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, Belgium
firstname.lastname@esat.kuleuven.be
[2] Square, Inc., USA
[3] NXP Semiconductors, Belgium
[4] University of Bristol, UK

Abstract. In this paper we introduce CAPA: a combined countermeasure against physical attacks. Our countermeasure provides security against higher-order SCA, multiple-shot DFA and combined attacks, scales to arbitrary protection order and is suitable for implementation in embedded hardware and software. The methodology is based on an attack model which we call *tile-probe-and-fault*, which is an extension (in both attack surface and capabilities) of prior work such as the wire-probe model. The tile-probe-and-fault leads one to naturally look (by analogy) at actively secure multi-party computation protocols such as SPDZ. We detail several proof-of-concept designs using the CAPA methodology: a hardware implementation of the KATAN and AES block ciphers, as well as a software bitsliced AES S-box implementation. We program a second-order secure version of the KATAN design into a Spartan-6 FPGA and perform a side-channel evaluation. No leakage is detected with up to 18 million traces. We also deploy a second-order secure software AES S-box implementation into an ARM Cortex-M4. Neither first- nor second-order leakage is detected with up to 200 000 traces. Both our implementations can detect faults within a strong adversarial model with arbitrarily high probability.

Keywords: MPC, masking, SCA, DFA, countermeasure, threshold implementation, AES, KATAN, leakage, physical attacks, side-channel, SCA

1 Introduction

Side-channel analysis attacks (SCA) [44] 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 [45] or the electromagnetic radiation [26]) 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, 29] is an established solution that stands out as a provably secure yet practically useful countermeasure.

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 suffices to recover the whole secret key [10, 51]. 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. It should be mentioned that a duality relation between probing and fault attacks was pointed out in [25]. 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 [37]. 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 [23]. 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 device itself by using, for example, active or passive shields and protection of the algorithm. 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 [57]. 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.

A second method is to use an error correcting or detecting code [49, 42, 8, 41, 40, 38, 39, 13, 12, 35]. 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 [47].

Another approach is that of infective computation [27, 46], 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 [58], 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 [37], where an attacker is allowed to probe, receive and combine the noiseless information about t wires within a time period [24]. Finally, it has been shown in [4] that a (semi-)parallel hardware 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, 32, 37, 54, 59–61] 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 aparent 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.

A natural countermeasure against combined attacks is found in leakage resilient schemes [48]. 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, 36], the authors use redundancy on top of a circuit that already resists SCA (private circuits I [37]) to add protection against FA. In ParTI [64], 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 [65], 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 only up to $t < n/3$ parties can be corrupt. The modern approach, adopted by protocols such as BODZ, SPDZ, Tiny-OT, MASCOT etc. [7, 23, 43, 53], 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 the scheme with implementations and experiments.

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 Figure 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.

The tile architecture satisfies the independent leakage assumption [24] 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 inherits theoretical aspects of the MPC protocol SPDZ [23]. Like SPDZ, CAPA computes 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. The methodology is scalable and suitable for implementation in hardware and software.

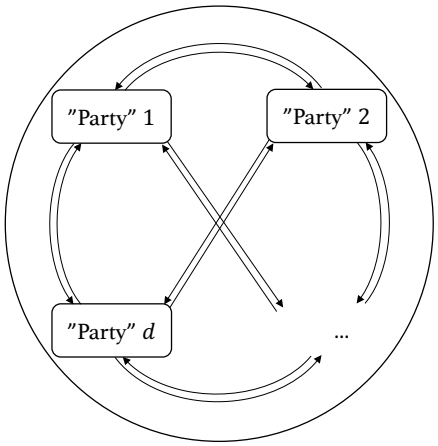

Fig. 1. Partition of the integrated circuit area into tiles, implementing MPC "parties".

Experimental Results. We provide examples of CAPA designs in hardware of the KATAN and AES block ciphers and a software bitsliced implementation of the AES S-box. We confirm our theoretical claims with a side-channel evaluation of hardware and software implementations of various cryptographic primitives using the CAPA methodology. We program 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 18 million traces. Furthermore, we deploy a second order secure implementation of the AES S-box on an ARM Cortex-M4 and take electromagnetic measurements. Neither first nor second order leakage is detected with up to 200 000 traces. Finally, using toy parameters, we verify our claimed fault detection probability for the AES S-box software implementation.

