M&M: Masks and Macs against Physical Attacks

Lauren De Meyer[1], Victor Arribas[1],
Svetla Nikova[1], Ventzislav Nikov[2] and Vincent Rijmen[1]

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

Abstract. Cryptographic implementations on embedded systems need to be protected against physical attacks. Today, this means that apart from incorporating countermeasures against side-channel analysis, implementations must also withstand fault attacks and combined attacks. Recent proposals in this area have shown that there is a big tradeoff between the implementation cost and the strength of the adversary model. In this work, we introduce a new combined countermeasure M&M that combines Masking with information-theoretic MAC tags and infective computation. It works in a stronger adversary model than the existing scheme ParTI, yet is a lot less costly to implement than the provably secure MPC-based scheme CAPA. We demonstrate M&M with a SCA- and DFA-secure implementation of the AES block cipher. We evaluate the side-channel leakage of the second-order secure design with a non-specific t-test and use simulation to validate the fault resistance.

Keywords: SCA, DFA, combined, countermeasure, masking, CAPA, ParTI, embedded, infective computation

1 Introduction

The implementation of cryptographic algorithms in embedded systems should be done with extreme care. Physical attacks are proliferating considerably and they are becoming easier and cheaper to perform. The most important physical attacks are Side-Channel Analysis (SCA), a non-invasive attack that exploits the physical leakages emanating from the device (power consumption or electromagnetic radiation among others) and Fault Attacks (FA), in which an adversary induces and exploits logical errors in the computation. These attacks are commonly used to retrieve secret data from the embedded device and can be executed either separately or combined. The most threatening attacks are differential power analysis (DPA) [KJJ99] for SCA and differential fault analysis (DFA) [BS97] and fault sensitivity analysis (FSA) [LSG+10] for FA.

In the case of SCA, a popular and established countermeasure is masking [ISW03, NRR06, PR11, NRS11, BGN+14, RBN+15, GMK16, GM17], a secret sharing-based method in which intermediate variables are stochastically split into multiple shares in order to make the side-channel-leaked information independent of sensitive data. To protect against fault injections there are two major countermeasures, as noted in [LRT12]: The first, *Detection*, checks whether the algorithm was faulted during the execution by using either area or time redundancy (*e.g.* duplication [BECN+06], concurrent error detection [BBK+03, KKG03, KKT04], . . .). The problem with duplication is that it does not provide security when faults are duplicated as well. Even with error-detecting codes, a powerful attacker can avoid detection if the injected faults result in valid codewords. The second approach, *Infection*, prevents an adversary from extracting secret information from a faulty ciphertext by ensuring that any induced fault results in a garbage output [GST12]. So far, all infective computations schemes have been broken [BG13].

The research direction of combined countermeasures - that is, countermeasures against both SCA and FA - is quite young and experimental. A popular methodology is to superpose two techniques that separately resist one family of attacks. Examples of schemes that combine masking against SCA with redundancy against FA are ParTI [SMG16] and Private Circuits II [IPSW06, CN16]. These countermeasures naturally inherit the drawbacks of redundancy, that is, they are vulnerable against the injection of *smart* undetectable faults. Moreover, implementing a checking mechanism that does not reveal sensitive information under combined attacks is a difficult task. More recently, an actively secure multi-party computation protocol was adapted to the context of embedded systems in order to provide security against combined attacks [RDB$^+$18]. The resulting combined countermeasure benefits from very strong formal security guarantees, but is extremely expensive to implement in hardware. A combination of duplication and infection is explored in [LRT12], but this scheme was broken in [BG13]. Infective computation is also combined with polynomial masking in [SFRES18]. These schemes alleviate the need for a checking mechanism, but as a result cannot give an honest user any indication on whether or not the chip has been tampered with.

Our Contribution In this work, we describe M&M, a new family of countermeasures that extends any SCA-secure masking scheme with information-theoretic MAC tags against DFA (*i.e.* **M**asks & **M**ACs) and combines them with an infective computation mechanism. By instantiating M&M with a d^{th}-order secure masking scheme, one achieves generic order of protection for SCA. The M&M construction then ensures generic order of protection against DFA and the combination of SCA and DFA. As opposed to error detecting codes, the MAC mapping is perfectly unpredictable, eliminating the possibility of *smart* undetectable faults. This makes M&M secure against stronger adversaries than when error detecting codes are used. We demonstrate M&M with first- and second-order secure implementations of the AES cipher. This example shows that M&M can be very efficient in area with an overhead factor of merely 2.53 compared to an implementation that protects only against SCA. We perform a SCA evaluation of our implementations where no leakage is found with up to 100 million traces. Additionally, we design and perform a fault evaluation to confirm our theoretically claimed fault coverage.

Scheme Overview We revisit the infective computation scheme of [LRT12], which uses a redundant encryption of the plaintext and uses the difference between the two ciphertexts to infect the output. That is, if the ciphertexts match, the output is exactly that ciphertext. If the ciphertexts do not match, the output is randomized so the attacker cannot get any information from it. The general idea is illustrated in Figure 1. For more details, we refer to the original work.

This scheme was broken in [BG13] because of a bias on the randomized output. We make two important changes. First, instead of using redundancy, which is vulnerable to the injection of identical faults, we replace the second instantiation of the cipher with a computation on information-theoretic MAC tags of the plaintext. If faults occur anywhere in the computation, the output of this block does not correspond to a valid MAC tag of the ciphertext with arbitrarily high probability. We use the difference between what the MAC tag should be and what it actually is, to randomize the ciphertext without any bias. This is illustrated in Figure 2. We also ensure that one can find out whether the ciphertext is correct or not. In a way, we thus combine the advantages of detection and infection.

The computation on masks and MACs resembles the approach of [RDB$^+$18]. However, instead of using expensive MPC machinery, we devise new constructions for generic field operations using existing SCA-secure gadgets.

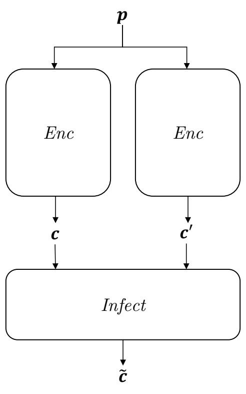

Figure 1: Infective Computation Scheme such as that of [LRT12]

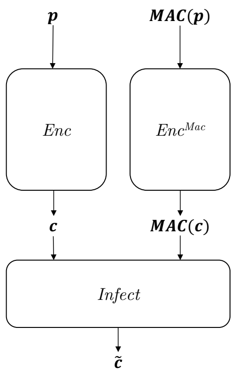

Figure 2: Our scheme

In Section 2, we introduce our adversarial model. Section 3 presents our framework of shared data and information-theoretic MAC tags and the basic M&M building blocks that are subsequently used in Section 4 to do more complex computations. In particular, we describe M&M blocks for elementary Galois Field operations, which can be used to construct the encryption blocks Enc and Enc^{Mac}. In Section 5 we describe how the shared ciphertext and MAC tags are used in an infective computation. This is followed by a discussion of the security in Section 6. Finally in Section 7, we demonstrate our scheme with an implementation and practical evaluation of the AES cipher.

2 Adversarial Model

In this work, we consider a semi-invasive adversary with probing and faulting capabilities. On the one hand, we work under the d-probing model introduced in [ISW03] for SCA, providing security against d^{th}-order side-channel analysis attacks under the independent leakage assumption. The model can include or exclude hardware glitches, but in this work, we specifically instantiate M&M considering glitches.

On the other hand, we consider two types of faults. We model faults as stochastic additive errors: this means the effect of a fault is the XOR of the current state with an error variable following some random distribution. This adversary model is very similar to the one described in [SMG16]. However, in this work, we do not limit the adversary in the number of bits he can alter, since we present a scheme which can tolerate *multiple* faults with *any Hamming weight.*

In addition, we allow the attacker to inject non-stochastic faults (for example very precise laser injections or stuck-at faults). In that case however, the faults must be restricted to affect at most d of the $d+1$ shares. We can justify this limitation by a proper placement of the circuit on the chip and the more complex setup of these kinds of faults.

3 M&M: The Basics

In this section, we describe the M&M framework and its most fundamental M&M building blocks, *i.e.* field multiplication and squaring in $\mathrm{GF}(2^k)$. We omit descriptions of trivial linear operations such as addition and scaling. Using these blocks, it is possible to secure any circuit against both SCA and DFA. Indeed, one only has to replace each AND-gate (resp. XOR-gate) with a M&M multiplication (resp. addition) in the field GF(2). We note that the proposed approach is only meant for the datapath. The control logic and public constants do not require a combined countermeasure as they are only vulnerable to FA.

3.1 The M&M Framework

Notation. $\boldsymbol{x}$ denotes a $d+1$-sharing $(x_0, \ldots, x_d)$ of an element $x \in \mathrm{GF}(2^k)$ such that $x = x_0 + \ldots + x_d$, with "+" denoting addition in the Galois Field $\mathrm{GF}(2^k)$. Additionally, "$\cdot$" is a field multiplication in $\mathrm{GF}(2^k)$ and "$\odot$" a shared (*i.e.* SCA protected) field multiplication in $\mathrm{GF}(2^k) : \boldsymbol{x} \odot \boldsymbol{y} = \boldsymbol{z} \Leftrightarrow x \cdot y = z$. Upright bold font is used for bit vectors $\mathbf{x} \in (\mathrm{GF}(2))^k$ and matrices $\mathbf{M} \in (\mathrm{GF}(2))^{k \times k}$.

