CAPA: The Spirit of Beaver Against Physical Attacks

Oscar Reparaz[1,2](✉), Lauren De Meyer[1], Begül Bilgin[1], Victor Arribas[1],
Svetla Nikova[1], Ventzislav Nikov[3], and Nigel Smart[1,4]

[1] KU Leuven, imec - COSIC, Leuven, Belgium
{oscar.reparaz,lauren.demeyer,begul.bilgin,victor.arribas,
svetla.nikova,nigel.smart}@esat.kuleuven.be
[2] Square Inc., San Francisco, USA
[3] NXP Semiconductors, Leuven, Belgium
venci.nikov@gmail.com
[4] University of Bristol, Bristol, UK

Abstract. In this paper we introduce two things: On one hand we introduce the Tile-Probe-and-Fault model, a model generalising the wire-probe model of Ishai et al. extending it to cover both more realistic side-channel leakage scenarios on a chip and also to cover fault and combined attacks. Secondly we introduce CAPA: a combined Countermeasure Against Physical Attacks. Our countermeasure is motivated by our model, and aims to provide security against higher-order SCA, multiple-shot FA and combined attacks. The tile-probe-and-fault model leads one to naturally look (by analogy) at actively secure multi-party computation protocols. Indeed, CAPA draws much inspiration from the MPC protocol SPDZ. So as to demonstrate that the model, and the CAPA countermeasure, are not just theoretical constructions, but could also serve to build practical countermeasures, we present initial experiments of proof-of-concept designs using the CAPA methodology. Namely, a hardware implementation of the KATAN and AES block ciphers, as well as a software bitsliced AES S-box implementation. We demonstrate experimentally that the design can resist second-order DPA attacks, even when the attacker is presented with many hundreds of thousands of traces. In addition our proof-of-concept can also detect faults within our model with high probability in accordance to the methodology.

1 Introduction

Side-channel analysis attacks (SCA) [41] are cheap and scalable methods to extract secrets, such as cryptographic keys or passwords, from embedded electronic devices. They exploit unintended signals (such as the instantaneous power consumption [42] or the electromagnetic radiation [24]) stemming from a cryptographic implementation. In the last twenty years, plenty of countermeasures to mitigate the impact of side-channel information have been developed. Masking [15,26] is an established solution that stands out as a provably secure yet practically useful countermeasure.

H. Shacham and A. Boldyreva (Eds.): CRYPTO 2018, LNCS 10991, pp. 121–151, 2018.
https://doi.org/10.1007/978-3-319-96884-1_5

Fault analysis (FA) is another relevant attack vector for embedded cryptography. The basic principle is to disturb the cryptographic computation somehow (for example, by under-powering the cryptographic device, or by careful illumination of certain areas in the silicon die). The result of a faulty computation can reveal a wealth of secret information: in the case of RSA or AES, a single faulty ciphertext pair makes key recovery possible [10,48]. Countermeasures are essentially based on adding some redundancy to the computation (in space or time). In contrast to masking, the countermeasures for fault analysis are mostly heuristic and lack a formal background.

However, there is a tension between side-channel countermeasures and fault analysis countermeasures. On the one hand, fault analysis countermeasures require redundancy, which can give out more leakage information to an adversary. On the other hand, a device that implements first-order masking offers an adversary double the attack surface to insert a fault in the computation. A duality relation between SCA and FA was pointed out in [23]. There is clearly a need for a combined countermeasure that tackles both problems simultaneously.

In this work we introduce a new attack model to capture this combined attack surface which we call the *tile-probe-and-fault* model. This model naturally extends the wire-probe model of [34]. In the wire-probe model individual wires of a circuit may be targetted for probing. The goal is then to protect against a certain fixed set of wire-probes. In our model, inspired by modern processor designs, we allow whole areas (or tiles) to be probed, and in addition we add the possibility of the attacker inducing faults on such tiles.

Protection against attacks in the wire-probe model is usually done via masking; which is in many cases the extension of ideas from *passively secure* secret sharing based Multi-Party Computation (MPC) to the side-channel domain. It is then natural to look at *actively secure* MPC protocols for the extension to fault attacks. The most successful modern actively secure MPC protocols are in the SPDZ family [20]. These use a pre-processing or *preparation* phase to produce so called *Beaver triples*, named after Beaver [6]. These auxiliary data values, which will be explained later, are prepared either before a computation, or in a just-in-time manner, so as to enable an efficient protocol to be executed. This use of *prepared* Beaver triples also explains, partially, the naming of our system, CAPA (a Combined countermeasure Against Physical Attacks), since Capa is also the beaver spirit in Lakota mythology. In this mythology, Capa is the lord of domesticity, labour and *preparation*.

1.1 Previous Work

Fault Attack Models and Countermeasures: Faults models typically describe the characterization of an attacker's ability. That is, the fault model is constructed as a combination of the following: the precision of the fault location and time, the number of affected bits which highly depends on the architecture, the effect of the fault (flip/set/reset/random) and its duration (transient/permanent). Moreover, the fault can target the clock or power line, storage units, combinational or control logic.

When it comes to countermeasures, one distinguishes between protection of the algorithm on the one hand and protection of the device itself by using, for example, active or passive shields on the other. No countermeasure provides perfect security at a finite cost; it is the designer's responsibility to strive for a balance between high-level (algorithmic) countermeasures and low-level ones that work at the circuit level and complement each other. In this paper, we discuss the former.

One algorithmic technique is to replicate the calculation m times in either time or space and only complete if all executions return the same result [54]. This countermeasure has the important caveat that there are conceptually simple attacks, such as m identical fault injections in each execution, that break the implementation with probability one. However, it should be stated that these attacks are not trivial to mount in practice when the redundancy is in space.

A second method is to use an error correcting or detecting code [8,12,13,32, 35–39,46]. This means one performs all calculations on both data and checksum. A drawback is that error correcting/detecting codes only work in environments in which errors are randomly generated, as opposed to maliciously generated. Thus, a skilled attacker may be able to carefully craft a fault that results in a valid codeword and is thus not detected. A detailed cost comparison between error detection codes and doubling is given in [44].

Another approach is that of infective computation [25,43], where any fault injected will affect the ciphertext in a way that no secret information can be extracted from it. This method ensures the ciphertext can always be returned without the need for integrity checks. While infective methods are very efficient, the schemes proposed so far have all been broken [5].

Side-Channel Attack Models and Countermeasures: A side-channel adversary typically uses the noisy leakage model [55], where side-channel analysis (SCA) attacks are bounded by the statistical moment of the attack due to a limited number of traces and noisy leakages. Given enough noise and an independent leakage assumption of each wire, this model, when limited to the t^{th}-order statistical moment, is shown to be comparable to the t-probing model introduced in [34], where an attacker is allowed to probe, receive and combine the noiseless information about t wires within a time period [21]. Finally, it has been shown in [4] that a (semi-)parallel implementation is secure in the t^{th}-order bounded moment model if its complete serialization is secure at the t-probing model.

While the countermeasures against fault attacks are limited to resist only a small subset of the real-world adversaries and attack models, protection against side-channel attacks stands on much more rigorous grounds and generally scales well with the attacker's powers. A traditional solution is to use masking schemes [9,29,34,51,56,58,59] to implement a function in a manner in which higher-order SCA is needed to extract any secret information, *i.e.* the attacker must exploit the joint leakage of several intermediate values. Masking schemes are analogues of the passively secure threshold MPC protocols based on secret sharing. One can thus justify their defence by appealing to the standard MPC literature. In MPC, a number of parties can evaluate a function on shared data, even in the presence of adversaries amongst the computing parties. The maximum number of dishonest parties which can be tolerated is called the threshold. In an embedded scenario, the basic idea is that different parts of a chip simulate the parties in an MPC protocol.

Combining Faults and Side-Channels Models and Countermeasures. The importance of combined countermeasures becomes more apparent as attacks such as [2] show the feasibility of combined attacks. Being a relatively new threat, combined adversarial models lack a joint description and are typically limited to the combination of a certain side-channel model and a fault model independently.

One possible countermeasure against combined attacks is found in leakage resilient schemes [45], although none of these constructions provide *provable* security against FA. Typical leakage resilient schemes rely on a relatively simple and easy to protect key derivation function in order to update the key that is used by the cryptographic algorithm within short periods. That is, a leakage resilient scheme acts as a specific "mode of operation". Thus, it cannot be a drop-in replacement for a standard primitive such as the AES block cipher. The aforementioned period can be as short as one encryption per key in order to eliminate fault attacks completely. However, the synchronization burden this countermeasure brings, makes it difficult to integrate with deployed protocols.

There are a couple of alternative countermeasures proposed for embedded systems in recent years. In private circuits II [16,33], the authors use redundancy on top of a circuit that already resists SCA (private circuits I [34]) to add protection against FA. In ParTI [62], threshold implementations (TI) are combined with concurrent error detection techniques. ParTI naturally inherits the drawbacks of using an error correction/detection code. Moreover, the detectable faults are limited in hamming weight due to the choice of the code. Finally, in [63], infective computation is combined with error-preserving computation to obtain a side-channel and fault resistant scheme. However, combined attacks are not taken into account.

Given the above introduction, it is clear that both combined attack models and countermeasures are not mature enough to cover a significant part of the attack surface.

Actively Secure MPC. Modern MPC protocols obtain active security, *i.e.* security against malicious parties which can actively deviate from the protocol. By mapping such protocols to the on-chip side-channel countermeasures, we would be able to protect against an eavesdropping adversary that inserts faults into a subset of the simulated parties. An example of a practical attack that fits this model is the combined attack of Amiel et al. [2]. We place defences against faults on the same theoretical basis as defences against side-channels.

To obtain maliciously secure MPC protocols in the secret-sharing model, there are a number of approaches. The traditional approach is to use Verifiable Secret Sharing (VSS), which works in the information theoretic model and requires that strictly less than $n/3$ parties can be corrupt. The modern approach, adopted by protocols such as BODZ, SPDZ, Tiny-OT, MASCOT etc. [7,20,40,50], is to work in a full threshold setting (*i.e.* all but one party can be corrupted) and attach information theoretic MACs to each data item. This approach turns out to be very efficient in the MPC setting, apart from its usage of public-key primitives. The computational efficiency of the use of information theoretic MACs and the active adversarial model of SPDZ lead us to adopt this philosophy.