2 The Tile-Probe-and-Fault Model

In this section we outline the underlying security model which we assume in this work.

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 Figure 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 variable in the system. 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 Figure 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 strong 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 d_p-probing adversary which gives access to only d_p intermediate *values* within a certain amount of time [37] and ϵ-probing adversary where the information about d_p intermediate *values* is gained with certain probability. 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 close proximity on the chip.

Faults. We also consider two types of fault models. Firstly, a d_f-faulting adversary which can induce known-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 known) offset *or* setting the intermediate values to a known 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 inserts a random-value fault in *all* variables that *each* party holds. 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 known 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) 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 infinitely 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 notion is applicable in our hardware situation because of the experimental difficulty of setting up probes and lasers that target specific parts on the chip. 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 $\alpha[j] \in F_q$ 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 auxilary 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 $(\boldsymbol{z}, \boldsymbol{\tau^z})$ of $(\boldsymbol{x}, \boldsymbol{\tau^x})$ and $(\boldsymbol{y}, \boldsymbol{\tau^y})$, 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 $(\boldsymbol{z}, \boldsymbol{\tau^z})$ of $(\boldsymbol{x}, \boldsymbol{\tau^x})$ 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 $(\boldsymbol{x}, \boldsymbol{\tau^x})$ and $(\boldsymbol{y}, \boldsymbol{\tau^y})$ 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 $(\boldsymbol{\varepsilon}, \boldsymbol{\tau^\varepsilon})$ (resp. $(\boldsymbol{\eta}, \boldsymbol{\tau^\eta})$), i.e. the value ε (resp. η) is unmasked but its tag $\boldsymbol{\tau^\varepsilon}$ ($\boldsymbol{\tau^\eta}$) 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 $\boldsymbol{\tau^\varepsilon}$ ($\boldsymbol{\tau^\eta}$) 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 $(\boldsymbol{z}, \boldsymbol{\tau^z})$ 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.

Squaring. Squaring is a linear operation over GF(2). 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 $(\boldsymbol{x}, \boldsymbol{\tau^x})$ into $(\boldsymbol{z}, \boldsymbol{\tau^z})$, we therefore require an auxiliary *tuple* $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle)$ such that $b = a^2$. The procedure to obtain $(\boldsymbol{z}, \boldsymbol{\tau^z})$ 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 $(\boldsymbol{z}, \boldsymbol{\tau^z})$ satisfying $z = A(x)$, 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* and provide in Appendix A.1 more details for verification.

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.

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 $\boldsymbol{\tau^a}$, $\boldsymbol{\tau^b}$, $\boldsymbol{\tau^c}$. 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, 55]. In the last decade, there has been significant improvement on passively secure shared multipliers that can be used in both hardware and software [9, 30, 32, 54, 59]. 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.

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 ensures that the generated 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 $r_1 \in F_q$
- 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 $r_2 \in F_q$ 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.

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

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 security against the following types of attacks as well as a combined attack of the two

1. Side-Channel Analysis (i.e. a $d-1$-probing adversary).
2. Fault Attacks (i.e. an adversary introducing 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. Since this requires knowledge of the MAC key $\alpha \in GF(2^{km})$, the adversary cannot forge a valid tag for a faulty value. His best option is to guess the MAC key. 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 wires $\in \mathcal{P}_1$ 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 $\mathscr{A}_1$ 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 CAPA does not allow injected faults to propagate through a calculation into a difference that depends on sensitive information.

What Does Our MAC Security Mean? We stress that CAPA provides significantly higher security than existing approaches. 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 $1-2^{-km}$.

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 tags are expected to leak very little information.

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; otherwise the attack will be detected with probability one. 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.

Selective Failure Attack. We point out a specific attack that targets any countermeasure against a probing and faulting adversary. In a selective failure attack (also known as safe-error attack in the embedded security community [67]), 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.* doesn't 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) 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 Attacks. In our description we are assuming that during the broadcast phase there are no "races" between tiles: 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 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 §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. 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			
	·	+	$L(.)$	·	+	·	+	·	+
Add.					d		dm		
Add. with C					1	dm	dm		
Multip.				d		dm			
Multip. with C	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 [19, 37, 60]. 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 §3.2. The same comments apply to FPGAs with plenty of BlockRAM.