Information-theoretic MAC tags. Detection of faults in the computation is achieved by accompanying each intermediate variable $x \in \mathrm{GF}(2^k)$ with an information-theoretic MAC tag. Let $\alpha \in \mathrm{GF}(2^k)$ denote a MAC key, which must be fresh for every encryption. For each $x \in \mathrm{GF}(2^k)$, we have a MAC tag $\tau^x = \alpha \cdot x$. Note that, if α were fixed and identical for all encryptions, the values and tags would be equivalent to an error detecting code. Security against faults is based on the fact that the MAC key α is secret. Without knowledge of α, an adversary cannot forge a valid tag $\tau^{\tilde{x}}$ for a faulty $\tilde{x}$. Its best strategy is guessing α, which offers a success probability of 2^{-k}. If the field $\mathrm{GF}(2^k)$ is too small, one can assign to each intermediate x multiple MAC tags $\tau^x[j]$, each for a different MAC key $\alpha[j]$ for $j = 1, \ldots, m$. The success probability of the adversary is then at most 2^{-km}. For readability, we will assume $m = 1$ unless otherwise mentioned.

Data representation against SCA and DFA. With security against both side-channel analysis and faults in mind, we port the information-theoretic MAC tags to the shared domain. This means that every intermediate $x \in \mathrm{GF}(2^k)$ is represented by $\langle \boldsymbol{x} \rangle = (\boldsymbol{x}, \boldsymbol{\tau^x})$ with value shares $\boldsymbol{x} = (x_0, \ldots, x_d)$ such that $x_0 + \ldots + x_d = x$ and tag shares $\boldsymbol{\tau^x} = (\tau_0^x, \ldots, \tau_d^x)$ such that $\tau_0^x + \ldots + \tau_d^x = \tau^x$. The MAC key itself is also shared: $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_d)$. Note that the MAC key α authenticates the sensitive value x itself and not merely its shares x_i. Hence, the tag shares τ_i^x are not tags of the value shares x_i but rather a share of the tag τ^x:

$$\tau_i^x \neq \alpha \cdot x_i$$
$$\sum_i \tau_i^x = \alpha \cdot \sum_i x_i$$

Shared multiplication. In what follows, we describe how M&M extends an existing SCA-secure masking scheme with protection against faults. Literature provides us with many secure Boolean masking schemes to choose from [ISW03, NRR06, PR11, NRS11, BGN+14, RBN+15, GMK16]. Each of those is defined by how it performs nonlinear operations (*i.e.* a multiplication) on Boolean shares. A specific instantiation of M&M thus depends on the choice of how to implement the shared multiplication operation $\boldsymbol{x} \odot \boldsymbol{y} = \boldsymbol{z}$. We assume that this operation transforms $d+1$-sharings of two variables x and y into a $d+1$-sharing of their product $z = xy$. The latency and randomness cost of this operation depends on the choice of SCA countermeasure. However, we assume for now that the latency is one clock cycle, since this is the case in most schemes. For further discussion on the latency of the multiplication gadgets, see Section 7.1.

3.2 Basic M&M Building Blocks

M&M Multiplication. Given two operands $\langle \boldsymbol{x} \rangle$ and $\langle \boldsymbol{y} \rangle$, we want to compute the value shares $\boldsymbol{z}$ and tag shares $\boldsymbol{\tau^z}$ of $z = xy$. The value shares $\boldsymbol{z}$ can naturally be obtained using a shared multiplier with $\boldsymbol{x}$ and $\boldsymbol{y}$ as inputs. From this point, for ease of notation, we use $\boldsymbol{xy}$ to denote a sharing of the product xy.

$$\boldsymbol{x} \odot \boldsymbol{y} = \boldsymbol{xy} = \boldsymbol{z}$$

Deploying the same multiplier for the tag shares does not result in a valid tag for z

$$\boldsymbol{\tau^x} \odot \boldsymbol{\tau^y} = \boldsymbol{\alpha x} \odot \boldsymbol{\alpha y} = \boldsymbol{\alpha^2 z} \neq \boldsymbol{\tau^z}$$

However, if we use the result above in another shared multiplication with a sharing of α^{-1}, we can obtain the correct tag shares:

$$\boldsymbol{\alpha^{-1}} \odot (\boldsymbol{\tau^x} \odot \boldsymbol{\tau^y}) = \boldsymbol{\alpha z} = \boldsymbol{\tau^z}$$

Note that we could also obtain these tag shares by either $\boldsymbol{x} \odot \boldsymbol{\tau^y}$ or $\boldsymbol{\tau^x} \odot \boldsymbol{y}$, but we want to avoid crossing the datapaths of value and tag shares such that faults introduced in the values cannot automatically propagate to the tags. Consider for example a fault injected on input $\boldsymbol{x}$, resulting in $\tilde{\boldsymbol{x}}$. Then $\tilde{\boldsymbol{x}} \odot \boldsymbol{\tau^y}$ is a valid tag for $\tilde{\boldsymbol{x}} \odot \boldsymbol{y}$.

The M&M multiplication is summarized in the left side of Figure 3. We assume the operation $\odot$ includes one register stage, since this is the case for most state-of-the-art d-secure multipliers. The value-datapath of a M&M multiplication thus requires one clock cycle whereas that of the tags requires two.

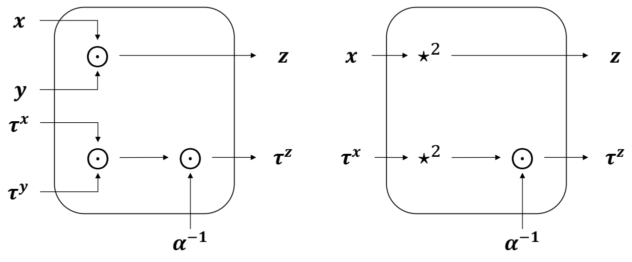

Figure 3: M&M nonlinear operations: Obtaining the value and tag shares of $z = xy$(left) and $z = x^2$(right)

This multiplication uses a sharing of the inverse of the MAC key $\boldsymbol{\alpha^{-1}}$. We assume this is made available together with the sharing of the MAC key itself. If this is not the case, $\boldsymbol{\alpha^{-1}}$ can be precomputed and stored.

M&M Squaring. Note that squaring in M&M follows the same procedure. That is, to obtain $\langle \boldsymbol{z} \rangle$ from $\langle \boldsymbol{x} \rangle$ such that $z = x^2$, we first square the value shares $\boldsymbol{x}$ and tag shares $\boldsymbol{\tau^x}$. Since a characteristic-two finite field allows $(a+b)^2 = a^2 + b^2$, squaring in the shared domain is a local operation that requires no registers: $\boldsymbol{x^2} = (x_0^2, \ldots, x_d^2)$ and $\boldsymbol{(\tau^x)^2} = ((\tau_0^x)^2, \ldots, (\tau_d^x)^2)$. We then again calculate the tag shares $\boldsymbol{\tau^z}$ by a shared multiplication with the inverse of the MAC key $\boldsymbol{\alpha^{-1}}$. The M&M squaring operation therefore takes one clock cycle and is depicted in the right side of Figure 3. The local squaring operation of shared data is depicted as $\star^2$. Extending this means that exponentiations by a power of two (x^{2^l}) take l clock cycles for the tags.

The Field GF(2). Note that in the case of bits (the field GF(2)), no correction of the MAC tag is needed since we then have that $\alpha^2 = \alpha$, *i.e.* $\boldsymbol{\tau^x} \odot \boldsymbol{\tau^y} = \boldsymbol{\tau^z}$. As a result, the M&M multiplication in GF(2) has the same latency as the SCA-secure multiplication and M&M squaring (as well as exponentiation by a power of two) is local.

4 Building circuits with M&M

In this section we describe how to use the above building blocks to construct circuits for more complex operations in the M&M framework. Specifically, we demonstrate the methodology for an inversion in $\mathrm{GF}(2^k), k > 2$[1], which is of course of particular interest for implementing the AES S-box. Furthermore, we introduce a method for processing an affine transformation over bits, as used as well in the AES S-box.

4.1 Galois Field Inversion

We discuss two methods to do an inversion in Galois Field $\mathrm{GF}(2^k)$. The first constructs a multiplication chain from the M&M multiplication and squaring blocks described above. This methodology is generic and can be applied to any S-box since any S-box can be presented as a polynomial over the considered field. The second version uses a SCA-secure inversion implementation from existing literature to build a secure M&M block. This approach is specific for the AES S-box, but results in a more efficient implementation.

4.1.1 Version 1: "Generic"

For $x \in \mathrm{GF}(2^8)$, the inversion x^{-1} is equivalent to the power map x^{254}. We can obtain this function via the following power chain [GPS14]:

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

Since $x^5 = (x^2)^2 \cdot x$, this inversion requires seven M&M squares and four M&M multiplications. Using the above squaring and multiplication blocks, obtaining the inversion output thus requires 15 clock cycles.

Optimization 1. The calculation can be sped up using a specialized block for the exponentiation to the power five, shown in Figure 4. This is also done in [GPS14] and justifies the choice of multiplication chain. In our case however, it is not trivial to do the same optimization for the tag calculation. The operation $\star^5$ raises the shares of x to the power five in one clock cycle. This requires a local computation of $\boldsymbol{x^4} = (x_0^4, \ldots, x_d^4)$, followed by a shared multiplication $\boldsymbol{x} \odot \boldsymbol{x^4} = \boldsymbol{x^5}$. This must be done with care since $\boldsymbol{x}$ and $\boldsymbol{x^4}$ are essentially the same variable and multiplying them may break non-completeness due to

[1]In GF(2) and $\mathrm{GF}(2^2)$, inversion is trivial as it corresponds respectively to the identity and squaring function

the dependencies among these two variables. We therefore precompute and refresh $\boldsymbol{x^4}$ one cycle before it is used.