1.2 Our Contributions

Our contributions are threefold. We first introduce the *tile-probe-and-fault* model, a new adversary model for physical attacks on embedded systems. We then use the analogy between masking and MPC to provide a methodology, which we call CAPA, to protect against such a tile-probe-and-fault attacker. Finally, we illustrate that the CAPA methodology can be prototyped by describing specific instantiations of the CAPA methodology, and our experimental results.

Tile-probe-and-fault model. We introduce a new adversary model that expands on the wire-probe model and brings it closer to real-world processor designs. Our model is set in an architecture that mimics the actively secure MPC setting that inspires our countermeasures (see Fig. 1). Instead of individual wires at the foundation of the model, we visualize a separation of the chip (integrated circuit) into areas or tiles, consisting of not only many wires, but also complete blocks of combinational and sequential logic. Such tiled designs are inherent in many modern processor architectures, where the tiles correspond to "cores" and the wires correspond to the on-chip interconnect. This can easily be related to a standard MPC architecture where each tile behaves like a separate party. The main difference between our architecture and the MPC setting is that in the latter, parties are assumed to be connected by a complete network of authenticated channels. In our architecture, we know exactly how the wires are connected in the circuit.

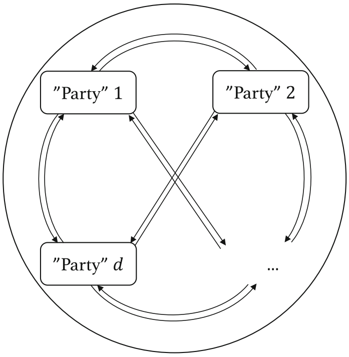

Fig. 1. Partition of the integrated circuit area into tiles, implementing MPC "parties"

The tile architecture satisfies the independent leakage assumption [21] amongst tiles. That is, leakage is local and thus observing the behaviour of a tile by means of probing, faulting or observing its side-channel leakage, does not give unintended information about another tile through, for example, coupling.

As the name implies, the adversary in our model exploits side-channels and introduces faults. We stress that our goal is to *detect* faults as opposed to *tolerate* or *correct* them. That is, if an adversary interjects a fault, we want our system to abort without revealing any of the underlying secrets.

CAPA Methodology. We introduce CAPA, a countermeasure against the tile-probe-and-fault-attacker, which is suitable for implementation in both hardware and software. CAPA inherits theoretical aspects of the MPC protocol SPDZ [20] by similarly computing on shared values, along with corresponding shared MAC tags. The former prevents the adversary from learning sensitive values, while the latter allows for detection of any faults introduced. Moreover, having originated from the MPC protocol SPDZ, CAPA is the first countermeasure with *provable* security against *combined* attacks. The methodology can be scaled to achieve an arbitrary fault detection probability and is suitable for implementation in both hardware and software.

Experimental Results. We provide examples of CAPA designs in hardware of the KATAN and AES block ciphers as well as a software bitsliced implementation of the AES S-box. Our designs show that our methodology is feasible to implement, and in addition our attack experiments confirm our theoretical claims. For example, we implemented a second-order secure hardware implementation of KATAN onto a Spartan-6 FPGA and perform a non-specific leakage detection test, which does not show evidence of first- or second-order leakage with up to 100 million traces. Furthermore, we deploy a second-order secure software based CAPA implementation of the AES S-box on an ARM Cortex-M4 and take electromagnetic measurements; for this implementation neither first-nor second-order leakage is detected with up to 200 000 traces. Using toy parameters, we verify our claimed fault detection probability for the AES S-box software implementation. It should be noted that our experimental implementations are to be considered only proof-of-concept; they are currently too expensive to be used in practice. But the designs demonstrate that the overall methodology can provide significant side-channel and fault protection, and they provide a benchmark against which future improvements can be measured.

2 The Tile-Probe-and-Fault Model

The purpose of this section is to introduce a new adversarial model in which our security guarantees are based. This model is strictly more powerful than the traditional DPA or DFA models.

Tile Architecture. Consider a partition of the chip in a number of tiles $\mathcal{T}_i$, with wires running between each pair of tiles as shown in Fig. 1. We call the set of all tiles $\mathcal{T}$. Each tile $\mathcal{T}_i \in \mathcal{T}$ possesses its own combinational logic, control logic (or program code) and pseudo-random number generator needed for the calculations of one share. In the abstract setting, we consider each tile as the set composed of all input and intermediate values on the wires and memory elements of those blocks. A probe-and-fault attacker may obtain, for a given subset of tiles, *all* the internal information at given time intervals on this set of tiles. He may also inject faults (known or random) into each tile in this set.

In our model, each sensitive variable is split into d shares through secret sharing. Without loss of generality, we use Boolean sharing in this paper.

We define each tile such that it stores and manipulates at most one share of each intermediate variable. Any wire running from one tile to another carries only blinded versions of a sensitive variables' share used by $\mathcal{T}_i$. We make minimal assumptions on the security of these wires. Instead, we include all the information on the unidirectional wires in Fig. 1 in the tile on the *receiving* and not the *sending* end. We thus assume only one tile is affected by an integrity failure of a wire. We assume that shared calculations are performed in parallel without loss of generalization. The redundancy of intermediate variables and logic makes the tiles completely independent apart from the communication through wires.

Probes. Throughout this work, we assume a powerful d_p-probing adversary where we give an attacker information about all intermediate values possessed by d_p specified tiles, *i.e.* $\cup_{i \in i_1,\ldots,i_{d_p}} \mathcal{T}_i$. The attacker obtains *all* the intermediate values on the tile (such as internal wire and register values) with probability one and obtains these values from the start of the computation until the end. Note that this is stronger than both the standard t-probing adversary which gives access to only t intermediate *values* within a certain amount of time [34] and ϵ-probing adversary where the information about t intermediate *values* is gained with certain probability. In our d_p-probing model, the adversary gets information from n intermediate values from d_p tiles where $n \gg d_p$. Therefore, our d_p-probing model is more generic and covers realistic scenarios including an attacker with a limited number of EM probes which enable observation of multiple intermediate values simultaneously within arbitrarily close proximity on the chip.

Faults. We also consider two types of fault models. Firstly, a d_f-faulting adversary which can induce chosen-value faults in any number of intermediate bits/values within d_f tiles, *i.e.* from the set $\cup_{i \in i_1,\ldots,i_{d_f}} \mathcal{T}_i$. These faults can have the nature of *either* flipping the intermediate values with a pre-calculated (adversarially chosen) offset *or* setting the intermediate values to a chosen fixed value. In particular, the faults are not limited in hamming weight. One can relate this type of faults with, for example, very accurate laser injections.

Secondly, we consider an ϵ-faulting adversary which is able to insert a random-value fault in *any* variable belonging to *any* party. This is a somehow new MPC model, and essentially means that all parties are randomly corrupted.

The ϵ-adversary may inject the random-value fault according to some distribution (for example, flip each bit with certain probability), but he cannot set all intermediates to a chosen fixed value. This adversary is different from the d_f-faulting adversary. One can relate the ϵ-faulting adversary to a certain class of non-localised EM attacks.

Time Periods. We assume a notion of time periods; where the period length is at least one clock cycle. We require that a d_f-fault to an adversarially chosen value cannot be preceded by a probe within the same time period. Thus adversarial faults can only depend on values from previous time periods. This time restriction is justified by practical experimental constraints; where the time period is naturally upper bounded by the time it takes to set up such a specific laser injection.

Adversarial Models. Given the aforementioned definitions, we consider on the one hand an active adversary $\mathscr{A}_1$ with both d_p-probing and d_f-faulting capabilities simultaneously. We define $\mathcal{P}_1$ the set of up to d_p tiles that can be probed and $\mathcal{F}_1$ the set of up to d_f tiles that can be faulted by $\mathscr{A}_1$. Since each tile potentially sees a different share of a variable and we use a d-sharing for each variable, we constrain the attack surface (the sets of adversarially probed and potentially modified tiles) as follows:

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

The constraint implies that at least one share remains unaccessed/honest and thus $|\mathcal{F}_1 \cup \mathcal{P}_1| \leq d-1$. Within those $d-1$ tiles, the adversary can probe and fault arbitrarily many wires, including the wires *arriving* at each tile. The adversary's d_f-faulting capabilities are limited in time by our definition of time periods, which implies that any d_f-fault cannot be preceded by another probe within the same time period.

We also consider an active adversary $\mathscr{A}_2$ that has d_p-probing and ϵ-faulting capabilities simultaneously. In this case, the constraint on the set of probed tiles $\mathcal{P}_2$ remains the same:

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

but the set of faulted tiles is no longer constrained:

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

Moreover, as ϵ-faults do not require the same set-up time as d_f-faults, they are not limited in time. Note that, ϵ-faults do not correspond to a standard adversary model in the MPC literature; thus this part of our model is very much an aspect of our side-channel and fault analysis focus. A rough equivalent model in the MPC literature would be for an honest-but-curious adversary who is able to replace the share or MAC values of honest players with values selected from a given random distribution. Whilst such an attack makes sense in the hardware model we consider, in the traditional MPC literature this model is of no interest due to the supposed isolated nature of computing parties.

As our constructions are based on MPC protocols which are *statically secure* we make the same assumptions in our tile-probe-and-fault model, *i.e.* the selection of tiles attacked must be fixed beforehand and cannot depend on information gathered during computation. This model reflects realistic attackers since it is infeasible to move a probe or a laser during a computation with today's resources. We thus assume that both adversaries $\mathscr{A}_1$ and $\mathscr{A}_2$ are static.

3 The CAPA Design

The CAPA methodology consists of two stages. A preprocessing step generates auxiliary data, which is used to perform the actual cryptographic operation in the evaluation step. We first present some notation, then the building blocks for the main evaluation, and finally the preprocessing components.

Notation. Although generalization to any finite field holds, in this paper we work over a field $\mathbb{F}_q$ with characteristic 2, for example $GF(2^k)$ for a given k, as this is sufficient for application to most symmetric ciphers. We use $\cdot$ and $+$ to describe multiplication and addition in $\mathbb{F}_q$ respectively. We use upper case letters for constants. The lower case letters x, y, z are reserved for the variables used only in the evaluation stage (e.g. sensitive variables) whereas $a, b, c, \ldots$ represent auxiliary variables generated from randomness in the preprocessing stage. The kronecker delta function is denoted by $\delta_{i,j}$. We use $L(.)$ to denote an additively homomorphic function and $A(.) = C + L(.)$ with C some constant.

