Extending Glitch-Free Multiparty Protocols to Resist Fault Injection Attacks

Okan Seker[1]*, Abraham Fernandez-Rubio[2]†, Thomas Eisenbarth[1,3] and Rainer Steinwandt[4]

[1] University of Lübeck, Germany
{okan.seker,thomas.eisenbarth}@uni-luebeck.de
[2] Intel
abraham.fernandez.rubio@intel.com
[3] Worcester Polytechnic Institute, USA
[4] Florida Atlantic University, USA
rsteinwa@fau.edu

Abstract. Side channel analysis and fault attacks are two powerful methods to analyze and break cryptographic implementations. At CHES 2011, Roche and Prouff applied secure multiparty computation to prevent side-channel attacks. While multiparty computation is known to be fault-resistant as well, the particular scheme used for side-channel protection does not currently offer this feature. This work introduces a new secure multiparty circuit to prevent both fault injection attacks and side-channel analysis. The new scheme extends the Roche and Prouff scheme to make faults detectable. Arithmetic operations have been redesigned to propagate fault information until a new secrecy-preserving fault detection can be performed. A new recombination operation ensures randomization of the output in the case of a fault, ensuring that nothing can be learned from the faulty output. The security of the new scheme is proved in the ISW probing model, using the reformulated t-SNI security notion. Besides the new scheme and its security proof, we also present an extensive performance analysis, including a proof-of-concept, software-based AES implementation featuring the masking technique to resist both *fault and side-channel attacks at the same time.* The performance analysis for different security levels are given for the ARM-M0+ MCU with its memory requirements. A comprehensive leakage analysis shows that a careful implementation of the scheme achieves the expected security level.

Keywords: Secure multiparty computation · Side-channel analysis · Fault attacks · Polynomial Masking · ARM

1 Introduction

Physical attacks are a common threat to cryptosystems if the adversary has physical access to the implementation. A wide range of such attacks has been shown to circumvent security assumptions and reveal cryptographic keys, often with little effort, especially if no special precautions were taken during implementation. Two commonly considered classes of physical attacks are *fault injection attacks* and passive *side-channel attacks.* Fault attacks require a fault to be induced into the (secret) state. The resulting faulty output can then reveal information about the state and the key [BDL97, BECN+06]. Similarly, data

*A significant part of the work was performed while the author was working at Worcester Polytechnic Institute, USA.

derived from power [KJJR11], sound [GST14], or electromagnetic emanation [GMO01] of a target implementation is measured in a side-channel analysis to learn information about the secret state. Another studied class of physical attacks are *probing attacks*. Ishai et al. [ISW03] construct a generic countermeasure for probing attacks, which has also been analyzed in the context of side-channels [DDF14, BDF+17].

Due to the effectiveness of physical attacks, countermeasures to both fault and side-channel attacks have been studied extensively. A common technique to prevent fault induction is error detection through adding redundancy. Often, reliable error detection requires the duplication of computation in space or time [BECN+06], especially for symmetric ciphers, to achieve the desired error detection ratio. For asymmetric ciphers, lower overheads are often possible [Sha99, Gir06]. Another technique proposed by Genkin et al. [GIP+14] uses algebraic manipulation detection (AMD) codes to protect sensitive variables. The idea is to compute a proof that the output is correct. However, this approach requires generating a MAC tag for each gate. Other branches of fault prevention are *infective* countermeasures, which aim at randomizing the secret state if an error occurs. Lomné et al. [LRT12] introduce an infective countermeasure using multiplicative random masking. Gierlichs et al. [GST12] present the idea of dummy rounds. However, these countermeasures were broken by Bastellini et al. [BG13]. The second scheme was improved by Tupsamudre et al. [TBM14]. The updated scheme has been analyzed in [BG16], showing that getting the countermeasure right and efficient is difficult.

To counteract side-channel analysis, one popular and effective countermeasure is to use secret shares, referred to as *masking* [CJRR99] in the SCA literature. The idea consists of splitting a sensitive variable x into n shares, such that x can be recovered from $d+1$ ($n \geq d+1$) shares, while no information can be recovered from fewer than $d+1$ shares. A basic example is the *Boolean masking* introduced by Ishai et al. [ISW03]. This approach based on sharing the sensitive variable x such that $x = x_1 \oplus \ldots \oplus x_n$ where $n-1$ of the shares are uniformly distributed. Therefore, x can be reconstructed using n shares, but no information can be gained form less than n shares. Also, AES implementations protected by first and second order Boolean masking schemes are given in [SP06, RDP08]. Later, Rivain and Prouff enhanced the Boolean masking schemes to work on any finite field [RP10] and implemented AES efficiently and securely in software using provable secure operations.

Earlier masking schemes were considered secure under specific security models such as [OMPR05, CB08], however, they would still leak detectable information under the presence of glitches in the hardware [MME10, MPO05]. Due to these facts, glitch resistance masking schemes were introduced. The *Threshold Implementations* (TI) are defined by Nikova et al. [NRS09] and then generalized by [BGN+14, RBN+15]. Gross et al. [GMK16] introduced *domain-oriented masking*. While the scheme had the same level of security as TI, it has lower implementation cost, since it has lower randomness cost.

Another approach uses polynomial masking schemes based on Shamir's secret sharing [Sha79]. This novel idea was employed by Roche and Prouff [RP11, RP12] and Goubin and Martinelli [GM11] (which is shown to be flawed in [CPR13]) using secure multi-party computations (SMC) defined by Ben-Or et al. [Ben88] and Gennaro et al. [GRR98]. The SMC proposed by Ben-Or et al. [Ben88] is both t-private and t-resilient, i.e. it guarantees that some subset of t parties can neither learn nor modify results. Roche and Prouff adopted the t-private property to achieve side-channel protection through glitch-free SMC [RP11, RP12]. It is known that the t-resilient property can be used to thwart fault attacks [RP12, GSF14], but this property has never been analyzed in that context.

To achieve both fault and side-channel resistance, two countermeasures can be combined. However, while the interactions between the countermeasures have not been studied in much depth, combining ad-hoc methods can have adverse effects [REB+08, LFZD14]. Furthermore, overheads are huge and become larger for combined methods. Nevertheless, resistance against both attacks is important, since attacks can be combined to have greater effects on partially protected implementations [RLK11, LRT12]. So far, the amount of research in this area is scarce. Ishai et al. extended private circuits [ISW03] to private circuits II by adding redundancy via encodings [IPSW06]. Therefore they were able to generate a circuit which resists both SCA and tampering attacks. Depending on the fault model, they defined two different encodings. Therefore they managed to detect faults by means of invalid encodings. Schneider et al. [SMG16] proposed a combined side-channel and fault countermeasure using TI and error detecting codes [MS77]. While their proposal is fairly efficient, the fault coverage depends on the fault distribution. The scheme does not by itself ensure the randomness of the output. The SCA resistance of scheme relies on security of a TI. Another countermeasure is introduced by De Cnudde et al. [DCN16] which enhanced private circuits II with TI. The idea is the same as private circuits II which forwards the valid inputs unless a faulty encoding is detected. A recent countermeasure introduced by Reparaz et al. [RMB$^+$17] builds on doing computations on shared values and MAC tags. The side channel resistance of the scheme relies on the secret sharing while fault resistance of the scheme relies on the MAC-tag shares.

Our contribution. This work examines the fault-resistance of the glitch-free secure multiparty circuits proposed by Roche and Prouff [RP11, RP12] and proposes a new combined protection scheme for both side-channel and fault attacks with its security proof in the ISW probing model [ISW03].

We start with analyzing the fault behavior of the operations, namely affine transformation and squaring of a secret share, addition of two secret shares, and multiplication of two secret shares. It is shown that, while most parts of the glitch-free SMCs can be naturally extended to detect faults, the multiplication operation makes faults undetectable. The circuits become vulnerable to fault attacks. We propose a new multiplication circuit that properly maintains fault information and thus allows for the composition of glitch-free fault-detecting SMCs in which errors are propagated by the algebraic operations. Thus, the fault information will be detectable until the end of the circuit. Therefore, we eliminate the cost of fault detection. We introduce a new recombination operation which randomizes the output, if an error occurred anywhere in the circuit, ensuring that the attacker cannot learn anything from the output.

We are able to construct arbitrary fault-resistant circuits using the basic arithmetic operations and a new recombination-fault detection operations. As a result, the attacker gets random outputs at the end of the cryptographic operations. Our scheme differs from previous proposals, as it does not have the same requirement of $n \geq 3d + 1$ to detect d *cheaters* in an (n, d)-secret sharing scheme. Instead, our scheme can detect up to ε errors, where $n > 2d + \varepsilon$ with very high probability. In fact, the detection probability is 1 after the first operation on faulty shares, but can slightly decrease, depending on the number of subsequent additions and multiplications. Even in the worst case, if only one of the inputs of the operations is faulty, the faulty output can always be detectable. Moreover, we provide a secrecy-preserving fault detection operation to increase the fault detection capabilities of the users as an option. This operation provides a trade-off between performance and security. It can be used securely (i.e., without leaking sensitive information) and therefore, perfect fault detection can be achieved.

To be able to prove the security against probing attacks, we follow the t-SNI security notions and give the formal proof of our schemes. First, we reformulate the t-SNI security notions to cover arbitrary (n, d)-secret sharing schemes. Then, we show that each operation defined in this paper satisfies t-probing security. Since, the new multiplication scheme is an extension of the one used in [RP12], the first formal security proof of the multiplication scheme [RP12] is provided within this work. Moreover, we introduce a refresh masking operation for polynomial masking to construct complete t-SNI algorithms.