4.5 Optimization of Preprocessing

As hinted in §4.4, it may be beneficial to store the output of the preprocessing stage §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 Implementation

CAPA is tailored to be used in embedded systems. In this section, we first emphasize concepts that need special attention in any implementation. Then, we elaborate on specific hardware and software implementations of KATAN-32 [14] and AES [1] which cover operations in different fields, possibility of bitsliced implementations, hardware and software specific timing and memory optimizations, and performance results.

Having d Copies of Unmasked Values/calculations. The most natural example of an unmasked calculation is that of the control unit. Each tile should have its own control logic to avoid single points of attack (for example in the round counter).

It is also important that each tile keeps its local copy of all public values. In particular, the differences $\varepsilon \cdot \alpha + \tau^{\varepsilon}$ during the MAC-tag checking phase and Δ and τ^{Δ} during the relation verification of auxiliary data are all unmasked by each tile. The same holds for ε and η during Beaver operations, as well as any computation on these values such as $\varepsilon \cdot \eta, L_1(\varepsilon)$, etc. Finally, each tile keeps its copy of the abort status.

Synchronization. In order to avoid leaking information on the sensitive data, all intermediate values to be (partially) unmasked, such as $\varepsilon_i = x_i + a_i$, need to be synchronized using memory elements before being released to other tiles.

Passively Secure Shared Multiplier. During the auxiliary data generation step, CAPA uses a passively secure shared multiplication with higher-order non-completeness. Literature provides us with a broad spectrum of multipliers to choose from [54, 32, 59, 30, 9]. However, in order to minimize the randomness requirement of the offline phase, our implementation uses the DOM-dep multiplier from [32].

5.1 Hardware Implementation

Timing optimizations. We optimize the calculations in the online phase such that they introduce minimum penalty on the number of clock cycles. We achieve this by providing a spatial separation between tiles, all of which perform their part of the same operations simultaneously using their own dedicated combinational circuit. This naturally causes a significant area increase compared to the use of the same combinational circuit by each tile consecutively.

In addition, we start the MAC-tag checking and computation of a Beaver operation in the same cycle. This is possible since the Beaver computation can

be computed without the knowledge of the authenticity of the input. Note that, the MAC-tag checking requires two cycles due to the synchronization of $\varepsilon \cdot \alpha + \tau^{\varepsilon}$ through a register. That implies that the MAC-tag check finishes one cycle later than its corresponding Beaver computation. In order to avoid doubling the latency of each Beaver calculation, we perform the second cycle of the tag check in parallel with the first cycle of the next operation (see Figure 2 for an example of the AES S-box).

Auxiliary data generation. Instead of using a single module to generate the auxiliary data prior to encryption and storing them in for example block RAM, we use several auxiliary data generation modules for convenience. The timing of these modules is arranged such that they provide the required amount of data in every clock cycle of encryption.

Library. For synthesis, we use Synopsis Design Compiler Version I-2013.12 using the NanGate 45nm Open Cell library [52] 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).

KATAN-32. This shift register based block cipher, which gets 80-bits of key and a 32-bit plaintext input, is designed for efficient hardware implementations. It performs 254 cycles consisting of four two-input AND and XOR operations. Hence, its natural shared data representation is in the field $F_q = \mathrm{GF}(2)$. We implement this cipher using different MAC-tag key sizes $\alpha \in F_2^m$.

Timing. 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 due to synchronization of ε_i and η_i in registers. It is implemented in a pipelined fashion which increases latency of the whole computation only by one clock cycle.

Area. Table 2 summarizes the area of our KATAN implementations. The first column shows the results of a shared implementation with $d = 3$ shares without tags ($m = 0$). The second two columns hold the results for 1 or 8 MAC keys $\alpha[j]$. 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 column, we extrapolate the area results, including the combinational logic, for any m. Note that the state array includes the non-linear Beaver multiplication.