Information Theoretic MAC Tags and the MAC Key α. We represent a value $a \in \mathbb{F}_q$ (similarly $x \in \mathbb{F}_q$) as a pair $\langle \boldsymbol{a} \rangle = (\boldsymbol{a}, \boldsymbol{\tau^a})$ of data and tag shares in the masked domain. The data shares $\boldsymbol{a} = (a_1, \ldots, a_d)$ satisfy $\sum a_i = a$. For each $a \in \mathbb{F}_q$, there exists a corresponding MAC tag τ^a computed as $\tau^a = \alpha \cdot a$, where α is a MAC key, which is secret-shared amongst the tiles as $\alpha = \sum \alpha_i$.

Analogously to the data, the MAC tag is shared $\boldsymbol{\tau^a} = (\tau_1^a, \ldots, \tau_d^a)$, such that it satisfies $\sum \tau_i^a = \tau^a$, but the MAC key itself does not carry a tag. Depending on a security parameter m, there can be m independent MAC keys $\alpha[j] \in \mathbb{F}_q$ for $j \in \{1, \ldots, m\}$. In that case, α as well as τ^a are in $\mathbb{F}_q^m$ and the tag shares satisfy $\sum \tau_i^a[j] = \tau^a[j] = \alpha[j] \cdot a$, $\forall j \in \{1, \ldots, m\}$. Further we assume $m = 1$ unless otherwise mentioned.

3.1 Evaluation Stage

We let each tile $\mathcal{T}_i$ hold the i^{th} share of each sensitive and auxiliary variable $(x_i, \ldots, a_i, \ldots)$ and the MAC key share α_i. We first describe operations that do not require communication between tiles.

Addition. To compute the addition $(\boldsymbol{z}, \boldsymbol{\tau}^{\boldsymbol{z}})$ of $(\boldsymbol{x}, \boldsymbol{\tau}^{\boldsymbol{x}})$ and $(\boldsymbol{y}, \boldsymbol{\tau}^{\boldsymbol{y}})$, each tile performs local addition of their data shares $z_i = x_i + y_i$ and their tag shares $\tau_i^z = \tau_i^x + \tau_i^y$. When one operand is public (for example, a cipher constant $C \in \mathbb{F}_q$), the sum can be computed locally as $z_i = x_i + C \cdot \delta_{i,1}$ for value shares and $\tau_i^z = \tau_i^x + C \cdot \alpha_i$ for tag shares.

Multiplication by a Public Constant. Given a public constant $C \in \mathbb{F}_q$, the multiplication $(\boldsymbol{z}, \boldsymbol{\tau}^{\boldsymbol{z}})$ of $(\boldsymbol{x}, \boldsymbol{\tau}^{\boldsymbol{x}})$ and C is obtained locally by setting $z_i = C \cdot x_i$ and $\tau_i^z = C \cdot \tau_i^x$.

The following operations, on the other hand, require auxiliary data generated in a preprocessing stage and also communication between the tiles.

Multiplication. Multiplication of $(\boldsymbol{x}, \boldsymbol{\tau}^{\boldsymbol{x}})$ and $(\boldsymbol{y}, \boldsymbol{\tau}^{\boldsymbol{y}})$ requires as auxiliary data a Beaver *triple* $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$, which satisfies $c = a \cdot b$, for random a and b. The multiplication itself is performed in four steps.

- Step A. In the *blinding* step, each tile $\mathcal{T}_i$ computes locally a randomized version of its share of the secret: $\varepsilon_i = x_i + a_i$ and $\eta_i = y_i + b_i$.
- Step B. In the *partial unmasking* step, each tile $\mathcal{T}_i$ broadcasts its own shares ε_i and η_i to other tiles, such that each tile can construct and store locally the values $\varepsilon = \sum \varepsilon_i$ and $\eta = \sum \eta_i$. The value ε (resp. η) is the *partial unmasking* of $(\boldsymbol{\varepsilon}, \boldsymbol{\tau}^{\boldsymbol{\varepsilon}})$ (resp. $(\boldsymbol{\eta}, \boldsymbol{\tau}^{\boldsymbol{\eta}})$), *i.e.* the value ε (resp. η) is unmasked but its tag $\boldsymbol{\tau}^{\boldsymbol{\varepsilon}}$ $(\boldsymbol{\tau}^{\boldsymbol{\eta}})$ remains shared. These values are blinded versions of the secrets x and y and can therefore be made public.
- Step C. In the *MAC-tag checking* step, the tiles check whether the tags $\boldsymbol{\tau}^{\boldsymbol{\varepsilon}}$ $(\boldsymbol{\tau}^{\boldsymbol{\eta}})$ are consistent with the public values ε and η, using a method which we will explain later in this section.
- Step D. In the *Beaver computation* step, each tile locally computes

$$z_i = c_i + \varepsilon \cdot b_i + \eta \cdot a_i + \varepsilon \cdot \eta \cdot \delta_{i,1}$$
$$\tau_i^z = \tau_i^c + \varepsilon \cdot \tau_i^b + \eta \cdot \tau_i^a + \varepsilon \cdot \eta \cdot \alpha_i.$$

It can be seen easily that the sharing $(\boldsymbol{z}, \boldsymbol{\tau}^{\boldsymbol{z}})$ corresponds to $z = x \cdot y$ unless faults occurred. Step B and C are the only steps that require communication among tiles. Step A and D are completely local. Note that to avoid leaking information on the sensitive data x and y, the shares ε_i and η_i must be synchronized using memory elements after step A, before being released to other tiles in step B. Moreover, we remark that step C does not require the result of step B and can thus be performed in parallel.

Squaring. Squaring is a linear operation in characteristic 2 fields. Hence, the output shares of a squaring operation can be computed locally using the input shares. However, obtaining the corresponding tag shares is non-trivial. To square $(\boldsymbol{x}, \boldsymbol{\tau}^{\boldsymbol{x}})$ into $(\boldsymbol{z}, \boldsymbol{\tau}^{\boldsymbol{z}})$, we therefore require an auxiliary *tuple* $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle)$ such that $b = a^2$. The procedure to obtain $(\boldsymbol{z}, \boldsymbol{\tau}^{\boldsymbol{z}})$ mimics that of multiplication with some modifications: there is only one partially unmasked value $\varepsilon = x + a$, whose tag needs to be checked, and each tile calculates $z_i = b_i + \varepsilon^2 \cdot \delta_{i,1}$ and $\tau_i^z = \tau_i^b + \varepsilon^2 \cdot \alpha_i$.

Following the same spirit, we can also perform the following operations.

Affine Transformation. Provided that we have access to a tuple $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle)$ such that $b = A(a)$, we can compute $(\boldsymbol{z}, \boldsymbol{\tau}^{\boldsymbol{z}})$ satisfying $z = A(x) = C + L(x)$, where $L(x)$ is an additively homomorphic function over the finite field, by computing the output sharing as $z_i = b_i + L(\varepsilon) \cdot \delta_{i,1}$ and $\tau_i^z = \tau_i^b + L(\varepsilon) \cdot \alpha_i$.

Multiplication following Linear Transformations. The technique used for the above additively homomorphic operations can be generalized even further to compute $z = L_1(x) \cdot L_2(y)$ in shared form, where L_1 and L_2 are additively homomorphic functions. A trivial methodology would require two tuples $(\langle \boldsymbol{a_i} \rangle, \langle \boldsymbol{b_i} \rangle)$ with $b_i = L_i(a_i)$ for $i \in \{1, 2\}$, plus a standard Beaver triple (*i.e.* requiring seven pre-processed data items). We see that we can do the same operation with five pre-processed items $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle, \langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle)$, such that $c = L_1(a)$, $d = L_2(b)$ and $e = L_1(a) \cdot L_2(b)$. The tiles partially unmask $\boldsymbol{x} + \boldsymbol{a}$ (resp. $\boldsymbol{y} + \boldsymbol{b}$) to obtain ε (resp. η) and verify them. Each tile computes its value share and tag share of z as $z_i = e_i + L_1(\varepsilon) \cdot d_i + L_2(\eta) \cdot c_i + L_1(\varepsilon) \cdot L_2(\eta) \cdot \delta_{i,1}$ and $\tau_i^z = \tau_i^e + L_1(\varepsilon) \cdot \tau_i^d + L_2(\eta) \cdot \tau_i^c + L_1(\varepsilon) \cdot L_2(\eta) \cdot \alpha_i$, respectively. We refer to $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle, \langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle)$ as a *quintuple.*

Proof

$$\begin{aligned}
\sum_{i=1}^{d} z_i &= \sum_{i=1}^{d} \left(e_i + L_1(\varepsilon) \cdot d_i + L_2(\eta) \cdot c_i\right) + L_1(\varepsilon) \cdot L_2(\eta) \\
&= \sum_{i=1}^{d} e_i + L_1(\varepsilon) \cdot \sum_{i=1}^{d} d_i + L_2(\eta) \cdot \sum_{i=1}^{d} c_i + L_1(\varepsilon) \cdot L_2(\eta) \\
&= L_1(a) \cdot L_2(b) + L_1(x + a) \cdot L_2(b) + L_2(y + b) \cdot L_1(a) + L_1(x + a) \cdot L_2(y + b) \\
&= L_1(a) \cdot L_2(b) + L_1(x) \cdot L_2(b) + L_1(a) \cdot L_2(b) + L_1(a) \cdot L_2(y) + L_1(a) \cdot L_2(b) \\
&+ L_1(x) \cdot L_2(y) + L_1(x) \cdot L_2(b) + L_1(a) \cdot L_2(y) + L_1(a) \cdot L_2(b) \\
&= L_1(x) \cdot L_2(y) \\
\sum_{i=1}^{d} \tau_i^z &= \sum_{i=1}^{d} \left(\tau_i^e + L_1(\varepsilon) \cdot \tau_i^d + L_2(\eta) \cdot \tau_i^c + L_1(\varepsilon) \cdot L_2(\eta) \cdot \alpha_i\right) \\
&= \sum_{i=1}^{d} \tau_i^e + L_1(\varepsilon) \cdot \sum_{i=1}^{d} \tau_i^d + L_2(\eta) \cdot \sum_{i=1}^{d} \tau_i^c + L_1(\varepsilon) \cdot L_2(\eta) \cdot \sum_{i=1}^{d} \alpha_i \\
&= \alpha \cdot e + L_1(\varepsilon) \cdot \alpha \cdot d + L_2(\eta) \cdot \alpha \cdot c + L_1(\varepsilon) \cdot L_2(\eta) \cdot \alpha \\
&= \alpha \cdot \left(e + L_1(\varepsilon) \cdot d + L_2(\eta \cdot c + L_1(\varepsilon) \cdot L_2(\eta))\right) \\
&= \alpha \cdot L_1(x) \cdot L_2(y)
\end{aligned}$$

Checking the MAC Tag of Partially Unmasked Values. Consider a public value $\varepsilon = x + a$, calculated in the partial unmasking step of the Beaver multiplication operation. Recall that we obtain its MAC-tag shares as follows: $\tau_i^\varepsilon = \tau_i^a + \tau_i^x$. During the MAC-tag checking step of the Beaver operation, the authenticity of τ^{ε} corresponding to ε is tested. As ε is public, each tile can calculate and broadcast the value $\varepsilon \cdot \alpha_i + \tau_i^{\varepsilon}$. For a correct tag, we expect $\sum \tau_i^{\varepsilon} = \alpha \cdot \varepsilon$, thus each tile computes $\sum(\varepsilon \cdot \alpha_i + \tau_i^{\varepsilon})$ and proceeds if the result is zero. Recall that the broadcasting must be preceded by a synchronization of the shares.