To analyze the fault detection properties of our scheme we use *Coverage* [SMG16] and define a new security notion *Propagation*, which states the probability of detecting faults using output shares if the input shares are faulty. We give a theoretical analysis of the proposed scheme using the new security notion. Moreover, we examine the fault resistance of the scheme by a simulation written in `SAGE`.

Implementation and leakage analysis of the original scheme have been performed in hardware [MM13]. Grosso et al. [GSF14] analyze its performance and present several methods for speeding up polynomial masking. To the best of our knowledge, however, there are no software implementations of the scheme tested under a comprehensive leakage assessment. Thus, we propose a practical C implementation tested on a popular ultra-low power architecture, the ARM Cortex M0+ core. Our analysis goes beyond the implementation itself by demonstrating its level of side-channel resistance and measuring its performance. The implementation provides multiple masking schemes easily portable to higher orders. However, by precomputing and inserting different public shares and corresponding Vandermonde matrices in the code, other orders of masking can be employed without modification of the operations and functions. To show the side-channel resistance of our implementation, we address a full leakage analysis including higher order moments on the SMC multiplication. High assurance of the mask quality is ensured through utilizing a built-in true random number generator available on the MCU. These tests enable us to see the relation between processed sensitive variables and side-channel leakage, the results show how they are statistically independent.

The code has also more advanced constant-time features: we present different types of field multiplication, some of which rely on input-dependent table accesses and thus give better performance. However, we also present true constant-time multiplication, which is slower, but is constant-time even on systems featuring caches. All of these different schemes execute in constant time regardless of the selected version of field operations. To that end, the code features a *fully constant execution flow with constant memory accesses* and is available as open source.[1]

2 Background

In this section, we describe Shamir's secret sharing and the multiparty circuit constructed in [RP12]. Also, we introduce the adversary models, namely, the ISW probing model [ISW03], the additive fault model and the re-formulation of t-SNI security notion.

2.1 Shamir's Secret Sharing and Secure Multiparty Computations

Shamir's secret sharing scheme [Sha79] allows players to split a secret using a polynomial. In the protocol, the trusted dealer generates a random polynomial, $F(x) = f_0 + f_1 x + \ldots + f_d x^d$ to share a secret f_0. The secret value is shared by evaluating $F(x)$ for n distinct and nonzero public points $(\alpha_0, \ldots, \alpha_{n-1})$. The shared representation of f_0 is shown as $\mathcal{F} = (F(\alpha_0), \ldots, F(\alpha_{n-1}))$ and throughout the paper, we denote them as *secret shares* or *secret states*. The secret reconstruction is done by polynomial interpolation using at least shares of $d+1$ players.

The most important feature of the polynomial masking is that, we can reconstruct all coefficients of $F(x)$. Let $\mathcal{A} = (f_0, f_1, \ldots, f_{n-1})$ be the coefficient vector of polynomial $F(x)$, then the relation between coefficients and secret shares can be formalized as $V\mathcal{A}^T = \mathcal{F}^T$ where V is the $n \times n$ Vandermonde matrix $V = (\alpha_j^i)_{i,j=0,\ldots,n-1}$. The coefficients of $F(x)$

[1]The code has been made publicly available at https://github.com/vernamlab/Robust-AES.

can be calculated as follows:

$$f_j = \sum_{i=0}^{n-1} F(\alpha_i)\lambda_i^j, \text{ where } V^{-1} = (\lambda_i^j)_{i,j=0,\dots,n-1}. \tag{1}$$

Remark that in a valid secret sharing scheme $f_{d+1} = \ldots = f_{n-1} = 0$. This fact is used in the following sections to detect faults.

Secure Multiparty Computations with secret sharing schemes (denoted by (n, d)-SMC) allow us to split a sensitive variable into shares in such a way that neither the shares nor the computations on them reveal any critical information. Relying on this fact, Roche and Prouff proposed SMC as *multiparty circuit* (MPC) to counteract higher-order side-channel analyses, even in the presence of *glitches*, and showed how to apply it to AES [RP12]. Moradi et al. [MM13] provided a first implementation of this scheme in hardware, as well as a practical side-channel analysis of their implementation. Similarly, Grosso et al. [GSF14] examined the performance of existing masking schemes in software for low-power microcontrollers. Both works concluded that the scheme comes with a significant overhead, even when compared to other side-channel protection schemes. The latter work proposed the usage of packed secret sharing to make the scheme more efficient for higher protection orders. They expanded SMC into a (n, d)-multiparty circuit using a sequence of sub-circuits.

2.2 The ISW Probing Model

Ishai et al. [ISW03] introduced how to build secure circuits against an adversary that can probe a portion of intermediate variables of the circuit. To prove the security of a circuit in the ISW probing model, any t probes should be perfectly simulatable without knowledge of the original variables in the circuit. If any set of t probed variables is perfectly simulated by $\leq t$ input shares, which are independently uniformly distributed elements from the field, then knowledge of the original circuit variables is not required to simulate t probes and no set of t probed variables brings any additional information. Next, we recall t-NI and t-SNI security notions originally defined by in [BBD+15] as they were restated by Coron et al. [CGPZ16].

Definition 1 (t-NI Security)**.** Let G be a gadget which takes as input n shares $(x_i)_{1\leq i\leq n}$ and as outputs n shares $(y_i)_{1\leq i\leq n}$. The gadget G is said to be t-NI secure if for any set of t probed intermediate variables and any subset $\mathcal{O} \subset [1, n]$ of output indices, such that $t+|\mathcal{O}| < n$, there exists a subset $I \subset [1, n]$ of input indices which satisfies $|I| \leq t+|\mathcal{O}|$, such that the t intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.

Definition 2 (t-SNI Security)**.** Let G be a gadget which takes as input n shares $(x_i)_{1\leq i\leq n}$ and as outputs n shares $(y_i)_{1\leq i\leq n}$. The gadget G is said to be t-SNI secure if for any set of t probed intermediate variables and any subset $\mathcal{O} \subset [1, n]$ of output indices, such that $t + |\mathcal{O}| < n$, there exists a subset $I \subset [1, n]$ of input indices which satisfies $|I| \leq t$, such that the t intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.

The main difference between the t-NI security notion and the stronger t-SNI security notion is that the size of the input subset does not depend on the size of the set of output shares $\mathcal{O}$. The t-SNI security notion ensures the security of a construction with $n \geq t + 1$, while the t-NI security notion enables a secure construction with $n \geq 2t + 1$. As shown in [DDF14], the probing adversary can be reduced to the *t-threshold-probing model*. Security in the *t-threshold-probing model* implies security in the *noisy leakage model*. Furthermore, the recent study by Barthe et al. [BDF+17] shows that security in the *probing* *model* for a serial implementation implies security in the *bounded leakage model* for the corresponding parallel implementation.

The t-SNI security notion becomes the standard way of proving the security against probing attacks. However, we cannot use the security notion directly. The notion is specialized for Boolean masking where $(n-1)$-tuples of n intermediate variables are uniformly distributed. On the other hand, an (n, d)-secret sharing corresponds to n intermediate variables such that every d-tuple of them is uniformly distributed and independent of any sensitive variable instead of $(n-1)$-tuple. Therefore, we extend the definition to cover (n, d)-SMC and through the paper we focus on the modified version of the security notion defined as follows:

Definition 3 (t-SNI^n_d Security)**.** Let G be a gadget which takes as input n shares $(x_i)_{1\leq i\leq n}$ and as outputs n shares $(y_i)_{1\leq i\leq n}$. The gadget G is said to be t-SNI^n_d secure if for any set of t probed intermediate variables and any subset $\mathcal{O} \subset [1, n]$ of output indices, such that $t + |\mathcal{O}| < d + 1$, there exists a subset $I \subset [1, n]$ of input indices which satisfies $|I| \leq t$, such that the t intermediate variables and the output variables $y_{|\mathcal{O}}$ can be perfectly simulated from $x_{|I}$.

Clearly, the original definition corresponds to t-SNI$^n_{n-1}$ in our notation. We show that it is possible to construct t probes as well as a set of output shares $\mathcal{O}$ such that $t + |\mathcal{O}| < d + 1$, using a subset of input shares with at most t elements. The elements in the subset will be uniformly distributed and be independent from the shared secret value. It should be noted that, the set of probed variables and the output shares can be perfectly simulated by d-tuple of random variables. Therefore, the modified security notion will be equivalent to the original definition and the *perfect t-probing security* defined by Carlet et al. [CPRR15] as well as Lemma 1 in [RP10].

2.3 The Additive Fault Model

Different types of fault models have been introduced in the literature. The models range from bit level faults to data path faults or control flow faults. Briefly, bit level faults can be summarized as *reset* (change a wire value to zero), *set* (change a wire value to one) or *toggle* (flipping the wire value) attacks [IPSW06]. More general faults, such as data path targeted faults can be classified according to distribution of the injected faults. An attacker can sample the injected faults from a uniform distribution or from a biased distribution, where one set of faults is significantly more probable then the set or remaining faults [SMG16]. Moreover, an extreme case of this idea is used by Reparaz et al. [RMB$^+$17] and it gives the attacker full control of the faults and defined as known-value faults. Depending on the security models, the distribution of the faults is combined with the number of injections or number of wires to inject the fault. For example, known-value faults can be used with limited number of wires to inject the faults or a uniform fault model can be used to injects faults to all wires [RMB$^+$17].