After this first stage, which takes one clock cycle, we thus obtain a sharing of x^5 in the value datapath:

$$\begin{aligned}(x_0^4, \ldots, x_d^4) &= \boldsymbol{x^4} \\ \boldsymbol{x} \odot \boldsymbol{x^4} &= \boldsymbol{x^5}\end{aligned}$$

and a sharing of $(\tau^x)^5 = \alpha^5 x^5$ in the tag datapath:

$$\begin{aligned}((\tau_0^x)^4, \ldots, (\tau_d^x)^4) &= \boldsymbol{(\tau^x)^4} \\ \boldsymbol{\tau^x} \odot \boldsymbol{(\tau^x)^4} &= \boldsymbol{(\tau^x)^5} \neq \boldsymbol{\tau^{x^5}}\end{aligned}$$

A valid tag for x^5 is obtained through one more shared multiplication with $\boldsymbol{\alpha^{-4}}$, which is easily obtained locally from $\boldsymbol{\alpha^{-1}}$:

$$\boldsymbol{\alpha^{-4}} \odot (\boldsymbol{\tau^x} \odot \boldsymbol{(\tau^x)^4}) = \boldsymbol{\tau^{x^5}}$$

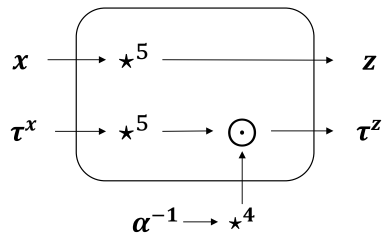

Figure 4: Obtaining the value and tag shares of $z = x^5$

With this specialized block, one obtains the exponentiation to the power five of $\langle \boldsymbol{x} \rangle$ in two clock cycles, if $(\boldsymbol{x^4}, \boldsymbol{(\tau^x)^4})$ is already refreshed beforehand. In those two clock cycles, we can obtain both the output $\langle \boldsymbol{z} \rangle$ and a refreshed $(\boldsymbol{z^4}, \boldsymbol{(\tau^z)^4})$ to be ready for the next block. The value shares can be refreshed using the second register stage (*i.e.* while the tag shares are being multiplied with α^{-4}). For the tag shares, there is no spare register stage for refreshing. In the shared multiplication with α^{-4}, we therefore raise each crossproduct to the power four. Before the register stage, we thus create a $(d+1)^2$-sharing of both τ^z and of $(\tau^z)^4$, each using its own randomness. As a result, the inversion result is available in only ten clock cycles:

- One cycle for the preparation of refreshed $\boldsymbol{x^4}$ and $\boldsymbol{(\tau^x)^4}$
- The calculation of $\langle \boldsymbol{x^{125}} \rangle = \langle ((\boldsymbol{x^5})^5)^5 \rangle$ requires six cycles.
- In the next cycle, we square $\langle \boldsymbol{x^{125}} \rangle$
- The last two cycles are spent on the multiplication of the result $\langle \boldsymbol{x^{250}} \rangle$ with $\langle \boldsymbol{x^4} \rangle$ to obtain $\langle \boldsymbol{x^{254}} \rangle$.

Optimization 2. By merging the last two operations into one step of two clock cycles, we reduce the total latency of the M&M inversion to nine cycles. This is possible because $x^{254} = f(x^4, x^{125})$ with $f(a,b) = a \cdot b^2$. We apply the same methodology as above. For the value shares:

$$\begin{aligned}(b_0^2, \ldots, b_d^2) &= \boldsymbol{b^2} \\ \boldsymbol{a} \odot \boldsymbol{b^2} &= \boldsymbol{f(a,b)}\end{aligned}$$

For the tag shares:

$$((\tau_0^b)^2, \dots, (\tau_d^b)^2) = (\boldsymbol{\tau}^{\boldsymbol{b}})^{\mathbf{2}}$$
$$\boldsymbol{\tau}^{\boldsymbol{a}} \odot (\boldsymbol{\tau}^{\boldsymbol{b}})^{\mathbf{2}} = \boldsymbol{\alpha}^{\mathbf{3}} \boldsymbol{f}(\boldsymbol{a}, \boldsymbol{b}) \neq \boldsymbol{\tau}^{\boldsymbol{f}(\boldsymbol{a},\boldsymbol{b})}$$
$$\boldsymbol{\alpha}^{\mathbf{-2}} \odot (\boldsymbol{\tau}^{\boldsymbol{a}} \odot (\boldsymbol{\tau}^{\boldsymbol{b}})^{\mathbf{2}}) = \boldsymbol{\tau}^{\boldsymbol{f}(\boldsymbol{a},\boldsymbol{b})}$$

Figure 5 summarizes the nine-stage pipeline that calculates the value and tag shares for an inversion of $\langle \boldsymbol{x} \rangle$.

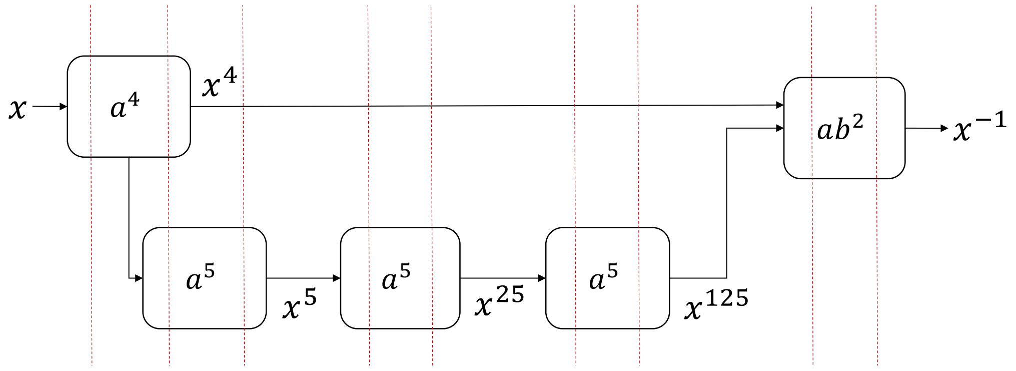

Figure 5: Inversion pipeline. (Register stages are depicted by red dotted lines.)

4.1.2 Version 2: "Custom"

The AES S-box has been extensively studied in literature and an abundance of SCA-protected implementations has already been proposed. When implementing the AES S-box in the M&M framework, it only makes sense to exploit the results from this research. We can take the above optimizations even further by merging all stages together into one. Consider applying a d^{th}-order secure shared inversion in GF(2^k) (denoted $\star^{-1}$) on both the value and tag shares. One obtains $\boldsymbol{x}^{\mathbf{-1}}$ and $(\boldsymbol{\tau}^{\boldsymbol{x}})^{\mathbf{-1}} = \boldsymbol{\alpha}^{\mathbf{-1}} \boldsymbol{x}^{\mathbf{-1}} \neq \boldsymbol{\tau}^{\boldsymbol{x}^{\mathbf{-1}}}$. Only a shared multiplication with $\boldsymbol{\alpha}^{\mathbf{2}}$ is required to calculate the correct tag shares of x^{-1}. This is illustrated in Figure 6. Again, it is easy to obtain $\boldsymbol{\alpha}^{\mathbf{2}}$ by locally squaring the shares of α.

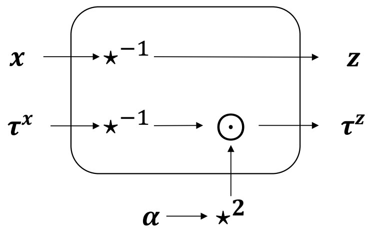

Figure 6: M&M inversion: Obtaining the value and tag shares of $z = x^{-1}$

4.2 Affine transformation over bits

The AES affine transformation at the end of the S-box $A(x) = L(x) + c$ is linear over GF(2). The linear part of the transform, $L(x)$ is a matrix multiplication operating on the bitvector of x. A sharing of $L(x)$ is trivially obtained by applying the transform locally to each share of x. The same cannot be said for the tag shares of x. In this section, we describe how to obtain the tag shares for any linear transform of this type.

Isomorphisms. Consider the isomorphism ϕ between the finite field $\mathrm{GF}(2^k)$ and the vector space $(\mathrm{GF}(2))^k$, that maps each element to its bitrepresentation vector, *i.e.* $\phi(2^i) = \mathbf{e}_i$ with $\mathbf{e}_i$ the i^{th} unit vector. We denote the bitvector of x as $\phi(x) = \mathbf{x}$. For the linear transform, we thus have

$$\phi(L(x)) = \mathbf{L}\phi(x) = \mathbf{L}\mathbf{x}$$

with $\mathbf{L} \in (\mathrm{GF}(2))^{k\times k}$ the matrix that defines the linear transformation. One of the consequences of this isomorphism is

$$\forall \alpha \in \mathrm{GF}(2^k), \exists \mathbf{M}_\alpha \in (\mathrm{GF}(2))^{k\times k} \text{ s.t. } \forall x \in \mathrm{GF}(2^k) : \phi(\alpha x) = \mathbf{M}_\alpha \phi(x) = \mathbf{M}_\alpha \mathbf{x}$$

Note that the opposite direction does not work: not for every $\mathbf{M} \in (\mathrm{GF}(2))^{k\times k}$ there exists an $\alpha \in \mathrm{GF}(2^k)$ such that this relation holds.

