StaMAC: Fault Protection via Stable-MAC Tags

Siemen Dhooghe*, Artemii Ovchinnikov, and Dilara Toprakhisar

COSIC, KU Leuven, Leuven, Belgium
firstname.lastname@esat.kuleuven.be

Keywords: Encoding · Fault Attacks · Masking · Side-Channel Analysis

Abstract. Fault attacks pose a significant threat to cryptographic implementations, motivating the development of countermeasures, primarily based on a combination of redundancy and masking techniques. Redundancy, in these countermeasures, is often implemented via duplication or linear codes. However, their inherent structure remains susceptible to strategic fault injections bypassing error checks. To address this, the CAPA countermeasure from CRYPTO 2018 leveraged information-theoretic MAC tags for protection against fault and combined attacks. However, a recent attack has shown that CAPA can only protect against either side-channel analysis or fault attacks, but not both simultaneously, and with significant hardware costs. Its successor, M&M, improves efficiency but lacks protection against ineffective faults.

In this paper, we propose StaMAC, a framework aimed at securely incorporating MAC tags against both side-channel and fault adversaries in a non-combined scenario. We extend the security notions outlined in StaTI from TCHES 2024, and propose the notion of *MAC-stability*, ensuring fault propagation in masked and MACed circuits, necessitating only a single error check at the end of the computation. Additionally, we show that the stability notion from StaTI is arbitrarily composable (whereas it was previously thought to be only serially composable), making it the first arbitrary composable fault security notion which does not require intermediate error checks or correction. Then, we establish the improved protection of masking combined with MAC tags compared to linear encoding techniques by showing bounds on the advantage considering several fault adversaries: a *gate/register faulting adversary*, an *arbitrary register faulting adversary*, and a *random register faulting adversary*. Then, we show how to transform any probing secure circuit to protect against fault attacks using the proposed MAC-stable gadgets implementing field operations. Finally, we demonstrate StaMAC on an AES implementation, evaluating its security and hardware costs compared to the countermeasures using MAC tags.

1 Introduction

Cryptographic algorithms are designed to resist cryptanalytic attacks, however, their vulnerability to physical attacks exploiting the physical characteristics of

* Authors listed in alphabetical order.

their implementations remains. These attacks can be categorized into two main types: (i) passive observation of the device's behavior, involving power consumption [KJJ99], timing [Koc96], and electromagnetic emanations [GMO01]; and (ii) the intentional introduction of errors in computation through physical manipulation, such as clock/voltage glitching [AK97], exposure to electromagnetic waves [DDRT12], and laser injections [Hab65].

The first type is called Side-Channel Analysis (SCA), a passive attack technique that exploits the observable leakage stemming from the physical characteristics of the implementations. To counter SCA, masking [CJRR99, ISW03, RBN+15] is widely employed as a countermeasure. Masking, based on secret-sharing, splits the secret input into a number of shares such that the side-channel information leakage from all-but-one share remains independent of the secret input. Domain-Oriented Masking (DOM), as proposed by Groß *et al.* [GMK16], is such a technique where each share of the secret input is linked to a distinct share domain. DOM is widely recognized for its low implementation costs and glitch resistance.

Unlike the passive observation involved in SCA, the second type of physical attack, Fault Attacks (FA), actively disturb the computations using physical fault injection mechanisms. Since the landmark work of Boneh *et al.* [BDL97], introducing fault attacks on RSA, a plethora of techniques have emerged that exploit injected faults in cryptographic implementations. In response, diverse countermeasures have been developed with redundancy emerging as a prevalent strategy to mitigate such attacks. Redundancy, in its temporal, spatial, or informational forms, is employed as a countermeasure to detect injected faults within circuits. Such countermeasures act by either suppressing or infecting the output when a fault is detected, rendering it unusable for exploitation by adversaries. Redundancy takes various forms, including duplication, concurrent error detection, or error correction mechanisms like majority voting. Duplication is effective against faults, but its security is compromised when identical faults are injected. Even with error detection codes, an adversary knowing the underlying code can strategically inject faults to bypass the detection.

Another prevalent approach involves the combination of redundancy with masking techniques. These countermeasures gained popularity for their effectiveness in protecting against SCA in addition to protection against fault attacks as an attacker will always abuse the weakest link of the implementation. Examples of such countermeasures include, among others, ParTI [SMG16], Impeccable Circuits I, II, III [AMR+20,SRM20,RSM21], Transform-and-Encode (TaE) [SJR+19], DOMREP I, II [GPK+21, PBGS24], and Private Circuits II [IPSW06]. All these countermeasures utilize redundancy through duplication or error detection codes, albeit with various levels of granularity in their error detection/correction and potentially providing protection against ineffective faults (*e.g.,* Statistical Ineffective Fault Attacks (SIFA) [DEK+18, DEG+18]). Daemen *et al.* [DDE+20] adopt a distinct approach utilizing reversible operations to ensure fault propagation, where a single error check is employed at the end of the circuit. Similarly, the StaTI work [DOT24] proposed a notion

called "stability" that ensures fault propagation allowing the implementation of a single error check at the end of the circuit. Even though the employed techniques prevent ineffective faults, all these countermeasures remain susceptible to strategic fault injections overcoming error detection due to their deterministic nature. Following a different approach, CAPA [RMB+18] and M&M [MAN+19] utilize information-theoretic MAC tags as a form of redundancy, where the MAC tags are randomized mappings, preventing strategic faults bypassing error detection. CAPA adapts a multiparty computation protocol to protect against fault attacks and their combination with SCA. Leveraging the strong formal security guarantees inherent to multiparty computation, CAPA's implementation on hardware devices is prohibitively costly. Moreover, a recently introduced attack [TNN24a] breaks its claimed security, limiting CAPA's effectiveness to protection against either SCA or fault attacks, and not their combination, while still necessitating a substantial hardware area overhead. Relaxing the security model of CAPA, M&M extends any probing secure masking scheme with MAC tags, albeit without claimed protection against ineffective faults as evidenced by a recent zero-value attack from TCHES 2024 [HMA+24]. As a result, the current schemes using MAC tags have limitations: CAPA, while secure against fault attacks, is prohibitively expensive, and M&M does not protect against specific SIFA-like attacks.

Contributions. As all the countermeasures utilizing MAC tags are either not secure against specific SIFA-like attacks, or have significant hardware costs, we propose a framework to securely utilize such information-theoretic MAC tags with comparable hardware costs.

We propose the notion of "MAC-stability" which is the natural extension of the security notions proposed by the StaTI work [DOT24]. The notion enables secure computation of masked and MACed data, where only a single error check is required at the end of computation alleviating the need for a complex abort mechanism on hardware. Moreover, we show that the original stability notion from StaTI, previously regarded as only serially composable, and the MAC-stability notion are in fact arbitrarily composable. This makes them the first arbitrarily composable security notions which provide fault protection, including ineffective fault attacks, without intermediate abort signals or error correction. This arbitrary composability enables the arbitrary decomposition of complex circuits into simpler components, thereby overcoming challenges of ensuring stability in the complex circuits such as AES S-box.

We prove that masking combined with MAC tags provides an improved protection against fault attacks compared to linear encoding techniques such as duplication as we provide bounds on the advantage considering three different fault adversaries: the *gate/register faulting adversary*, which is the typical threshold fault model used in several works [IPSW06, AMR+20, SRM20, RSM21, DOT24]; the *arbitrary register faulting adversary* which injects an arbitrary number of bitflips in a single register stage; and the *random register faulting adversary* from [DN23] which injects arbitrary faults in a single register stage with a limited probability for the faults to apply.

We provide a generic transformation to create arbitrary composable countermeasures secure against side-channel attacks (probing secure) (x)or fault attacks from probing secure solutions. This transformation is demonstrated using the well-known DOM method [GMK16] considering the SNI-secure variant from Faust *et al.* [FGP+18] using our novel non-masked MAC-secure gadgets of the field addition, multiplication, and of linear functions.

The above gadgets are then used to implement a masked and MACed full AES design on hardware using a serialized architecture and a multiplication chain-based S-box. Its implementation cost including area, latency, and randomness usage is then compared to the countermeasures using MAC tags, as well as its security.

2 Preliminaries

In this section, we introduce the probing and fault adversary models with their corresponding security models. We then introduce Boolean masking, domain-oriented masking, and information-theoretic MAC tags as countermeasures to address probing and fault adversaries. Lastly, we introduce StaTI [DOT24], a countermeasure framework introducing two key notions: *stability* and *fault non-completeness*, aimed at protecting against Statistical Ineffective Fault Attacks (SIFA) [DEK+18, DEG+18].

2.1 Adversary and Security Models

We consider an adversary with probing and faulting capabilities within a scenario where these capabilities are not combined. We adhere to the digital circuit model (*i.e.,* attack surface) outlined in [ISW03], represented as a Directed Acyclic Graph (DAG). In this model, the vertices represent Boolean logic and memory gates, and the edges represent the wires carrying elements in $\mathbb{F}_{2^n}$. These circuits also incorporate randomization, with certain vertices producing output bits distributed uniformly and independently. Additionally, in this paper, we work with gadgets that correctly execute a function $F : \mathbb{F}_q^{k_1} \to \mathbb{F}_q^{k_2}$. By composing gadgets, we construct more complex circuits, benefiting from their individual security attributes. The composition of gadgets forms a DAG whose vertices are the gadgets and whose edges are the connections between the gadgets. The incoming and outgoing edges correspond to the input and output of the respective gadgets.

We now go over the capabilities of the probing and faulting adversaries. For the adversary capable of probing, we consider the standard glitch-extended probing model. For faulting capabilities, we consider three different fault adversaries following the work of [TNN24b] that investigates the adversary models assumed in fault attacks and countermeasures, compared to the capabilities of the physical fault injection mechanisms.

According to Toprakhisar *et al.* [TNN24b], the physical fault injection mechanisms can be categorized into two groups. The first group involves injection

mechanisms with high precision, targeting small areas like a few gates on ASIC, such as laser fault injection. These injection mechanisms can target specific gates with specific fault types. To address such injection mechanisms, we consider a *gate/register faulting adversary* capable of precisely targeting a number of chosen gates/registers with specific fault types. The second group involves mechanisms with low precision but affecting larger areas, such as clock/voltage glitching and EM-fault injection. These injection mechanisms can affect more adjacent gates than the ones originally targeted, albeit with limited control over the fault types. To target such mechanisms, we consider an *arbitrary additive register adversary* capable of impacting a larger area with data-independent faults, resulting in additive faults. Additionally, we consider a *random register faulting adversary* addressing certain capabilities of both groups of injection mechanisms. This adversary is capable of injecting an arbitrary number of faults of their choosing with each having a certain probability of being successfully injected. While the random register faulting adversary addresses the precise faulting capabilities of the first group of injection mechanisms depending on the success of the fault injection, they also address the larger impact area of the second group of injection mechanisms. In Table 1, we compare the capabilities of the fault injections adversary models we use in this work.

Table 1. Comparison of the different fault injection adversaries in this work. Accuracy is a value between zero and one indicating the probability that the fault injected by the adversary matches the intended target fault.