Before stating the fault model, we introduce the notation that we use in the following sections. Throughout the paper, we denote the secret values as f_0 and g_0 from a fixed finite field $\mathbb{F}$ and the nonzero public points as $(\alpha_0, \ldots, \alpha_{n-1})$. To generate corresponding (n, d)-secret sharing schemes, the trusted dealer generates two random polynomials $F(x) = f_0 \oplus f_1 x \oplus \ldots \oplus f_d x^d \in \mathbb{F}[x]$ and $G(x) = g_0 \oplus g_1 x \oplus \ldots \oplus g_d x^d \in \mathbb{F}[x]$. After evaluating the polynomials at the public points, the dealer distributes the secret share of the $i+1^{th}$ player as $F(\alpha_i)$ and $G(\alpha_i)$. For simplicity, we denote the share indices as subscripts, such as $F(\alpha_i) := F_i$. Therefore, the sets $(F_0, \ldots, F_{n-1})$ and $(G_0, \ldots, G_{n-1})$ represent the sets of the secret states of f_0 and g_0 respectively. Moreover, through the paper, a valid secret sharing means when we perform a polynomial interpolation to the set $(F_0, \ldots, F_{n-1})$, the $deg(F(x))$ will be less then or equal to d. Similarly, an invalid secret sharing means $deg(F(x)) > d$.

In our model, we address data targeted faults on the secret state only. As studied in most of the works, fault injections on the control flow are excluded from the model and need to be prevented by other means. The attack model is similar to known-value faults defined in [RMB+17] or the extreme case of a biased fault model in [SMG16]. It can be described as *blindly-chosen*, *non-adaptive*, and *additive* faults. That is, the attacker can pre-calculate a set of faults and induce single or multiple faults from this set to the secret states. The attacker can target any instance of the operation, can choose the specific elements from the secret states, and can inject multiple faults in one clock cycle. More precisely, the logical effect of a fault σ to a share (F_i) will be $F_i \oplus \sigma$ and the faulty state denoted by F'_i. Therefore, the *additive fault model* describes a wide class of errors that can be observed in practice, such as flipping one bit of F_i. Using this model, we eliminate cases like limitation of Hamming weight or selecting faults from a distribution. Faults on randomness are also allowed due to the *additive* fault property: Adding a blindly-chosen fault to an unknown random value results in an unknown random value.

Since the computations continue with the faulty state F'_i, the degree of the polynomial generated by the faulty state (denoted by $F'(x)$) is greater than d with a high probability, so we can detect the faults by checking the degree of the secret sharing polynomial. Therefore the effect of multiple fault injections in one clock cycle will be the same in our analysis. The advantage of the additive model is that we can clearly define the relation between the secret state and the secret sharing polynomial using the Vandermonde representation. To examine this relation, we use the matrix representation of Equation 1.

$$V^{-1}\cdot \underbrace{\begin{pmatrix} F_0 \\ \vdots \\ F_{i-1} \\ F_i \\ \vdots \\ F_{n-1} \end{pmatrix}}_{\text{Secret states.}} = \underbrace{\begin{pmatrix} f_0 \\ \vdots \\ f_d \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{\substack{\text{The coefficients} \\ \text{of } F(x).}} \xrightarrow{\text{Fault Injection}} V^{-1}\cdot \underbrace{\begin{pmatrix} F_0 \\ \vdots \\ F_{i-1} \\ F'_i \\ \vdots \\ F_{n-1} \end{pmatrix}}_{\text{Faulty states.}} = \underbrace{\begin{pmatrix} f'_0 \\ \vdots \\ f'_d \\ f'_{d+1} \\ \vdots \\ f'_{n-1} \end{pmatrix}}_{\substack{\text{The coefficients} \\ \text{of } F'(x).}} \tag{2}$$

According to the additive fault model, the relation between $F'(x)$ and $F(x)$ can be summarized as $F'(x) = F(x) \oplus \Delta(x)$, where $\Delta(x)$ is the polynomial generated by faults i.e. interpolated by the points, $(0, \ldots, 0, \sigma, 0, \ldots, 0)$. For an (n, d)-secret sharing scheme, faults are undetectable if and only if the degree of $\Delta(x)$ is smaller than or equal to d, i.e., the coefficients of the terms of degree $d+1, \ldots, n-1$ are zero. We refer to these terms as *error detection terms*.

3 SMC as a Fault Injection Countermeasure

Another important feature of secure multiparty computation schemes is that they can be used to detect faults. Depending on the number of faulty shares, errors are *detectable*, *undetectable*, or *correctable*. To be able to *correct* faults on d shares, previously proposed schemes require at least $3d+1$ shares [Ben88]. Furthermore, *robust* multiplication requires cheater detection at the input and the output of every multiplication, which is a very costly operation [GRR98].

Fault injection countermeasures are less concerned with the correction of errors. The main goal is usually to *detect* faults and to ensure nothing can be learned from a faulty output. Therefore, our aim is to preserve faults and detect them only once the output is produced. To be able to detect the faults without conducting additional operations, we need to observe the propagation of the faulty state for each SMC component. In this section, we discuss the preservation of the faulty state and show the vulnerabilities of the computations. Remark that, in the following analysis, we assume that at least one of the input polynomials is faulty, i.e. $deg(F(x)) > d$ or $deg(G(x)) > d$.

- **Affine transformation of a secret (`Affine`):** An affine transformation $\mathsf{L}(x) = ax \oplus b$ with $a \neq 0$ can be computed on the secret value by applying L to secret shares locally; also each player P_i computes its component by $\mathsf{L}(F_i)$. If $deg(F(x)) > d$, then clearly $deg(\mathsf{L}(F(x))) > d$ as well. $\mathsf{L}(x)$ changes the faults only in magnitude while the localization of the faults is preserved. Moreover, the behavior of the faulty state is the same if the faults are injected during the computation of the affine transformation.

- **Addition of two secrets (`Add`):** Two secret values $f_0 \oplus g_0$ can be added by pairwisely adding shares. Each player P_i computes $F_i \oplus G_i$. According to our fault model, if only one polynomial is faulty, i.e. has a degree greater than d, then the degree of the resulting polynomial will be also greater than d and therefore the faults are propagated. However, an attacker can inject faults to both polynomials in different or in the same shares. In these cases, there is a probability that faults can become undetectable. The error detection terms can be zero if corresponding coefficients of $F(x)$ and $G(x)$ are equal, that is:

$$f_{d+1} \oplus g_{d+1} = \ldots = f_{n-1} \oplus g_{n-1} = 0.$$

  As we assumed, $deg(F(x)) > d$ and $deg(G(x)) > d$, $f_i \neq 0$ and $g_j \neq 0$ for at least one $i, j \in \{d+1, \ldots, n-1\}$. So, $Pr[(f_i \oplus g_i = 0)_{d+1 \leq i < n}] \approx (1/|\mathbb{F}|)^{d+\varepsilon}$. [2]

- **Efficient squaring operation ($\texttt{Sqr}_k$):** Efficient squaring operations can be used to eliminate costly multiplications in $\mathrm{GF}(2^m)$ [RP12]. Squaring can be defined as $\eta_k(y) = y^{2^k}$ and requires two conditions on public points α_i :

  1. $\alpha_i \neq 0$ for $i = 0, \ldots, n-1$.
  2. For every α_i there exists α_j such that $\alpha_i^2 = \alpha_j$.

  Each player calculates the operation on its share locally by $\eta_k(F_i)$. The family of shares $(\eta_k(F_i))_{i=0,\ldots n-1}$ is a valid secret share of $f_0^{2^k}$. However, communication between players is needed to do the reordering of the secret shares. Faulty shares are preserved as in the affine transformation.

- **Multiplication of two secrets (`Mult`):** The multiplication of two secret values $f_0 \cdot g_0$ requires communication between players. An efficient algorithm was proposed by Gennaro et al. [GRR98], simplifying the original SMC multiplication proposed by Ben-Or et al. [Ben88]:

  1. Each player P_i computes $H_i = F_i \cdot G_i$.
  2. Each player P_i generates a degree d polynomial $\mathcal{Q}_i(x)$ such that $\mathcal{Q}_i(0) = H(\alpha_i)$, and sends the value $\mathcal{Q}_i(\alpha_j)$ to player P_j. In this step, faults in $F(x)$ and $G(x)$ spread to all shares and they become undetectable, since $\mathcal{Q}_i(x)$ is a degree d polynomial.
  3. Each player P_i computes its secret share by $\mathbf{Q}_i = \sum_{j=0}^{n-1} \lambda_j^0 \mathcal{Q}_{j,i}$ and gets a *valid* (n, d)-secret sharing of the faulty secret value.

  The shares calculated in step 1 cannot be used as a valid secret sharing because of two main problems: (1) the polynomial is a degree $2d$ polynomial. (2) It is not a

[2] The exact probability can be calculated as $(1/|\mathbb{F}^*|) \cdot (1/|\mathbb{F}|)^{d+\varepsilon-1}$.

random polynomial [Ben88]. In step 2 and 3, degree reduction and randomization are done in order to generate a *proper* (n, d)-secret sharing of $f_0 g_0$. As a result, the output shares are always be a valid secret sharing and an adversary can inject faults without detection.