Given a value $x \in \mathrm{GF}(2^k)$ and a corresponding tag $\tau^x = \alpha x \in \mathrm{GF}(2^k)$, we wish to obtain a tag $\tau^{L(x)}$ satisfying $\tau^{L(x)} = \alpha L(x)$. We denote the bitvector of τ^x as $\phi(\tau^x) = \mathbf{t} = \mathbf{M}_\alpha \mathbf{x}$. We have

$$\begin{aligned}\phi(\tau^{L(x)}) = \phi(\alpha L(x)) &= \mathbf{M}_\alpha \phi(L(x)) \\ &= \mathbf{M}_\alpha \mathbf{L}\mathbf{x} \\ &= \mathbf{M}_\alpha \mathbf{L}(\mathbf{M}_{\alpha^{-1}}\mathbf{t}) \\ &= (\mathbf{M}_\alpha \mathbf{L} \mathbf{M}_{\alpha^{-1}})\mathbf{t}\end{aligned}$$

Hence, we can go from (x, τ^x) to $(L(x), \tau^{L(x)})$ by applying $\mathbf{L}$ to the bitvector of x and $\mathbf{M}_\alpha \mathbf{L} \mathbf{M}_{\alpha^{-1}}$ to the bitvector of τ^x. A similar approach is used in error detecting/correcting code schemes [BCC+14]. In our case however, the code depends on α and thus the matrix $\mathbf{M}_\alpha \mathbf{L} \mathbf{M}_{\alpha^{-1}}$ is secret and different in every encryption.

Calculating $\mathbf{M}_\alpha$. The matrix $\mathbf{M}_\alpha$ is straightforward to find. The i^{th} column of $\mathbf{M}_\alpha$ can be denoted by $\mathbf{M}_\alpha \mathbf{e}_i = \mathbf{M}_\alpha \phi(2^i) = \phi(\alpha 2^i)$. We therefore know that

$$\mathbf{M}_\alpha = \begin{bmatrix}\phi(128\alpha) & \phi(64\alpha) & \dots & \phi(2\alpha) & \phi(\alpha)\end{bmatrix}$$

By seeing a matrix product as a compact way to describe k matrix-vector products

$$\mathbf{AB} = \begin{bmatrix}\mathbf{A}\mathbf{b}_{k-1} & \mathbf{A}\mathbf{b}_{k-2} & \dots & \mathbf{A}\mathbf{b}_0\end{bmatrix}$$

we get that

$$\mathbf{L}\mathbf{M}_{\alpha^{-1}} = \begin{bmatrix}\phi(L(128\alpha^{-1})) & \phi(L(64\alpha^{-1})) & \dots & \phi(L(\alpha^{-1}))\end{bmatrix}$$

and thus

$$\mathbf{M}_\alpha \mathbf{L}\mathbf{M}_{\alpha^{-1}} = \begin{bmatrix}\phi(\alpha L(128\alpha^{-1})) & \phi(\alpha L(64\alpha^{-1})) & \dots & \phi(\alpha L(\alpha^{-1}))\end{bmatrix}$$

For each MAC key α, this matrix can be precomputed and stored in $d+1$ shares, similar to the precomputed sharing of α^{-1}. In the linear transformation, we obtain the tag shares $\boldsymbol{\tau}^{\boldsymbol{L(x)}}$ by a shared matrix-vector multiplication with the sharing of $\mathbf{M}_\alpha \mathbf{L}\mathbf{M}_{\alpha^{-1}}$. A shared matrix-vector multiplication can use the same equations as the SCA-secure multiplier $\odot$, but with one of the inputs a matrix and with the field multiplication '$\cdot$' replaced by matrix-vector products. Because of the register stage in the shared matrix-vector multiplication, the affine transformation requires one clock cycle.

Latency Optimization Note that in cases such as AES, where the affine transformation follows a power map, a little trick can ensure that the affine transformation does not increase the total latency of an S-box evaluation. The last clock cycle of the inversion in §4.1.1 (resp. §4.1.2) is spent on a shared multiplication of intermediate tag shares with $\boldsymbol{\alpha^{-2}}$ (resp. $\boldsymbol{\alpha^{2}}$). We can incorporate this tag correction in the affine transformation by replacing the matrix $\mathbf{M}_\alpha \mathbf{L}\mathbf{M}_{\alpha^{-1}}$ with $\mathbf{M}_\alpha \mathbf{L}\mathbf{M}_{\alpha^{-3}}$ (resp. $\mathbf{M}_\alpha \mathbf{L}\mathbf{M}_{\alpha}$).

5 Infective Computation

We have described above how to implement two encryption blocks: one that calculates ciphertext shares, given plaintext shares and another that calculates ciphertext tag shares from plaintext tag shares and MAC key shares α_i. We now consider the ciphertext block-per-block with blocksize k. Let $c_i \in \mathrm{GF}(2^k)$ be the shares of one ciphertext block and $\tau_i^c \in \mathrm{GF}(2^k)$ the shares of the corresponding tag block. If the tags are consistent with the data, then $\sum_i \tau_i^c = \alpha \cdot \sum_i c_i$.

5.1 The Problem with Error Checking

The use of infective computation in M&M is motivated by the difficulty of designing an error checking mechanism that is secure against combined attacks. Consider the following algorithm that computes the error on the tags $E = \tau^c + \alpha \cdot c$ and verifies that it is zero. We assume sufficient registers are in place to prevent glitch problems.

```
Check($\boldsymbol{c}, \boldsymbol{\tau^c}, \boldsymbol{\alpha}$)
  Let $\boldsymbol{\theta} \leftarrow \boldsymbol{\alpha} \odot \boldsymbol{c}$
  for all shares $i$ do
    Let $E_i \leftarrow \theta_i + \tau_i^c$
  end for
  $E = \sum_i E_i$
  Output $(E == 0)$
```

This checking algorithm is not secure in a combined adversary model with probing and faulting. When an attacker manages to insert a known fault Δ in one share of the shared multiplication such that the check is performed with $\alpha' = \alpha + \Delta$, the unshared (zero) error E is replaced by

$$\begin{aligned} E' &= \tau^c + \alpha' \cdot c \\ &= \tau^c + \alpha \cdot c + \Delta \cdot c \\ &= \Delta \cdot c \end{aligned}$$

A single probe on the unshared E' thus reveals the (faulty) ciphertext c. This attack defeats the very purpose of the error check, which is to stop a faulty ciphertext from being released to the adversary.

5.2 The Solution

We propose the following routine Infect, which is local except for one shared multiplication to obtain a sharing of the correct MAC tag $\boldsymbol{\alpha c}$. Let R be a uniformly random mask $\in \mathrm{GF}(2^k) \setminus \{0\}$. Each share of the output ciphertext block is modified using this random mask and the difference of the tags.

```
Infect ($\boldsymbol{c}, \boldsymbol{\tau^c}, \boldsymbol{\alpha}$)
  Let $\boldsymbol{\theta} \leftarrow \boldsymbol{\alpha} \odot \boldsymbol{c}$
  Draw $R \xleftarrow{\$} \mathrm{GF}(2^k) \setminus \{0\}$
  for all shares $i$ do
    Let $\tilde{c}_i \leftarrow c_i + R \cdot (\theta_i + \tau_i^c)$
  end for
  Output $\boldsymbol{\tilde{c}}$
```

The scheme outputs a sharing of the adapted ciphertext block $\tilde{c} = \sum_i \tilde{c}_i = c + R \cdot (\alpha \cdot c + \tau^c)$. Thus, if the tags are consistent $(\alpha \cdot c + \tau^c = 0)$, the scheme outputs a sharing of the computed block c. On the other hand, if the tags do not match ($\alpha \cdot c + \tau^c \neq 0$), the unshared output $\tilde{c}$ is random.

One may note that generating a nonzero mask R is nontrivial. However, there must be a PRNG with enough throughput to realize all the randomness for the computation of the S-box. The number of random bits available for the routine Infect is thus much higher than the amount required. From this, it is easy to generate one nonzero byte.

Unbiased Randomization We verify that the infected ciphertexts are uniformly distributed, so the attacker cannot obtain any information from them. Consider the case when the computation of *Enc* (and Enc^{Mac}) is disturbed by faults Δ^c (resp. Δ^τ), resulting in the unshared infected ciphertext block

$$\tilde{c} = c + \Delta^c + R \cdot (\alpha \cdot (c + \Delta^c) + \tau^c + \Delta^\tau)$$

We may assume Δ^c is non-zero and unkown to the attacker. In fact, knowing the faulty ciphertext, or indeed Δ^c, is the goal of the adversary in a DFA attack. Furthermore, we assume a strong probing adversary can know the value of the mask R, which is why we do not allow it to be zero. However, the MAC key α is always secret due to sharing. These introduced faults (Δ^c, Δ^τ) remain undetected with a probability of 2^{-km}, corresponding to the case when $\Delta^\tau = \alpha \cdot \Delta^c$. We therefore claim an error detection probability (EDP) of $1 - 2^{-km}$ and focus now on the case when $\Delta^\tau \neq \alpha \cdot \Delta^c$.

Using the fact that $\alpha \cdot c + \tau^c = 0$, we rewrite $\tilde{c} = c + \Delta^c \cdot (1 + R \cdot \alpha) + \Delta^\tau \cdot R$. Clearly, the unbiased randomization of the output depends on the uniformity of the mask $(1 + R \cdot \alpha)$. It can be verified that this mask is uniformly random in GF(2^k) when R is uniformly random in GF(2^k) $\setminus \{0\}$ and α uniformly random in GF(2^k). As a result, $\tilde{c}$ is uniformly random in GF(2^k) when $\Delta^c \neq 0$.

5.3 Combining Infection and Detection

In many applications, outputting garbage when the chip is under attack suffices. There are however some use cases, where one might want to know whether the outputted ciphertext is correct. If that is the case, we propose to do the infective computation twice, with different masks and with different MAC keys. The two resulting ciphertexts can be compared; If the computation was corrupted, the ciphertexts are distinct and randomized. Otherwise, they are identical.