	Accuracy	# Faults	Fault Type	Logic
k-Gate/Register Faulting	One	k	Any	Any
Additive Register Faulting	One	Unlimited	Additive	Registers
Random Register Faulting	κ	Unlimited	Any	Registers

Wire Probing. We consider the d-probing model, as introduced by Ishai *et al.* [ISW03]. In this model, the adversary is limited to observing a maximum of d predetermined wires within the Boolean circuit. To account for the physical effect of glitches on a hardware circuit, we extend the probing model to the glitch-extended robust probing model introduced by Faust *et al.* [FGP+18]. This model allows an adversary to obtain all the registered inputs leading to the probed gate/wire. In this work, we focus on a first-order (glitch-extended) probing adversary, where the adversary probes a single gate/wire.

Gate/Register Faulting. To capture the first type of faulting capabilities, we adopt the same approach as in StaTI [DOT24], where an adversary can manipulate the outputs of a total of k gates or registers within the circuit. This k-gate/register faulting adversary can manipulate the outputs in three ways: setting the output to zero (reset faults), setting it to one (set faults), or flipping the output value (bitflip faults). In this work, our focus lies specifically

on first-order gate/register-faulting security, where the adversary is capable of replacing only one gate or register throughout the entire computation process, and the injected fault is non-persistent (the next execution of the circuit the fault disappears). As each wire corresponds to either the output of a register or a gate, we consider wire faults as faults injected to corresponding registers or gates. This excludes the scenarios where an adversary cuts the connections of wires and targets specific sets of wires.

Arbitrary Additive Register Faulting. To capture the second type of faulting capabilities, we consider an adversary limited to targeting the state registers, which can only inject additive faults following a chosen distribution with the limitation that this distribution cannot depend on the state values of the circuit. In symbols, consider the registered state $\mathbf{x}$, the adversary is allowed to inject any additive fault $\mathbf{x} \to \mathbf{x} + \Delta$ for Δ drawn from a distribution which is independent of $\mathbf{x}$. A realistic example of such a faulting scenario includes an adversary bit flipping the entire state for any fixed value Δ or uniform randomly flipping each bit (drawing a uniform random Δ). Similar to gate/register faulting, we consider first-order arbitrary additive register faulting adversaries, where the adversary is capable of faulting a single (state) register layer.

Compared to the gate/register faulting adversary, which can inject a limited number of faults of any type, the additive register faulting adversary can inject an unlimited number of faults (limited to a single time frame), but is limited to additive faults only.

Random Register Faulting. The third faulting capability is based on the random fault model by Dhooghe and Nikova [DN23] where we constrain the fault model to work only on a single register layer. In this fault model, the adversary can inject a fault in every bit in the state registers by applying a chosen function $g : \mathbb{F}_2 \to \mathbb{F}_2$ where the adversary can choose a different function for each bit. Each injected fault, however, only has a limited probability κ to be injected and only one fault can be injected per state bit. More formally, the injected fault is a random function F applied to the bit x for which

$$F(x) = \begin{cases} g(x) \text{ with probability } \kappa\,, \\ x \text{ with probability } 1 - \kappa\,. \end{cases}$$

This model includes the realistic fault scenario of an adversary setting or resetting any partial set of bits of the elements in the registered state $\mathbf{x}$ (making the distribution Δ independent of the sharing of the secret x).

Compared to the previous adversaries, the random register faulting adversary can inject an unlimited number of faults (constrained to a single time frame), and of any fault type, but is limited in the accuracy of each fault as each fault only has a limited probability to be applied.

Security Model. For the security model, we consider two models: *correctness* and *privacy*.

Concerning the *correctness model*, the adversary $\mathcal{A}$ (either a probing or faulting adversary) interacts with a challenger $\mathcal{C}$ by providing it a secret value x. The challenger encodes this secret value in a circuit C which is then returned to the adversary. The adversary can then interact with the circuit. For ease of this paper, we only consider that the adversary makes a single query to C (*i.e.,* it only probes/faults the circuit once). In addition, the adversary can not interact with the encoding (masking, encoding) and decoding (error checking, unmasking) phases of C. The circuit C then returns after the interaction with the adversary any probed values and the state of its output being either 'correct', 'incorrect', or 'abort', however, the circuit does not return the actual output. The adversary wins the correctness game if C returns 'incorrect' meaning that the circuit gave an incorrect output. The advantage of $\mathcal{A}$ is then calculated as

$$Adv_{cor}(\mathcal{A}) = \Pr[C_{\mathcal{A}}[x] = \text{'incorrect'}]\,,$$

where $C_{\mathcal{A}}[x]$ denotes the output of the circuit encoded with the secret x after interaction of the adversary $\mathcal{A}$. We call the advantage of a circuit C, the maximal advantage over any possible adversary. It is clear that a probing adversary can not have a non-trivial advantage in the correctness game since it can not influence the state of the circuit.

The second model is the *privacy* model for which we use a left-right model where an adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$. The adversary can choose two secret values x^0, x^1, and the challenger chooses one of the two and encodes it in a circuit C (so either $C[x^0]$ or $C[x^1]$) which is then given to the adversary. Depending on the above adversaries, it can then interact with C by probing or faulting it. This interaction is the same as in the correctness model, meaning that the interaction only returns probed values and the state of the output. After the interaction, the adversary then decides which secret value (x^0 or x^1) the challenger chose. The adversary wins this game when it can correctly guess the secret value chosen. The advantage is calculated as

$$Adv_{priv}(\mathcal{A}) = |\Pr[\mathcal{A}^{C[x^0]} = x^0] - \Pr[\mathcal{A}^{C[x^1]} = x^0]|\,,$$

where $\mathcal{A}^{C[x]}$ denotes the output of the adversary $\mathcal{A}$ after it interacted with the circuit $C[x]$.

Finally, in case we call a circuit probing-secure, we mean that a probing adversary has a zero advantage when interacting with the circuit. Moreover, a circuit which is Strong Non-Interferent (SNI) [BBD+16] is also secure in the above models.

2.2 Boolean Masking and Domain-Oriented Masking

In order to thwart probing adversaries, we use Boolean masking which splits each variable $x \in \mathbb{F}_{2^n}$ in the circuit into s_x shares $\bar{x} = (x_0, x_1, ..., x_{s_x-1})$ such that $x = \sum_{i=0}^{s_x-1} x_i$ over $\mathbb{F}_{2^n}$. The core idea of splitting the secret variables is to perform computations independently of the processed data. The literature contains many

secure Boolean masking schemes [ISW03, NRR06, RBN+15, GMK16] which are identified by how they perform the nonlinear operations.

In this work, we use Domain-Oriented Masking (DOM) to demonstrate our countermeasure, a generic masking scheme chosen for its low implementation costs and glitch resistance. In DOM, each share of a secret variable is associated with a unique share domain. To provide $d^{th}-$order protection, DOM uses $d+1$ domains (*i.e.,* $d+1$ shares) ensuring the independence of shares across different domains. The independence of domains is straightforward for linear functions as the combination of the shares happens within a domain. Conversely, non-linear operations necessitate dedicated measures due to the combination of different share domains. In DOM, the integration of cross-domain terms into a domain is handled by refreshing (*i.e.,* adding a fresh random value to the terms composed of cross-domain terms). To prevent glitches from propagating from one domain to another, refreshed variables are stored in registers.

However, in this work, we are interested in arbitrary composability and the original DOM multiplier is not arbitrary composable. Instead, we also register the output of the multiplier making the multiplier Strong Non-interferent (SNI) (which will be defined further on in Section 5.2). The proof that the DOM multiplier with the output registered is SNI is given by Faust *et al.* [FGP+18, Section 5.2].

In Figure 1, we illustrate the first-order secure DOM multiplier that has two domains where $x = (x_0, x_1)$ and $y = (y_0, y_1)$ are the multiplication inputs, along with a fresh random value r, and $z = (z_0, z_1)$ is the multiplication output.

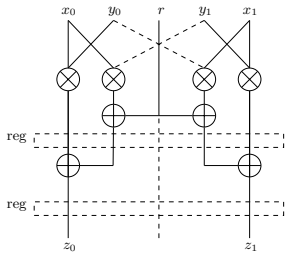


Fig. 1. Two-shared DOM multiplier where the output is registered.

2.3 Information-theoretic MAC Tags

In order to thwart faulting adversaries, we use information-theoretic Message Authentication Code (MAC) tags, which are associated pieces of information used to verify data integrity. The incorporation of MAC tags into fault attack countermeasures was initially achieved by CAPA [RMB+18]. CAPA achieves security against fault and combined attacks by leveraging the principles of the

MPC protocol SPDZ [DPSZ12]. However, a recently proposed attack CAPABARA [TNN24a] exploits the deviation of the CAPA design from SPDZ, effectively breaking CAPA. As a result, its protection is limited to either SCA or fault attacks, but it no longer provides security against combined attacks. Moreover, CAPABARA can be reduced to a fault attack requiring the injection of two faults: one during the preprocessing stage, as in CAPABARA, a second fault to probe the unmasked byte value by setting the variable to a specific value and observing whether the circuit aborts.

Building upon CAPA's methodology, M&M [MAN+19] extends any SCA-secure masking scheme by incorporating MAC tags to ensure data integrity, albeit under a more relaxed adversary model. In M&M, each secret variable is first associated with a MAC tag, after which both are shared, where also the MAC key α is shared. M&M makes use of a side-channel secure multiplier and transforms it to a side-channel and fault-secure one. Specifically, denoting $\odot$ as the side-channel secure multiplication gadget, the following is the multiplication gadget of M&M, given that x and y are the multiplication inputs, τ_x and τ_y are the corresponding MAC tags, and α is the MAC key (with all inputs shared)

$$z \mapsto x \odot y, \qquad \tau_z \mapsto \alpha^{-1} \odot \tau_x \odot \tau_y \,.$$

M&M employs a single error check at the end, however, it fails to detect ineffective faults, which is also not claimed in their security assumptions. A recently proposed zero-value attack and the countermeasure against it [HMA+24] underscores the necessity for additional measures beyond a straightforward combination of masking and MAC tags. The idea of this SIFA-like attack is that when the input of the S-box is zero, an injected fault within the inversion circuit is masked by the final multiplications involving the input in the M&M V2 implementation using tower field decomposition. This attack essentially exploits the structure of tower field decomposition, where the input of the inversion itself is multiplied by an intermediate value, which if faulty, the zero input cancels the injected fault. We note that M&M V1 using the multiplication chain is also vulnerable to a similar attack as one can perform a zero-value attack on the final step of the multiplication chain involving the inversion input.

In response to the proposed zero-value attack on M&M, Hirata *et al.* [HMA+24] extend M&M by introducing "λ-detection" placing intermediate cross-checks at criticial points inside the AES S-box V2. These cross-checks ensure the faults can be detected even when they are multipled by the zero input, which would otherwise mask the effect of the fault.