Remark 1. Castagnos et al. [CRZ13] suggested an improvement of the SMC multiplication by using the connection between Reed-Solomon codes [Ber68] and Shamir's secret sharing scheme. Although Reed-solomon codes provide extensions and generalization of the sharing [MS81], the improved secure multiplications have the same undetectable fault problem. The improvement focuses on the randomization part of the multiplication scheme (or *encoding* procedure as stated in [CRZ13]). The output is produced by the *Re-encoding* algorithm ([CRZ13], Algorithm 3). The output of the algorithm is always a valid secret state. Therefore, a faulty input will always result in a valid secret sharing and faults become undetectable.

4 Error Preserving Multiparty Computation

The error-preserving multiparty computation scheme below differs significantly from other proposals, such as robust SMC. Unlike, e. g., [GRR98, GIP$^+$14], detecting errors after each operation is not convenient in many cryptographic implementations, as it can reveal critical information. The basic ideas of our scheme are as follows:

- **Error Detection Only:** Our scheme does neither try to correct errors, nor detect where the error occurred. As in most application scenarios, the scheme only aims at detecting the errors and ensures that the attacker cannot learn anything from a faulty output.
- **Fault Detection Without Leaking Information:** The scheme aims to eliminate the leakages that can occur during the detection and keep the fault detection probability as one.
- **Error-Preserving Computation:** Once an errors occurs, it will spread through the state and remain part of the state. The advantage of this is that error detection can be performed once an output is produced.
- **Infective Computation:** If an error occurs, it is important to ensure the output does not reveal information to the attacker. We show that the randomization property of the secret sharing together with the redundant error detection coefficients ensure random outputs of faulty parts of the state if an error occurs.

Most of these goals can be achieved with the SMC described in [RP12] in a straightforward manner. However, the *multiplication* is difficult to construct in such a way that error detection is not performed once for each multiplication on each input and output. Instead, we propose a new multiplication engine that, in addition to the shared inputs and outputs, also uses additional shared error detection coefficients. The main advantage of error detection coefficients is that they add redundancy while only introducing minor overhead. In summary, all circuits can be represented by a classic SMC addition, our updated SMC multiplication, and a new recombination step. SMC squaring and affine transformation can still be used as before, as they do not influence the fault propagation.

4.1 Error Preserving Multiplication (`EPMult`)

Multiplication is the most critical SMC operation. Even without error detection, multiplication is the reason why $n > 2d$ is required, since the product of two degree-d polynomials is of degree-$2d$, To achieve error detection, more shares are needed. In fact, we show

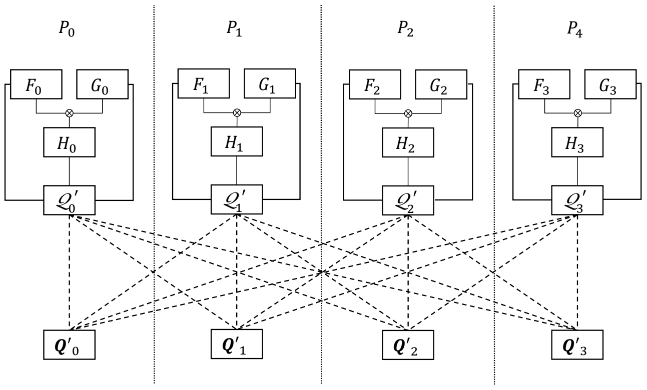

Figure 1: Detailed visualization of (4,1)-SMC Multiplication. Dashed lines show the communications between players (P_i). The shares of error detection terms of $H(x)$, $F(x)$, and $G(x)$ are forwarded to shares of f_0g_0 during the randomization step. Thus, each $\boldsymbol{Q}'_i$ contains a share of error detection terms of $F(x)$, $G(x)$, or $H(x)$.

that to detect ε errors, a total of $n > 2d + \varepsilon$ shares are needed. A representation of the (4,1)-error preserving multiplication can be seen in Figure 1.

In the new scheme, the error propagation is achieved by using the *error detection terms* of input polynomials and the intermediate polynomial $H(x)$. A step-by-step description of our new multiplication scheme that can resist ε faults can be introduced as follows:

1. Each player P_i locally computes $H_i = F_i \cdot G_i$.
2. Each player P_i generates a degree d polynomial $\mathcal{Q}_i(x)$ such that $\mathcal{Q}_i(0) = H_i$ and evaluates the polynomial for the public points $\mathcal{Q}_i(\alpha_j)$ (denoted by $\mathcal{Q}_{i,j}$). The main difference in our scheme occurs in this step: P_i also calculates a share of error detection coefficients (denoted by $\mathsf{E}_{i,j}$) of H_i or $G_i \oplus F_i$ with the corresponding Vandermonde element. The resulting $\mathcal{Q}'_{i,j} = \mathcal{Q}_{i,j} \oplus \mathsf{E}_{i,j}$ is sent to player P_j for $j = 0, \ldots, n-1$ and $j \neq i$. $\mathsf{E}_{i,j}$ is defined as follows:

$$\mathsf{E}_{i,j} = \begin{cases} \frac{\lambda_i^{n-j}}{\lambda_i^0} H_i & \text{if } 0 \leq j < \varepsilon \\ \frac{\lambda_i^{n-j}}{\lambda_i^0}(F_i \oplus G_i) & \text{if } \varepsilon \leq j < \varepsilon + d \\ 0 & \text{if } \varepsilon \leq j \leq n-1 \end{cases} \cdot \tag{3}$$

3. In the third step, each player calculates its share by $\mathbf{Q}'_i = \sum_{j=0}^{n-1} \lambda_i^0 \mathcal{Q}'_{j,i}$.

Remark 2. The 11th line in the Algorithm 1 corresponds to the error propagation. Without it, the `EPMult` corresponds to the SMC-Multiplication defined by Gennaro et al. [GRR98] and applied by Roche and Prouff [RP12]. The advantage of $\mathsf{E}_{i,j}$ is that it ensures the propagation of error detection terms as faults to the output by using only the local information. Therefore, the first $\varepsilon + d$ players implicitly get an error detection coefficient of $H(x)$ or $F(x) \oplus G(x)$. For example, player P_i $(0 \leq i < \varepsilon)$ calculates its share $\mathbf{Q}'_i$ by:

$$\begin{aligned} \mathbf{Q}'_i &= \sum_{j=0}^{n-1} \lambda_i^0 \mathcal{Q}'_{j,i} = \sum_{j=0}^{n-1} \lambda_i^0 (\mathcal{Q}_{j,i} \oplus \mathsf{E}_{j,i}) \\ &= \left[\lambda_0^0 \mathcal{Q}_{0,i} \oplus \lambda_0^{n-i} H_0\right] \oplus \ldots \oplus \left[\lambda_{n-1}^0 \mathcal{Q}_{n-1,i} \oplus \lambda_{n-1}^{n-i} H_{n-1}\right] \\ &= \underbrace{\left[\lambda_0^0 \mathcal{Q}_{0,i} \oplus \ldots \oplus \lambda_{n-1}^0 \mathcal{Q}_{n-1,i}\right]}_{=\mathbf{Q}_i \text{ in Section 3 SMC mult.}} \oplus \underbrace{\left[\lambda_0^{n-i} H_0 \oplus \ldots \oplus \lambda_n^{n-i} H_{n-1}\right]}_{=h_{n-i-1} \text{ by Equation (1)}} \\ &= \mathbf{Q}_i \oplus h_{n-i-1}. \end{aligned}$$

Algorithm 1 Error Preserving Multiplication (`EPMult`)

```
Input: Shares of $f_0$ as $(F_i)_{0 \leq i < n}$ and shares of $g_0$ as $(G_i)_{0 \leq i < n}$.
Output: Shares of $f_0 g_0$ as $(\mathbf{Q}_i)_{0 \leq i < n}$.
 1: for $i = 0$ to $n-1$ do
 2:     $H_i \leftarrow F_i G_i$
 3: end for
 4: for $i = 0$ to $n-1$ do
 5:     $(r_{i,1}, \ldots, r_{i,d}) \leftarrow \mathbb{F}_{2^m}$          ▷ Coefficients of the random polynomial.
 6:     for $j = 0$ to $n-1$ do
 7:         $\mathcal{Q}_{i,j} \leftarrow H_i$                              ▷ referred to as $\mathcal{Q}^0_{j,i}$.
 8:         for $k = 1$ to $d$ do                                        ▷ Evaluate the polynomial.
 9:             $\mathcal{Q}_{i,j} \leftarrow \mathcal{Q}_{i,j} \oplus r_{i,k}\alpha_j^k$    ▷ referred to as $\mathcal{Q}^k_{j,i}$.
10:         end for
11:         $\mathcal{Q}_{i,j} \leftarrow \mathcal{Q}_{i,j} \oplus \mathtt{E}_{i,j}$        ▷ Add a share of an error detection term.
12:         $\mathbf{Q}_j \leftarrow \mathbf{Q}_j \oplus \lambda_i^0 \mathcal{Q}_{i,j}$       ▷ referred to as $\mathbf{Q}_{j,i}$.
13:     end for
14: end for
15: return $(\mathbf{Q}_0, \ldots, \mathbf{Q}_{n-1})$
```

The propagation of the faults within the output shares can be summarized as follows:

$$\mathbf{Q}'_i = \begin{cases} \mathbf{Q}_i \oplus h_{n-i-1} & \text{if } 0 \leq i < \varepsilon \\ \mathbf{Q}_i \oplus g_{n-i-1} \oplus f_{n-i-1} & \text{if } \varepsilon \leq i < \varepsilon + d \\ \mathbf{Q}_i & \text{if } \varepsilon + d \leq i < n \end{cases},$$

where h_i, g_i and f_i represent i^{th} degree the coefficients of H, G and F, respectively. Remark that, $(h_i)_{2d<i<n} = 0$ and $(f_i \oplus g_i)_{d<i<n} = 0$ for valid secret sharing schemes.