Randomness. Each Beaver multiplication in GF(2) (AND operations) requires one triple, and each triple needs $2d$ random bits for generating $\boldsymbol{a}$ and $\boldsymbol{b}$. A 3-share masked DOM multiplication over GF(2) requires 3 bits of randomness. In general, a d-share DOM multiplication requires $\binom{d}{2}$ units of randomness. The construction of one triple requires $1 + 3m$ masked multiplications: one to obtain the multiplication $\boldsymbol{c}$ of $\boldsymbol{a}$ and $\boldsymbol{b}$; and $3m$ to obtain the m tags $\boldsymbol{\tau^a}$,$\boldsymbol{\tau^b}$ and $\boldsymbol{\tau^c}$.

Table 2. 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	$\approx 3\,560 + 3\,580m$
* State Array	1 363	2 812	12 890	$\approx 1\,363 + 1\,450m$
* Key Schedule	2 197	4 327	19 478	$\approx 2\,197 + 2\,130m$
- Preprocessing	638	1 468	7 124	$\approx 638 + 830m$
* Two triple generation	428	952	4 694	$\approx 428 + 524m$
* Relation verification	210	516	2 430	$\approx 210 + 306m$
Total	5 971	12 083	55 254	$\approx 5\,971 + 6\,112m$

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 (three Beaver multiplications) is $3 \cdot 2 \cdot (2d + (1+3m)\frac{d(d-1)}{2}))$.

AES. We choose to work in GF(2^8) with $m = 1$ for AES, i.e. the MAC-key, data and the MAC-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 mainly 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. The combination of the two operations can be expressed by the following polynomial in GF(2^8) [22]:

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

We distinguish two methodologies for the S-box implementation. We either evaluate this polynomial alltogether or we do the inversion, followed by the affine transformation separately. Initial estimations reveal the former method is more expensive than the latter. Following further cost estimations on different multiplication chains [20, 60], we decide to use the one from [33] which is also depicted in Figure 5 in Appendix.

The Affine Transform. Since $A(x)$ is linear over GF(2), we can use the Beaver operation described in §3.1 to evaluate the polynomial at once, using auxiliary affine tuples $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle)$ such that $b = A(a)$.

Multiplication Chain. A typical implementation of the proposed multiplication chain uses three types of operations: x^2, x^5 and $x \cdot y$. The squaring and multiplication can be done as described in §3.1. 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 can calculate exponentiation to the power five using the generalized multiplication of §3.1 with linear input transformations $L_1(x) = x^4$ and $L_2(x) = x$. This way, we obtain the inversion output in 5 cycles, using 2 squaring tuples, 3 exponentiation triples and 1 multiplication triple. We optimize the chain further by merging the squares and the

multiplication. That is, let $y = x^{125}$, then $x^{254} = x^4 \cdot y^2$. We can use quintuples $(\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle, \langle \boldsymbol{d} \rangle, \langle \boldsymbol{e} \rangle)$ from §3.1 such that $c = a^4$, $d = b^2$ and $e = c \cdot d = a^4 \cdot b^2$ to perform this operation in one cycle. As a result, an inversion in 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 Figure 2). This approach does not only decrease the number of cycles but also the amount of required randomness.

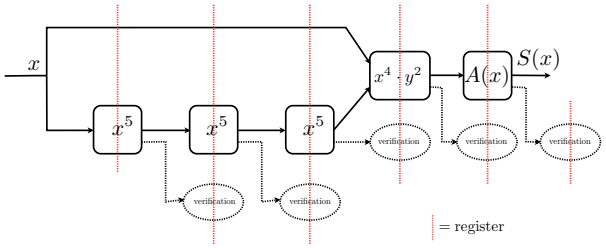

Fig. 2. AES S-box pipeline

Timing. We use a serialized AES implementation, based on that in [31]. The S-box is implemented as a five stage pipeline. One round of the cipher requires 21 clock cycles and the complete encryption of one plaintext takes 226 clock cycles. Given that the unprotected serialised implementation of [50] also requires 226 cycles, the timing performance is very good.

Area. Table 3 presents the area for the different blocks that make up our AES implementation. The results correspond to a $d = 3$ share and $m = 1$ byte MAC 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.

Randomness. Table 4 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.

5.2 Software

As a proof-of-concept we implement the AES S-box based on CAPA with computations over GF(2).