In this paper, similar to the CAPA and M&M countermeasures, a MAC tag is generated for each registered variable in the state $x \in \mathbb{F}_{2^n}$ by computing $\tau_x = \alpha \cdot x$, where α is the MAC key, and $\cdot$ is the field multiplication in $\mathbb{F}_{2^n}$. Considering the calculation on MAC tag pairs (x, τ_x), the MAC key α (or its inverse α^{-1}) is considered as a global variable which means that the circuit only stores it once where the same value is used throughout the computation (and distinct for each encryption). These MAC tag pairs allow for the verification of

any errors present on values by checking whether

$$\alpha^{-1}\tau_x + x = 0\,.$$

More specifically, in this work, we consider performing this error check only at the end of the circuit (called the *decoding phase*) where the entire state is error checked. If one of the MAC tag pairs does not pass the error check, the circuit does not return an output[1]. Otherwise, if all error checks pass, the circuit returns the first value of each MAC tag pair, meaning it returns x from the pair (x, τ_x).

From the definition of the security models in Section 2.1, recall that the adversary can not fault the above decoding phase as assumed in every countermeasure. This is done to avoid trivial attacks such as faulting the output after the final error check.

2.4 StaTI: Protecting against Fault Attacks Using Stable Threshold Implementations

StaTI [DOT24] is a fault countermeasure framework that is based on threshold implementations [NRR06] and linear encoding techniques. It requires circuits to be *correct*, *non-complete*, *uniform*, *stable*, and *fault non-complete* to be secure against a single gate/register-faulting adversary. Correctness, non-completeness, and uniformity are notions stemming from threshold implementations. Correctness ensures that the sum of the output shares produces the correct output. A shared function is non-complete if each of its coordinate functions operates on data independent of the secret input, and it is uniform if it outputs a uniform sharing given the input sharing is uniform. These notions are detailed further in [NRR06]. In this section, we only focus on stability and fault non-completeness since these will be further explored.

Stability, as introduced in [DOT24], is a composable notion that aims to thwart SIFA [DEK+18, DEG+18] by ensuring the faults present in the input of a register-to-register function propagate to its corresponding output. Stability is initially defined for encoded functions using an arbitrary linear encoding, specifically employing systematic codes where the input data is embedded in the encoded output. An encoded function $\tilde{F}$, encoding F, takes an input codeword (x, x'), where $x' = P(x)$ is the parity data corresponding to x, computed by the function $P(\cdot)$. $\tilde{F}$ then computes the output codeword $(y = F(x), y')$. In this context, an incorrect (*i.e.,* faulty) codeword is defined as an element (x, x') that does not belong to the underlying code, indicating that x' does not match the corresponding parity data of x (*i.e.,* $x' \neq P(x)$).

Definition 1 (Stability). *Given an encoded register-to-register function and a code, the gadget implementing the encoded function is said to be stable if any incorrect input codeword is mapped to an incorrect output codeword.*

[1] This can be done by raising a flag or outputting a special value.

In particular, the above definition holds for any invertible correct encoding since each correct output codeword is mapped from a correct input codeword.

StaTI leverages the composability of the stability notion to ensure that the injected faults propagate to the output of the whole circuit without getting ineffective, particularly in scenarios where the stable gadgets are serially composed.

Theorem 1. *[DOT24, Theorem 1] The serial composition of stable gadgets is stable.*

In addition to ensuring the propagation of faults originating from the inputs, StaTI also limits the impact of injected faults on the output through the implementation of fault non-completeness. This ensures that the propagation of a fault remains independent of secret values, particularly the faults injected to gates between two register layers.

Definition 2 (Fault non-completeness). *A circuit is called fault non-complete when, per register stage, each gate of the circuit drives only a non-complete set of output shares given that these shares can not influence the correctness of the output codeword.*

3 Stability of Arbitrarily Composed Circuits

In the work of StaTI [DOT24], the stability of the serial composition of stable gadgets was proven. This serial composition property is useful to make stable gadgets for example by decomposing the Keccak S-box into several Toffoli gates. However, for many complex components like the AES S-box, achieving stability with only serial composition can be challenging. Therefore, building upon this groundwork, this section now shifts the focus to the stability of circuits *arbitrarily* composed of stable gadgets. This allows us to arbitrarily decompose the complex circuits, and work on simpler gadgets (*e.g.,* multiplication) to achieve stability. We extend Theorem 1, and prove the stability of the circuits arbitrarily composed of stable gadgets with an arbitrary number of inputs and outputs in Theorem 2.

Theorem 2. *The arbitrary composition of stable gadgets with an arbitrary number of inputs and outputs is stable.*

Before proving the theorem, we introduce the notion of an *incorrect codeword* encoding multiple data elements.

Definition 3 (Incorrect codeword). *For a given set of data* $x = (x_0, x_1, ..., x_{k_1})$*, and the corresponding parity data* $x' = P(x) = (x'_0, ..., x'_{k_2})$*, the codeword* (x, x') *is said to be an incorrect codeword if* $x' \neq P(x)$*.*

Given the definition of *incorrect codeword* encoding multiple data, we can prove Theorem 2 as following.

Proof. We begin by proving stability for a circuit composed of two stable gadgets. The proof for the arbitrary number of gadgets recursively follows this.

Let G_1 and G_2 be stable gadgets with an arbitrary number of inputs and outputs, and G_3 be an arbitrary composition of G_1 and G_2 as depicted in Figure 2, where x_1 and x_2 are the input codewords encoding some input data. Due to G_3 being a DAG, we assume that G_1 exclusively derives its inputs from the initial inputs of G_3, while G_2 can derive its inputs from both the initial inputs of G_3 and the outputs of G_1. Then, the outputs of G_3 consist of the outputs of G_2 and a subset of the outputs of G_1.

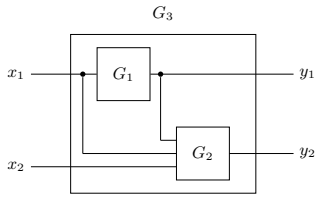


Fig. 2. An arbitrarily composed circuit model.

Let G_3 receive an incorrect input codeword. In this scenario, there are three cases over the connection of the initial inputs of G_3: (1) only G_1 receives a faulty input due to x_1 being an incorrect codeword (2) only G_2 receives a faulty input due to x_2 being an incorrect codeword, and (3) and both G_1 and G_2 receive a faulty input due to both x_1 and x_2, or only x_1 being incorrect codewords. We investigate the error propagation for these three cases:

(1) As G_1 is stable, its output codeword is also incorrect. This incorrect output codeword is then connected to either (a) the input of G_2, and/or (b) the output of G_3. Consequently, the output of G_3 is also an incorrect codeword due to (a) (G_2 propagating the incorrect codeword to the output of G_3) and/or (b).
(2) As G_2 is stable, its output codeword is also incorrect. This incorrect output codeword is then directly connected to the output of G_3.
(3) Since both G_1 and G_2 are stable, their output codewords will also be incorrect. Consequently, the output codeword of G_3 also becomes incorrect.

In all three cases, G_3 outputs an incorrect codeword, making it stable. □

4 Stability Over MAC Tags

In this section, we extend the stability notion to *MAC-stability* over gadgets implementing functions encoded using MAC tags. For such a gadget, the input/output is composed of the secret data $x \in \mathbb{F}_{2^n}$ and its associated tag $\tau_x = \alpha x \in \mathbb{F}_{2^n}$, where $\alpha \in \mathbb{F}_{2^n}$ is the MAC key, and the multiplication is defined

over the field $\mathbb{F}_{2^n}$. Before we describe MAC-stability, we extend the definition of *incorrect codewords* (Definition 3) using MAC tags.

Definition 4 (Incorrect codeword in MAC encodings). *For a given set of data $x = (x_0, x_1, ..., x_k)$, $(x, \tau_x) = (x_0, ..., x_k, \tau_{x_0}, ..., \tau_{x_k})$ is the corresponding systematic codeword where $\tau_{x_i} = \alpha x_i$, and α is global to all x_is. Then, (x, τ_x) is said to be an incorrect codeword if at least one element in (x, τ_x) is faulty (i.e., for at least one i, $\tau_{x_i} \neq \alpha x_i$).*

Then, we describe MAC-stability by extending the definition of stability (Definition 1):

Definition 5 (MAC-stability). *Consider a gadget G implementing an encoded register-to-register function using information-theoretic MAC tags, G is said to be* MAC-stable *if it maps any incorrect input codeword to an incorrect output codeword.*

The stability of the arbitrary composition of MAC-stable gadgets follows Theorem 2 as the proof does not rely on any specific encoding.

Having established the feasibility of constructing MAC-stable circuits from MAC-stable gadgets, we shift our focus to the process of devising MAC-stable gadgets themselves. As stated in [DOT24], the straightforward encodings of the XOR and AND gates do not exhibit stability due to their non-injective nature. In other words, one can fault the inputs of the XOR and AND gates such that the output is still a correct codeword, resulting in undetected (ineffective) faults. Similarly, in $\mathbb{F}_{2^n}$, the straightforward implementations of the field addition and multiplication operations (*i.e.*, $(x, \tau_x) + (y, \tau_y) = (x + y, \tau_x + \tau_y)$ and $(x, \tau_x)(y, \tau_y) = (xy, \alpha^{-1}\tau_x\tau_y)$) also do not exhibit (MAC-)stability due to their non-injective nature. In this section, we demonstrate how we can adapt the implementation of the addition and multiplication operations in $\mathbb{F}_{2^n}$, and a linear operation in $\mathbb{F}_2^n$ such that they are MAC-stable. Note that each of these MAC-stable gadgets is made fault non-complete (Definition 2) by partitioning/duplicating its combinatorial gates such that no gate is used to compute more than one output variable.

In the description of the algorithms, the tag of x is denoted by $\tau_x = \alpha x$ (and equivalently, $\alpha^{-1}\tau_x = x$), where α is the MAC key and global to the whole circuit.

4.1 A MAC-Stable Addition Gadget

In this section, we describe the MAC-stable gadget G_{add} that implements the addition in $\mathbb{F}_{2^n}$ which is detailed in Algorithm 1. & denotes the bitwise AND operation.

Algorithm 1: G_{add} : A MAC-stable addition gadget

Input: x, τ_x, y, τ_y
Global Variable: α^{-1}
Output: z, τ_z
$z = x + y + (\tau_x\alpha^{-1} + x)\&(\tau_y\alpha^{-1} + y)$
$\tau_z = \tau_x + \tau_y$
return z, τ_z

G_{add} is correct when the input codeword is correct (*i.e.*, $(\alpha^{-1}\tau_x + x) = 0$ and $(\alpha^{-1}\tau_y + y) = 0$) since it returns $z = x + y$ and $\tau_z = \tau_x + \tau_y = \alpha z$. In the following, we prove that the proposed addition gadget, G_{add}, is MAC-stable.

Theorem 3. *The gadget G_{add} as defined in Algorithm 1 is MAC-stable.*

Proof. In order to prove the MAC-stability of G_{add}, we show that any incorrect input codeword is mapped to an incorrect output codeword. Let $\Delta_{\alpha^{-1}}, \Delta_x, \Delta_{\tau_x}, \Delta_y, \Delta_{\tau_y}$ be the additive fault values present in $\alpha^{-1}, x, \tau_x, y, \tau_y$, respectively. In that case, the input codeword is incorrect if at least one of the following inequalities holds

$$E_x = (\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_x + \Delta_{\tau_x}) + (x + \Delta_x) \neq 0$$
$$E_y = (\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_y + \Delta_{\tau_y}) + (y + \Delta_y) \neq 0.$$

Then, given the input codeword is incorrect, we show that the output codeword is also incorrect.