Note on Unmasked Values/Calculations. There are several components in a cipher which do not need to be protected against SCA (*i.e.* masked), because their specific values are not sensitive. One prominent example is the control unit which decides what operations should be performed (*e.g.* the round counter). Other examples are constants such as the AES affine constant `0x63` or public values such as ε in a Beaver calculation and the difference $\varepsilon \cdot \alpha + \tau^{\varepsilon}$ during the MAC-tag checking phase.

While these *public* components are not sensitive in a SCA context, they *can* be targeted in a fault attack. It is therefore important to introduce some redundancy. Each tile should have its own control logic and keep a local copy of all public values to avoid single points of attack. The shares ε_i are distributed to *all* tiles so that ε can be unmasked by each tile separately and any subsequent computation performed on these public values is repeated by each tile. Finally, each tile also keeps its own copy of the abort status. This is in fact completely analogous to the MPC scenario.

3.2 Preprocessing Stage

The auxiliary data $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \ldots)$ required in the Beaver evaluations, is generated in a preprocessing stage. This preparation corresponds to the *offline* phase in SPDZ. However, CAPA's preprocessing stage is lighter and does not require a public key calculation due to the differences in adversary model. As in SPDZ, this stage is completely independent from the sensitive data of the main evaluation. Below, we describe the generation of a Beaver triple used in multiplication. This can trivially be generalized to tuples and quintuples.

Auxiliary Data Generation. To generate a triple $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ satisfying $c = a \cdot b$, we draw random shares $\boldsymbol{a} = (a_1, \ldots, a_d)$ and $\boldsymbol{b} = (b_1, \ldots, b_d)$ and use a passively secure shared multiplier to compute $\boldsymbol{c}$ s.t. $c = a \cdot b$. We then use another such multiplication with the shared MAC key $\boldsymbol{\alpha}$ to generate tag shares $\boldsymbol{\tau}^{\boldsymbol{a}}$, $\boldsymbol{\tau}^{\boldsymbol{b}}$, $\boldsymbol{\tau}^{\boldsymbol{c}}$. We note that the shares a_i, b_i are randomly generated by tile $\mathcal{T}_i$. There are thus d separate PRNG's on d distinct tiles.

Passively Secure Shared Multiplier. For a secure implementation of a shared multiplication, no subset of $d-1$ tiles should have access to all shares of any variable. This concept, which is used in the context of secure implementations against SCA on hardware, is precisely called $d-1^{\text{th}}$-order non-completeness in [9,52]. In the last decade, there has been significant improvement on passively secure shared multipliers that can be used in both hardware and software [9,27,29,51,56]. In principle, CAPA can use any such multiplier as long as the tile structure still holds.

A close inspection of existing multipliers show that they require the calculation of the cross products $a_i b_j$. In order to make these multipliers compatible with the CAPA tile architecture, we define tiles $\mathcal{T}_{i,j}$ which receive a_i from $\mathcal{T}_i$ and b_j from $\mathcal{T}_j$ where $i \neq j$ in order to handle the pair (a_i, b_j) to be used during tuple, triple and quintuple generation. This implies $d(d-1)$ smaller tiles used only during auxiliary data generation in addition to d tiles used for both auxiliary data generation and evaluation. The output wires from $\mathcal{T}_{i,j}$ are only connected to $\mathcal{T}_i$ and carry randomized information.

The multipliers used in the preprocessing phase are only *passively* secure. We also ensure resistance against *active* adversaries because on the one hand, deterministic faults are limited to $d-1$ tiles and on the other, because of a relation verification step, which is explained in the next section.

3.3 Relation Verification of Auxiliary Data

The information theoretic MAC tags provide security against faults induced in the evaluation stage. To detect faults in the preprocessing stage, we perform a relation verification of the auxiliary data. This relation verification step is done for each generated triple that is passed from the preprocessing to the evaluation stage and ensures that the triple is functionally correct (*i.e.* $c = a \cdot b$) by sacrificing another triple. That is, we take as input two triples $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ and $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$, that should satisfy the same relation, in this example $c = a \cdot b$ and $f = d \cdot e$. The following Beaver computation holds if and only if both relations are satisfied:

- Draw a random $r_1 \in \mathbb{F}_q$
- Use triple $(\langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle, \langle \boldsymbol{f} \rangle)$ to calculate the multiplication of $r_1 \cdot \langle \boldsymbol{a} \rangle$ and $\langle \boldsymbol{b} \rangle$ using a constant multiplication with r_1, followed by the Beaver equation for multiplication described above. The result $\langle \tilde{\boldsymbol{c}} \rangle$ is a shared representation of $\tilde{c} = r_1 \cdot a \cdot b$.
- For each share i, calculate the difference with the shares and tags of $r_1 \cdot c$: $\Delta_i = r_1 \cdot c_i + \tilde{c}_i$ and $\tau_i^{\Delta} = r_1 \cdot \tau_i^{c} + \tau_i^{\tilde{c}}$.
- Unmask the resulting differences Δ and τ^{Δ}.
- If a difference is nonzero, reject $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ as a valid triple.
- Pick another $r_2 \in \mathbb{F}_q$ such that $r_2 \neq r_1$ and repeat a second time.

Note that this relation verification ensures that the second triple is functionally correct too. However, it is burnt (or "sacrificed") in this process in order to ensure that the first triple can be used securely further on. Note that this relation verification or "sacrificing" step is mandatory in each Beaver-like operation.

Why We Need Randomization. This sacrificing step involves two values r_1 and r_2. We present the following attack to illustrate why this randomization is needed. Again, we elaborate on triples, but the same can be said for tuples and quintuples. As the security does not rely on the secrecy of r_1 and r_2, we assume for simplicity that they are known to the attacker. We only stress that they are different: $r_1 \neq r_2$.

Consider two triples $(\langle \boldsymbol{a}\rangle, \langle \boldsymbol{b}\rangle, \langle \boldsymbol{c}'\rangle)$ and $(\langle \boldsymbol{d}\rangle, \langle \boldsymbol{e}\rangle, \langle \boldsymbol{f}'\rangle)$ at the input of the sacrificing stage. We assume that the adversary has introduced an additive difference into one share of c' and f' such that $c' = a \cdot b + \Delta_c$ and $f' = d \cdot e + \Delta_f$. This fault is injected before the MAC tag calculation, so that $\tau^{c'}$ and $\tau^{f'}$ are valid tags for the faulted values c' and f' respectively. In particular, this means we have $\tau^{c'} = \tau^{c} + \alpha \cdot \Delta_c$ and $\tau^{f'} = \tau^{f} + \alpha \cdot \Delta_f$.

The sacrificing step calculates the following four differences (for $r_j = r_1$ and r_2) and only succeeds if all are zero.

$$\begin{aligned}
\Delta_j &= \sum_{i=1}^{d} \big(r_j \cdot c'_i + f'_i + \varepsilon \cdot e_i + \eta \cdot d_i\big) + \varepsilon \cdot \eta \\
&= r_j \cdot \Delta_c + \Delta_f \stackrel{?}{=} 0 \\
\tau^{\Delta_j} &= \sum_{i=1}^{d} \big(r_j \cdot \tau_i^{c'} + \tau_i^{f'} + \varepsilon \cdot \tau_i^{e} + \eta \cdot \tau_i^{d} + \varepsilon \cdot \eta \cdot \alpha_i\big) \\
&= r_j \cdot \alpha \cdot \Delta_c + \alpha \cdot \Delta_f \stackrel{?}{=} 0
\end{aligned}$$

Without randomization (*i.e.* $r_1 = r_2 = 1$), the attacker only has to match the differences $\Delta_f = \Delta_c$ to pass verification. With a random r_1, the attacker can fix $\Delta_f = r_1 \cdot \Delta_c$ to automatically force Δ_1 and τ^{Δ_1} to zero. Even if he does not know r_1, he has probability as high as 2^{-k} to guess it correctly.

Only thanks to the repetition of the relation verification with r_2, the adversary is detected with a probability $1 - 2^{-km}$. Assuming he fixed $\Delta_f = r_1 \cdot \Delta_c$, it is impossible to also achieve $\Delta_f = r_2 \cdot \Delta_c$. Even if the attacker manages to force Δ_2 to zero with an additive injection (since he knows all components r_2, Δ_c and Δ_f), he cannot get rid of the difference $\tau^{\Delta_2} = r_2 \cdot \alpha \cdot \Delta_c + \alpha \cdot \Delta_f$ without knowing the MAC key. Since α remains secret, the attacker only has a success probability of 2^{-km} to succeed.

4 Discussion

4.1 Security Claims

With both described adversaries $\mathscr{A}_1$ and $\mathscr{A}_2$, our design CAPA claims provable security against the following types of attacks as well as a combined attack of the two

1. Side-Channel Analysis (*i.e.* against $d-1$ tile probing adversary).
2. Fault Attacks (*i.e.* an adversary introducing either known faults into $d-1$ tiles or random faults everywhere).