Table 3. Areas for AES implementation with $d = 3$ and $m = 1$ in (GE)

Evaluation		Preprocessing	
	GE		GE
S-box	28 234	Quintuples	53 212
* Beaver x^5 (x3)	5 875	* Generation	32 241
* Beaver x^4y^2	7 427	* Sacrificing	20 971
* Beaver Affine	2 344	Triples (x3)	34 954
State array	7 466	* Generation	21 112
* MixColumns	1 584	* Sacrificing	13 842
Key array	4 835	Affine tuples	14 657
Others	1 839	* Generation	10 444
		* Sacrificing	4 213
Total	42 374	Total	172 731
TOTAL			215 105

Table 4. 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			$12d + 2(4 + 16m)\frac{d(d-1)}{2}$

Model fit. CAPA is a suitable technique for software implementations if we map different tiles to different processors. We do 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. 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.

AES S-box implementation. 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]. Our high-level implementation processes 32 blocks simultaneously. As the AES 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, §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).

6 Evaluation

6.1 DPA: hardware

We program one of our designs onto a Xilinx Spartan-6 FPGA. Our platform is a Sakura-G board specifically designed for side-channel evaluation. To minimize platform noise, our design is splitted into two different FPGAs: a control FPGA handles I/O with the host computer and supplies data in masked representation to the crypto FPGA. The crypto FPGA implements both the preprocessing and evaluation as in §5.1.

As a proof of concept, we implement KATAN with $d = 3$ shares and $m = 2$ MAC keys. The parameter $m = 2$ is insufficient in practice, but serves for our proof-of-concept purpose. The design is clocked at $3,072$ MHz, we sample power traces of 2500 time samples each at 500 MS/s. Each trace comprises the first 14 rounds of KATAN. Exemplary traces are shown in Figure 3, top.

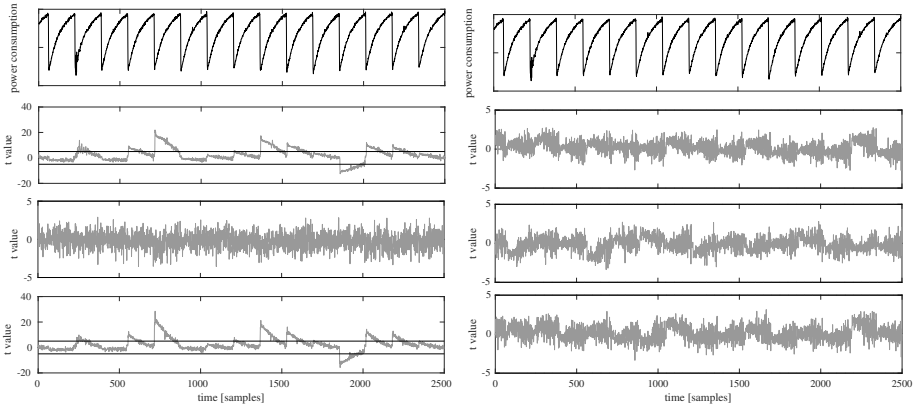

Fig. 3. Non-specific leakage detection on the first 14 rounds of KATAN hardware implementation. Left column: masks off. Right column: masks on. Rows (top to bottom): one exemplary power trace, first-order t-test; second-order t-test; third-order t-test.

We perform a non-specific leakage detection test [17, 18, 21, 28] on the input plaintext following the standard methodology [63]: first, we test the design without masks to verify that our setup is indeed sound and able to detect leakage. Then we switch on masks and corroborate that the design does not leak. In Figure 3, left column, we plot the results of a univariate t-test at first, second and third

order when the masks are off. As expected, the design presents severe leakage (the t-statistic crosses the threshold $C = \pm 5$. In the right column, we repeat the procedure with masks on. No univariate leakage is detected with up to 18 million traces, since the t-statistic does not surpass the threshold C. (Eventually we expect to detect leakage in the third order since our implementation handles 3 shares. However, due to platform noise, third-order leakage is not yet visible.)

6.2 DPA: software

Our processor is an STM32F407 32-bit ARM Cortex-M4 running the implementation from §5.2. 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 Figure 4.