To check whether the output codeword is correct, we check whether $(\alpha^{-1} + \Delta_{\alpha^{-1}})\tau_{z'} + z'$, with the (faulted) outputs of G_{add} (z', τ'_z), equals to zero. We can write this out as follows.

$$\begin{aligned}(\alpha^{-1} + \Delta_{\alpha^{-1}})\tau_{z'} + z' &= (\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_x + \Delta_{\tau_x}) + x + \Delta_x \\ &\quad + (\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_y + \Delta_{\tau_y}) + y + \Delta_y + E_x\&E_y \\ &= E_x + E_y + E_x\&E_y \qquad (1)\end{aligned}$$

Considering the truth table of $E_x + E_y + (E_x\&E_y)$, it is zero only when both $E_x = 0$ and $E_y = 0$ which implies that the input codeword is correct.

This implies that G_{add} maps any incorrect input codeword to an incorrect output codeword and is thus MAC-stable. □

In addition to the above proof, we also verified the MAC-stability of the above gadget over $\mathbb{F}_{2^4}$ using software by going over all incorrect codewords for the input[2].

[2] Exhaustive verification in $\mathbb{F}_{2^8}$ requires significant time and resources.

4.2 A MAC-Stable Multiplication Gadget

In this section, we describe the MAC-stable multiplication gadget G_{mult} that implements multiplication in $\mathbb{F}_{2^n}$ which is detailed in Algorithm 2. The equality operation == denotes the equality check operation which evaluates to $1 \in \mathbb{F}_{2^n}$ if both of the operands are identical, and $0 \in \mathbb{F}_{2^n}$ otherwise. The operation $\neq$ does the opposite of the equality operation and evaluates to $0 \in \mathbb{F}_{2^n}$ only if both operands are the same and to $1 \in \mathbb{F}_{2^n}$ otherwise.

Algorithm 2: G_{mult} : A MAC-stable multiplication gadget

Input: x, τ_x, y, τ_y
Global Variable: α^{-1}
Output: z, τ_z
if $(\alpha^{-1}\tau_y + y) \neq 0$ **then**
  $z = 1$
  $\tau_z = 0$
else if $y == 0$ **then**
  $z = xy + \big((\alpha^{-1}\tau_x + x) \neq 0\big)$
  $\tau_z = \alpha^{-1}\tau_x\tau_y$
else
  $z = xy$
  $\tau_z = \alpha^{-1}\tau_x\tau_y$
end
return z, τ_z

G_{mult} is correct when the input codeword is correct (*i.e.,* $(\alpha^{-1}\tau_x + x)$ and $(\alpha^{-1}\tau_y + y)$ are both zero) since it returns $z = xy$ and $\tau_z = \alpha^{-1}\tau_x\tau_y = \alpha z$. In the following, we prove that the proposed multiplication gadget, G_{mult}, is MAC-stable.

Theorem 4. *The gadget G_{mult} as defined in Algorithm 2 is MAC-stable.*

Proof. Similar to the MAC-stable addition gadget, to prove the MAC-stability of G_{mult}, we show that any incorrect input codeword is mapped to an incorrect output codeword. Let $\Delta_{\alpha^{-1}}, \Delta_x, \Delta_{\tau_x}, \Delta_y, \Delta_{\tau_y}$ be the additive fault values present in $\alpha^{-1}, x, \tau_x, y, \tau_y$, respectively. In that case, the input codeword is incorrect if at least one of the following inequalities holds

$$E_x = (\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_x + \Delta_{\tau_x}) + (x + \Delta_x) \neq 0,$$
$$E_y = (\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_y + \Delta_{\tau_y}) + (y + \Delta_y) \neq 0.$$

Given the input codeword is incorrect, we show that the output codeword is also incorrect. We check whether $(\alpha^{-1}+\Delta_{\alpha^{-1}})\tau_{z'}+z'$, with the (faulted) outputs of G_{mult} (z', τ'_z), equals to zero.

First, we assume $E_y \neq 0$ holds. Plugging this in the error check on the output, we find

$$(\alpha^{-1} + \Delta_{\alpha^{-1}})\tau_{z'} + z' = (\alpha^{-1} + \Delta_{\alpha^{-1}})0 + 1 = 1 \neq 0$$

which implies that the output codeword is incorrect. Now, we assume $E_y = 0$ and $E_x \neq 0$ holds. Then, the algorithm reduces to

$$z = (x + \Delta_x)(y + \Delta_y) + ((y + \Delta_y) == 0)$$
$$\tau_z = (\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_x + \Delta_{\tau_x})(\tau_y + \Delta_{\tau_y})\,.$$

Then, when checking for errors $((\alpha^{-1} + \Delta_{\alpha^{-1}})\tau_{z'} + z' == 0)$, we obtain the following:

$$\begin{aligned}&(\alpha^{-1} + \Delta_{\alpha^{-1}})^2(\tau_x + \Delta_{\tau_x})(\tau_y + \Delta_{\tau_y}) + (x + \Delta_x)(y + \Delta_y) + ((y + \Delta_y) == 0)\\ &= E_xE_y + E_x(y + \Delta_y) + E_y(x + \Delta_x) + ((y + \Delta_y) == 0)\,.\end{aligned}$$

When $(y + \Delta_y) = 0$ (and $E_y = 0$, $E_x \neq 0$), we find

$$(\alpha^{-1} + \Delta_{\alpha^{-1}})\tau_{z'} + z' = E_xE_y + E_y(x + \Delta_x) + 1 = 1 \neq 0\,.$$

When $(y + \Delta_y) \neq 0$, the algorithm reduces to

$$z = (x + \Delta_x)(y + \Delta_y)$$
$$\tau_z = (\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_x + \Delta_{\tau_x})(\tau_y + \Delta_{\tau_y})\,.$$

Then, we find

$$(\alpha^{-1} + \Delta_{\alpha^{-1}})\tau_{z'} + z' = E_xE_y + E_x(y + \Delta_y) + E_y(x + \Delta_x) = E_x(y + \Delta_y) \neq 0$$

since both $E_x \neq 0$ and $(y + \Delta_y) \neq 0$.

This implies that G_{mult} maps any incorrect input codeword to an incorrect output codeword, and is thus MAC-stable. □

Similar to the addition gadget, we also verified the MAC-stability of the multiplication gadget over $\mathbb{F}_{2^4}$ using software.

4.3 A MAC-Stable Linear Gadget

In this section, we describe the MAC-stable gadget G_L that implements an invertible linear function L in $\mathbb{F}_2^n$ which is detailed in Algorithm 3. We base the method on the gadget used in the M&M work [MAN+19]. Namely, the multiplication by α over $\mathbb{F}_{2^n}$ can be represented as a (invertible) matrix multiplication by M_α over $\mathbb{F}_2^n$.

Algorithm 3: G_L : A MAC-stable linear gadget

Input: x, τ_x
Global Variable: M_α, M_α^{-1}
Output: z, τ_z
$z = Lx$
$\tau_z = (M_\alpha L M_\alpha^{-1})\tau_x$
return z, τ_z

G_L is correct when the input codeword is correct (*i.e.*, $(\alpha^{-1}\tau_x + x)$ is zero) since it returns $z = Lx$ and $\tau_z = (M_\alpha L M_\alpha^{-1})\tau_x = \alpha(L\alpha^{-1}\tau_x) = \alpha Lx$. Note that $M_\alpha L M_\alpha^{-1}$ can be precomputed for efficiency reasons. In the following, we prove that the proposed linear gadget, G_L, is MAC-stable.

Theorem 5. *The gadget G_L as defined in Algorithm 3 is MAC-stable.*

Proof. Given that L and M_α are invertible, the operation in Algorithm 3 is also invertible and thus MAC-stable. □

5 MAC-Stability in the Masked Domain

As stated in Section 2.1, we assume an adversary with probing capabilities *(x)or* faulting capabilities. Therefore, we now shift our focus to the MAC-stability in masked domains, and show how to secure circuits against *probing*, *gate/register-faulting*, *arbitrary additive register faulting*, and *random register faulting* adversaries in a non-combined setting.

5.1 Security of Masked MAC Pairs

In this section, we quantify the security (two-share) masked MAC pairs can offer and provide proofs for the security of the computation of MAC-stable gadgets against the adversaries described in Section 2.1.

Similar to StaTI [DOT24], we first share the input of the cipher and then encode these shares. Thus, a secret variable $x \in \mathbb{F}_{2^n}$ is first shared as (x_0, x_1) such that $x_0 + x_1 = x$, and then encoded to (x_0, x_1), (τ_{x_0}, τ_{x_1}), α^{-1} such that $x_0 = \alpha^{-1}\tau_{x_0}$ and $x_1 = \alpha^{-1}\tau_{x_1}$. Note that, we do not share the MAC key α used to compute the MAC tags of the shares, nevertheless, akin to M&M, we require distinct MAC keys for each encryption to avoid linear encoding characteristics.

For the first lemma, we provide the advantage of a gate/register faulting adversary[3] against masked MAC tag pairs.

Lemma 1. *The maximal advantage of a gate/register faulting adversary $\mathcal{A}$ against k MAC tag pairs over $\mathbb{F}_{2^n}$ in the correctness and privacy model is 2^{-kn}.*

Proof. Consider k masked MAC tag pairs $(x_0[i], x_1[i]), (\tau_{x_0[i]}, \tau_{x_1[i]})$ of the variables $x[i] \in \mathbb{F}_{2^n}$ for $i \in \{0, ..., k-1\}$ and the inverse of the MAC key α^{-1}. Finally, recall that the gate/register faulting adversary $\mathcal{A}$ can inject any fault (bitflip, set-to-zero, etc.) but can only fault one variable out of the MAC tag pairs or the MAC key. We provide the advantage of $\mathcal{A}$ in the correctness and the privacy game.

Correctness: For the correctness game, the goal of the adversary is to fault the state such that, after decoding, the output is incorrect and the circuit did not abort. It is clear that faulting the MAC tags $\tau_{x_0[i]}, \tau_{x_1[i]}$ or the inverse MAC

[3] Since we only consider faulting the encoding and not the computation, this adversary reduces to a single register faulting adversary.

key α^{-1} does not change the output correctness. That is, the circuit can still abort from the injected fault but, when an output is given, it is never incorrect since the output, $x = x_0 + x_1$, contained no fault. Thus, consider a fault on $x_j[i]$ for $j \in \{0, 1\}$ and $i \in \{0, ..., k-1\}$. We model this fault as an additive fault $x_j[i] + \Delta$. If $\Delta = 0$, then clearly the output stays correct. In case $\Delta \neq 0$, then the error check on $x_j[i]$ calculates

$$\alpha^{-1}\tau_{x_j[i]} + x_j[i] + \Delta = \Delta \neq 0\,.$$

Similarly, when only a single fault in a MAC tag $\tau_{x_j[i]}$ is placed, the circuit would also always abort. Thus, the error is always detected and the advantage of the adversary in the correctness game is $Adv_{cor}(\mathcal{A}) = 0$.