The above adaption requires that the number of MAC keys m is at least two. If $m = 1$ suffices for security, we propose the following solution to avoid duplicating the tagsize just for the sake of outputting two ciphertexts. A second MAC key β is created only for the infective computation part and not for the cipher evaluation. The procedure Infect$_2$ below describes how to obtain the second ciphertext block $\boldsymbol{\tilde{c}'}$ apart from the original $\boldsymbol{\tilde{c}}$ obtained with Infect.

```
**Infect$_2$** $(\boldsymbol{c}, \boldsymbol{\tau^c}, \boldsymbol{\alpha^{-1}}, \boldsymbol{\beta}, \boldsymbol{R})$
  Let $\boldsymbol{\theta'} \leftarrow \boldsymbol{\beta} \odot \boldsymbol{c}$
  Let $\boldsymbol{\tau^{c\prime}} \leftarrow \boldsymbol{\alpha^{-1}} \odot (\boldsymbol{\beta} \odot \boldsymbol{\tau^c})$
  Draw $R' \xleftarrow{\$} \mathrm{GF}(2^k) \setminus \{0, R\}$
  **for all** shares $i$ **do**
    Let $\tilde{c}_i' \leftarrow c_i + R' \cdot ({\theta_i}' + \tau_i^{c\prime})$
  **end for**
  Output $\boldsymbol{\tilde{c}'}$
```

The two unshared outputs are thus

$$\tilde{c} = \sum_i \tilde{c}_i = c + R \cdot (\alpha \cdot c + \tau^c)$$
$$\tilde{c}' = \sum_i \tilde{c}'_i = c + R' \cdot (\beta \cdot c + \beta \cdot \alpha^{-1} \cdot \tau^c)$$

6 Security Analysis

In this section, we discuss the security of the M&M scheme. Note that M&M can be based on several different Boolean masking schemes providing SCA secure multiplication, inversion and refreshing gadgets in the chosen adversary model and thus the security of any instantiation depends heavily on those choices.

6.1 Security against SCA.

By adhering to security principles such as ensuring non-completeness [BGN+14] and proper refreshing is satisfied everywhere, M&M inherits the SCA security of the shared multiplication and inversion mechanism used. A Boolean masking scheme that is secure in the considered SCA attacker model thus provides security against d^{th}-order SCA. Since the model can include or exclude hardware glitches if the shared multiplication and inversion mechanism used are also secure in the presence of glitches, M&M inherits this.

The computation of the tag shares follows the same design principles as the value share calculations. The two datapaths operate completely independently of each other and receive their own distinct fresh randomness (see Figure 7). It is important to note that the input sharings $\boldsymbol{p}$ and $\boldsymbol{\tau^p}$ must be independent as well, which is easily achieved if the initial maskings of p and τ^p are obtained separately. The independence of the two datapaths ensure that their merging in the *Infect* block does not induce leakage on p or τ^p.

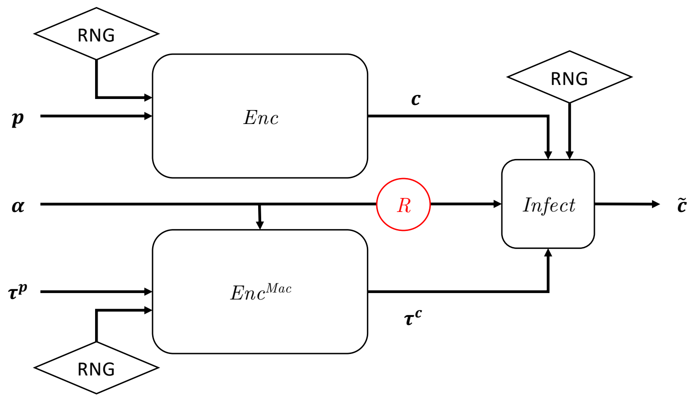

Figure 7: Overview of the scheme

The Refreshing Gadget. It is important that any refreshing mechanism used (cf. § 4.1.1) ensures the same security as is provided by the used masking scheme. The kind of refreshing thus depends on the targeted security order d [BBP+16] and the considered attacker model. In general, one can always make use of the multiplication-based refresh gadget of Ishai *et al.* [ISW03]. It has been shown in [BBD+16, Gadget 4b] that this refreshing ensures composability at any order. For a specific target security level, randomness can be consumed more efficiently. For example, the *ring refreshing* approach of [CRB+16] uses

$d+1$ fresh masks in a circular manner to refresh $d+1$ shares. This method suffices for second- and third-order security. At certain higher orders, one can use its variant, *offset refreshing*, which still uses only $d+1$ units of fresh randomness but rotates with an offset of more than 1 [BBD+18, Alg. 2]. Finally, *additive refreshing* using only d fresh masks is sufficient when first-order security is targeted. For a more detailed treatment of refreshing gadgets, we refer to [BBD+18].

6.2 Security against FA.

The introduction of MAC tags to the circuit results in resistance against FA, in which faults are not limited in Hamming weight nor in quantity. Faults are only undetected if both the value and MAC tag shares are modified in such a way that the relation

$$\sum_i \tau_i^x = \alpha \cdot \sum_i x_i \tag{1}$$

remains true. Since the MAC key $\alpha \in \mathrm{GF}(2^{km})$ is unkown to the adversary, any number of stochastic additive errors result in this relation with probability at most 2^{-km}. If the attacker has the ability to inject non-stochastic faults, our model restricts these to affect at most d of the $d+1$ shares. In that case, the success probability still depends on the probability of guessing the secret MAC key $\alpha \in \mathrm{GF}(2^{km})$ correctly. We therefore claim an error detection probability (EDP) of $1-2^{-km}$.

As stated in the adversarial model in §2, the faults we consider are neither limited in Hamming weight nor in quantity. The worst-case probability that (1) is satisfied is 2^{-km} regardless of the Hamming weight of a single fault. The accumulated effect of multiple faults also does not change this probability. The adversary obtains no additional information after injecting faults hence random shooting or guessing α remains the best strategy for subsequent faults. In the end, the same equation (1) in $\mathrm{GF}(2^{km})$ must hold for the faults to remain undetected. It holds with probability 2^{-km}. We experimentally investigate the effect of multiple faults in Section 7.3.2.

The zero MAC key. The event that the MAC key is zero occurs with probability 2^{-km}. Since α is secret and shared, the adversary cannot know when the tags are zero and must still guess α to determine what fault to inject in the tag computation. An adversary strategy of not injecting faults in the tag computation, *i.e.* injecting faults only on the value computations, corresponds to guessing that $\alpha = 0$ and succeeds with probability 2^{-km}. This is completely analogous to guessing for example that $\alpha = 1$ and injecting identical faults in both the tag and value computation accordingly. Either by guessing α or by injecting a random fault, the adversary hits the correct value with probability 2^{-km}, corresponding to our claimed EDP of $1-2^{-km}$. Hence, in theory, the case $\alpha = 0$ is equivalent to any other nonzero MAC key. In practice however, the strategy corresponding to guessing $\alpha = 0$ is easier since it requires only fault injections in the value datapath and not the tag datapath. To avoid it, one could exclude the zero MAC key and reduce the EDP to $1-(2^{km}-1)^{-1}$. This difference is negligible if km is sufficiently large. However, note that in that case, the infective computation output phase can no longer be used, since it requires α to be uniformly random in $\mathrm{GF}(2^{km})$.

Ineffective Faults. Apart from DFA and FSA, there is also an interesting branch of fault attacks that exploits so-called *ineffective faults*. For example, a stuck-at-zero fault on a wire or set of wires is *ineffective* when those wires already carry the zero value. This type of faults are naturally undetectable at algorithm level, which makes them immune to both detection and infection countermeasures. A flavour of Ineffective Fault Analysis (IFA) [Cla07] called Statistical Ineffective Fault Analysis (SIFA) [DEK+18] has recently been proposed. SIFA collects a subset of correct ciphertexts from a large number of faulted encryptions and exploits the fact that the intermediate state of the algorithm is not uniformly distributed in this subset. This attack has been extended to masked implementations in [DEG+18]. Ineffective faults (and thus SIFA) fall outside of our adversary model since they are impossible to detect. Protection against such attack can be provided at a different level, for example using a protocol that erases the key as soon as a certain threshold of faulty ciphertexts has been detected.

6.3 Security against combined attacks.

Having brought the MAC tags to the shared domain, our scheme also provides security against combined attacks. Thanks to the fact that the MAC key α is shared, it remains secret even to a probing adversary. As a result, the detection probability of injected faults does not change, even when the adversary combines them with SCA. We recall that our model limits the injection of deterministic faults to d of the $d+1$ shares. In case of a combined attack, the total number of affected shares by either faults or probes should thus still not exceed d.

Modern combined attacks such as PACA [AVFM07, CFGR10], which require a faulty ciphertext to succeed, are prevented since faulty ciphertexts are only released with probability 2^{-km}. The effective complexity of such attacks thus increases by a factor at least 2^{km}. Thanks to the infective computation, M&M is also secure against combined attacks that target the checking mechanism or that exploit correlations on the faulty ciphertext [RLK11, DV12].

Although we are not aware of any combined attack against M&M, we cannot formally prove it as CAPA [RDB+18] does. The only provable approach against combined attacks known so far is to adapt an actively secure MPC protocol. No other formal techniques have yet been found.

7 AES Case Study

M&M has been designed in a way that allows it to use both provably secure (e.g. SNI [BBD+15]) gadgets or more efficient but non-provably secure blocks. We now present specific AES implementations confirming the latter. We target first- and second-order security as those are relevant for realistic attacks and use TVLA for these specific orders to demonstrate the security.

Our implementations are mere examples of the many different ways to instantiate M&M. Any existing or future SCA-secure gadgets can be used as the underlying building blocks. In this section, we first detail our choice of gadgets and investigate the implementation cost of the resulting AES constructions. We then empirically validate our SCA claims using univariate and bivariate test vector leakage assessment and we perform a simulation-based verification of the DFA security.