We follow the same procedure as in §6.1. First, we 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 Figure 4. Severe leakage is detected, which confirms that the setup is sound. When we plug in the mask PRNG, no leakage is detected with up to 200 000 traces (the statistic does not surpass the threshold $C = \pm 5$). This serves to confirm that the implementation effectively masks all intermediates, and that DPA is not possible on this implementation.

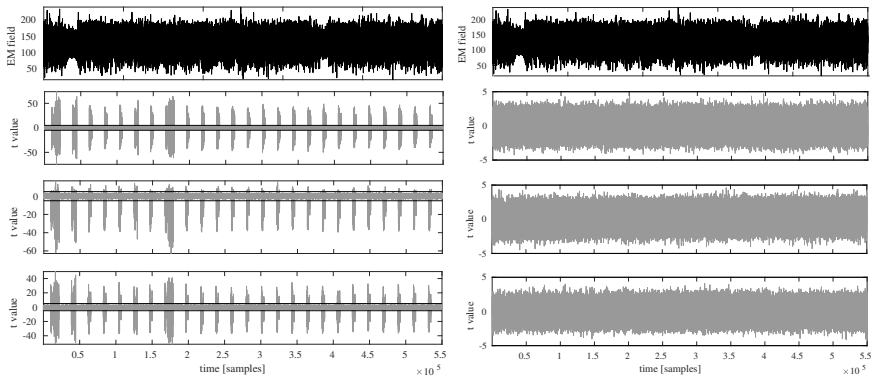

Fig. 4. Non-specific leakage detection on SubBytes. Left column: masks off. Right column: masks on. Rows (top to bottom): one exemplary EM trace, first-order t-test; second-order t-test; third-order t-test. SPA features within an electromagnetic trace are better visible in the cross-correlation matrix shown in Figure 6.

6.3 Fault analysis

Detection probability. For the purposes of validating our theoretical security claims, 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 execution of the S-box.

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 d tiles are modified and fixed to a known value, the fault escapes detection with probability one, as expected.

7 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 developped 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.

References

1. Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce, Nov. 2001.
2. F. Amiel, K. Villegas, B. Feix, and L. Marcel. Passive and active combined attacks: Combining fault attacks and side channel analysis. In L. Breveglieri, S. Gueron,

I. Koren, D. Naccache, and J. Seifert, editors, *FDTC 2007*, pages 92–102. IEEE Computer Society, 2007.
3. J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede. DPA, bitslicing and masking at 1 GHz. In Güneysu and Handschuh [34], pages 599–619.
4. G. Barthe, F. Dupressoir, S. Faust, B. Grégoire, F. Standaert, and P. Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In J. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 535–566, 2017.
5. A. Battistello and C. Giraud. Fault analysis of infective AES computations. In W. Fischer and J. Schmidt, editors, *FDTC 2013*, pages 101–107. IEEE Computer Society, 2013.
6. D. Beaver. Precomputing oblivious transfer. In D. Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, Heidelberg, Aug. 1995.
7. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In Paterson [56], pages 169–188.
8. G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Trans. Computers*, 52(4):492–505, 2003.
9. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-order threshold implementations. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 326–343. Springer, Heidelberg, Dec. 2014.
10. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14(2):101–119, 2001.
11. J. Boyar, P. Matthews, and R. Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, Apr. 2013.
12. J. Bringer, C. Carlet, H. Chabanne, S. Guilley, and H. Maghrebi. Orthogonal direct sum masking - A smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks. In D. Naccache and D. Sauveron, editors, *WISTP 2014. Proceedings*, volume 8501 of *LNCS*, pages 40–56. Springer, 2014.
13. J. Bringer, H. Chabanne, and T. Le. Protecting AES against side-channel analysis using wire-tap codes. *J. Cryptographic Engineering*, 2(2):129–141, 2012.
14. C. D. Cannière, O. Dunkelman, and M. Knežević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In C. Clavier and K. Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, Heidelberg, Sept. 2009.
15. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Wiener [66], pages 398–412.
16. T. D. Cnudde and S. Nikova. More efficient private circuits II through threshold implementations. In *FDTC 2016*, pages 114–124. IEEE Computer Society, 2016.
17. J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. Test Vector Leakage Assessment (TVLA) methodology in practice. International Cryptographic Module Conference, 2013.
18. J. Coron, D. Naccache, and P. C. Kocher. Statistics and secret leakage. *ACM Trans. Embedded Comput. Syst.*, 3(3):492–508, 2004.
19. J.-S. Coron. Higher order masking of look-up tables. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 441–458. Springer, Heidelberg, May 2014.
20. J.-S. Coron, A. Greuet, E. Prouff, and R. Zeitoun. Faster evaluation of SBoxes via common shares. In B. Gierlichs and A. Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 498–514. Springer, Heidelberg, Aug. 2016.