Privacy: In the privacy game, the adversary picks two secret values and attempts to determine which of the two was embedded in the circuit by observing the probability of the circuit aborting after faulting it. We consider $\mathcal{A}$ picking two secret values to distinguish (denoted by x^0 and x^1). We calculate the probability of the circuit aborting when a fault is injected during the computation on either x^0 or x^1. As mentioned in the proof of the correctness model, faulting $x_j[i]$ or the tags $\tau_{x_j}[i]$ such that their value changes always results in an abort. However, setting these shares to a fixed value results in an abort only if the fault led to an additive change in the value, and it will always abort when the value is changed. Thus, from the abort status, the adversary only learns the value of the faulted register which is a single share or tag, and is uniform randomly distributed revealing no information about the secrets. This leaves faulting α^{-1} which provides the following equations for error checks

$$(\alpha^{-1} + \Delta)\tau_{x_0[i]} + x_0[i] == 0, \qquad (\alpha^{-1} + \Delta)\tau_{x_1[i]} + x_1[i] == 0\,,$$

which always detect an error unless $\tau_{x_0[i]} = \tau_{x_1[i]} = 0$ for $i \in \{0, ..., k-1\}$ which means $x_0[i] = x_1[i] = 0$ since $\alpha^{-1} \neq 0$. More specifically, an error can only pass when each $x[i] = 0$. As a result, the maximal advantage of this adversary is $Adv_{priv}(\mathcal{A}) = 2^{-kn}$. Namely, the above equality holds due to the probability of the error check passing when the secret is zero (when all $x[i] = 0$) being 2^{-kn} since it requires that all $x_0[i] = x_1[i]$, and since the circuit with $x \neq 0$ always aborts. □

The above lemma shows that masked MAC tags provide sufficient security against a single gate/register faulting adversary by achieving security equivalent to the state size of the cryptographic primitive that is implemented. For example, since the AES has a 128-bit state size, a masked MACed AES implementation provides 128-bit security against a register/gate faulting adversary (without including a masked MAC for the key).

We now do the same for an arbitrary additive register faulting adversary.

Lemma 2. *The maximal advantage of an arbitrary additive register faulting adversary $\mathcal{A}$ against k MAC tag pairs over $\mathbb{F}_{2^n}$ in the correctness and privacy model is $\frac{1}{2^n-1}$.*

Proof. Recall that the arbitrary additive register faulting adversary $\mathcal{A}$ can inject any additive fault to the entire state (all shares, tags, and MAC key). We provide the advantage of $\mathcal{A}$ in the correctness and the privacy game.

Correctness: In the correctness model, faulting $x_j[i]$, $\tau_{x_j[i]}$, and α^{-1} by additive faults $\Delta_{x_j[i]}, \Delta_{\tau_{x_j[i]}}, \Delta_{\alpha^{-1}}$ provides the following error check

$$(\alpha^{-1} + \Delta_{\alpha^{-1}})(\tau_{x_j[i]} + \Delta_{\tau_{x_j[i]}}) + x_j[i] + \Delta_{x_j[i]} == 0.$$

When $\Delta_{\alpha^{-1}} \neq 0$, the adversary has to guess for each i, j the following equality

$$\Delta_{\alpha^{-1}} = \frac{\Delta_{\tau_{x_j[i]}}\alpha^{-1} + \Delta_{x_j[i]}}{\tau_{x_j[i]} + \Delta_{\tau_{x_j[i]}}},$$

which can only be done by guessing α^{-1} with probability $\frac{1}{2^n-1}$ (since $\alpha \neq 0$) or by guessing each pair $(x_j[i], \tau_{x_j[i]})$ for a total probability 2^{-kn}. When the adversary does not fault α^{-1}, the adversary can still guess the value of α^{-1} and fault one pair $(x_j[i], \tau_{x_j[i]})$ with probability $\frac{1}{2^n-1}$ which provides the advantage $Adv_{cor}(\mathcal{A}) = \frac{1}{2^n-1}$. It is important to note that the above bound does not depend on the parameter k or, as a side-note, on the number of shares.

Privacy: Similar to the proof of the gate/register faulting adversary in the privacy model, we consider $\mathcal{A}$ picking the two secret values to distinguish as x^0 and x^1. When additively faulting x_0, x_1, τ_{x_0}, and τ_{x_1} the error is detected when the adversary guesses α^{-1} incorrectly. However, whether or not the circuit aborts is independent of the secret value given to the circuit. Thus, in order to break the privacy of the circuit, the adversary has to fault the inverse MAC key α^{-1} which leads to the same advantage as in Lemma 1, $Adv_{priv}(\mathcal{A}) = \frac{1}{2^{kn}}$. □

Finally, the advantage is given for the third adversary, the random register fault adversary. Note that calculating the advantage for this adversary is more complex, thus, we discuss several attack vectors but do not conclude on the attack vector with the highest advantage.

Lemma 3. *The advantage of a κ- random register faulting adversary $\mathcal{A}$ against k MAC tag pairs over $\mathbb{F}_{2^n}$ in the correctness and privacy model is equal or higher than $\kappa^2/2$.*

Proof. Recall that the random register faulting adversary $\mathcal{A}$ with parameter κ can inject any fault (modeled as the application of a function $g : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$) to each separate bit of the state (all shares, tags, and MAC key), however, each fault only has a κ probability of being applied. We provide the advantage of $\mathcal{A}$ in the correctness and the privacy game.

Correctness: In the correctness model, we consider an adversary which injects a bitflip in both $x_j[i]$ and $\tau_{x_j[i]}$ while guessing α. This fault has a κ^2 probability for both bitflips to occur and a $\frac{1}{2^n-1}$ probability that α maps the first bitflip to the other. The attack is improved when κ is small by guessing α (for example guessing $\alpha = 1$) and repeating the attack over multiple (say ℓ) shared

pairs $(x_j[i], \tau_{x_j[i]})$, for which we have $\ell = 2k$ since there are k secret values and 2 shares. This provides the advantage

$$Adv_{cor}(\mathcal{A}) = \max_{\ell \in \{1,\ldots,2k\}} \frac{\ell\kappa^2(\kappa^2 + (1-\kappa)^2)^{\ell-1}}{2^n - 1},$$

since one faulted shared pair (out of the ℓ pairs) needs to be faulted such that the fault is again a correct codeword ($\ell\kappa^2$ after correctly guessing α) and the other pairs are either faulted to a correct codeword or not faulted at all (probability $(\kappa^2 + (1-\kappa)^2)^{\ell-1}$ after guessing α). For ease, we assume $\kappa \approx 1$ which provides the maximal advantage as $\frac{\kappa^2}{2^n-1}$.

Privacy: In the privacy model, we consider the attack mentioned in [DN23] from the "mask-then-duplicate" case which provides the advantage $Adv_{priv}(\mathcal{A}) = \kappa^2/2$. This attack simply sets all Boolean masked shares to a specific value to uncover the secret (using faults as probes). When κ is small, this attack can be repeated over the kn 2-shared bits of the state (x_0, x_1) (when all $x^0[i] = 0$ versus all $x^1[i] = 2^n - 1$) for a maximal advantage

$$\mathrm{Adv}(\mathcal{A}) = \max_{\ell \in \{1,\ldots,kn\}} \left|(1-\kappa)^\ell - \frac{1}{2}(1 + (1+\kappa^2)^\ell)\right|.$$

Again, for ease, we consider $\kappa \approx 1$ such that the maximal advantage is $\kappa^2/2$. □

Lemma 2 shows why MAC tags offer improved security over code-based encodings such as duplication, which offer no security (advantage one) against an arbitrary additive register faulting adversary. Namely, an arbitrary additive adversary can always additively inject correct codewords into the state to change its correctness. Additionally, compared to the results in Lemma 3, linear encodings also perform worse than MAC tags in the correctness model against a random register faulting adversary, as the advantage for linear encodings does not depend on the field size (*e.g.,* in duplication, one changes the correctness via two bitflips regardless of the field size). This is reflected in Table 2, where MAC-tag based countermeasures demonstrate an improved security in the correctness model over duplication/linear codes/error correction. This exponential improvement in correctness security with increasing field size makes MAC-tag based countermeasures particularly worthy of further investigation.

Finally, we note that the advantage in the correctness model of Lemmas 2 and 3 can be improved by using multiple MAC keys; with ℓ MAC keys, we get an advantage $\frac{1}{(2^n-1)^\ell}$. However, in this work, we focus on using one MAC key for efficiency reasons.

While Lemmas 1-3 provide security bounds for the encoding itself, we now move to the security of circuits computing on masked MAC-encoded data. We show that the computation does not degrade the security, as shown in the lemmas, provided that each register stage is fault non-complete and MAC-stable, and that the computation is glitch-extended probing secure.

Table 2. Correctness and privacy security against a gate/register faulting adversary $\mathcal{A}_1$, an arbitrary additive register faulting adversary $\mathcal{A}_2$, and a random fault adversary $\mathcal{A}_3$ for different encoding schemes (following a mask-then-encode technique) for k pairs of n-bit values. Triplication performs error correction. For the linear code countermeasure, we consider a code with minimal distance $t > 1$. For the random fault model, for simplicity, we consider κ close to one.

		$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_3$
Correctness	Masking + Duplication	0	1	κ^2
	Masking + Triplication	0	1	$1-(1-\kappa^2)^{nk}$
	Masking + Linear Code	0	1	κ^t
	Masking + MAC tag	0	$\frac{1}{2^n-1}$	$\frac{\kappa^2}{2^n-1}$
Privacy	Masking + Duplication	0	0	$\kappa^2/2$
	Masking + Triplication	0	0	$\kappa^2/2$
	Masking + Linear Code	0	0	$\kappa^2/2$
	Masking + MAC tag	2^{-kn}	2^{-kn}	$\kappa^2/2$

Theorem 6. *A glitch-extended probing secure masked gadget G where each register stage is fault non-complete and MAC-stable is as secure against the gate/register faulting, the arbitrary additive register faulting, and the random register faulting adversaries as the encoding of its state.*

Proof. Denote the gate/register faulting adversary by $\mathcal{A}_1$, the arbitrary additive register faulting adversary by $\mathcal{A}_2$, and the random register faulting adversary by $\mathcal{A}_3$.

Consider that $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ fault a register layer in G. We distinguish three cases. (i) The fault did not change the state. Then, G provides the expected output since the state remains unchanged. (ii) The fault modified the state into a correct encoding of another secret. As G is correct, it outputs a correct encoding of the new secret value. (iii) The fault resulted in an incorrect state encoding. Since G is MAC-stable, the output of G is again an incorrect encoding which, after an error check, results in an abort. Consequently, G's output state (correct, incorrect, or abort) does not change from the state of the faulted register. This ensures that the advantage of $\mathcal{A}_1$, $\mathcal{A}_2$, or $\mathcal{A}_3$ is equivalent to faulting the intermediate register. These advantages were accounted for in the lemmas in Section 5.1.