7.1 Implementation Details

In Section 4, we described essentially all components that are needed to construct an AES SubBytes implementation: the inversion in $GF(2^8)$ and the affine transformation over bits. We distinguish two versions of the S-box implementation. One follows a rather generic methodology using multiplication chains and the other is customized for the inversion. We investigate the implementation cost of both implementations for first- and second-order SCA security ($d = 1$ or 2). For our specific instantiations of M&M, we only claim first- and second-order security. Nevertheless, it is extendable to higher-orders given a suitable choice of building blocks.

The remaining AES blocks (ShiftRows, MixColumns and AddRoundKey) consist exclusively of linear operations and are thus trivially implemented for the M&M framework. More specifically, the datapaths in both the *Enc* and Enc^{Mac} blocks each consist of $d+1$ copies of the AES state and key arrays. These arrays contain data shares in the encryption block *Enc* and tag shares in case of the Enc^{Mac} block. In total, the area cost of these blocks increases with a factor $(d+1)(m+1)$ compared to an unprotected implementation. Our implementation uses the same byte-serialized architecture as used in [GMK17].

We now detail our choice of multiplication and refresh gadgets which are used in the multiplication chain version of the inversion (cf. version 1, Figure 5) and the inversion gadget used in version 2 (cf. Figure 6).

Multiplication gadgets. We choose a $d+1$-share multiplier as our SCA-secure multiplier. The following equations describe for example a three-share, second-order ($d=2$) secure multiplication of shared variables $\boldsymbol{x} = (x_0, x_1, x_2)$ and $\boldsymbol{y} = (y_0, y_1, y_2)$ into $\boldsymbol{z} = (z_0, z_1, z_2)$, using three units of randomness $r_0, r_1, r_2 \in \mathrm{GF}(2^k)$ as in [GMK16]. We first calculate nine intermediate values t_{ij} for $i, j \in \{0, 1, 2\}$. After a register stage for synchronization of t_{ij}, we compute the output shares $\boldsymbol{z} = (z_0, z_1, z_2)$. The latency of the operation $\odot$ is thus 1 clock cycle.

$$\begin{aligned}
t_{00} &= x_0 \cdot y_0 \\
t_{01} &= x_0 \cdot y_1 + r_0 \\
t_{02} &= x_0 \cdot y_2 + r_1
\end{aligned}$$

$$\begin{aligned}
t_{10} &= x_1 \cdot y_0 + r_0 & \qquad z_0 &= [t_{00}]_{reg} + [t_{01}]_{reg} + [t_{02}]_{reg} \\
t_{11} &= x_1 \cdot y_1 & \qquad z_1 &= [t_{10}]_{reg} + [t_{11}]_{reg} + [t_{12}]_{reg} \\
t_{12} &= x_1 \cdot y_2 + r_2 & \qquad z_2 &= [t_{20}]_{reg} + [t_{21}]_{reg} + [t_{22}]_{reg}
\end{aligned}$$

$$\begin{aligned}
t_{20} &= x_2 \cdot y_0 + r_1 \\
t_{21} &= x_2 \cdot y_1 + r_2 \\
t_{22} &= x_2 \cdot y_2
\end{aligned}$$

The corresponding first-order construction is given in [GMK16]. Each multiplication requires $\binom{d+1}{2}$ fresh units of randomness. It has been shown in [FGMDP⁺18] that such a multiplication gadget is composable if the result is stored in a register. As shown on Figure 5, we present in this work a construction with such registers in the value share datapath but *without* extra registers in the tag share datapath. Note that, the intermediate registers of [FGMDP⁺18] are a requirement for SNI schemes in HW, but the SNI property is not a prerequisite for a scheme to be secure and we customized our design for better performance. We see currently no formal way to prove the security of our construction, but because the tag shares can be seen as Boolean shares of a multiplicative share of the secret, we judge that the tag shares do not need to be stored in registers in order to obtain security. We verify our approach empirically using TVLA and do not detect any leakage (cf. §7.3.1). Provable security can be achieved by adding registers to each of the tag share and value share datapaths.

Refreshing gadgets. When $d=1$, a $d+1$-share variable $\boldsymbol{x}$ can be refreshed with d fresh random units using additive refreshing:

$$(x_0, x_1, \ldots, x_d) \to (x_0 + r_0, x_1 + r_1, \ldots, x_{d-1} + r_{d-1}, x_d + \sum_{i=0}^{d-1} r_i)$$

For second-order security, we use ring refreshing as in [CRB+16], which consumes $d+1$ fresh random units.

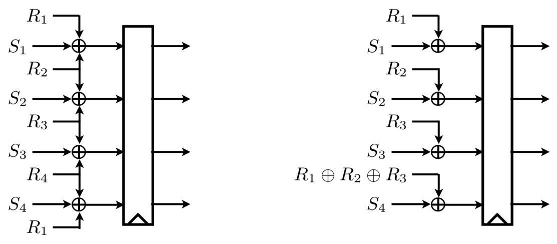

Figure 8: Ring and Additive Refreshing [CRB+16]

We use these refreshings for the shares of x^4, x^{20} and x^{100} in Figure 5 as well as for the shares of $(\tau^x)^4$. We let $r(d)$ be the corresponding randomness cost for refreshing $d+1$ shares with security order d, *i.e.* $r(1) = 1$ and $r(2) = 3$. The shares of $(\tau^{x^5})^4$ and $(\tau^{x^{25}})^4$ are refreshed using $\binom{d+1}{2}$ units during the last multiplication in the M&M power five block (cf. Figure 4). For $d \in \{1, 2\}$, we actually have that $r(d) = \binom{d+1}{2}$, so the cost of each refreshing is $r(d)$. We note again that these types of refreshing gadgets are only to be used for respectively first- and second-order security and are not secure for higher orders.

Inversion gadget. For the shared inversion in GF(2^8), $\star^{-1}$, we can use De Cnudde *et al.*'s $d+1$ [CRB+16] or Gross *et al.*'s Domain Oriented Masking [GMK17] AES S-box implementations. Both are based on Canright's compact S-box [Can05] using the tower field approach. We opt for the first, which requires five register stages. Together with the final shared multiplication with α^2, the latency of the M&M inversion in Figure 6 is thus six cycles.

7.2 Implementation Cost

Randomness We summarize the randomness cost in Table 1.

Recall that for $d \in \{1, 2\}$, each shared multiplication $\odot$ and each refreshing consume $\binom{d+1}{2}$ units of randomness, where each unit is a byte in the case of AES. The inversion pipeline of Figure 5 performs four M&M nonlinear operations (three times a^5 and once ab^2). Each of these requires exactly three $\odot$'s, although we count one less for the $f(a, b) = ab^2$ block (because this operation is included in the affine transform). We count two additional $\odot$'s with $\boldsymbol{\alpha^{-1}}$ for the computation of the tag shares of $\boldsymbol{x^4}$ and one $\odot$ for the tag shares in the affine transform. This brings the total number of shared multiplications $\odot$ to 14. In addition, we need 6 refreshings to preserve the non-completeness in the power five exponentiations $\star^5$.

The inversion circuit from [CRB+16] consumes 54 (resp. 162) bits of randomness in first-(resp. second-)order. The M&M inversion in Figure 6 uses this circuit twice. Furthermore, the affine transformation adds one additional shared (matrix) multiplication.

Finally, the infective computation uses randomness for one shared multiplication and an additional unit for the mask R. However, since the infection takes place when all SubBytes evaluations have finished, the total randomness does not increase.

Latency The first inversion from §4.1 requires nine clock cycles. In version 2, we use the $\star^{-1}$ implementation from [CRB+16], which results in a M&M inversion of six clock cycles. Recall from §4.2 that the AES affine transformation in M&M requires no additional cycles.

Table 1: Randomness Cost for the AES S-box implementations

	# $\odot$	# $\star^{-1}$	# Fresh	# Random Bits d	$d=1$	$d=2$
Shared Mult. ($\odot$)/Refresh	1	-	-	$8\binom{d+1}{2}$	8	24
Shared Inv. ($\star^{-1}$) [CRB+16]	-	1	-	-	54	162
Fresh Mask	-	-	1	8	8	8
S-box V1	14+6	-	-	$160\binom{d+1}{2}$	160	480
S-box V2	1	2	-	-	116	348
Infective Computation	1	-	1	$8\binom{d+1}{2}+8$	16	32

In total, the AES S-box output is thus obtained in nine clock cycles with version 1 and six clock cycles with version 2.

The byte-serialized architecture from [GMK17] is very efficient as it performs the MixColumns, ShiftRows and AddRoundKey stages in parallel with the SubBytes stage. As a result, when the S-box latency is $C \geq 4$ cycles, one round of encryption (including key schedule) requires exactly $16 + C$ clock cycles: During the first 16 cycles, all the state bytes are fed to the S-box pipeline input. The MixColumns operation is done in parallel every four cycles. The remaining C cycles are spent waiting for the last S-box output. During the first four of these, the S-box can be used by the key schedule. In the last cycle, the last S-box output is shifted into the state at the same time as ShiftRows is performed.

Our two versions of the AES Encryption therefore require respectively 25 and 22 clock cycles per encryption round.

Area We report our area results for first- and second-order security in Table 2 together with latency and randomness cost. As expected, the more customized version of the S-box results in a much more efficient implementation. Note that many more tradeoffs between area, latency and randomness cost are possible depending on design choices (*e.g.* multiplication chain) and used building blocks (*e.g.* shared inversion block $\star^{-1}$).