4.2 Fault Detection Operation (`FDect`)

In order to maintain the error detection probability as one, faults need to be detected before each multiplication and addition. Detection can be performed according to the Vandermonde representation. However, this operation can leak sensitive information. Therefore, we add a randomization step to provide the confidentiality of the secret value.

1. ***Randomization:*** This step adds a random degree-d polynomial to the shared secret, thereby masking the secret value, but not deleting fault information.
   (a) A random degree d polynomial $\mathcal{R}(x)$ is generated, the value $\mathcal{R}(\alpha_i)$ is sent (denoted by $\mathcal{R}_i$) to player P_i.
   (b) Each player calculates the new share by simply adding the random share to their own share.
   $$F_{\mathcal{R}_i} = F_i \oplus \mathcal{R}_i \text{ for } i = 0, \ldots n-1.$$
2. ***Detection:*** Error detection coefficients are reconstructed in the natural way as given in Equation (1):
   $$f_j = \sum_{i=0}^{n-1} \lambda_i^j F_{\mathcal{R}_i} \text{ for } j = d+1, \ldots, n-1.$$

The randomized share of a player (denoted by $F_{\mathcal{R}_i}$) corresponds to a secret sharing of a random value. Therefore, the reconstruction cannot leak information about the real secret value. Moreover, $F_{\mathcal{R}}(x) = F(x) \oplus \mathcal{R}(x)$, and we know that $deg(\mathcal{R}_i(x)) \leq d$. Therefore, if $F(x)$ is faulty, i.e. $deg(F(x)) > d$, then $deg(F_{\mathcal{R}}(x)) > d$, and fault detection can be achieved easily in the *detection* step by reconstructing and checking the error detection terms. The details and security features of the operation can be found in Appendix A.1.

Remark 3. While in-circuit fault detection is an option, our aim is not to give any error message or stop the execution. The scheme outputs the faulty ciphertext even if the fault is detected. The degree of the polynomial is used as a fault flag. The output can then be randomized using this flag.

4.3 Recombination Operation (ReComb) and Infective Computation

In order to avoid costly fault detection operations, we first propose a recombination operation which detects the faults when the output is produced. We explain the *infectiousness* of the faults while introducing a recombination algorithm.

The recombination operation is composed of two main steps, ***re-sharing*** and ***reconstruction***. The main idea is to share the secret variable while adding randomized error detection terms. The first part can be seen as a modified version of `EPMult` using a different $\mathsf{E}_{i,j}$. The inputs of the operation are secret shares F_i for $0 \leq i < n$ and a random vector $(\mathsf{r}_0, \ldots, \mathsf{r}_{\varepsilon+d-1})$ where $\mathsf{r}_i \in \mathrm{GF}(2^8) \setminus 0$. The outputs are the secret value f_0 and a fault decision.

1. ***Re-Sharing***: Players share the secret value as in the second part of the `EPMult`, the only difference is that we update $\mathsf{E}_{i,j}$ as follows:

$$\mathsf{E}^R_{i,j} = \begin{cases} \mathsf{r}_j \frac{\lambda_i^{n-(j-1)}}{\lambda_i^0} F_i & \text{if } 0 \leq j < d + \varepsilon \\ 0 & \text{if } d + \varepsilon \leq j \leq n-1 \end{cases}.$$

   The adversary is still able to get information from the output, so we ensure the randomization of the secret value by using fresh random values.

   (a) Each player P_i generates a degree d polynomial $\mathcal{Q}_i(x)$ such that $\mathcal{Q}_i(0) = F_i$ and evaluates the polynomial for the public shares $\mathcal{Q}_i(\alpha_i)$ (denoted by $\mathcal{Q}_{i,j}$) and sends the value $\mathcal{Q}'_i(x) = \mathcal{Q}_i(x) \oplus \mathsf{E}^R_{i,j}$ to player P_j.

   (b) Each player calculates its new share $\mathbf{Q}'_i$ by $\sum_{j=0}^{n-1} \lambda_i^0 \mathcal{Q}'_{j,i}$.

   Remark 4. As in the `EPMult`, the first $\varepsilon + d$ players implicitly get the randomized error detection coefficient of $F(x)$.

$$\mathbf{Q}'_i = \begin{cases} \mathbf{Q}_i \oplus \mathsf{r}_i f_{n-i-1} & \text{if } 0 \leq j < \varepsilon + d \\ \mathbf{Q}_i & \text{if } \varepsilon + d \leq j < n \end{cases}.$$

Reconstruction: In the last step, the secret value and error detection coefficients are reconstructed using Equation (1):

$$f_j = \sum_{i=0}^{n-1} \lambda_i^j \mathbf{Q}'_i \text{ for } j = d+1, \ldots, n-1 \text{ and } 0.$$

Clearly, if F is faulty, then at least one of the error detection terms is non-zero. In the second step, the secret value is randomized by these terms, and, therefore, the output is randomized in case of fault injections. Thus, infective computation is achieved. The details and security features of the operation can be found in Appendix A.2.

5 Security Analysis

Previously proposed schemes that combined countermeasures against side channel analysis and fault injection have different security claims depending on their used adversarial models. The SCA security of works like [IPSW06] are based on the ISW transformation [ISW03], while others [SMG16, DCN16] are based on Threshold implementations. Both the ISW transformation and TI can achieve t^{th}-order security. However, it was recently shown Moos et al. [MMSS18] that TI and derived schemes can have security issues due to the insufficient refreshing in higher order variants and thus require special care during implementation. There are bigger differences in the fault resistance properties of the proposed schemes. For example, in [IPSW06] security against *reset attacks* and *set, reset* and *toggle attacks* are formally proven. In [SMG16] authors defined a notion called *Coverage* to quantify the fault coverage of their scheme. They analyzed the fault resistance of the scheme using this notion. Similarly, in [RMB$^+$17] the authors examined the conditions where faults are undetectable. In this section, we discuss the security features of our combined countermeasure under specific attack models. We formally prove the t-probing security of individual operations and analyze the side channel resistance of a combination of operations. Then a similar discussion in [SMG16, RMB$^+$17] will be done to explain the fault resistance of our scheme. First we define a new notion called *Propagation*, which states the probability of detecting faults using output shares if the input shares are faulty. Then we examine the conditions where faults are undetectable for each individual operation and for a combination of operations.

5.1 Side Channel Resistance

In this section, we formally prove the t-probing security of the error preserving multiplication scheme defined in Section 4.1. The following theorem shows the security of `EPMult`.

Theorem 1 (t-$\mathbf{SNI}^n_d$ of `EPMult`)**.** *Let $(F_i)_{0\leq i<n}$ and $(G_i)_{0\leq i<n}$ be the input shares of the Error Preserving Multiplication operation, and let $(\boldsymbol{Q}_i)_{0\leq i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 < d + 1$, there exist two subsets I and J of indices with $|I| \leq t_1$ and $|J| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $\boldsymbol{Q}_{|\mathcal{O}}$ can be perfectly simulated from $F_{|I}$ and $G_{|J}$.*

Proof. In the first part of the proof, we construct the sets of the input share indices I and J depending on the intermediate variables that are probed. We denote U as the intersection of I and J. We divide the probes into 2 groups.

- **Group 1:** If F_i or G_i is probed, add i to I or J, respectively. If H_i, $\mathcal{Q}^0_{i,j}$ or $\mathsf{E}_{i,j}$ is probed, add i to I and J.
- **Group 2:** If $r_{i,j}$ or $\mathcal{Q}^k_{i,j}$ where $k \in \{1, \dots, d\}$ is probed, add i to I and J.

According to our selection, we add at most one index to I and J for each probe and, therefore, $|I| \leq t_1$ and $|J| \leq t_1$.

1. The simulations of the probed variables in group 1 are straightforward. Since $i \in I$ ($i \in J$ resp.), we can perfectly simulate F_i (G_i resp.), because F_i and G_i are known values. Similarly, we can simulate H_i and $\mathcal{Q}^0_{i,j}$, since $i \in I$ and $i \in J$. Finally, since the elements λ_i of the inverse Vandermonde matrix are public variables, we can simulate $\mathsf{E}_{i,j}$ as defined in Equation (3).
2. If $\mathcal{Q}^k_{i,j}$ is probed, we need to consider two cases:

Algorithm 2 Mask Refreshing (`RefreshM`)

```
Input: Shares of $f_0$ as $(F_i)_{0 \leq i < n}$.
Output: Shares of $f_0$ as $(C_i)_{0 \leq i < n}$.
1: $(r_1, \ldots, r_d) \leftarrow \mathbb{F}_{2^m}$          ▷ Coefficients of the random polynomial.
2: for $j = 0$ to $n - 1$ do
3:     for $k = 1$ to $d$ do                                ▷ Evaluate the polynomial.
4:         $\mathcal{Q}_j \leftarrow \mathcal{Q}_j \oplus r_k \alpha_j^k$      ▷ referred to as $\mathcal{Q}_j^k$.
5:     end for
6:     $C_j \leftarrow F_j \oplus \mathcal{Q}_j$
7: end for
8: return $(C_0, \ldots, C_{n-1})$
```

- If $r_{i,k}$ is also probed, we leave $r_{i,k}$ as in the real circuit, therefore, we can simulate $\mathcal{Q}_{i,j}^k$ as $H_i \oplus r_{i,k}\alpha_j^k$ where α_j is a public value.
- If $r_{i,k}$ is not probed, it does not enter into the computations of $\mathcal{Q}_{i,j}^k$, therefore, we can perfectly simulate $\mathcal{Q}_{i,j}^k$ with a random value.

