d = 2$ (*i.e.,* 2 shares/tiles). It is important to highlight that the attack scenario is independent of the number of shares, and can be generalized for any arbitrary order d as it only requires faulting a single share/tile. Additionally, the adversary can probe the same faulted tile, where the probed variable is unmasked, making it adaptable for different values of d.

3.1 The Attack Scenario

The combined attack scenario consists of two steps: the fault injection step, and the subsequent probing step that exploits the manifestation of the injected fault. We first describe the assumptions regarding the faulting and probing capabilities of the adversary, then, describe the fault injection step targeting the preprocessing stage. Following the fault injection step, we describe the probing step exploiting the injected fault, and outline the success conditions for the attack.

Capabilities

We describe the faulting capabilities of the assumed adversary performing the fault injection step as follows:

– The adversary injects a single-shot fault to a variable in $\mathbb{F}_{2^k}$, potentially affecting a random number of bits within the targeted variable.

- The fault location is loose, allowing the injection of a fault affecting any set of bits within any share of the target secret variable.
- Precise timing is crucial for the injected fault, necessitating its occurrence before the tag computation of the targeted secret variable.
- The effectiveness period of the fault is transient.
- The fault is required to be injected to a registered value, and can be of a random type.

Additionally, the adversary is assumed to be capable of probing a variable of their choosing.

Fault Injection Step

The fault injection in the preprocessing stage is formalized as follows:

(1) Each tile draws randoms $\mathcal{T}_0 \rightarrow a_0, b_0 \in \mathbb{F}_{2^k}$, $\mathcal{T}_1 \rightarrow a_1, b_1 \in \mathbb{F}_{2^k}$, such that $\boldsymbol{a} = (a_0, a_1)$ where $a = a_0 + a_1$ and $\boldsymbol{b} = (b_0, b_1)$ where $b = b_0 + b_1$.

(2) The tiles compute $\boldsymbol{c} = (c_0, c_1)$ where $c = c_0 + c_1$ such that $c = ab$ using a passively secure shared multiplier.

(3) The tiles compute the corresponding MAC tags $\boldsymbol{\tau_a}, \boldsymbol{\tau_b}, \boldsymbol{\tau_c}$ using a passively secure shared multiplier (*i.e.,* $\tau_x = \alpha x$).

(3.1) Before the computation of $\boldsymbol{\tau_a}$, a fault is injected to a share of $\boldsymbol{a}$. Assuming a_0 is the targeted share, $\mathcal{T}_0$ now stores the faulty value $a'_0 = a_0 + \Delta$, and uses it in the subsequent computations.

(3.2) Then, the incorrect MAC tag $\boldsymbol{\tau_{a'}}$ consistent with a' is computed, where $\tau_{a'} = \alpha a'$ and $a' = a + \Delta$, while $\boldsymbol{\tau_b}, \boldsymbol{\tau_c}$ are correct and consistent with $\boldsymbol{b}, \boldsymbol{c}$, respectively.

(4) Relation verification for the triple $(\langle \boldsymbol{a'} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ is performed utilizing another triple $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$ that is assumed to be correct:

(4.1) A random $r_1 \in \mathbb{F}_{2^k}$ is drawn.

(4.2) $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$ is used to compute $\langle \tilde{\boldsymbol{c}} \rangle$ where $\tilde{c} = r_1 a' b$. As $\boldsymbol{\tau_{a'}}$ and $\boldsymbol{\tau_b}$ are consistent with the values $\boldsymbol{a'}$ and $\boldsymbol{b}$, the actively secure multiplication is executed with a successful MAC tag check step.

(4.3) Each tile locally computes and broadcasts $\mathcal{T}_0 \rightarrow \Theta_0 = r_1 c_0 + \tilde{c}_0, \tau_{\Theta_0} = r_1 \tau_{c_0} + \tau_{\tilde{c}_0}$, $\mathcal{T}_1 \rightarrow \Theta_1 = r_1 c_1 + \tilde{c}_1, \tau_{\Theta_1} = r_1 \tau_{c_1} + \tau_{\tilde{c}_1}$. $\Theta = \Theta_0 + \Theta_1$ and $\tau_\Theta = \tau_{\Theta_0} + \tau_{\Theta_1}$ are then unmasked.