21. J.-S. Coron, P. C. Kocher, and D. Naccache. Statistics and secret leakage. In Y. Frankel, editor, *FC 2000*, volume 1962 of *LNCS*, pages 157–173. Springer, Heidelberg, Feb. 2001.
22. I. Damgård, M. Keller, E. Larraia, C. Miles, and N. P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In I. Visconti and R. D. Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 241–263. Springer, Heidelberg, Sept. 2012.
23. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini and Canetti [62], pages 643–662.
24. A. Duc, S. Faust, and F.-X. Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 401–429. Springer, Heidelberg, Apr. 2015.
25. B. M. Gammel and S. Mangard. On the duality of probing and fault attacks. *J. Electronic Testing*, 26(4):483–493, 2010.
26. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya. Koç, D. Naccache, and C. Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, Heidelberg, May 2001.
27. B. Gierlichs, J.-M. Schmidt, and M. Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In A. Hevia and G. Neven, editors, *LATINCRYPT 2012*, volume 7533 of *LNCS*, pages 305–321. Springer, Heidelberg, Oct. 2012.
28. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop, 2011.
29. L. Goubin and J. Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya. Koç and C. Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172. Springer, Heidelberg, Aug. 1999.
30. H. Groß and S. Mangard. Reconciling d+1masking in hardware and software. *IACR Cryptology ePrint Archive*, 2017:103, 2017.
31. H. Groß, S. Mangard, and T. Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *IACR Cryptology ePrint Archive*, 2016:486, 2016.
32. H. Groß, S. Mangard, and T. Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In H. Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *LNCS*, pages 95–112. Springer, 2017.
33. V. Grosso, E. Prouff, and F.-X. Standaert. Efficient masked S-boxes processing - A step forward -. In D. Pointcheval and D. Vergnaud, editors, *AFRICACRYPT 14*, volume 8469 of *LNCS*, pages 251–266. Springer, Heidelberg, May 2014.
34. T. Güneysu and H. Handschuh, editors. *CHES 2015*, volume 9293 of *LNCS*. Springer, Heidelberg, Sept. 2015.
35. X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri. Security analysis of concurrent error detection against differential fault analysis. *J. Cryptographic Engineering*, 5(3):153–169, 2015.
36. Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private circuits II: Keeping secrets in tamperable circuits. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 308–327. Springer, Heidelberg, May / June 2006.

37. Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, Aug. 2003.
38. N. Joshi, K. Wu, and R. Karri. Concurrent error detection schemes for involution ciphers. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 400–412. Springer, Heidelberg, Aug. 2004.
39. M. Joye, P. Manet, and J. Rigaud. Strengthening hardware AES implementations against fault attacks. *IET Information Security*, 1(3):106–110, 2007.
40. M. G. Karpovsky, K. J. Kulikowski, and A. Taubin. Differential fault analysis attack resistant architectures for the advanced encryption standard. In J. Quisquater, P. Paradinas, Y. Deswarte, and A. A. E. Kalam, editors, *CARDIS 2004, 22-27 August 2004, Toulouse, France*, volume 153 of *IFIP*, pages 177–192. Kluwer/Springer, 2004.
41. R. Karri, G. Kuznetsov, and M. Gössel. Parity-based concurrent error detection of substitution-permutation network block ciphers. In C. D. Walter, Çetin Kaya. Koç, and C. Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 113–124. Springer, Heidelberg, Sept. 2003.
42. R. Karri, K. Wu, P. Mishra, and Y. Kim. Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(12):1509–1517, 2002.
43. M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 16*, pages 830–842. ACM Press, Oct. 2016.
44. P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, Aug. 1996.
45. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In Wiener [66], pages 388–397.
46. V. Lomné, T. Roche, and A. Thillard. On the need of randomness in fault attack countermeasures - application to AES. In G. Bertoni and B. Gierlichs, editors, *FDTC 2012*, pages 85–94. IEEE Computer Society, 2012.
47. T. Malkin, F. Standaert, and M. Yung. A comparative cost/security analysis of fault attack countermeasures. In L. Breveglieri, I. Koren, D. Naccache, and J. Seifert, editors, *FDTC 2006*, volume 4236 of *LNCS*, pages 159–172. Springer, 2006.
48. M. Medwed, F.-X. Standaert, J. Großschädl, and F. Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In D. J. Bernstein and T. Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 279–296. Springer, Heidelberg, May 2010.
49. S. Mitra and E. J. McCluskey. Which concurrent error detection scheme to choose ? In *Proceedings IEEE International Test Conference 2000, Atlantic City, NJ, USA, October 2000*, pages 985–994. IEEE Computer Society, 2000.
50. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of AES. In Paterson [56], pages 69–88.
51. D. Mukhopadhyay. An improved fault based attack of the advanced encryption standard. In B. Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 421–434. Springer, Heidelberg, June 2009.
52. NANGATE. The NanGate 45nm Open Cell Library. Available at http://www.nangate.com.