3. If $\mathbf{Q}_{j,i}$ is probed, we need to consider two cases as in the previous step:
   - If all the values $r_{i,k}$ for $1 \leq k \leq d$ are probed, we can perfectly simulate the values $\mathcal{Q}_{i,j}^k$, and hence $\mathbf{Q}_{j,i}$ can be simulated. Remark that, the λ_i^0 is an element of the inverse Vandermonde matrix so it is a public value.
   - If at least $r_{i,k}$ for $1 \leq k \leq d$ is not probed, that means $r_{i,k}$ does not enter into the computation of $\mathbf{Q}_{j,i}$, therefore $\mathbf{Q}_{j,i}$ can be simulated by a random value.

Now we explain how to simulate output shares $\mathbf{Q}_i$ for all $i \in \mathcal{O}$ where $\mathcal{O}$ is an arbitrary subset of $[1, n]$ with t_2 elements such that $t_1 + t_2 < d + 1$. Clearly, using t_1 probes, we can observe at most t_1 intermediate variables of $\mathbf{Q}_i$, where $\mathbf{Q}_i$ can be written as: $\mathbf{Q}_i = \sum_{j=0}^{n-1} \lambda_i^0 \mathcal{Q}_{j,i}$. Since $t_1 + t_2 < d + 1$, at least one intermediate variable of $\mathbf{Q}_i$ is not probed. Therefore, we can simulate $\mathcal{Q}_{j,i}$ with $j \notin U$ by generating a random degree d polynomial and evaluating it for α_i. Hence, we can simulate $\mathbf{Q}_i$ for each $i \in \mathcal{O}$. □

We show that any set of t_1 intermediate variables and any subset of t_2 output shares can be perfectly simulated by at most d independent and uniformly distributed variables. Therefore, we can say that `EPMult` is secure in the d-probing model which is followed by security in the noisy leakage and bounded moment leakage models [DDF14, BDF+17].

Next, we give the security notion of SMC-addition, affine transformation, and efficient squaring. These operations perform sharewise computations. Therefore, they can be computed using affine gadgets as defined in [BBD+15]. As stated in [BBD+15], an algorithm is said to be t-NI if all gadgets are t-NI and every non-linear usage of a secret state is guarded by t-SNI (or t-SNI$_d^n$ as in our re-formulation) refreshing gadgets. Moreover, it is sufficient to make the algorithm t-SNI, if every input or the output of a t-NI algorithm is processed by a t-SNI gadget. The mask refreshing operation denoted by `RefreshM` can be defined as in Algorithm 2. `RefreshM` ensures the independence of the inputs of the `EPMult` operation and we can implement an arbitrary function with t-SNI$_d^n$ security.

The following theorem provides the security of our `RefreshM` operation. We provide the proof in Appendix A.3.

Theorem 2 (t-SNI$_d^n$ of `RefreshM`)**.** *Let $(F_i)_{0 \leq i < n}$ be the input shares of `RefreshM` and let $(C_i)_{0 \leq i < n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 < d + 1$, there exist a subset I of indices with $|I| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $C_{|\mathcal{O}}$ can be perfectly simulated from $F_{|I}$.*

5.2 Fault Resistance

First, we discuss the resistance capabilities of the systems using fault detection operations. As given in Section 3, affine transformation and efficient squaring operations can be listed as fault preserving, while addition and multiplication are fault preserving with high probability. Therefore, fault detection operations should be used before all multiplication and addition operations to ensure perfect fault detection. However, depending on the circuit and the number ε, this number can be decreased. Therefore, an optimal point between performance and security can be achieved. The advantage of using fault detection operations is that it can be carried out without leaking sensitive information.

Next, we discuss the fault resistance features of the proposed scheme without using the fault detection operation. The fault detection mechanism relies on the degree of the polynomial generated by the secret state. We illustrate the methodology with the following example: Assume $(4,1)$-secret sharing $(F_0,\ldots,F_3)$ is used to share a secret. Clearly, the secret sharing polynomial $deg(F(x)) \leq 1$. Assume the fault, denoted by σ, is injected to the second share. After the injection, the secret sharing polynomial $F'(x)$ becomes a polynomial generated by the points $(F_0, F_1 \oplus \sigma, F_2, F_3)$. Remark that, as stated by the additive fault model, the relation between polynomials can be seen as $F'(x) = F(x) \oplus \Delta(x)$. Error polynomial $\Delta(x)$ has one nonzero point and three zero points. In other words $\Delta(x)$ is generated by the points $(0, \sigma, 0, 0)$. Clearly these points belong to an at least degree-3 polynomial. Hence, we can detect the fault by looking at the degree of the polynomial resulting $F'(x)$. Clearly, the only way of generating undetectable faults is arranging $\Delta(x)$ as a degree-1 polynomial.

Using this motivation, we introduce the following lemma which constitutes a basis of our error detection method.

Lemma 1. *Let $(F_i)_{i=0\ldots,n-1} \in \mathbb{F}$ represent an (n,d)-secret sharing of $f_0 \in \mathbb{F}$ with $n = d+\varepsilon+1$ and $\Delta(x)$ represents the polynomial generated by the faults. If k faults are injected to secret states, generating an error polynomial degree greater than d is,*

$$Pr[deg(\Delta(x)) > d] = \begin{cases} 1 & k \leq d+\varepsilon \\ 1 - \frac{|\mathbb{F}|^{k-\varepsilon}-1}{|\mathbb{F}|^{k}-1} & k > d+\varepsilon \end{cases}.$$

Proof. Let $\alpha_0,\ldots,\alpha_{n-1} \in \mathbb{F}$ be public evaluation points and $F(x) = f_0 \oplus f_1 x \oplus \ldots \oplus f_d x^d \in \mathbb{F}[x]$ be the secret sharing polynomial. Without loss of generality, assume there exist k faults in the first k secret shares and let us denote the corresponding error polynomial by $\Delta(x) = \delta_0 \oplus \delta_1 x \oplus \ldots \oplus \delta_{d+1} x^{d+1} \oplus \ldots \oplus \delta_{d+\varepsilon} x^{d+\varepsilon} \in \mathbb{F}[x]$. From Equation (2), the relation between faulty shares and coefficients of $F(x)$ and $\Delta(x)$ can be seen as follows:

$$V^{-1} \underbrace{\begin{pmatrix} F_0 \oplus \sigma_0 \\ \vdots \\ F_{k-1} \oplus \sigma_{k-1} \\ F_k \\ \vdots \\ F_{n-1} \end{pmatrix}}_{\text{Faulty state}} = V^{-1} \underbrace{\begin{pmatrix} F_0 \\ \vdots \\ F_{k-1} \\ F_k \\ \vdots \\ F_{n-1} \end{pmatrix}}_{\text{Points of } F(x)} \oplus V^{-1} \underbrace{\begin{pmatrix} \sigma_0 \\ \vdots \\ \sigma_{k-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{\text{Points of } \Delta(x)} = \begin{pmatrix} f_0 \\ \vdots \\ f_d \\ 0 \\ \vdots \\ 0 \end{pmatrix} \oplus \begin{pmatrix} \delta_0 \\ \vdots \\ \delta_d \\ \delta_{d+1} \\ \vdots \\ \delta_{n-1} \end{pmatrix}$$

where $V = (\alpha_j^i)$ is the $n \times n$ Vandermonde matrix. As seen in the above equation $\Delta(x)$, is the polynomial generated by the points $(\sigma_0,\ldots,\sigma_{k-1},0,\ldots,0)$, i.e. it has $n-k$ zero points and ,therefore, it generates at least a degree $n-k$ polynomial.

Assume that $k \leq d+\varepsilon$. Since $n = d+\varepsilon+1$, it is clear that $n-k \geq d+1$. Therefore, $deg(\Delta(x)) > d$ with probability 1 and faults are always detectable.

Assume $k > d + \varepsilon$. Then we need to focus on the Vandermonde representation of the secret shares. Using $V^{-1}_{i,j} = \lambda^i_j$, we can form a system of linear equations for error detection terms of $\Delta(x)$:

$$\begin{aligned}
\sigma_0\lambda_0^{n-1} \oplus \ldots \oplus \sigma_{k-1}\lambda_{k-1}^{n-1} &= \delta_{n-1} \\
\sigma_0\lambda_0^{n-2} \oplus \ldots \oplus \sigma_{k-1}\lambda_{k-1}^{n-2} &= \delta_{n-2} \\
&\vdots \\
\sigma_0\lambda_0^{n-\varepsilon} \oplus \ldots \oplus \sigma_{k-1}\lambda_{k-1}^{n-\varepsilon} &= \delta_{d-\varepsilon}
\end{aligned}$$

To arrange $\Delta(x)$ as degree d polynomial, the above equations should be solved for $\delta_i = 0$ for $i = (d+1, d+2, \ldots, d+\varepsilon)$. The parameters of this homogeneous system of linear equations are k unknowns and ε equations. Since $k > \varepsilon$, the number of non-trivial solutions for this system is $|\mathbb{F}|^{k-\varepsilon} - 1$. Therefore, $Pr[deg(\Delta(x)) \leq d] = \frac{|\mathbb{F}|^{k-\varepsilon}-1}{|\mathbb{F}|^k-1}$. Hence,

$$Pr[deg(\Delta(x)) > d] = 1 - \frac{|\mathbb{F}|^{k-\varepsilon} - 1}{|\mathbb{F}|^k - 1}.$$

□

Next, we can state the following theorem, to clarify our fault detection properties. Remark that the following theorem was already proven in [YO13, Sec. 4.1].