(4.4) Then, the relation verification checks whether at least one of unmasked Θ and τ_Θ is non-zero. If one of them is non-zero, then the triple is rejected as it is not valid. That is, if b is non-zero, the faulty triple $(\langle \boldsymbol{a'} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ will not pass the check as the fault injected to $\boldsymbol{a}$ is effective (*i.e.,* $a'b = c + \Delta b \neq c$), making the triple invalid. However, the faulty triple will pass the check if $b = 0$ as the injected fault is ineffective due to b nullifying it (*i.e.,* $a'b = c + \Delta 0 = c$), maintaining the validity of the triple.

Probing Step

In this step of the attack, the adversary exploits the injected fault by probing to deduce information about the secrets. Considering that $b = 0$ and $c = 0$, the triple $(\langle \boldsymbol{a'} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ successfully passes the relation verification for $a' = a + \Delta$. This triple is then employed in the multiplication (Equation 1) to blind the shares of the secret inputs. As a result, one of the secret variables (consider y in our example) in the multiplication $z = xy$ is blinded by a zero value: $\eta = y + b = y$. Subsequently, the adversary probes the unmasked variable $\eta = y$ to reveal the secret value y.

This described attack involves a single fault injection in one of the randoms (*i.e., a*) of a Beaver triple, coupled with a single probe. The attack is successful if and only if the injected fault to $\boldsymbol{a}$ (where any of the shares can be faulted) is ineffective due to the value of $b = 0$, occurring with a probability of 2^{-k}. Consequently, the leakage results from an unmasked byte value, which is not directly targeted by the fault. In essence, as long as the fault injection step is successful, *i.e.,* the injected fault is ineffective, an unmasked variable occurs in the subsequent multiplication some cycles after the fault injection event. It is crucial to note that the fault does not need to be repeatable for a successful attack. The ineffectiveness of the fault injected into $\boldsymbol{a}$ is solely contingent on the value of b, and the fault can take any random fault type.

It is essential to highlight that, in the CAPA countermeasure, faults are undetected only if both the value and its corresponding MAC tag are altered such that they are consistent, which requires two faults to be injected. For an $\mathcal{A}_1$ adversary to successfully obtain a pair of a faulty value and a consistent MAC tag, knowledge of the MAC key (α) is required. Due to the MAC key being secret, the adversary is limited to probabilistic guessing, which is successful with a probability of 2^{-k}. For an $\mathcal{A}_2$ adversary, the faults go undetected only when the value and the corresponding tag happen to be consistent, occurring with a probability of 2^{-k}. In this context, the success probability of the proposed attack, conditioned on $b = 0$, remains the same at 2^{-k}. Nevertheless, it exploits a specific vulnerability that arises in the preprocessing stage, requiring a single fault and a subsequent probe, in contrast to the need for two faults. Moreover, employing multiple MAC tags for each variable reduces the success probability of obtaining consistent variable and MAC tags to 2^{-lk}, where l denotes the number of tags. However, this does not impact the success probability of the proposed attack, as the tags are computed based on the faulty value.

4 Fixes Against the Proposed Attack

In this section, we propose a few fixes against the aforementioned attack. It is important to emphasize that while the proposed fixes do not completely eliminate the vulnerability, they do increase the complexity of the attack.

Computing the Tags of $\boldsymbol{a}$ and $\boldsymbol{b}$ Prior to Forming the Triple

In response to this attack, one proposed fix involves precomputing the tags of $\boldsymbol{a}$ and $\boldsymbol{b}$ before forming the triple in the preprocessing stage, prior to the computation of $\boldsymbol{c}$ such that $c = ab$. As the proposed attack requires the fault to be injected after $\boldsymbol{c}$ is computed, this fix aims to prevent faults injected to $\boldsymbol{a}$ and $\boldsymbol{b}$ from going undetected by computing the associated MAC tags based on their non-faulty values. Then, the faults injected to $\boldsymbol{a}$ and $\boldsymbol{b}$ will be detected in the MAC tag check step of the actively secure multiplication of the relation verification step, thus circumventing the proposed attack.