In case $\mathcal{A}_1$ faults the combinatorial logic of G, due to the fault non-completeness, this fault affects only a single share or tag of the next register. This is equivalent to $\mathcal{A}_1$ faulting the next register in the correctness model. Since G is glitch-extended probing secure, any additive fault on the next register layer is independent of the secrets of G. Furthermore, due to the MAC-stability of each stage of G, the fault will be propagated to the output of G causing an abort that is independent of any secret value. As a result, $\mathcal{A}_1$'s advantage is again equivalent to faulting the intermediate register as described in the lemmas in Section 5.1. □

The above theorem shows that in order to secure against the adversaries from Section 2.1, it is sufficient to have a gadget which is glitch-extended probing secure, fault non-complete, and MAC-stable. In the next section, we investigate how to create such gadgets.

5.2 Transforming Probing Secure Circuits

What makes MAC-stability with masking easy to implement is that there is a simple transformation of any glitch-extended probing secure circuit to a MAC-stable one. Since MAC-stability is an arbitrary composable notion as shown in Theorem 2, we show the transformation of an arbitrary composable probing secure gadget for which we make use of the SNI security notion by Barthe *et al.* [BBD+16]. We thus quickly introduce the notion of simulatability and SNI.

Definition 6 (Simulatability [CS20]). *Let $P = \{p_1, ..., p_\ell\}$ be a set of ℓ probes of a gadget C and C_P the tuple of values of the probes for an execution of C. Let $I = \{(i_1, j_1), ..., (i_k, j_k)\} \subset \{0, ..., d-1\} \times \{0, ..., m-1\}$ be a set of input wires of C. A simulator is a randomized function $\mathcal{S} : \mathbb{F}_q^k \to \mathbb{F}_q^\ell$. The set of probes P can be simulated with the set of input wires I if there exists a simulator $\mathcal{S}$ such that for any inputs $x_{*,*}$, the distributions $C_P(x_{*,*})$ and $\mathcal{S}(x_{i_1,j_1}, ..., x_{i_k,j_k})$ are equal, where the probability is over the random coins in C and $\mathcal{S}$.*

The above definition defines security in terms of a simulation game. This framework is extended to SNI security where we define which information is given to the simulator.

Definition 7 (d-Strong Non-Interferent (d-SNI) [BBD+16]). *A gadget G is d-SNI if any set of d_1 (glitch-extended) probes on its intermediate variables and every set of d_2 (glitch-extended) probes on its output shares such that $d_1 + d_2 \leq d$, the totality of the probes can be simulated by only d_1 shares of each input.*

More specifically, we focus on 1-SNI which we compactly denote as SNI.

Moving on to the MAC-stable transformation, the transformation works by replacing each $\mathbb{F}_{2^n}^k \to \mathbb{F}_{2^n}^\ell$ function (register-to-register) in the original gadget with a MAC-stable counterpart. A key requirement of this transformation is that it operates without introducing any additional inputs beyond those of the original function, except for the MAC tags of the inputs and the MAC key α. This is achieved by separately replacing the computation of each n-bit output word.

Theorem 7. *Given a glitch-extended SNI secure gadget G, when replacing each $\mathbb{F}_{2^n}$ output function with its MAC-stable variant, the new gadget G_{stable} is SNI and MAC-stable.*

Proof. We first show that G_{stable} is MAC-stable. From Theorem 2, we know that the arbitrary composition of MAC-stable circuits is again MAC-stable, thus G_{stable} is MAC-stable.

Next, we show that G_{stable} is again SNI secure. Take any glitch-extended probe in G_{stable}, it reads the inputs from one register-to-register coordinate function. From the construction of G_{stable}, that coordinate function is the MAC-stable transformation of a coordinate function of G which is SNI. However, the MAC-stable transformation does not use any more inputs than the original register-to-register function together with the MAC tags of those inputs (which provide no extra information). Thus, there exists a simulator that adheres to the SNI properties (*i.e.,* which provides the specific input shares depending on the probe locations following Definition 7) which simulates the probed variable. Since the probe in G_{stable} observes no additional information except for the MAC key α which can be safely given to the simulator, the simulator can also simulate the probed variable in G_{stable} showing that the gadget is again SNI. □

We note that the above proof would be similar for Probe-Isolating Non-Interference (PINI) by Cassiers and Standaert [CS20], and thus the MAC-stable transformation would also keep PINI security untouched.

In addition, the above proof also works for d-SNI secure gadgets, meaning that there is a transformation that provides countermeasures that are arbitrary composable secure against adversaries which either place d probes (x)or which fault the circuit following the faulting adversaries from Section 2.1.

It is also clear that the above transformation is easily made fault non-complete by using different combinatorial gates for each output share or tag and by copying constant values.

Recall that the above transformed MAC-stable, non-complete, and SNI (thus glitch-extended probing secure) circuit is secure against a gate/register faulting, arbitrary additive register faulting, or random register faulting adversary as shown in Theorem 6. In the following section, we apply the transformation to the well-known Domain-Oriented Masking (DOM) method.

5.3 MAC-Stable Domain-Oriented Masking

In this section, we provide a MAC-stable SNI-secure multiplier. The multiplier is based on the DOM multiplier, however, we note that (as mentioned in Theorem 7) any masked multiplier can be transformed to be MAC-stable. The output of the DOM multiplier is considered to be registered such that the gadget becomes SNI secure as shown by Faust *et al.* [FGP+18]. While it is possible to make a MAC-stable DOM multiplier by replacing each addition and multiplication in the algorithm by the MAC stable variant from Sections 4.1 and 4.2, we choose to further improve the efficiency by still incorporating more efficient non-MAC-stable components while keeping the overall multiplier MAC-stable.

We depict the first-order masked MAC-stable multiplication gadget in Figure 3. The multiplication inputs are split into two shares such that $x = x_0 + x_1$ and $y = y_0 + y_1$, where each share is associated with a tag τ_{x_i/y_i} with α the MAC key which is global to the whole circuit. r is a fresh random value associated with τ_r as its MAC tag. The cross products x_0y_0 and x_1y_1 (and the corresponding tags) are calculated using the MAC-stable multipliers from Algorithm 2, and

the cross products x_0y_1 and x_1y_0 (and the corresponding tags) are calculated using (unstable) field multiplication in $\mathbb{F}_2^8$. The cross products x_0y_1 and x_1y_0 are then added with the MAC tagged randomness (r, τ_r) using (unstable) field addition in $\mathbb{F}_2^8$. These partially refreshed cross products are then stored in registers to stop glitches after which the cross products x_iy_0 and x_iy_1 are added using a MAC-stable addition from Algorithm 1. Since it follows the DOM multiplier procedure, it is clear that the MAC-stable transformation remains correct. In addition, since we only replaced some multipliers and adders by their MAC-stable variants, the SNI-security follows from Theorem 7.

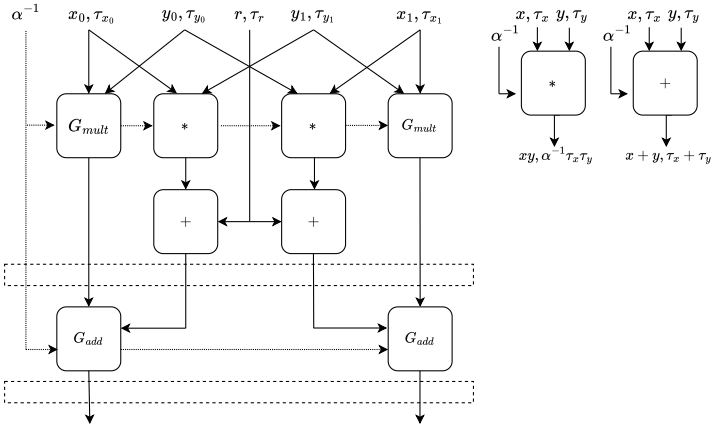


Fig. 3. Masked MAC-stable multiplication gadget. G_{add} and G_{mult} are the stable addition and multiplication gadgets (Algorithm 1 and Algorithm 2, respectively). The gadget $*$ consists of three (unstable) field multiplications in $\mathbb{F}_2^8$, and the gadget $+$ consists of two (unstable) field additions in $\mathbb{F}_2^8$.

We prove that the above masked multiplication is MAC-stable.

Theorem 8. *The masked multiplier depicted in Figure 3 is MAC-stable from register layer to register layer.*

Proof. We first show that any incorrect input codeword generates an incorrect state codeword (the state of the middle register layer of the multiplier). We distinguish two cases of the incorrect input.

– At least one of the values $(x_0, \tau_{x_0}), (x_1, \tau_{x_1}), (y_0, \tau_{y_0}), (y_1, \tau_{y_1})$ is an incorrect codeword which may also result from a fault in α^{-1}. Due to the stable multiplication between (x_0, τ_{x_0}) and (y_0, τ_{y_0}) or the one between (x_1, τ_{x_1}) and (y_1, τ_{y_1}), an incorrect input codeword is mapped to an incorrect state codeword in the middle register layer.

– Only (r, τ_r) is an incorrect codeword. Then, due to the addition giving an incorrect output codeword when only one of the two input codewords is incorrect, the middle register layer has an incorrect state codeword.

The above two cases include all cases of incorrect input codewords (the case where both (r, τ_r) and the encoded input x, y contained errors is still handled by the first case). This shows that the first phase of the multiplier is MAC-stable. The second phase where the values in the middle register layer are mapped to the output is also MAC stable since it consists only of two MAC stable additions. As a result, the masked multiplier is MAC stable. □

Finally, we note that it is also possible to use the transformed HPC1 [CGLS21], HPC2 [CGLS21], or HPC3 [KM22] PINI-secure gadgets (similarly, all SNI-secured gadgets as well) instead of the DOM multiplier.

6 Implementing a Masked MAC-Stable AES

In this section, we describe how to construct more complex circuits using gadgets within the StaMAC framework. Specifically, we detail the hardware implementation of AES based on the MAC-stable domain-oriented masking, as detailed in Section 5.3. We first elaborate on its side-channel and fault security, particularly compared to M&M, for which we detail the countermeasure and the attack against it. Additionally, we discuss the fault securities of CAPA and the λ-detection M&M countermeasure [HMA+24], which is proposed against the attack targeting M&M. Finally, we present the hardware cost of our implementation in comparison to these countermeasures, as well as StaTI.

6.1 MAC-Stable AES S-box

The AES S-box consists of an inversion in $\mathbb{F}_{2^8}$ where zero is mapped to zero, followed by an affine transformation. We first evaluate the inversion. This operation is equivalent to $x \mapsto x^{254}$ in $\mathbb{F}_{2^8}$ for which we use the multiplication chain described in [GPS14] and depicted in Figure 4.

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

The affine transformation $A(x) = L(x) + c$ following the inversion is performed over $\mathbb{F}_2$. This affine transformation is implemented using the MAC-stable gadget described in Algorithm 3. We note that the constant addition is performed by adding the constant to the first share of the state together with its MAC tag which was precomputed during the encoding phase of the AES cipher.

The computation of the multiplication chain relies on three distinct gadgets. Below, we outline these gadgets that work over the shared data encoded using MAC tags.