Side-channel Analysis. One can check that no union of $d-1$ tiles $\cup_{j \in j_1,\ldots,j_{d-1}} \mathcal{T}_j$ has all the shares of a sensitive value. Very briefly, we can reason to this $d-1^{\text{th}}$-order non-completeness as follows. All computations are local with the exception of the unmasking of public values such as ε. However, the broadcasting of all shares of ε does not break non-completeness since $\varepsilon = x + a$ is not sensitive itself but rather a blinded version of a sensitive value x, using a random a that is shared across all tiles. Unmasking the public value ε therefore gives each tile $\mathcal{T}_i$ only one share $\varepsilon + a_i$ of a new sharing of the secret x:

$$\boldsymbol{x} = (a_1, \ldots, a_{i-1}, \varepsilon + a_i, a_{i+1}, \ldots, a_d)$$

In this sharing, no union of $d - 1$ shares suffices to recover the secret. Our architecture thus provides non-completeness for all sensitive values. As a result, our d-share implementation is secure against $d-1$-probing attacks. Any number of probes following the adversaries' restrictions leak no sensitive data. Our model is related to the wire-probe model, but with wires replaced by entire tiles. We can thus at least claim security against $d - 1^{\text{th}}$ order SCA.

Fault Attacks. A fault is only undetected if both value and MAC tag shares are modified such that they are consistent. Adversary $\mathcal{A}_1$ can fault at most $d_f < d$ tiles, which means he requires knowledge of the MAC key $\alpha \in GF(2^{km})$ to forge a valid tag for a faulty value. Since α is secret, his best option is to guess the MAC key. This guess is correct with probability 2^{-km}. Adversary $\mathcal{A}_2$ has ϵ-faulting abilities only and will therefore only avoid detection if the induced faults in value and tag shares *happen* to be consistent. This is the case with probability 2^{-km}. We can therefore claim an error detection probability (EDP) of $1 - \frac{1}{2^{km}}$. The EDP does not depend on the number of faulty bits (or the hamming weight of the injected fault).

Combined Attacks. In a combined attack, an adversary with d_f-faulting capabilities can mount an attack where he uses the knowledge obtained from probing some tiles $\in \mathcal{P}_1$ to carefully forge the faults. In SPDZ, commitments are used to avoid the so called "rushing adversary". CAPA does not need commitments as the timing limitation on $\mathscr{A}_1$ adversary ensures a d_f-fault cannot be preceded by a probe in the same clock cycle. As a result, we inherit the security claims of SPDZ and the claimed EDP is not affected by probing or SCA. Also, the injection of a fault in CAPA does not change the side-channel security. Performing a side-channel attack on a perturbed execution does not reveal any additional information because the Beaver operations do not allow injected faults to propagate through a calculation into a difference that depends on sensitive information. We can claim this security, because of the aspects inherited from MPC. CAPA is essentially secure against a very powerful adversary that has complete control (hence *combined* attacks) over all but one of the tiles.

What Does Our MAC Security Mean? We stress that CAPA provides significantly higher security than existing approaches against faults. An adversary that injects errors in up to d_f tiles cannot succeed with more than the claimed detection probability. This means that our design can stand $d'_f \gg d_f$ shots if they affect at most d_f tiles. This is the case even if those d_f tiles leak their entire state; hence our resistance against combined attacks. The underlying reason for this is that to forge values, an attacker needs to know the MAC key, but since this is also shared, the attacker does not gain any information on the MAC key and their best strategy is to insert a random fault, which is detected with probability $1 - 2^{-km}$. Moreover, our solution is incredibly scalable compared to for example error detection code solutions.

How Much Do Tags Leak? The tag shares τ_i^a form a Boolean masking of a variable τ^a. This variable τ^a itself is an information theoretic MAC tag of the underlying value a and can be seen as a multiplicative share of a. We therefore require the MAC key to change for each execution. Hence MAC tag shares are a Boolean masking of a multiplicative share and are expected to leak very little information in comparison with the value shares themselves.

Forbidding the All-0s MAC Key. If the MAC key size mk is small, we should forbid the all-0 MAC key. This ensures that tags are injective: if an attacker changes a value share, he must change the tag share. We only pay with a slight decrease in the claimed detection probability. By excluding 1 of the 2^{km} MAC key possibilities, we reduce the fault detection probability to $1 - 2^{-\kappa}$, where $\kappa = \log_2(2^{km} - 1)$.

4.2 Attacks

The Glitch Power Supply or Clock Attack. The solution presented in this paper critically depends on the fact that there is no single point where an attacker can insert a fault that affects all d tiles deterministically. An attacker may try to glitch the chip clock line that is shared among all tiles. In this case, the attacker could try to carefully insert a glitch so that writing to the abort register is skipped or a test instruction is skipped. Since all tiles share the same clock, the attacker can bypass in this way the tag verification step. Similar comments apply, for example, to glitches in the power line. The bottom line is that one should design the hardware architecture accordingly, that is, deploy low-level circuit countermeasures that detect or avoid this attack vector.

Skipping Instructions. In software, when each tile is a separate processor (with its own program counter, program memory and RAM memory), skipping one instruction in up to $d - 1$ shares would be detected. The unaffected tiles will detect this misbehavior when checking partially unmasked values.

Safe Error Attack. We point out a specific attack that targets any countermeasure against a probing and faulting adversary. In a safe error attack [65], the attacker perturbs the implementation in a way that the output is only affected if a sensitive variable has a certain value. The attacker learns partial secret information by merely observing whether or not the computation succeeds (*i.e.* does not abort). Consider for example a shared multiplication of a variable x and a secret y and call the resulting product $z = xy$. The adversary faults one of the inputs with an additive nonzero difference such that the multiplication is actually performed on $x' = x + \Delta$ instead of x. Such an additive fault can be achieved by affecting only one share/tile. The multiplication results in the faulty product $z' = z + \Delta \cdot y$. The injected fault has propagated into a difference that depends on sensitive data (y). As a result, the success or failure of any integrity check following this multiplication depends on y. In particular, if nothing happens (all checks pass), the attacker learns that y must be 0.

Among existing countermeasures against combined attacks, none provide protection against this kind of selective failure attack as they cannot detect the initial fault Δ. The attacker can always target the wire running from the last integrity check on x to the multiplication with y. We believe CAPA is currently unique in preventing this type of attack. One can verify that the MAC-tag checking step in a Beaver operation successfully prevents Δ from propagating to the output. This integrity check only passes if all tiles have a correct copy of the public value ε. Any faults injected after this check have a limited impact as the calculation finishes locally. That is, once the correct public values are established between the tiles, the shares of the multiplication output z are calculated without further communication among tiles. The adversary is thus unable to elicit a fault that depends on sensitive data.

PACA. We claim security against the passive and active combined attack (PACA) on masked AES described in [2] because CAPA does not output faulty ciphertexts. A second attack in this work uses another type of safe errors (or ineffective faults as they are called in this work) which are impossible to detect. The attacker fixes a specific wire to the value zero (this requires the d_f-faulting capability) and collects power traces of the executions that succeed. This means the attacker only collects traces of encryptions in which that specific wire/share was already zero. The key is then extracted using $d-1^{\text{th}}$-order SCA on the remaining $d-1$ shares. This safe error attack however falls outside our model since the adversary gets access (either by fault or SCA) to all d shares and thus $(\mathcal{F}_1 \cup \mathcal{P}_1) = \mathcal{T}$.

Advanced Physical Attacks. In our description we are assuming that during the broadcast phase there are no "races" between tiles: by design, each tile sets its share to be broadcasted at clock cycle t and captures other tiles' share in the same clock cycle t. We are implicitly assuming that tiles cannot do much work between these two events. If this assumption is violated (for example, using advanced circuit editing tools), a powerful adversary could bypass any verification. This is why in the original SPDZ protocol there are commitments prior to broadcasting operations; if this kind of attack is a concern one could adapt the same principles of commitments to CAPA. This is a very strong adversarial model that we consider out of scope for this paper.

4.3 Differences with SPDZ

Offline Phase. In SPDZ, the auxiliary data is generated using a somewhat homomorphic encryption scheme. The mapping onto a chip environment thus seems prohibitive due to the need for this expensive public-key machinery to obtain full threshold and the large storage required. We avoid this by generating the Beaver triples using passively secure shared multipliers. Furthermore, to avoid the large storage requirement, we produce the auxiliary data on the fly whenever required.

MAC Tag Checking. SPDZ delays the tag checking of public values until the very end of the encryption by using commitments. For this, each party keeps track of publicly opened values. This is to avoid a slowdown of the computation and because in the MPC setting, local memory is cheaper than communication costs. In an embedded scenario the situation is opposite so we check the opened values on the fly at the cost of additional dedicated circuit. In hardware, we "simulate" the broadcast channel by wiring between all tiles. Each tile keeps a local copy of those broadcasted values.

Adversary. Although MPC considers mainly the "synchronous" communication model, the SPDZ adversary model also includes the so-called "rushing" adversary, which first collects all inputs from the other parties and only then decides what to send in reply. In our embedded setting, as already pointed out, the "rushing" adversary is impossible. Due to the nature of the implementation, the computational environment and storage is very much restricted. On the other hand, communication channels are very efficient and can be assumed to be automatically synchronous with all tiles progressing in-step in the computation.

4.4 Cost Analysis and Scalability

The computation as described in Sect. 3.1 scales nicely with the masking order d and the security parameter m. For any fixed number of shares d, the circuit area scales linearly in m (see for example Table 2). Storage increases with a factor $(m+1)d$ compared to a plain implementation. We note that our implementations run in almost the same amount of cycles as that of a plain implementation. There is almost no loss in throughput and only negligible in latency. In software as well, the timing scales linearly if tiles run in parallel (Table 1).

Table 1. Overview of the number of $\mathbb{F}_q$ multiplications (.), $\mathbb{F}_q$ additions (+) and linear operations in $GF(2)$ ($L(.)$) required to calculate all building blocks with d shares and m tags

	Public Values			Output calculation				MAC check	
				Value		Tags			
	$\cdot$	$+$	$L(.)$	$\cdot$	$+$	$\cdot$	$+$	$\cdot$	$+$
Add.					d		dm		
Add. with C					1	dm	dm		
Multip. with C				d		dm			
Multip.	d	$2d+2(d-1)d$		$2d$	$2d+1$	$3dm$	$3dm$	$2dm$	$4dm+2(d-1)dm$
Square/Affine		$d+(d-1)d$	d		1	dm	dm	dm	$2dm+(d-1)dm$
$L_1(x) \cdot L_2(y)$	d	$2d+2(d-1)d$	$d+d$	$2d$	$2d+1$	$3dm$	$3dm$	$2dm$	$4dm+2(d-1)dm$

This efficiency does not come for free. The complexity is shifted to the preprocessing stage; indeed the generation of auxiliary triples is the most expensive part of the implementation. There is a trade-off to be made here between the online and offline complexity. The more auxiliary data we prepare "offline", the more efficient the online computation.