Theorem 3. *Let $F'(x)$ be the faulty secret sharing polynomial. If $deg(F'(x)) > d$, the faults can be detectable.*

Using this theorem and the notation given by Schneider et al. [SMG16], we can define the fault coverage of our scheme. Let $F'(x)$ be the faulty secret sharing polynomial, then the probability of a set of faults to be undetectable is defined as our fault coverage:

$$Coverage_\varepsilon = 1 - Pr[deg(F'(x)) \leq d].$$

Assume the number of injected faults to the system is k, in the first multiplication, faults are propagated with probability 1 if $k \leq \varepsilon$ as given in Lemma 1. However, we cannot use Lemma 1 as the fault coverage for a set of operations, since faults can be injected in different instances or one fault can spread to a large number of shares. As a result, faults become unstable and potentially undetectable. In a sequence of operations, faults can become undetectable after an SMC addition or multiplication. In the following, we perform the security analysis and derive the probability of undetectable faults in SMC multiplication.

Corollary 1 ($Propagation_\varepsilon$(`EPMult`)). *Let $(F_i)_{0\leq i<n} \in \mathbb{F}$ and $(G_i)_{0\leq i<n} \in \mathbb{F}$ be the input shares of the Error Preserving Multiplication operation and let $(\boldsymbol{Q}_i)_{0\leq i<n}$ be the output shares. And let us denote the sets of faulty indices as k_F and k_G respectively, with $k = |k_F \cup k_G|$. If the fault is detectable using both of the set of input shares $(F_i)_{0\leq i<n}$ and $(G_i)_{0\leq i<n}$, the faults can be detectable using output shares $(\boldsymbol{Q}_i)_{0\leq i<n}$ with the following probability:*

$$Propagation_\varepsilon(\texttt{EPMult}) = \begin{cases} 1 - \frac{1}{q^{d+k}} & k \leq \varepsilon \\ 1 - \frac{1}{q^d}\left(\frac{1}{(q+1)^e} + \frac{1}{q^k}\right) & k > \varepsilon \end{cases}, \text{ where } |\mathbb{F}^*| \text{ is denoted by } q.$$

Proof. Assume there exists 2 sets of faults within the input shares $(F_i)_{0\leq i<n}$ and $(G_i)_{0\leq i<n}$, such that if i^{th} share F_i (resp. G_i) is faulty, then $\sigma_{F_i} \neq 0$ (resp. $\sigma_{G_i} \neq 0$) otherwise $\sigma_{F_i} = 0$ (resp. $\sigma_{G_i} = 0$). And clearly, $k_F = \{i|\sigma_{F_i} \neq 0\}$ and $k_G = \{i|\sigma_{G_i} \neq 0\}$. The

polynomial $H(x)$ is generated by the shares $F_i \cdot G_i$ for $0 \le i < n$ and the faults in H_i can be calculated as follows:

$$H_i = [F_i \oplus \sigma_{i_F}] \cdot [G_i \oplus \sigma_{i_G}] = F_i G_i \oplus \underbrace{F_i \sigma_{i_G} \oplus G_i \sigma_{i_F} \oplus \sigma_{i_F}\sigma_{i_G}}_{\text{The fault in } H_i \text{ denoted by } \sigma_{H_i}.}$$

Remark that $\mathsf{E}_{i,j}$ is used to propagate the faults. The faults in $(\mathbf{Q}_i)_{0 \le i < n}$ become undetectable if and only if the following equations, which correspond to the partial sums in step 3, hold:

$$\sum_{j=0}^{n-1} \lambda_i^0 \mathsf{E}_{i,j} = 0 \text{ for } 0 \le i < \varepsilon + d. \tag{4}$$

From the definition of $\mathsf{E}_{i,j}$ we know that the variables correspond to shares error detection terms of $F(x) \oplus G(x)$ or $H(x)$. Hence, faults become undetectable if and only if error detection terms are zero, i.e. the following equations hold:

1. $f_{d+1} \oplus g_{d+1} = \ldots = f_{2d} \oplus g_{2d} = 0$. Remark that we assumed the fault is detectable using both of the sets of input shares $(F_i)_{0 \le i < n}$ and $(G_i)_{0 \le i < n}$, therefore, at least one $f_i \neq 0$ and $g_i \neq 0$ where $i \in \{d+1, \ldots, 2d\}$. Therefore,

$$Pr[(f_i \oplus g_i = 0)_{d < i \le 2d}] = \frac{1}{q}\left(\frac{1}{q+1}\right)^{d-1} \approx \frac{1}{q^d}.$$

2. $h_{2d+1} = \ldots = h_{2d+\varepsilon+1} = 0$. In other words $deg(H(x) \le 2d)$. And $deg(H(x) \le 2d)$ if and only if

   (a) The polynomial generated by σ_{H_i}'s is at most a degree $2d$ polynomial or

   (b) $\sigma_{H_i} = 0$ for $i = 0, \ldots, n-1$.

   From Lemma 1, the probability of condition (a) is:

$$Pr[deg(H(x)) \le 2d] = \begin{cases} 0 & k \le \varepsilon \\ \frac{(q+1)^{k-\varepsilon}-1}{(q+1)^k - 1} \approx \frac{1}{(q+1)^\varepsilon} & k > \varepsilon \end{cases}.$$

   And the probability of condition (b) is:

$$Pr[(\sigma_{H_i} = 0)_{0 \le i < n}] \approx \frac{1}{q^k}.$$

Using the conditions listed above we can calculate the probability of the Equation (4) to be hold is:

$$Pr\left[(\sum_{j=0}^{n-1} \lambda_i^0 \mathsf{E}_{i,j} = 0)_{0 \le i < \varepsilon + d}\right] = \begin{cases} \frac{1}{q^{d+k}} & k \le \varepsilon \\ \frac{1}{q^d}\left(\frac{1}{(q+1)^\varepsilon} + \frac{1}{q^k}\right) & k > \varepsilon \end{cases}.$$

Therefore,

$$Propagation_\varepsilon(\texttt{EPMult}) = \begin{cases} 1 - \frac{1}{q^{d+k}} & k \le \varepsilon \\ 1 - \frac{1}{q^d}\left(\frac{1}{(q+1)^\varepsilon} + \frac{1}{q^k}\right) & k > \varepsilon \end{cases}.$$

□

Using propagation probabilities of individual operations, we can analyze the fault resistance of a composition of operations. The following theorem formally analyzes the fault resistance properties of a combination of operations. The main idea of the theorem is to examine the propagation of fault indices and calculate the individual $Propagation_\varepsilon$.

Theorem 4. *Let $A_1, \ldots, A_t$ be a sequence of operations and let us denote the sets of faulty indices of A_j as k_{F^j} and k_{G^j} respectively, with $k_j = |k_{F^j} \cup k_{G^j}|$. Without loss of generality let's assume that the fault is detectable using the inputs of A_i where $1 \le i \le t$. Then, $Propogation_\varepsilon$ can be calculated as follows:*

$$Propogation_\varepsilon(A_1, \ldots, A_t) = \prod_{j=i}^{t} Propogation_\varepsilon^{k_j}(A_j).$$

Proof. First, let us categorize operations into to two sets depending on the number of inputs as follows: $\mathcal{A}_1 = \{$`Affine`, `Sqr`, `RefreshM`, `FDect`$\}$[3] and $\mathcal{A}_2 = \{$`EPMult`, `Add`$\}$. Since a fault is detectable using the inputs of A_i, we know that the degree of the secret sharing polynomial is greater than d and $k_i > 0$ from Theorem 3. First, we analyze propagation of the number of faulty indices in three cases:

- **Case 1:** $k_{i+1} = k_i$ if $A_{i+1} \in \mathcal{A}_1$ and $Propagation_\varepsilon(A \in \mathcal{A}_1) = 1$.
  - $A_{i+1} \in \{$`Affine`, `Sqr`$\}$, then the magnitude of faults is changed however the number of faulty indices is preserved, as explained in Section 3.
  - $A_{i+1} \in \{$`RefreshM`, `FDect`$\}$. Since `RefreshM` and `FDect` can be seen as additions with a valid (i.e degree-d) secret sharing, the number of faulty indices are preserved with their magnitudes.
- **Case 2:** $0 \le k_{i+1} \le k_i$ if $A_{i+1} =$ `Add`, The number of fault indices can be decreased depending on the magnitudes and the indices of the faults. As explained in Section 3 $Propagation_\varepsilon(\texttt{Add}) = (1/q)^{d+\varepsilon}$.
- **Case 3:** $0 \le k_{i+1} \le \varepsilon + d$ if $A_{i+1} =$ `EPMult`, then the number of faulty indices changes depending of the Equation (4) in Corollary 1.