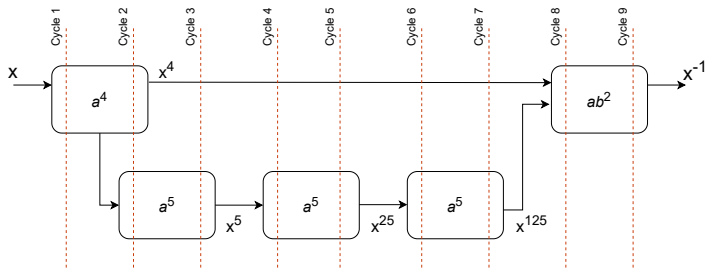


Fig. 4. Pipelined multiplication chain for $\mathbb{F}_{2^8}$ inversion.

Gadget 1: $x \mapsto x^4$. The first gadget is a linear gadget that computes the fourth power of its input. It can be calculated without register layers, as shown in Algorithm 3:

$$(x_0, x_1) \mapsto (x_0^4, x_1^4), \qquad (\tau_{x_0}, \tau_{x_1}) \mapsto (\alpha^{-3} \cdot \tau_{x_0}^4, \alpha^{-3} \cdot \tau_{x_1}^4)\,.$$

Since the above operation is applied share-wise and is invertible, it is glitch-extended probing secure and MAC-stable. While this operation is not SNI secure, it is Non-Interferent (NI) which can securely sequentially compose with an SNI gadget. The output of this gadget is typically refreshed.

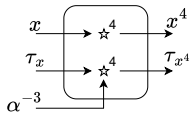
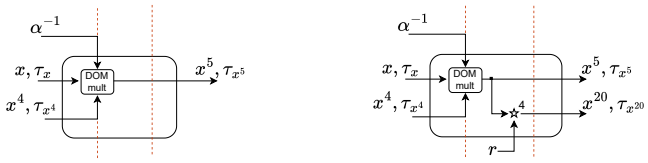


Fig. 5. MAC-stable gadget implementing $x \mapsto x^4$.

Gadget 2: $x \mapsto x^5$. The second gadget computes a MAC-stable DOM multiplication of the refreshed output and initial input of Gadget 1 as depicted in Figure 6a. This gadget requires two register stages and it is SNI because the transformation $x \mapsto x^4$ is linear, followed by the computation of the SNI secure DOM multiplier.

The calculation of the multiplication chain is optimized by combining the second register stage of the DOM multiplier with the computation and storage of the refreshed output of Gadget 1, precomputed for use in the subsequent Gadget 2, as shown in Figure 6b. This approach is especially beneficial for cycles 2, 3, and 4, 5, where the inputs x and x^4 are needed for the subsequent gadgets.



(a) Used in cycles 6,7 in Figure 4. (b) Used in cycles 2-5 in Figure 4.

Fig. 6. MAC-stable gadget implementing $x \mapsto x^5$ using the MAC-stable multiplication from Figure 3 on the input x and on the refreshed and registered x^4.

Gadget 3: $(x, y) \mapsto x \cdot y^2$. The third gadget used in cycles 8 and 9 in Figure 4, computes a MAC-stable DOM multiplication of the first input x, and the square of the second input y over two register layers. The second input is squared using a similar approach described in Gadget 1 (*i.e.,* $x \mapsto x^2$, $\tau_x \mapsto \alpha^{-1}\tau_x^2$), and is followed by a MAC-stable DOM multiplication.

6.2 MAC-Stable AES Architecture

We describe our AES architecture, utilizing a single MAC key α, which is similar to the architecture used in the M&M countermeasure (Galois field inversion version 1). The precomputation involves the calculation of α^{-1}, α^{-3}, and the $M_\alpha L M_\alpha^{-1}$ matrix. Overall, the implementation is represented by the byte-serialized architecture which includes the key-schedule as described by Groß *et al.* [GMK17]. Similar to M&M and CAPA, the control logic of our design is not protected against injected faults.

Regarding randomness[4], each of the four MAC-stable DOM multiplications, which are subroutines in Gadget 2 and 3, in the AES S-box require *one* byte of fresh randomness. Moreover, the gadgets used in cycles 2-5 (Figure 6b) require *one* byte of randomness to refresh after a *to-the-power four* operation. Similarly, the Gadget 2 used in cycles 2, 3 requires the refreshed output of the Gadget 1 which uses *one* byte of randomness. The refreshing is done in an additive manner where the tags are refreshed with the tag of the random value $(x \mapsto x + r), (\tau_x \mapsto \tau_x + (\alpha r))$. We reduce the fresh randomness cost approximately a factor of three compared to M&M's version 1 implementation, as we do not share the α key. Consequently, no randomness is required for the multiplications involving α. More specifically, we need only $(4 + 3) \cdot 8 = 56$ bits of randomness for an S-box.

Regarding latency, the inversion, as depicted in Figure 4, has a latency of 9 clock cycles, matching that of the M&M pipeline. However, minor differences exist between the pipelines. Namely, StaMAC avoids overhead since the affine transformation of the inverse does not require a shared multiplication, as α is

[4] For randomness generation, we use an unrolled Keccak implementation.

not shared. The overall latency of one round is $16 + C$ cycles, where C is the S-box latency. That is achieved by performing the MixColumns, ShiftRows, and AddRoundKey operations in parallel with the S-box. During the first 16 cycles, the state bytes are fed into the S-box's pipeline. The MixColumns operation runs in parallel every 4 cycles, while retrieving the S-box output requires C cycles. Meanwhile, the S-box can be used for the key schedule and ShiftRows is performed at the end.

6.3 Side-Channel Security

In this section, we discuss the SCA security of the MAC-stable AES implementation. For that purpose, we use PROLEAD, a probing-based leakage detection tool, introduced by Müller and Moradi [MM22]. This software is aligned with the adversarial models described in Section 2.1, making it a reliable tool for assessing our design. The evaluation was performed on a system equipped with Intel® Xeon® Gold 6244 Processor with 8 cores allowing 16 parallel threads and with 256 Gb of RAM available.

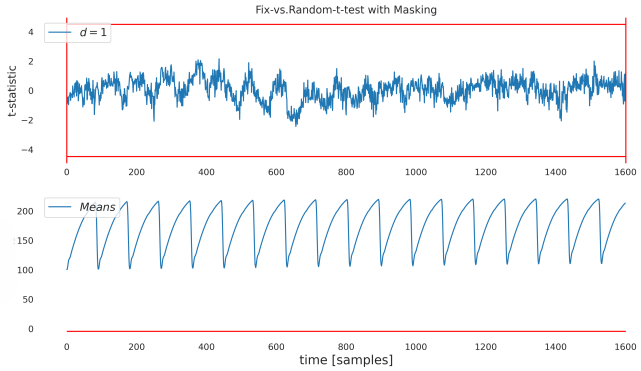
The evaluation employs G-test of independence and compares the encryption of two sets of plaintext inputs (typically, fixed versus randomly chosen) masked using specified number of shares. The software requires the design netlist, generated as described in Section 6.5, along with a cell library and a configuration file. The configuration file specifies the experiment settings, such as the number of simulations, the probing model extension (glitches only or transitions included), the order of the test, and related settings. We conducted 200 million experiments to assess the first-order security of our MAC-Stable AES S-box implementation under a glitch-extended probing model. The results reporting no leakage demonstrated its security against first-order attacks.

To complete the evaluation, we supplement PROLEAD tests with TVLA (Test Vector Leakage Assessment) [CDG+13]. This technique allows to detect a possible side-channel leakage without implementing actual attacks. We perform the assessment for StaMAC S-box by supplying it with fixed versus random inputs and running it for 18 clock cycles, which corresponds to two complete pipeline cycles. The S-box is placed on a Xilinx Spartan-6 FPGA located on a SAKURA-G evaluation board [SAK], whereas masks generation and precomputation logic are located on a separate control FPGA. The devices are supplied with a stable 6.144 MHz clock and an oscilloscope samples power consumption traces at a rate of 500MS/sec. The result of the first-order test is depicted in Figure 7.

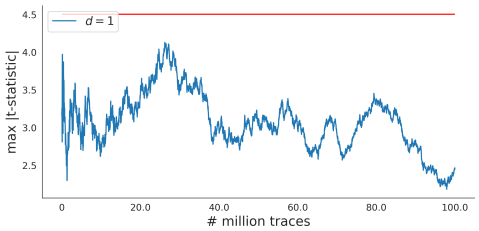
6.4 Fault Security

In this section, we discuss the fault security of the MAC-stable AES implementation.

To assess the MAC-stability of the proposed MAC-stable AES S-box implementation, we verified the stability of the simpler gadgets (*e.g.,* addition and multiplication) used in the implementation via a simple software tool. This tool



(a) Top: First-order t-test result, 100 million traces.
Bottom: Sample trace.



(b) Maximum absolute t-test value change.

Fig. 7. StaMAC S-box TVLA result, first-order security, 100 million traces, 2 runs in a row.

checks each incorrect faulty codeword and verifies that the output codeword is also incorrect. Then, the MAC-stability of the entire AES S-box follows Theorem 2. Following that, Theorem 6 proves the security of a glitch-extended probing secure masked gadget, which is fault non-complete and MAC-stable, against the gate/register faulting, the arbitrary additive register faulting, and the random register faulting adversaries. Therefore, the fault security of the entire AES S-box against these adversaries is established by its MAC-stability.

In addition to the stability verification, we performed an attack simulation on the C implementation of the MAC-stable AES S-box. We cover the arbitrary additive register faulting and the random register fault adversaries (targeting the register layer only) in the assessment of the MAC-stability of our design. Thus, we only verify the gate/register adversary in this fault simulation. Faults were modeled as XOR additions applied to variables in $\mathbb{F}_2^8$. The simulation involved extending the C implementation to inject faults by XORing a fault variable (ranging from 0 to 255) to selected critical variables (*e.g.,* inputs of the DOM multiplications) aligning with the gate/register faulting adversary. The results verified that all injected faults were propagated to the S-box output yielding an

incorrect output codeword regardless of the S-box input, showing the resistance of StaMAC against SIFA-like attacks.

Although translating clock glitching to the theoretical fault models is not trivial, we note that the zero-value attack proposed against M&M (aligning with the adversaries we consider) is not feasible in our architecture implemented using the StaMAC framework. This is due to the MAC-stability of the gadgets that compose the S-box. As a result, any fault appearing in an input of the multiplication involving the S-box input is propagated to the output (the advantages of the adversaries were discussed earlier). Additionally, the countermeasure, λ-detection M&M, proposed against this attack is specifically designed against clock glitching and does not provide security against a gate/register-faulting adversary. For instance, with a high precision fault injection setup, it is possible for the ineffectiveness of a fault injected to the inversion in $\mathbb{F}_2^2$ (in tower field construction) to depend on the other multiplication input in Stage 4, which would indicate that the S-box input is zero. As a result, only StaMAC and CAPA remain secure against first-order gate/register faulting adversaries (SIFA-like attacks).