However, it is crucial to note that an adversary can still execute the described attack, albeit with multiple faults. Specifically, the adversary needs to inject three faults. First, the adversary injects a fault to a to obtain $a' = a + \Delta$ before computing τ_a to get a faulty tag associated with a'. Then, they inject the same fault to a' to revert it to a, allowing c to be computed using the correct a and b. Lastly, they inject one more fault to a after computing c as in the original attack. We formalize this fault injection step as follows:

(1) Each tile draws randoms $\mathcal{T}_0 \rightarrow a_0, b_0 \in \mathbb{F}_{2^k}$, $\mathcal{T}_1 \rightarrow a_1, b_1 \in \mathbb{F}_{2^k}$, such that $\boldsymbol{a} = (a_0, a_1)$ where $a = a_0 + a_1$ and $\boldsymbol{b} = (b_0, b_1)$ where $b = b_0 + b_1$.
(2) The tiles compute the corresponding MAC tags $\boldsymbol{\tau_a}, \boldsymbol{\tau_b}$.
  (2.1) Before the computation of $\boldsymbol{\tau_a}$, a fault is injected to a share of $\boldsymbol{a}$. Assuming a_0 is the targeted share, $\mathcal{T}_0$ now stores the faulty value $a'_0 = a_0 + \Delta$, and uses it in subsequent computations.
  (2.2) Then, the incorrect MAC tag $\boldsymbol{\tau_{a'}}$ consistent with a' is computed, where $\tau_{a'} = \alpha a'$ and $a' = a + \Delta$, while $\boldsymbol{\tau_b}$ is correct and consistent with $\boldsymbol{b}$.
  (2.3) After the computation of $\boldsymbol{\tau'_a}$, the same fault Δ is injected to a share of $\boldsymbol{a'}$ to obtain the initial value a (*i.e.* $a = a' + \Delta$).
(3) The tiles compute $\boldsymbol{c} = (c_0, c_1)$ where $c = c_0 + c_1$ such that $c = ab$, and the corresponding MAC tag τ_c.
(4) The same fault Δ is injected to any share of a once more, obtaining $a' = a + \Delta$.
(5) Relation verification for the triple $(\langle \boldsymbol{a'} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ is executed utilizing another triple $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$ that is assumed to be correct:
  (5.1) A random $r_1 \in \mathbb{F}_{2^k}$ is drawn.
  (5.2) $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$ is used to compute $\langle \tilde{\boldsymbol{c}} \rangle$ where $\tilde{c} = r_1 a' b$. As $\boldsymbol{\tau_{a'}}$ and $\boldsymbol{\tau_b}$ are consistent with the values $\boldsymbol{a'}$ and $\boldsymbol{b}$, the actively secure multiplication is executed with a successful MAC tag check step.
  (5.3) Each tile locally computes and broadcasts $\mathcal{T}_0 \rightarrow \Theta_0 = r_1 c_0 + \tilde{c}_0, \tau_{\Theta_0} = r_1 \tau_{c_0} + \tau_{\tilde{c}_0}$, $\mathcal{T}_1 \rightarrow \Theta_1 = r_1 c_1 + \tilde{c}_1, \tau_{\Theta_1} = r_1 \tau_{c_1} + \tau_{\tilde{c}_1}$. $\Theta = \Theta_0 + \Theta_1$ and $\tau_\Theta = \tau_{\Theta_0} + \tau_{\Theta_1}$ are then unmasked.
  (5.4) Then, the relation verification checks whether at least one of unmasked Θ and τ_Θ is non-zero. If one of them is non-zero, then the triple is rejected as it is not valid. That is, the triple is rejected if b is non-zero, and the triple is used for blinding if $b = 0$.

We note that, in certain cases, a copy of a rather than the value itself may be used to compute c (step 3). In such instances, step 4 becomes unnecessary, given that a' is already stored and utilized in subsequent computations.

The probing step follows the same process described in the original attack which is conditioned on the injected fault to a being ineffective (*i.e.,* $b = 0$), occurring with a probability of 2^{-k}. In both adversarial models, $\mathcal{A}_1$ and $\mathcal{A}_2$, the adversary is able to inject the same additive fault ($\varDelta$) repeatedly, as it is independent of the value a. Although this attack maintains the success probability of the original attack, the proposed fix increases the complexity by necessitating three faults to be injected.

Randomly Choosing the Beaver Triple To Be Used in the Multiplication

In a manner similar to SPDZ, the CAPA countermeasure can exert control over the selection of the Beaver triple used for blinding secrets in the multiplication (Equation 1). That is, the CAPA countermeasure can choose between the two Beaver triples $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$, $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$ to be used for blinding in the multiplication (Equation 1) or to be sacrificed. Consequently, for an adversary to execute the same attack with equivalent success probability 2^{-k}, faults must be injected to both Beaver triples ($1/2 \cdot 2^{-k} + 1/2 \cdot 2^{-k} = 2^{-k}$). The injected faults do not need to be identical, as achieving an ineffective fault hinges solely on the value of b (*i.e.,* the fault-free variable of the Beaver triple inputs). Therefore, akin to the first proposed fix, this one also increases the complexity of the attack, requiring a total of two fault injections, albeit maintaining the same success probability. It is essential to highlight that an alternative attack strategy still incorporates a single fault injection, as assumed in the proposed attack. In this scenario, the adversary can still be successful, albeit with half of the initial success rate (2^{-k-1}), when the faulty Beaver triple is chosen with a probability of 2^{-1}.