Complexity for Passive Attacker Scenario. It is remarkable that if active attackers are ruled out, and only SCA is a concern, then the complexity of the principal computation is linear in d. This may seem like a significant improvement over previous masking schemes which have quadratic complexity on the security order [18,34,58]. However, this complexity is again pushed into the preprocessing stage. Nevertheless, this can be interesting especially for software implementations in platforms where a large amount of RAM is available to store the auxiliary data generated in Sect. 3.2. The same comments apply to FPGAs with plenty of BlockRAM.

Optimization of Preprocessing. It may be beneficial to store the output of the preprocessing stage Sect. 3.2 in a table for later usage. One could optimize this process by recycling auxiliary data (sample elements with replacement from the table). Of course, this would void the provable security claims; but if performed with care (with appropriate table shuffling and table elements refresh), this can give rise to an implementation that is secure in practice.

5 Proof-of-Concept

In this section we detail a proof-of-concept implementation of the CAPA methodology in both a hardware and a software environment. We emphasize specific concepts for hardware and software implementations and provide case studies of KATAN-32 [14] and AES [1], which cover operations in different fields, possibility of bitsliced implementations, specific timing and memory optimizations, and performance results.

5.1 Hardware Implementations

We now describe two case studies for applying CAPA in hardware. Our implementations are somewhat optimized for latency rather than area with d tiles spatially separated and operating in parallel, each with its own combinational and control logic and auxiliary data preparation module. These preparation modules are equipped with a passively secure shared multiplication with higher-order non-completeness. Literature provides us with a broad spectrum of multipliers to choose from [9,27,29,51,56]. In order to minimize the randomness requirement, our implementation uses the one from [29], hereafter referred to as DOM.

Library. For synthesis, we use Synopsis Design Compiler Version I-2013.12 using the NanGate 45 nm Open Cell library [49] for ease of future comparisons. We choose the compile option `- exact_map` to prevent optimization across tiles. The area results are provided in 2-input NAND-gate equivalents (GE).

Table 2. Area (GE) of 2-share KATAN-32 implementations with m MAC keys $\alpha[j] \in \mathbb{F}_q$

	No tags	$m = 1$	$m = 8$	Any m
- Evaluation	2 315	4 708	21 404	$\approx 2\,315 + 2\,390\,\mathrm{m}$
* Shift Register	888	1 823	8 419	$\approx 888 + 935\,\mathrm{m}$
* Key Schedule	1 427	2 885	12 985	$\approx 1\,427 + 1\,455\,\mathrm{m}$
- Preprocessing (x3)	363	679	2 727	$\approx 363 + 315\,\mathrm{m}$
* Two triple generation	237	431	1 786	$\approx 237 + 195\,\mathrm{m}$
* Relation verification	126	248	941	$\approx 126 + 120\,\mathrm{m}$
Total	3 672	7 103	30 596	$\approx 3\,672 + 3\,430\,\mathrm{m}$

Case Study: KATAN-32. KATAN-32 is a shift register based block cipher, which has a 80-bits key and processes 32-bit plaintext input. It is designed specifically for efficient hardware implementations and performs 254 cycles of four AND-XOR operations. Hence, its natural shared data representation is in the field $\mathbb{F}_q = \mathrm{GF}(2)$, which makes the mapping into CAPA operations relatively straightforward. However, the small finite field means that we need to utilize a vectorized MAC-tag operation ($m > 1$) to ensure a good probability of detecting errors. Our implementation is round based, as in [14] with three AND-XOR Beaver operations and one constant AND-XOR calculated in parallel. Each Beaver AND-XOR operation requires two cycles, and is implemented in a pipelined fashion such that the latency of the whole computation increases only by one clock cycle.

Implementation Cost. Tables 2 and 3 summarize the area of our KATAN implementations. Naturally, compared to a shared implementation without MAC tags, the state registers grow with a factor $m + 1$ as the MAC-key size increases. In the last columns, we extrapolate the area results for any m.

Each Beaver multiplication in GF(2) requires one triple, and each triple needs $2d$ random bits for generating $\boldsymbol{a}$ and $\boldsymbol{b}$. A d-share DOM multiplication requires $\binom{d}{2}$ units of randomness. The construction of one triple requires $1 + 3m$ masked

Table 3. Area (GE) of 3-share KATAN-32 implementations with m MAC keys $\alpha[j] \in \mathbb{F}_q$

	No tags	$m = 1$	$m = 8$	Any m
- Evaluation	3 560	7 139	32 368	$\approx 3\,560 + 3\,580\,\mathrm{m}$
* Shift Register	1 363	2 812	12 890	$\approx 1\,363 + 1\,450\,\mathrm{m}$
* Key Schedule	2 197	4 327	19 478	$\approx 2\,197 + 2\,130\,\mathrm{m}$
- Preprocessing (x3)	638	1 468	7 124	$\approx 638 + 830\,\mathrm{m}$
* Two triple generation	428	952	4 694	$\approx 428 + 524\,\mathrm{m}$
* Relation verification	210	516	2 430	$\approx 210 + 306\,\mathrm{m}$
Total	5 971	12 083	55 254	$\approx 5\,971 + 6\,112\,\mathrm{m}$

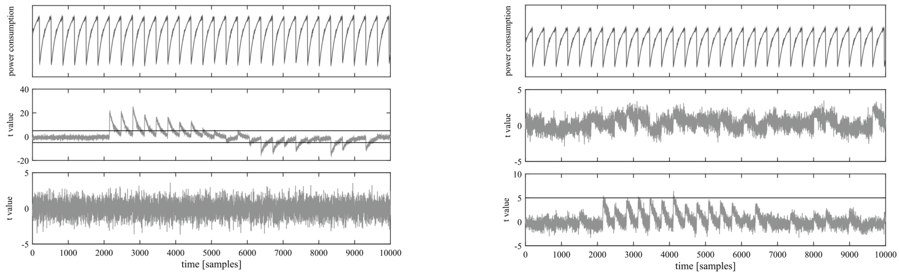

Fig. 2. Non-specific leakage detection on the first 31 rounds of first-order KATAN. Left column: PRNG off (24K traces). Right column: PRNG on (100M traces). Rows (top to bottom): exemplary power trace; first-order t-test; second-order t-test

multiplications: one to obtain the multiplication $\boldsymbol{c}$ of $\boldsymbol{a}$ and $\boldsymbol{b}$; and $3m$ to obtain the m tags $\boldsymbol{\tau}^{\boldsymbol{a}}$,$\boldsymbol{\tau}^{\boldsymbol{b}}$ and $\boldsymbol{\tau}^{\boldsymbol{c}}$. Due to the relation verification through the sacrificing of another triple, the randomness must be doubled. Hence, the total required number of random bits per round of KATAN is $3 \cdot 2 \cdot (2d + (1 + 3m)\frac{d(d-1)}{2}))$.

Experimental Validation. The goal of the prior proof-of-concept implementation is to experimentally validate the protection against side-channel attacks offered by the CAPA methodology. We deploy a first- and second-order secure KATAN instance onto a Xilinx Spartan-6 FPGA. Our platform is a Sakura-G board specifically designed for side-channel evaluation with two FPGA's to minimize platform noise: a control FPGA handles I/O with the host computer and supplies masked data to the crypto FPGA, which implements both the preprocessing and evaluation. The KATAN implementations use $d = 2$ (resp. $d = 3$) shares and $m = 2$ MAC keys. The parameter $m = 2$ is insufficient in practice, but serves for this experiment since m has no influence on SCA security. The designs are

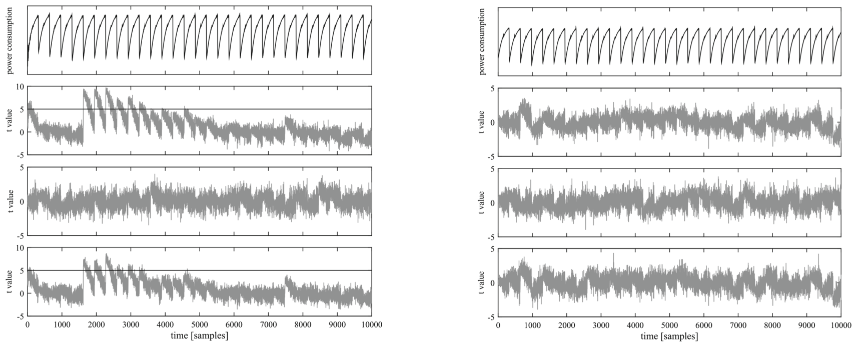

Fig. 3. Non-specific leakage detection on the first 31 rounds of second-order KATAN. Left column: PRNG off (24K traces). Right column: PRNG on (100M traces). Rows (top to bottom): exemplary power trace; first-order t-test; second-order t-test; third-order t-test

clocked at 3 MHz and we sample power traces of 10 000 time samples each at 1 GS/s. Exemplary traces are shown in Fig. 2, top.

We perform a non-specific leakage detection test [17] following the methodology from [57,61]. First, we test the designs without masks to verify that our setup is indeed sound and able to detect leakage. Then we switch on the PRNG and corroborate that the design does not leak with high confidence.

In Fig. 2, we show the results for the first-order secure design ($d = 2$). In the left column, the PRNG is turned off, emulating an unmasked design. Indeed, we see clear leakage at first order, since the t-statistics cross the threshold 4.5. With the PRNG on (right column), no first-order leakage is detected with up to 100 million traces. As expected, we do see second-order leakage. Figure 3 exhibits the results for the second-order secure design ($d = 3$). The left column shows clear leakage at first, second and third order when the PRNG is turned off. In the right column, we repeat the procedure with PRNG on and no univariate leakage is detected with up to 100 million traces.[1]

Case Study: AES. There has been a great deal of work on MPC and masked implementations of the basic AES operations. We take what has now become the traditional approach and work in the field GF(2^8) with $m = 1$ for AES, *i.e.* the MAC key, data and tag shares α_i, a_i and τ_i^a are $\in$ GF(2^8). The ShiftRows and MixColumns operations are linear in GF(2^8), hence are straightforward. Here, we only describe the S-box calculation.