Table 2: Cost of first- and second-order AES implementations

	Area [kGE]		Latency	# Random bits/cycle	
	$d=1$	$d=2$	[# cycles]	$d=1$	$d=2$
State Array	5.6	8.5	-	-	-
Key Array	4.2	6.3	-	-	-
S-box					
• V1	19.2	42.0	9	160	480
• V2	6.5	13.5	6	116	348
Control	0.2	0.2	-	-	-
Infective Comp.	1.7	3.3	-	16	32
Other	1.0	1.4	-	-	-
Total					
• V1	31.9	61.7	266	160	480
• V2	19.2	33.2	236	116	348

Comparison to State-of-the-art. In Table 3, we report our area results next to other state-of-the-art schemes. Some of these protect only against SCA [CRB+16, GMK17] and some are combined countermeasures, such as ParTI [SMG16] and CAPA [RDB+18]. For the latter, it is not easy to compare the results given the difference in cipher implemented and synthesis libraries used. We try to overcome these differences by also reporting the overhead factor of the combined countermeasure, compared to an implementation that provides only protection against SCA.

The ParTI countermeasure is applied to the LED scheme in [SMG16]. The authors report an area of 20.2 kGE obtained with a UMC 0.18μm library [Inc04], compared to 7.9 kGE for the SCA-only first-order secure LED implementation. This signifies an area overhead factor of $\frac{20.2}{7.9} = 2.56$. For the combined countermeasure CAPA, we can compute the overhead over a SCA-secure KATAN implementation [RDB+18, Table 2]. Finally, we compare our first-order M&M AES (V2) with De Cnudde's [CRB+16] first-order implementation against SCA only. Table 3 reports the area of the implementation that is only secure against SCA in the fourth column and the area of the combined countermeasure in the fifth column. The overhead factor for those can be found in the last column. We note that the dependency on the synthesis library cannot completely be eliminated this way. Furthermore, all schemes consider very different adversary models.

Table 4 does the same for second-order secure implementations.

Table 3: Area comparison for first-order secure implementations

Countermeasure	Synthesis Library	Cipher	SCA-only [kGE]	Combined [kGE]	**Overhead factor**
[CRB+16]	Nangate 45*nm* [NAN]	AES	7.6[2]	-	-
[GMK17]	UMC 90nm Low-K	AES	6.0	-	-
CAPA [RDB+18]	Nangate 45*nm* [NAN]	KATAN	3.6	30.5	**8.47**
ParTI [SMG16]	UMC 0.18μm [Inc04]	LED	7.9	20.2	**2.56**
M&M	NanGate 45*nm* [NAN]	AES	7.6	19.2	**2.53**

Table 4: Area comparison for second-order secure implementations

Countermeasure	Synthesis Library	Cipher	SCA-only [kGE]	Combined [kGE]	**Overhead factor**
[CRB+16]	Nangate 45*nm* [NAN]	AES	12.6[1]	-	-
[GMK17]	UMC 90nm Low-K	AES	10.0	-	-
CAPA [RDB+18]	Nangate 45*nm* [NAN]	KATAN	5.9	55.2	**9.35**
M&M	NanGate 45*nm* [NAN]	AES	12.6	33.2	**2.63**

7.3 Evaluation

Evaluation of our M&M implementations is done separately for SCA and DFA, since no comprehensive method for verifying against combined attacks has been published to our knowledge. For the first, we program both versions of a second-order protected AES on an FPGA and evaluate the leakage coming from the power consumption with a non-specific t-test. The state-of-the-art on evaluating fault countermeasures is less advanced. We evaluate the scheme's EDP through a simulation of the circuits, in which we model additive faults in the RTL.

7.3.1 SCA evaluation

Setup. To assess the security of our implementations we use a SAKURA-G board, which is specifically designed for side-channel evaluation. On this board there are two distinct Spartan-6 FPGA's. The control FPGA handles the communication with the host computer and generates the shares for the cryptographic FPGA. In the crypto FPGA, we deploy the actual encryption scheme. This way we isolate the power consumption of the actual

[2]This number differs from the one reported in [CRB+16]. We contacted the authors and obtained their code in order to synthesize with the same software and library as our M&M implementation.

encryption, reducing considerably the noise in the experiment. We use a very slow 3 MHz clock to ensure clear power traces with minimal overlap between consecutive time samples. The synthesis of the design is done using Xilinx tools with the `KEEP HIERARCHY` constraint, in order to avoid optimizations across different shares. We sample the power consumption at 1.0GS/s with 10 000 points per frame, which includes 30 clock cycles. This is equivalent to approximately 1.2 rounds of V1 and 1.4 rounds of V2.

TVLA. We perform a non-specific test vector leakage assessment (TVLA) [BCD+13] using the methodology described in [RGV17]. This assessment is not used to mount an attack but to detect correlations of the instantaneous power consumption with the secret. We gather power traces for two distinct plaintext classes (one fixed and one random) and compare the two sets using the t-test statistic. When the t-statistic exceeds the threshold 4.5 in absolute value, one can conclude with confidence 99.9995% that the two sets of power traces follow different distributions and thus, that the design leaks. This is a necessary but not sufficient condition for a successful attack to exist. When the t-statistic remains below this threshold, the designer can conclude with high confidence that the design is secure. We choose the fixed plaintext equal to the key so that all S-box inputs in the first round are zero.

We first perform the t-test on an unprotected AES implementation to verify that our setup is sound and able to detect leakage. We emulate the unprotected implementation by disabling the PRNG. Leakage is then expected in every order. When we turn the PRNG on and activate the countermeasures, we expect only third-order leakage since the implementation is second-order secure.

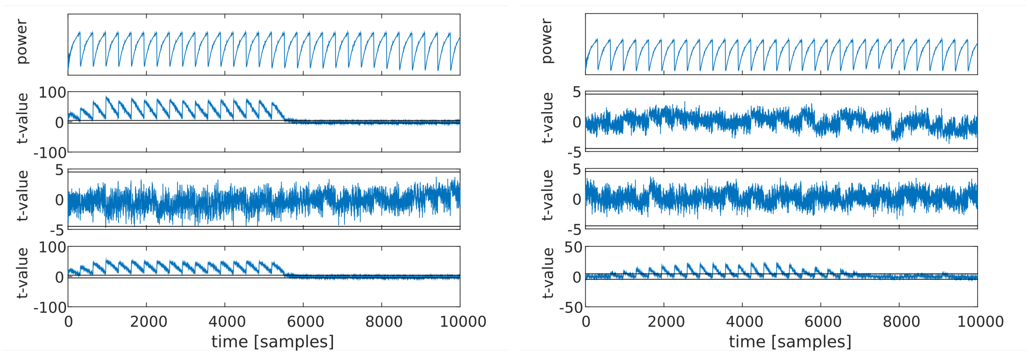

Figure 9: Non-specific t-test on second-order secure M&M AES implementation, V1. Left: PRNG off (24K traces); Right: PRNG on (100M traces). Rows (top to bottom): one exemplary power trace; first-order t-test; second-order t-test; third-order t-test.

The t-test results for version 1 and 2 of our AES implementation are shown in Figures 9 and 10 respectively. In both cases, we see clear evidence of leakage at only 24 000 traces when the PRNG is turned off. When we enable the PRNG, neither the first- nor second-order t-test statistics surpass the threshold 4.5 with up to 100 million power traces.

In addition, we perform a bivariate analysis by combining time samples. For memory efficiency, we reduce the resolution of the oscilloscope to 100MS/s, resulting in power traces of 1 000 time samples each. We then perform the t-test as described above on $1\,000 \times 1\,000$ matrices, formed by a centered product of the traces. The results are shown in Figure 11 and confirm that there is no bivariate leakage with up to 50 million traces.

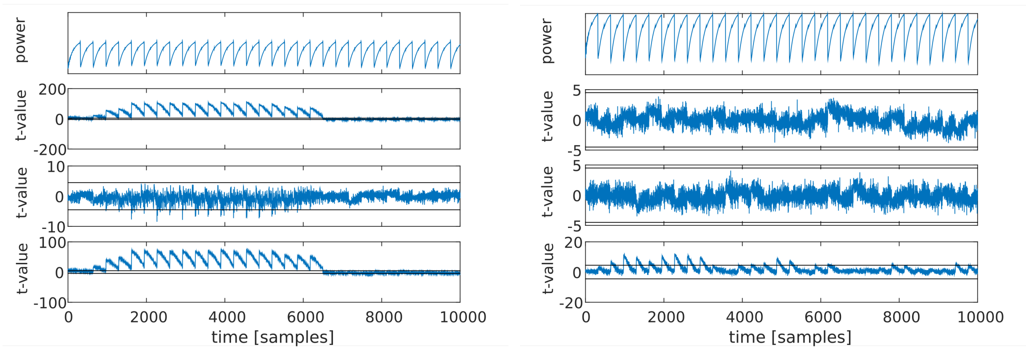

Figure 10: Non-specific t-test on second-order secure M&M AES implementation, V2. Left: PRNG off (24K traces); Right: PRNG on (100M traces). Rows (top to bottom): one exemplary power trace; first-order t-test; second-order t-test; third-order t-test.

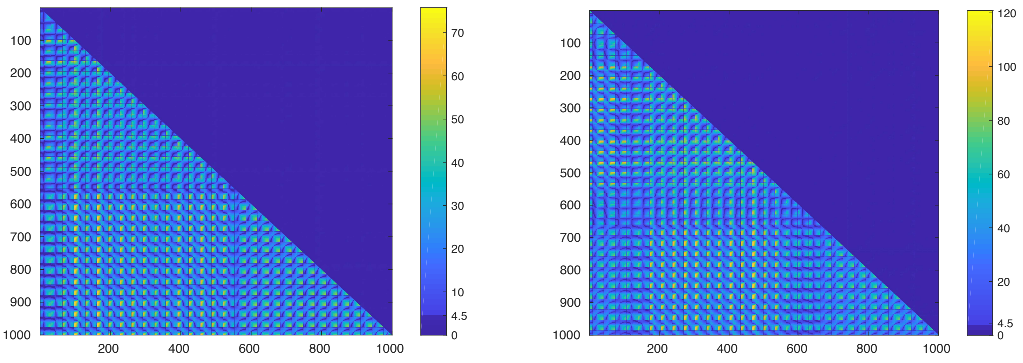

Figure 11: Bivariate t-test on second-order secure M&M AES imlementation, V1 (left) and V2 (right). Below diagonal: PRNG off (20K traces); Above diagonal: PRNG on (50M traces).