Another fix involves the generalization of the above described fix. During the preprocessing stage, the countermeasure can generate multiple Beaver triples. To be used in the evaluation phase, two of the triples can be randomly selected for the relation verification, and consequently, for blinding the secrets. This further increases the complexity of the attack, meaning that more faults (*i.e.,* the number of Beaver triples) must be injected to execute the same attack with the same success rate. Alternatively, the adversary can still execute the attack with a single fault, albeit with a probability of ($1/m \cdot 2^{-k}$), where m is the number of Beaver triples available to be selected for blinding in the multiplication (Equation 1). If the number of used Beaver triples for the selection is less than $2^{(l-1)k}$, this alternative attack still proves more efficient than the straightforward attack of injecting two faults to achieve a consistent value and a tag, when more than one MAC tag is employed.

Zero-check on $\boldsymbol{c}$

The CAPA countermeasure can implement a zero-check on $\boldsymbol{c}$, indirectly checking whether either a or b is zero, as $c = ab$, constituting the condition for the ineffectiveness of the injected fault. In this way, the countermeasure selectively forms the Beaver triple $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ only for non-zero inputs. While implementing a zero-check on masked values is not inherently efficient, this fix does serve as a preventive measure against the proposed attack, assuming the zero-check protocol and its result remain unaffected by the faults.

Nevertheless, it is important to note that excluding zero inputs from the Beaver triple generation process will compromise the uniformity of the partially unmasked multiplication inputs. The potential impact of this strategy on the susceptibility of the non-uniformly blinded values to exploitation by a probing adversary should be carefully considered.

5 Conclusion

Our work presents the first known attack, CAPABARA, targeting the CAPA countermeasure within its own adversarial model, the tile-probe-and-fault model. We highlight the distinct preprocessing stage of CAPA that differs from the offline phase of SPDZ, as discussed in Section 2.2. While these distinctions render the CAPA preprocessing stage more lightweight, it sacrifices certain security properties provided by somewhat homomorphic encryption utilized in SPDZ.

Our proposed attack exploits the lack of error checks of the generated randoms (a_i, b_i) forming the Beaver triples after they are fed to the passively secure multiplier. Despite CAPA's assumption of a robust adversary model, CAPABARA exploits only a single fault injected during the preprocessing stage and a probe performed during the evaluation stage, where the probed value is unmasked due to the ineffective fault.

In response to the proposed attack, we also propose a few fixes to the CAPA design. While these fixes do not fully eliminate the exploited vulnerability, they enhance the attack complexity. The task of eliminating this vulnerability while meeting efficiency requirements for deployment is left as a future work that merits further investigation.

Acknowledgements This work was supported by CyberSecurity Research Flanders with reference number VR20192203.

References

1. Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997.

2. Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.
3. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.
4. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
5. Joan Daemen, Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Florian Mendel, and Robert Primas. Protecting against statistical ineffective fault attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):508–543, 2020.
6. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
7. Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In Guido Bertoni and Benedikt Gierlichs, editors, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, pages 7–15. IEEE Computer Society, 2012.
8. Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 315–342. Springer, 2018.
9. Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):547–572, 2018.
10. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.
11. Hannes Groß and Stefan Mangard. Reconciling d+1 masking in hardware and software. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei,*

Taiwan, September 25-28, 2017, Proceedings, volume 10529 of *Lecture Notes in Computer Science*, pages 115–136. Springer, 2017.

12. Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.
13. Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.
14. D. H. Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions on Nuclear Science*, 12(5):91–100, 1965.
15. Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
16. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
17. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
18. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.
19. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.
20. Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel P. Smart. CAPA: the spirit of beaver against physical attacks. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 121–151. Springer, 2018.
21. Sayandeep Saha, Arnab Bag, Dirmanto Jap, Debdeep Mukhopadhyay, and Shivam Bhasin. Divided we stand, united we fall: Security analysis of some SCA+SIFA

countermeasures against sca-enhanced fault template attacks. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 62–94. Springer, 2021.