Design Choices. The AES S-box consists of an inversion in GF(2^8), followed by an affine transformation over bits. We distinguish two methodologies for the S-box implementation: It is well known that the combination of the two operations can be expressed by the following polynomial in GF(2^8) [19]:

$$\begin{aligned}\text{S-box}(x) = \texttt{0x63} + \texttt{0x8F} \cdot x^{127} + \texttt{0xB5} \cdot x^{191} + \texttt{0x01} \cdot x^{223} + \texttt{0xF4} \cdot x^{239} \\ + \texttt{0x25} \cdot x^{247} + \texttt{0xF9} \cdot x^{251} + \texttt{0x09} \cdot x^{253} + \texttt{0x05} \cdot x^{254}\end{aligned} \tag{1}$$

This polynomial can be implemented using 6 squares and 7 multiplications in GF(2^8) with a latency of 13 clock cyles. A second approach is to evaluate the inversion $x \longrightarrow x^{254}$ using the following multiplication chain from [30]:

$$x^{254} = x^4 \cdot \left(\left((x^5)^5\right)^5\right)^2$$

Since the AES affine transform $A(x)$ is linear over GF(2), we can then use the Beaver operation described in Sect. 3.1 to evaluate it in one cycle, using auxiliary affine tuples $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle)$ such that $b = A(a)$. Initial estimations reveal the former method is more expensive than the latter, so we adopt the latter technique.

[1] Since our implementation handles 3 shares, we expect to detect leakage in the third order. Due to platform noise, this is not visible.

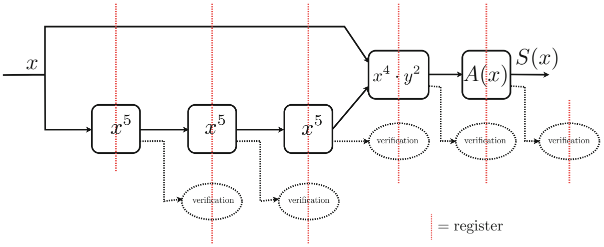

Fig. 4. AES S-box pipeline

Multiplication Chain. Our implementation of the proposed multiplication chain uses two types of operations: x^5 and $x^4 \cdot y^2$, which can both be computed as described in Sect. 3.1 (Multiplication following Linear Transformations). Given an input $\langle \boldsymbol{x} \rangle$ and a triple $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle)$ such that $b = a^4$ and $c = a^5$, we calculate the CAPA exponentiation to the power five. Likewise, we perform the map $x^4 \cdot y^2$ (with $y = x^{125}$) in one cycle, using quintuples $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle, \langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle)$ such that $c = a^4$, $d = b^2$ and $e = c \cdot d = a^4 \cdot b^2$. As a result, an inversion in $\mathrm{GF}(2^8)$ costs only 4 cycles, using 3 exponentiation triples and 1 quintuple. Combined with the affine stage, we obtain the S-box output in 5 cycles (see Fig. 4). This approach does not only optimize the number of cycles but also the amount of required randomness. The S-box is implemented as a five stage pipeline.

Implementation Cost. We use a serialized AES architecture, based on that in [28]. One round of the cipher requires 21 clock cycles, making the latency of one complete encryption 226 clock cycles. Since the unprotected serialised implementation of [47] also requires 226 cycles, the timing performance is very good.

Table 4 presents the area for the different blocks that make up our AES implementation. We can see a significant difference between the preprocessing and evaluation stages, *i.e.* the efficient calculation phase comes at the cost of expensive resource generation machinery.

Table 5 summarizes the required number of random bytes for the generation of the triples/tuples for the AES S-box as a function of the number of MAC keys m and the number of shares d. Recall that the S-box needs three exponentiation triples, one quintuple and one affine tuple per cycle (doubled for the sacrificing). Each of these uses d initial bytes of randomness per input for the shares of a (and b). Furthermore, recall that each masked multiplication requires $\binom{d}{2}$ bytes or randomness. That is, for $d = 3$ and $m = 1$, we need 156 bytes of randomness per S-box evaluation.

Table 4. Areas for first- and second-order AES implementations with $m = 1$ in 2-NAND Gate Equivalents (GE)

Evaluation	$d = 2$	$d = 3$	Preprocessing	$d = 2$	$d = 3$
S-box	18 810	28 234	Quintuples	29 147	53 212
* Beaver x^5 (x3)	3 914	5 875	* Generation	15 092	32 241
* Beaver x^4y^2	4 944	7 427	* Sacrificing	14 055	20 971
* Beaver Affine	1 563	2 344	Triples (x3)	19 106	34 954
State array	4 962	7 466	* Generation	9 804	21 112
* MixColumns	1 056	1 584	* Sacrificing	9 302	13 842
Key array	3 225	4 835	Affine tuples	7 603	14 657
Others	1 296	1 839	* Generation	4 821	10 444
			* Sacrificing	2 782	4 213
Total	28 293	42 374	Total	94 068	172 731
TOTAL				122 361	215 105

Table 5. The number of randomness in bytes for the initial sharing, shared multiplication and the sacrifice required for AES S-box

	Initial sharing	Shared mult.	Total
Exp. triple	d	$1 + 3m$	$2(d + (1 + 3m)\frac{d(d-1)}{2})$
Quintuple	$2d$	$1 + 5m$	$2(2d + (1 + 5m)\frac{d(d-1)}{2})$
Affine tuple	d	$2m$	$2(d + 2m\frac{d(d-1)}{2})$
Total			$12d + 2(4 + 16m)\frac{d(d-1)}{2}$

5.2 Software Implementation

CAPA is a suitable technique for software implementations if we map different tiles to different processors/cores. We do, however, need to place some constraints on the underlying hardware architecture; namely each processor should have an independent memory bank. Otherwise, a single affected tile (processor) could compromise the security of the whole system by for example dumping the entire memory contents (including all shares for sensitive variables).

This model therefore does not perfectly fit commercial off-the-shelf multi-core architectures, but we think isolated memory regions is a reasonable assumption for future micro-processors. While we do not have access to such architecture, as a proof of concept we emulate the proposed multi-processor architecture by time-sharing a 32-bit single-core ARM Cortex-M4 processor. This proof-of-concept does not provide resistance against attacks such as the memory dump example above.

Case Study: AES S-box. Even though it is possible to implement the AES S-box using GF(2^8) operations in SW also, we base our bitsliced software implementation on the principles of gate-level masking and we use the depth-16 AES S-box circuit by Boyar et al. [11] in order to provide competitive throughput. Our high-level implementation processes 32 blocks simultaneously which is compatible with the word size of our processor and can naturally be reduced. As the circuit boils down to a series of XOR and AND operations over pairs of value and tag shares, we redefine these elementary operations in the same way as previous works [3, Sect. 4]. We note that this technique is independent from the concrete design, and one could apply the same principles to different ciphers.

We create a prototype implementation in C99. This is an unoptimized implementation meant for functionality and security testing. We compile with `gcc-arm 4.8.4`. The 32 parallel SubBytes operations are performed in 2.52 million cycles (15ms) at 168MHz with $m = 8$ MAC tags and $d = 3$ shares. The implementation holds 41 intermediate variables in the stack (but this can be optimized); each takes $d \cdot w$ bytes for value shares and $m \cdot d \cdot w$ bytes for tag shares ($w = 4$ is number of bytes per word).

Experimental Validation of DPA Security. We use an STM32F407 32-bit ARM Cortex-M4 processor running the C99 implementation. We take EM measurements with an electromagnetic probe on top of a decoupling capacitor. This platform is very low noise: a DPA attack on the unprotected byte-oriented AES implementation succeeds with only 15 traces. Each trace is slightly above 500 000 time samples long and covers the entire execution of SubBytes. An exemplary trace is depicted at the top of Fig. 5.

Following the same procedure as in Sect. 5.1, we first perform a non-specific leakage detection test with the masking PRNG turned off. The results of the first-, second- and third-order leakage tests are shown on the left side of Fig. 5. Severe leakage is detected, which confirms that the setup is sound. When we plug in the PRNG, no leakage is detected with up to 200 000 traces (the statistic does not surpass the threshold $C = \pm 4.5$). This serves to confirm that the implementation effectively masks all intermediates, and that first- nor second-order DPA is not possible on this implementation. SPA features within an electromagnetic trace are better visible in the cross-correlation matrix shown in Fig. 6.

Experimental Validation of DFA Security. For the purposes of validating our theoretical security claims on CAPA's protection against fault attacks, we scale down our software AES SubBytes implementation, reducing the MAC key size to $m = 2$ and scaling down words to bits ($k = 1$). Note that this parameter choice lowers the detection probability; the point of using these toy parameters is only to verify more comfortably that the detection probability works as expected. It is easier to verify that the detection probability is $1-2^{-2}$ rather than $1-2^{-40}$. This concrete parameter choice is naturally not to be used in a practical deployment.

When barring the all zeroes key, we expect the attacker to succeed with probability at most $\frac{1}{2^{mk}-1} = \frac{1}{2^2-1} = 33\%$. The instrumented implementation conditionally inserts faults in value and/or tag shares. We repeat the SubBytes execution 1000 times, each iteration with a fresh MAC key. Faults are inserted in a random location during the execution of the S-box.

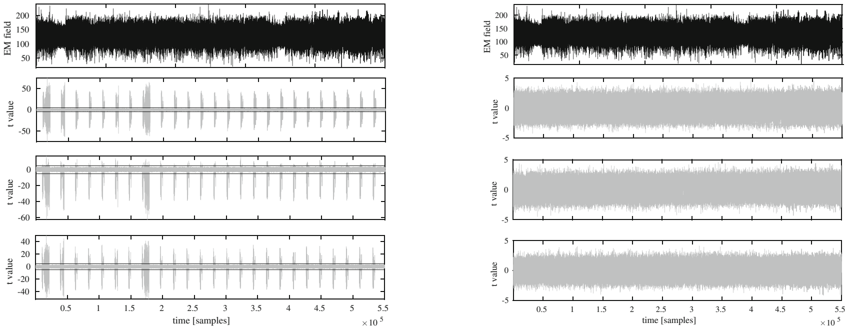

Fig. 5. Non-specific leakage detection on second-order SubBytes. Left column: masks off. Right column: masks on (200K traces). Rows (top to bottom): one exemplary EM trace, first-order t-test; second-order t-test; third-order t-test

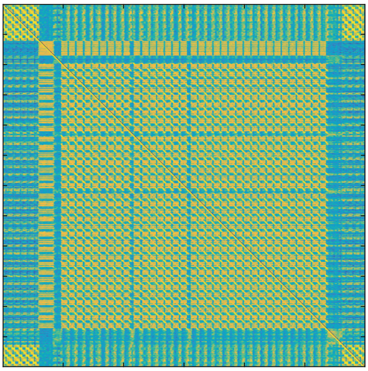

Fig. 6. Cross-correlation for second-order SubBytes. One can identify the 34 AND gates in the SubBytes circuit of Boyar et al. [11].