Regarding CAPABARA, the attack does not apply to StaMAC as combined attacks are not accounted for in its security model. We note that the combined attack CAPABARA can also be performed by injecting two faults: one fault in the preprocessing stage as in CAPA, then using a second fault to probe the unmasked variable (setting the variable to a certain value and checking if the circuit aborts). Therefore, this attack renders CAPA comparable to StaMAC with its security limited to only fault attacks with high hardware cost as discussed in Section 6.5.

6.5 Hardware Benchmarks

In this section, we evaluate the hardware benchmarks of our MAC-stable AES implementation compared with the AES implementations protected with CAPA, M&M, and λ-detection M&M countermeasures. We use the Synopsys Design Compiler (version S-2021.06-SP3) together with the open source NANGATE 45nm library. The hierarchy is preserved during evaluation by enabling the *set_dont_touch* constraint and the compile option *exact_map* is used to avoid optimizations. We note that the precomputation costs are considered for the cost comparison of the full AES designs, whereas the generation of fresh masks is not included.

Table 3 compares the AES implementations protected with the listed countermeasures in terms of their latency, fresh randomness and area costs, and overhead relative to SCA-only AES implementations.[5] We additionally include StaTI in the comparison, focusing primarily on the overhead factor, as only a Keccak [BDPA13] implementation is available. We note that, since the StaMAC AES S-box implementation employs multiplication chain inversion, our design is

[5] We thank the authors of [RMB+18], [MAN+19] and [DOT24] for providing code of their designs for further comparison.

more comparable to the CAPA and M&M V1 countermeasures, as they utilize similar pipelines. The remaining AES S-box implementations utilize the tower-field decomposition approach [Can05].

For the SCA-only designs, we refer to the work of De Cnudde *et al.* [CRB+16] for comparison with M&M V2 and λ-detection AES S-box implementations. CAPA, M&M V1, and StaMAC designs are compared to the SCA-only multiplication chain that we implemented ourselves. The linear layers of SCA-only AES implementation are based on a serialized architecture from the work of Groß *et al.* [GMK16]. Keccak-f[1600] StaTI implementations are compared to their respective 2-share and 4-share SCA-only implementations, which we designed based on the works [SM21] and [BDN+13].

Table 3. Hardware cost comparison of various designs.

Block	Design	Latency (cycles)	Fresh Rand. (bits/cycle)	Area (kGE)	Overhead (factor)
AES	CAPA[1*]	226	96	122.4[&]	10.94
	M&M V1[2*]	266	160	35.1[&]	3.14
	M&M V2[2*]	236	116	23.7[&]	3.12
	λ-detection M&M[3**]	244	564	44.0	3.49
	This work*[*]	266	56	42.9[&]	3.83
Keccak-f	StaTI[4*]	72	0	285.4	2.25
		24	0	267.8	2

[*] first-order security, [**] second-order security,
[&] with precomputation cost, [1] [RMB+18], [2] [MAN+19] , [3] [HMA+24], [4] [DOT24]

Table 3 shows that M&M (both versions) and λ-detection M&M offer lower area costs compared to StaMAC, while StaMAC achieves lower randomness cost. However, as discussed in Section 6.4, M&M does not protect against ineffective faults, and although λ-detection M&M protects against ineffective faults caused by clock glitches, it does not protect against a single gate/register-faulting adversary. Moreover, StaMAC achieves significantly lower area costs compared to CAPA.

Comparing the hardware costs of StaMAC and StaTI is challenging since StaTI only has a Keccak implementation available. Even in terms of overhead, we do not achieve a fair overhead, as the structure of the Keccak S-box is inherently stable for 4-share (or nearly stable for 2-share), the overhead is comparable to simple duplication.

7 Conclusions and Future Work

In this paper, we presented a framework to create masked and MACed gadgets that are arbitrary composable secure against a probing adversary (x)or a faulting adversary. The framework includes proofs of arbitrary composability, security proofs including advantage bounds of extensive fault adversaries, the provision of stable gadgets of MACed data, and a generic transformation of SNI or PINI secure gadgets to MAC-stable ones which is worked out on a domain-oriented masked multiplier. Finally, we provide a hardware implementation of our framework applied to AES where we provide efficiency numbers compared to the previous work utilizing MAC tags. StaMAC achieves competitive security and efficiency relative to M&M, λ-detection M&M, and CAPA.

While our work tackled the theoretical aspects of physical security using MAC tags, there are practical improvements which are left as future work. The MAC-stable AES implementation is heavier in area versus the (insecure) solutions from the M&M countermeasure. While efficiency and area optimization was not the goal of the work, we do believe the efficiency of the masked and MACed S-box can be made much lighter since the current area overhead originates from the heavier MAC-stable addition (Alg. 1) and multiplication (Alg. 2). Finding lighter MAC-stable gadgets results in a plug-and-play reduction to a more efficient AES implementation.

This work only tackles the injection of faults at one time-frame and a non-combined setting of probes and faults. To tackle faults in multiple time-frames, a stronger notion than stability, or intermediate is needed as one fault can be used to make a function non-stable and the second fault can be used for the injection of a SIFA-like attack. Similarly, it is unclear how to model two or more faults at two different time frames such that the output's correctness is guaranteed without any intermediate error-checks and abort signals. From that same view, it is unclear how to model the security of an adversary using both a probe and a fault when these are injected at two different time-frames when no additional error checks/correction are used. For example, a probe can be used before a fault to read α and improve the success probability of the fault, which would require α to be shared, conversely, a fault can be used to remove randomness in the computation leaving the operations vulnerable to a probe, which would require intermediate error checks to mitigate. These challenges are left as future work to expand the work.

Acknowledgements. This work was supported by CyberSecurity Research Flanders with reference number VR20192203, and partially by FWO with reference number VS07724N.

References

AK97. Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas,

and Michael Roe, editors, *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997.

AMR+20. Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. Impeccable circuits. *IEEE Trans. Computers*, 69(3):361–376, 2020.

BBD+16. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

BDL97. Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

BDN+13. Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and first-order DPA resistant implementations of keccak. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2013.

BDPA13. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.

Can05. David Canright. A very compact s-box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

CDG+13. Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Pankaj Rohatgi, et al. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, volume 20, 2013.

CGLS21. Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

CRB+16. Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

CS20. Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

DDE+20. Joan Daemen, Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Florian Mendel, and Robert Primas. Protecting against statistical ineffective fault attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):508–543, 2020.

DDRT12. Amine Dehbaoui, Jean-Max Dutertre, Bruno Robisson, and Assia Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In Guido Bertoni and Benedikt Gierlichs, editors, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, pages 7–15. IEEE Computer Society, 2012.

DEG+18. Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 315–342. Springer, 2018.

DEK+18. Christoph Dobraunig, Maria Eichlseder, Thomas Korak, Stefan Mangard, Florian Mendel, and Robert Primas. SIFA: exploiting ineffective fault inductions on symmetric cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):547–572, 2018.

DN23. Siemen Dhooghe and Svetla Nikova. The random fault model. In *SAC*, volume 14201 of *Lecture Notes in Computer Science*, pages 191–212. Springer, 2023.

DOT24. Siemen Dhooghe, Artemii Ovchinnikov, and Dilara Toprakhisar. Stati: Protecting against fault attacks using stable threshold implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):229–263, 2024.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

FGP+18. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

GMK16. Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Pro-*

ceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016, page 3. ACM, 2016.

GMK17. Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.

GMO01. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

GPK+21. Michael Gruber, Matthias Probst, Patrick Karl, Thomas Schamberger, Lars Tebelmann, Michael Tempelmeier, and Georg Sigl. Domrep-an orthogonal countermeasure for arbitrary order side-channel and fault attack protection. *IEEE Trans. Inf. Forensics Secur.*, 16:4321–4335, 2021.

GPS14. Vincent Grosso, Emmanuel Prouff, and François-Xavier Standaert. Efficient masked s-boxes processing - A step forward -. In David Pointcheval and Damien Vergnaud, editors, *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, volume 8469 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2014.

Hab65. D. H. Habing. The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions on Nuclear Science*, 12(5):91–100, 1965.

HMA+24. Haruka Hirata, Daiki Miyahara, Victor Arribas, Yang Li, Noriyuki Miura, Svetla Nikova, and Kazuo Sakiyama. All you need is fault: Zero-value attacks on AES and a new λ-detection m&m. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):133–156, 2024.

IPSW06. Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David A. Wagner. Private circuits II: keeping secrets in tamperable circuits. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 308–327. Springer, 2006.

ISW03. Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

KM22. David Knichel and Amir Moradi. Low-latency hardware private circuits. In *CCS*, pages 1799–1812. ACM, 2022.

Koc96. Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

MAN+19. Lauren De Meyer, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. M&m: Masks and macs against physical attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):25–50, 2019.

MM22. Nicolai Müller and Amir Moradi. PROLEAD A probing-based hardware leakage detection tool. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):311–348, 2022.

NRR06. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

PBGS24. Matthias Probst, Manuel Brosch, Michael Gruber, and Georg Sigl. DOM-REP II. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2024, Tysons Corner, VA, USA, May 6-9, 2024*, pages 112–121. IEEE, 2024.

RBN+15. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

RMB+18. Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel P. Smart. CAPA: the spirit of beaver against physical attacks. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 121–151. Springer, 2018.

RSM21. Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, and Amir Moradi. Impeccable circuits III. In *IEEE International Test Conference, ITC 2021, Anaheim, CA, USA, October 10-15, 2021*, pages 163–169. IEEE, 2021.

SAK. SAKURA. Side-channel Attack User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html.

SJR+19. Sayandeep Saha, Dirmanto Jap, Debapriya Basu Roy, Avik Chakraborti, Shivam Bhasin, and Debdeep Mukhopadhyay. Transform-and-encode: A countermeasure framework for statistical ineffective fault attacks on block ciphers. *IACR Cryptol. ePrint Arch.*, page 545, 2019.

SM21. Aein Rezaei Shahmirzadi and Amir Moradi. Second-order SCA security with almost no fresh randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):708–755, 2021.

SMG16. Tobias Schneider, Amir Moradi, and Tim Güneysu. Parti - towards combined hardware countermeasures against side-channel and fault-injection attacks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference,*

Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II, volume 9815 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2016.

SRM20. Aein Rezaei Shahmirzadi, Shahram Rasoolzadeh, and Amir Moradi. Impeccable circuits II. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020.

TNN24a. Dilara Toprakhisar, Svetla Nikova, and Ventzislav Nikov. CAPABARA: A combined attack on CAPA. In Romain Wacquez and Naofumi Homma, editors, *Constructive Side-Channel Analysis and Secure Design - 15th International Workshop, COSADE 2024, Gardanne, France, April 9-10, 2024, Proceedings*, volume 14595 of *Lecture Notes in Computer Science*, pages 76–89. Springer, 2024.

TNN24b. Dilara Toprakhisar, Svetla Nikova, and Ventzislav Nikov. Sok: Parameterization of fault adversary models connecting theory and practice. In Elisabeth Oswald, editor, *Topics in Cryptology - CT-RSA 2024 - Cryptographers' Track at the RSA Conference 2024, San Francisco, CA, USA, May 6-9, 2024, Proceedings*, volume 14643 of *Lecture Notes in Computer Science*, pages 433–459. Springer, 2024.