7.3.2 FA evaluation

In this section we evaluate the behaviour of our design when *multiple* stochastic faults are injected. We describe our experiment, which aims to measure M&M's detection rate of faults. For simplicity, we evaluate our first-order secure AES implementations.

Fault modeling. Traditionally, fault modeling theory distinguishes between faults that affect the logic function on the one hand and delay faults on the other. Moreover, faults can be clasified as structural (modifying the interconnections among components in the circuit) or functional (modifying the functionality of certain parts of the circuit) [ABF94].

Faults in cryptographic devices are typically injected with a laser or introduced by clock or power line glitches. In our experiments we consider functional faults in the logic functions to model the adversary of §2 (*i.e.* additive errors). We model faults using XOR additions. This does not only allow us to flip one specific bit, but also to XOR entire offsets to k-bit words.

Fault injection. We enable a fault injection on a wire by extending the original VHDL code with an additional *fault gate* on that wire. Such a gate is simply an XOR with a *fault selector*, indicating whether or not we want to inject a fault.

In each design to be tested, we select a number of critical bytes where an attacker is most likely to inject a fault. These points are: the input to the state register; the state and key byte before AddRoundKey; the SubBytes input from the key schedule and four different points inside SubBytes. Faults can be inserted in every data share and every tag share of those bytes.

This means that 256 fault gates are installed in the first order implementations. We collect the corresponding fault selectors in a fault vector, which is controlled by the testbench. Each bit set to '1' in the fault vector corresponds to a fault on a single bit. By setting multiple bits in the vector, we enable *multiple faults* in the implementation. When several faults are activated in the same byte, it implies the XOR of an offset to that variable.

We want to be able to randomly draw fault vectors with a chosen Hamming weight H. For this, we draw inspiration from the basic principles of address decoding. We draw one random bit; if it is 'zero', a fault occurs in the first half of the vector and if it is 'one', in the second half. We draw a second bit and follow the same procedure to decide which of the two quarters. We continue this way until a single fault bit is selected. Thus, $log_2(256) = 8$ random bits are needed to set a one-bit fault in a 256-bit vector. By repeating this method H times, we can draw random fault vectors with Hamming weight H. In our experiments we choose $H = 128$.

For each selected bit, we flip one more coin that decides whether the selected fault is activated or not. This means that of the 128 selected faults, approximately half will be active. Since most of the fault attacks in literature target one of the last rounds of AES, we similarly "inject" our faults in the last round of encryption.

Results. We simulate the fault-augmented VHDL code with Xilinx ISIM for 50 000 iterations and measure the fault detection rate. In each experiment, approximately 64 bits are altered in the computation. We are thus faulting one or more bits in multiple bytes. We consider the faults *detected* if the returned ciphertext is infected. In version 1 of our M&M AES, the experiment shows that 210 faulty ciphertexts are not infected. This means that the experimental rate of detection of our M&M implementation is 0.9958, compared to the theoretical $1 - 2^{-8} = 0.9961$. In our second AES version, 189 faulty ciphertexts are not infected, which means the experimental detection probability is 0.9962.

8 Conclusion

We introduce a new family of countermeasures to provide security against both SCA and DFA. M&M can extend any masking countermeasure with information-theoretic MAC tags and infective computation. We demonstrate how to construct basic M&M building blocks and how to build a secure implementation of any cipher. We illustrate our proposal with first- and second-order secure implementations of AES and we experimentally verify the SCA and DFA security. We show that M&M implementations can be very efficient while providing resistance against both SCA and DFA in a strong but realistic adversary model.

Acknowledgements

The authors would like to thank Dusan Bozilov, Begül Bilgin and Nigel Smart for fruitful discussions and also the CHES reviewers for their helpful comments. This work was supported in part by the Research Council KU Leuven: C16/15/058 and OT/13/071, by the NIST Research Grant 60NANB15D346 and the EU H2020 project FENTEC. Lauren De Meyer is funded by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO).

References

[ABF94] M. Abramovici, M. Breuer, and A. Friedman. Digytal systems testing and testable design. Wiley-IEEE Press, September 1994.

[AVFM07] F. Amiel, K. Villegas, B. Feix, and L. Marcel. Passive and active combined attacks: Combining fault attacks and side channel analysis. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007)*, pages 92–102, Sept 2007.

[BBD$^+$15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Veriffied proofs of higher-order masking. *EUROCRYPT, IACR Cryptology ePrint Archive*, 2015:060, 2015.

[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.

[BBD$^+$18] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Improved parallel mask refreshing algorithms - generic solutions with parametrized non-interference & automated optimizations. *IACR Cryptology ePrint Archive*, 2018:505, 2018.

[BBK$^+$03] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Trans. Computers*, 52(4):492–505, 2003.

[BBP$^+$16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

[BCC$^+$14] Julien Bringer, Claude Carlet, Hervé Chabanne, Sylvain Guilley, and Houssem Maghrebi. Orthogonal direct sum masking - A smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks. In David Naccache and Damien Sauveron, editors, *Information Security Theory and Practice. Securing the Internet of Things - 8th IFIP WG 11.2 International Workshop, WISTP 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings*, volume 8501 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2014.

[BCD$^+$13] G. Becker, J. Cooper, E. De Mulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi, et al. Test vector leakage assessment (tvla) methodology in practice. In *International Cryptographic Module Conference*, volume 1001, page 13, 2013.

[BECN$^+$06] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, Feb 2006.

[BG13] Alberto Battistello and Christophe Giraud. Fault analysis of infective AES computations. In Wieland Fischer and Jörn-Marc Schmidt, editors, *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*, pages 101–107. IEEE Computer Society, 2013.

[BGN$^+$14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[BS97] Eli Biham and Adi Shamir. *Differential fault analysis of secret key cryptosystems*, pages 513–525. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[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.

[CFGR10] C. Clavier, B. Feix, G. Gagnerot, and M. Roussellet. Passive and active combined attacks on aes combining fault attacks and side channel analysis. In *2010 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 10–19, Aug 2010.

[Cla07] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[CN16] Thomas De Cnudde and Svetla Nikova. More efficient private circuits II through threshold implementations. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*, pages 114–124. IEEE Computer Society, 2016.

[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.

[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. *IACR Cryptology ePrint Archive*, 2018:357, 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 Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):547–572, Aug. 2018.

[DV12] F. Dassance and A. Venelli. Combined fault and side-channel attacks on the aes key schedule. In *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 63–71, Sept 2012.

[FGMDP+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 Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, Aug. 2018.

[FH17] Wieland Fischer and Naofumi Homma, editors. *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*. Springer, 2017.

[GM17] Hannes Groß and Stefan Mangard. Reconciling d+1 masking in hardware and software. In Fischer and Homma [FH17], pages 115–136.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *IACR Cryptology ePrint Archive*, 2016:486, 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.

[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.

[GST12] Benedikt Gierlichs, Jörn-Marc Schmidt, and Michael Tunstall. Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings*, volume 7533 of *Lecture Notes in Computer Science*, pages 305–321. Springer, 2012.

[Inc04] Virtual Silicon Inc. 0.18 μm VIP Standard cell library tapeout ready, partnumber: UMCL18G212T3, process: UMC logic 0.18μm generic II technology: 0.18μm, July 2004.

[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] Y. Ishai, A. Sahai, and D. Wagner. *Private Circuits: Securing Hardware against Probing Attacks*, pages 463–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[KJJ99] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

[KKG03] Ramesh Karri, Grigori Kuznetsov, and Michael Gössel. Parity-based concurrent error detection of substitution-permutation network block ciphers. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2003.

[KKT04] Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. Differential fault analysis attack resistant architectures for the advanced encryption standard. In Jean-Jacques Quisquater, Pierre Paradinas, Yves Deswarte, and Anas Abou El Kalam, editors, *Smart Card Research and Advanced Applications VI, IFIP 18th World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), 22-27 August 2004, Toulouse, France*, volume 153 of *IFIP*, pages 177–192. Kluwer/Springer, 2004.

[LRT12] Victor Lomné, Thomas Roche, and Adrian Thillard. On the need of randomness in fault attack countermeasures - application to AES. In Guido Bertoni and Benedikt Gierlichs, editors, *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*, pages 85–94. IEEE Computer Society, 2012.

[LSG$^+$10] Yang Li, Kazuo Sakiyama, Shigeto Gomisawa, Toshinori Fukunaga, Junko Takahashi, and Kazuo Ohta. *Fault Sensitivity Analysis*, pages 320–334. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[NAN] NANGATE. The NanGate 45nm Open Cell Library. Available at http://www.nangate.com.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, pages 529–545, 2006.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.

[PR11] Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2011.

[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.

[RDB$^+$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.

[RGV17] Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. Fast leakage assessment. In Fischer and Homma [FH17], pages 387–399.

[RLK11] Thomas Roche, Victor Lomné, and Karim Khalfallah. *Combined Fault and Side-Channel Attack on Protected Implementations of AES*, pages 65–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[SFRES18] Okan Seker, Abraham Fernandez-Rubio, Thomas Eisenbarth, and Rainer Steinwandt. Extending glitch-free multiparty protocols to resist fault injection attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):394–430, Aug. 2018.

[SMG16] Tobias Schneider, Amir Moradi, and Tim Güneysu. Parti: Towards combined hardware countermeasures against side-channeland fault-injection attacks. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 39. ACM, 2016.