We verify that single faults on only values or only tags are detected unconditionally when we bar the all-0s key. When a single-bit offset (fault) is inserted in a single tile in both the value and tag share, it is indeed detected in approximately 66% of the iterations. Inserting a single-bit offset in value share and a random-bit offset in tag share is a worse attack strategy and is detected in around 83% of the experiments. The same results hold when faults are inserted in up to $d - 1$ tiles. When the value and tag shares in all d tiles are modified and fixed to a known value, the fault escapes detection with probability one, as expected.

6 Conclusion

In this paper, we introduced the first adversary model that jointly considers side-channels and faults in a unified and formal way. The *tile-probe-and-fault* security model extends the more traditional wire-probe model and accounts for a more realistic and comprehensive adversarial behavior. Within this model, we developed the methodology CAPA: a new combined countermeasure against physical attacks. CAPA provides security against higher-order DPA, multiple-shot DFA and combined attacks. CAPA scales to arbitrary security orders and borrows concepts from SPDZ, an MPC protocol. We showed the feasibility of implementing CAPA in embedded hardware and software by providing prototype implementations of established block ciphers. We hope CAPA provides an interesting addition to the embedded designer's toolbox, and stimulates further research on combined countermeasures grounded on more formal principles.

Acknowledgements. This work was supported in part by the Research Council KU Leuven: C16/15/058 and OT/13/071, by the NIST Research Grant 60NANB15D346 and the EU H2020 project FENTEC. Oscar Reparaz and Begül Bilgin are postdoctoral fellows of the Fund for Scientific Research - Flanders (FWO) and Lauren De Meyer is funded by a PhD fellowship of the FWO. The work of Nigel Smart has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070, and by EPSRC via grants EP/M012824 and EP/N021940/1.

References

1. Advanced Encryption Standard (AES): National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, November 2001
2. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and active combined attacks: combining fault attacks and side channel analysis. In: Breveglieri, L., Gueron, S., Koren, I., Naccache, D., Seifert, J. (eds.) FDTC 2007, pp. 92–102. IEEE Computer Society (2007)
3. Balasch, J., Gierlichs, B., Reparaz, O., Verbauwhede, I.: DPA, bitslicing and masking at 1 GHz. In: Güneysu and Handschuh [31], pp. 599–619
4. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F.-X., Strub, P.-Y.: Parallel implementations of masking schemes and the bounded moment leakage model. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 535–566. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_19
5. Battistello, A., Giraud, C.: Fault analysis of infective AES computations. In: Fischer, W., Schmidt, J., (eds.) FDTC 2013, pp. 101–107. IEEE Computer Society (2013)
6. Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 97–109. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-44750-4_8
7. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson [53], pp. 169–188
8. Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., Piuri, V.: Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. IEEE Trans. Comput. **52**(4), 492–505 (2003)

9. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_18
10. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of eliminating errors in cryptographic computations. J. Cryptol. **14**(2), 101–119 (2001)
11. Boyar, J., Matthews, P., Peralta, R.: Logic minimization techniques with applications to cryptology. J. Cryptol. **26**(2), 280–312 (2013)
12. Bringer, J., Carlet, C., Chabanne, H., Guilley, S., Maghrebi, H.: Orthogonal direct sum masking- a smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks. In: Naccache, D., Sauveron, D. (eds.) WISTP 2014. LNCS, vol. 8501, pp. 40–56. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43826-8_4
13. Bringer, J., Chabanne, H., Le, T.: Protecting AES against side-channel analysis using wire-tap codes. J. Cryptogr. Eng. **2**(2), 129–141 (2012)
14. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — a family of small and efficient hardware-oriented block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_20
15. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener [64], pp. 398–412
16. Cnudde, T.D., Nikova, S.: More efficient private circuits II through threshold implementations. In: FDTC 2016, pp. 114–124. IEEE Computer Society (2016)
17. Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Rohatgi, P.: Test Vector Leakage Assessment (TVLA) methodology in practice. In: International Cryptographic Module Conference (2013)
18. Coron, J.-S.: Higher order masking of look-up tables. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 441–458. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_25
19. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer, Heidelberg (2002). https://doi.org/10.1007/978-3-662-04722-4
20. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini and Canetti [60], pp. 643–662
21. Duc, A., Faust, S., Standaert, F.-X.: Making masking security proofs concrete. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 401–429. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_16
22. Fischer, W., Homma, N. (eds.): CHES 2017. LNCS, vol. 10529. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4
23. Gammel, B.M., Mangard, S.: On the duality of probing and fault attacks. J. Electron. Test. **26**(4), 483–493 (2010)
24. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44709-1_21
25. Gierlichs, B., Schmidt, J.-M., Tunstall, M.: Infective computation and dummy rounds: fault protection for block ciphers without check-before-output. In: Hevia, A., Neven, G. (eds.) LATINCRYPT 2012. LNCS, vol. 7533, pp. 305–321. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33481-8_17

26. Goubin, L., Patarin, J.: DES and differential power analysis the "Duplication" method. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_15
27. Groß, H., Mangard, S.: Reconciling d+1 masking in hardware and software. In: Fischer and Homma [22], pp. 115–136
28. Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: compact masked hardware implementations with arbitrary protection order. IACR Cryptology ePrint Archive, 2016:486 (2016)
29. Gross, H., Mangard, S., Korak, T.: An efficient side-channel protected AES implementation with arbitrary protection order. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 95–112. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_6
30. Grosso, V., Prouff, E., Standaert, F.-X.: Efficient masked S-boxes processing – a step forward –. In: Pointcheval, D., Vergnaud, D. (eds.) AFRICACRYPT 2014. LNCS, vol. 8469, pp. 251–266. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06734-6_16
31. Güneysu, T., Handschuh, H. (eds.): CHES 2015. LNCS, vol. 9293. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4
32. Guo, X., Mukhopadhyay, D., Jin, C., Karri, R.: Security analysis of concurrent error detection against differential fault analysis. J. Cryptogr. Eng. **5**(3), 153–169 (2015)
33. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.: Private circuits II: keeping secrets in tamperable circuits. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 308–327. Springer, Heidelberg (2006). https://doi.org/10.1007/11761679_19
34. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
35. Joshi, N., Wu, K., Karri, R.: Concurrent error detection schemes for involution ciphers. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 400–412. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_29
36. Joye, M., Manet, P., Rigaud, J.: Strengthening hardware AES implementations against fault attacks. IET Inf. Secur. **1**(3), 106–110 (2007)
37. Karpovsky, M., Kulikowski, K.J., Taubin, A.: Differential fault analysis attack resistant architectures for the advanced encryption standard. In: Quisquater, J.J., Paradinas, P., Deswarte, Y., El Kalam, A.A. (eds.) Smart Card Research and Advanced Applications VI. IFIP, vol. 153, pp. 177–192. Springer, Boston (2004). https://doi.org/10.1007/1-4020-8147-2_12
38. Karri, R., Kuznetsov, G., Goessel, M.: Parity-based concurrent error detection of substitution-permutation network block ciphers. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 113–124. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45238-6_10
39. Karri, R., Wu, K., Mishra, P., Kim, Y.: Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. IEEE Trans. CAD Integr. Circ. Syst. **21**(12), 1509–1517 (2002)
40. Keller, M., Orsini, E., Scholl, P.: MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 830–842. ACM Press, October 2016

41. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
42. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener [64], pp. 388–397
43. Lomné, V., Roche, T., Thillard, A.: On the need of randomness in fault attack countermeasures - application to AES. In: Bertoni, G., Gierlichs, B. (eds.) FDTC 2012, pp. 85–94. IEEE Computer Society (2012)
44. Malkin, T.G., Standaert, F.-X., Yung, M.: A comparative cost/security analysis of fault attack countermeasures. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 159–172. Springer, Heidelberg (2006). https://doi.org/10.1007/11889700_15
45. Medwed, M., Standaert, F.-X., Großschädl, J., Regazzoni, F.: Fresh re-keying: security against side-channel and fault attacks for low-cost devices. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 279–296. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12678-9_17
46. Mitra, S., McCluskey, E.J.: Which concurrent error detection scheme to choose? In: Proceedings IEEE International Test Conference 2000, Atlantic City, NJ, USA, October 2000, pp. 985–994. IEEE Computer Society (2000)
47. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: a very compact and a threshold implementation of AES. In: Paterson [53], pp. 69–88
48. Mukhopadhyay, D.: An improved fault based attack of the advanced encryption standard. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 421–434. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02384-2_26
49. NANGATE: The NanGate 45nm Open Cell Library. http://www.nangate.com
50. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini and Canetti [60], pp. 681–700
51. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 529–545. Springer, Heidelberg (2006). https://doi.org/10.1007/11935308_38
52. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of non-linear functions in the presence of glitches. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 218–234. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00730-9_14
53. Paterson, K.G. (ed.): EUROCRYPT 2011. LNCS, vol. 6632. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4
54. Patranabis, S., Chakraborty, A., Nguyen, P.H., Mukhopadhyay, D.: A biased fault attack on the time redundancy countermeasure for AES. In: Mangard, S., Poschmann, A.Y. (eds.) COSADE 2014. LNCS, vol. 9064, pp. 189–203. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21476-4_13
55. Prouff, E., Rivain, M.: Masking against side-channel attacks: a formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_9
56. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_37

57. Reparaz, O., Gierlichs, B., Verbauwhede, I.: Fast leakage assessment. In: Fischer and Homma [22], pp. 387–399
58. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_28
59. Roche, T., Prouff, E.: Higher-order glitch free implementation of the AES using secure multi-party computation protocols - extended version. J. Cryptogr. Eng. **2**(2), 111–127 (2012)
60. Safavi-Naini, R., Canetti, R. (eds.): CRYPTO 2012. LNCS, vol. 7417. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5
61. Schneider, T., Moradi, A.: Leakage assessment methodology - a clear roadmap for side-channel evaluations. In: Güneysu and Handschuh [31], pp. 495–513
62. Schneider, T., Moradi, A., Güneysu, T.: ParTI – towards combined hardware countermeasures against side-channel and fault-injection attacks. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 302–332. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_11
63. Seker, O., Eisenbarth, T., Steinwandt, R.: Extending glitch-free multiparty protocols to resist fault injection attacks. IACR Cryptology ePrint Archive, 2017:269 (2017)
64. Wiener, M. (ed.): CRYPTO 1999. LNCS, vol. 1666. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1
65. Yen, S., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Trans. Comput. **49**(9), 967–970 (2000)