53. J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In Safavi-Naini and Canetti [62], pages 681–700.
54. S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In P. Ning, S. Qing, and N. Li, editors, *ICICS 06*, volume 4307 of *LNCS*, pages 529–545. Springer, Heidelberg, Dec. 2006.
55. S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of non-linear functions in the presence of glitches. In P. J. Lee and J. H. Cheon, editors, *ICISC 08*, volume 5461 of *LNCS*, pages 218–234. Springer, Heidelberg, Dec. 2009.
56. K. G. Paterson, editor. *EUROCRYPT 2011*, volume 6632 of *LNCS*. Springer, Heidelberg, May 2011.
57. S. Patranabis, A. Chakraborty, P. H. Nguyen, and D. Mukhopadhyay. A biased fault attack on the time redundancy countermeasure for AES. In S. Mangard and A. Y. Poschmann, editors, *COSADE 2015. Revised Selected Papers*, volume 9064 of *LNCS*, pages 189–203. Springer, 2015.
58. E. Prouff and M. Rivain. Masking against side-channel attacks: A formal security proof. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.
59. O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. Consolidating masking schemes. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 764–783. Springer, Heidelberg, Aug. 2015.
60. M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In S. Mangard and F.-X. Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, Heidelberg, Aug. 2010.
61. T. Roche and E. Prouff. Higher-order glitch free implementation of the AES using secure multi-party computation protocols - extended version. *J. Cryptographic Engineering*, 2(2):111–127, 2012.
62. R. Safavi-Naini and R. Canetti, editors. *CRYPTO 2012*, volume 7417 of *LNCS*. Springer, Heidelberg, Aug. 2012.
63. T. Schneider and A. Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Güneysu and Handschuh [34], pages 495–513.
64. T. Schneider, A. Moradi, and T. Güneysu. ParTI – towards combined hardware countermeasures against side-channel and fault-injection attacks. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 302–332. Springer, Heidelberg, Aug. 2016.
65. O. Seker, T. Eisenbarth, and R. Steinwandt. Extending glitch-free multiparty protocols to resist fault injection attacks. *IACR Cryptology ePrint Archive*, 2017:269, 2017.
66. M. J. Wiener, editor. *CRYPTO'99*, volume 1666 of *LNCS*. Springer, Heidelberg, Aug. 1999.
67. S. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000.

A Supplementary materials

A.1 Proof

Proof. Beaver operation for a multiplication with linear input transformations.

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

A.2 Figures

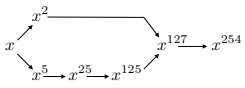

Fig. 5. Multiplication chain from [33]

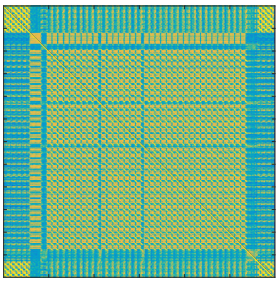

Fig. 6. Cross-correlation for SW implementation, $d = 3$, SubBytes. One can identify the 34 AND gates in the SubBytes circuit of Boyar et al. [11].