Using these discussions we analyzed the propagation of the number of faulty indices of each operation. Depending on the operation and number of fault indices we can calculate $Propogation_\varepsilon$ of operations individually. Hence, the following equation holds:

$$Propogation_\varepsilon(A_1, \ldots, A_t) = \prod_{j=i}^{t} Propogation_\varepsilon^{k_j}(A_j). \tag{5}$$

$\square$

Remark 5. In Theorem 4 we assumed that faults are injected before the i^{th} operation. The attacker can inject additional faults into the scheme, which can change the number of faulty indices. However, the propagation of faulty indices of individual operations works as analyzed in Theorem 4 and the propagation of faults can be calculated using the Equation (5) for a composition of individual operations in the presence of faults.

The infective computation property of our scheme is based on the $Propogation_\varepsilon$. As given in Section 4.3, the infective property of our scheme is provided by $\mathsf{E}^R_{i,j}$. If a fault is detectable using input shares $(F_i)_{0 \le i < n}$, then $f_i \le 0$ for at least one $i \in \{d+1, \ldots, n-1\}$

[3]We excluded the `ReComb` operation from the analysis, since it can only be the last operation. The case where a fault is detectable using the inputs of `ReComb` is already explained in Section 4.3.

Table 1: Number of operations in Gennaro et al. [GRR98] and `EPMult` *in Section 4.1, where Mul., Add. and Rand. represents the field multiplication, field addition, and randomness requirements respectively.*

	Gennaro et al. [GRR98]			`EPMult`			Overhead
	step 1	step 2	step 3	step 1	step 2	step 3	
Mul.	n	n^2d	n^2	n	$n^2d + n(\varepsilon + d)$	n^2	$n(\varepsilon + d)$
Add.	-	n^2d	$(n-1)n$	-	$n^2d + n(\varepsilon + 2d)$	$(n-1)n$	$n(\varepsilon + 2d)$
Rand.	-	nd	-	-	nd	-	-

from Theorem 3. As a result, the output shares are randomized by at least one nonzero coefficient and a random value. Therefore, the infective computation is achieved.

The analyzed fault model considers faults on intermediate states only. However, it has been shown that faulting the control flow, e.g. the number of rounds of a cipher, is also sufficient for key recovery [CT05, DMN+12]. We consider such attacks out of the scope of this work. Indeed, such attacks might not be a concern for fully unrolled circuits, but other implementation styles would require additional protection of the control flow to prevent such attacks.

5.3 Resistance Against Combined Attacks

After describing the side channel and fault resistance of our scheme, a natural question arises: what will be the security properties if an attacker is able to mount SCA and FA together? In this section we focus on two attacks described in [CFGR10]. Due to the infective properties of our scheme, the attacker will not be able to collect useful faulty ciphertexts. Secondly, even if the attacker successfully chooses ε faults in such a way that the shared values are fixed to a predefined value, the attacker should probe $d+1$ variables to recover the secret, which is not possible in our model. Therefore, the combined attacks as defined in [CFGR10] are naturally eliminated by the scheme. Another advantage of the scheme is that our fault model is defined as *blindly-chosen* and *non-adaptive*. Therefore, the attacker cannot observe the secret states and forge a fault to inject. The model inherently creates a timing limitation which eliminates *rushing adversary* [RMB+17]. As a result, *Propagation* probabilities are not affected by probing d variables. Moreover, fault injections targeting randomness sources to disable masking could be a serious threat to the system. In our model, the faults on randomness would change the randomness in a way the attacker cannot control, thereby keeping the side channel protection intact.

5.4 Performance Analysis

Next we analyze the performance of our scheme in terms of basic operations such as field multiplications, field additions and randomness requirements, and compare the perormance to related work. Table 1 compares the SMC multiplication of Gennaro et al. [GRR98] to the `EPMult` defined in Section 4.1 in terms of field additions (XOR), multiplications, and required fresh randomness. As shown in Table 1, performance overhead is only introduced in the second step, while calculating the $\mathsf{E}_{i,j}$. The additional costs of adding $\mathsf{E}_{i,j}$ are $n(\varepsilon + d)$ field multiplications and $n(\varepsilon + 2d)$ field additions. Except this overhead, both schemes have identical cost: each player generates a random degree-d polynomial and sends the corresponding values to the other players, requiring nd random values and n^2 polynomial evaluations, where each evaluation costs d field multiplications and d additions.

An overview of the computational cost for the SMC operations described in Sections 3 and 4.1 is provided in Table 2. The table lists the required number of field multiplications, additions, and the randomness requirements for every secure operation of an *(n,d)* masking scheme. Besides the arithmetic operations, the scheme requires the recombination and

Table 2: Number of field multiplications, additions, and randomness requirements for the SMC operations.

	`EPMult`	$\texttt{Sqr}_k$	`Add`	`Affine`	`RefreshM`
Field Mul.	$n^2(d+1)+n(\varepsilon+d+1)$	nk	-	n	nd
Field Add	$n^2(d+1)+n(\varepsilon+2d-1)$	-	n	n	nd
Randomness	nd	-	-	-	d

Table 3: Recombination Operation and Fault Detection Operation

	Recombination		Fault Detection	
	Re-Sharing	Reconstruction	Randomization	Detection
Mul.	$n^2(d+1)+n(2\varepsilon+2d+1)$	$n(\varepsilon+d+1)$	$n(d+1)$	$n(\varepsilon+d)$
Add.	$n^2(d+1)+n(\varepsilon+d-1)$	$(n-1)(\varepsilon+d+1)$	$n(d+2)$	$(n-1)(\varepsilon+d)$
Rand.	$\varepsilon+d+nd$	-	$d+1$	-

fault detection operations. The costs for both are listed in Table 3. A single recombination is more costly than an error preserving multiplication; however, only one recombination operation per secret value is needed, keeping the contribution to the overall cost small. The total overhead of the fault detection operation depends on the sequence of operations and security level of the implementation. In the first step, $(d+1)$ random values are needed to generate a random polynomial. The evaluation of the polynomial for all players costs n^2d field multiplications and n^2d field additions. Thus, the cost is small compared to the multiplication, allowing for frequent checks of the state.

Using the above performance analysis, we compare our results with other side channel countermeasures and one other combined side-channel fault countermeasure in Table 2. We consider the total number of field multiplications and additions of the side-channel protected scheme by Roche and Prouff [RP11], the scheme by Rivain and Prouff [RP10] and combined side-channel fault countermeasure (CAPA) by Reparaz et al. [RMB+17]. Remark that, our work and [RP11] operate on polynomial masking while [RMB+17] and [RP10] use Boolean masking, hence $d=n-1$. The only comparable numbers corresponds to a combined countermeasure are given in Table 1 in [RMB+17]. For CAPA, the number of operations shown in Table 2 include the computations required for public values, output calculations and MAC check functionalities [RMB+17]. While some operations like addition and square transformation cost more in [RMB+17], the secure multiplication is the bottleneck of our scheme when applied for higher order masking.

Table 4: Performance Comparison of Secure Operations in terms o ffield operations; field additions are shown in normal font while the number Field Multiplications are in bold font.

	SMC Multiplication	SMC Square	SMC Addition
Our Work	$n^2(d+1)+n(\varepsilon+d+1)$	n	-
	$\boldsymbol{n^2(d+1)+n(\varepsilon+2d-1)}$	**-**	$\boldsymbol{n}$
Roche-Prouff [RP11]	$n^2(d+1)+n$	n	-
	$\boldsymbol{n^2(d+1)-n}$	**-**	$\boldsymbol{n}$
CAPA [RMB+17]	$8n$	$2n$	-
	$\boldsymbol{11n+4n(n-1)+1}$	$\boldsymbol{2n^2+3n+1}$	$\boldsymbol{2n}$
Rivain-Prouff [RP10]	n^2	n	-
	$\boldsymbol{2n(n-1)}$	**-**	$\boldsymbol{n}$

Algorithm 3 $Exp254((F_i)_{0\le i<n})$

```
Input: Shares of $f_0$ as $(F_i)_{0\le i<n}$.
Output: Shares of $f_0^{254}$ as $(Y_i)_{0\le i<n}$.
1: $(Z_i)_{0\le i<n} = \texttt{Sqr}_1((F_i)_{0\le i<n})$    $\triangleright z \leftarrow f_0^2$
2: $(Z_i)_{0\le i<n} = \texttt{RefreshM}((Z_i)_{0\le i<n})$
3: $(Y_i)_{0\le i<n} = \texttt{EPMult}((Z_i)_{0\le i<n}, (F_i)_{0\le i<n})$    $\triangleright y \leftarrow f_0^3$
4: $(W_i)_{0\le i<n} = \texttt{Sqr}_2((Y_i)_{0\le i<n})$    $\triangleright w \leftarrow f_0^{12}$
5: $(W_i)_{0\le i<n} = \texttt{RefreshM}((W_i)_{0\le i<n})$
6: $(Y_i)_{0\le i<n} = \texttt{EPMult}((Y_i)_{0\le i<n}, (W_i)_{0\le i<n})$    $\triangleright y \leftarrow f_0^{15}$
7: $(Y_i)_{0\le i<n} = \texttt{Sqr}_4((Y_i)_{0\le i<n})$    $\triangleright y \leftarrow f_0^{240}$
8: $(Y_i)_{0\le i<n} = \texttt{EPMult}((Y_i)_{0\le i<n}, (W_i)_{0\le i<n})$    $\triangleright y \leftarrow f_0^{252}$
9: $(Y_i)_{0\le i<n} = \texttt{EPMult}((Y_i)_{0\le i<n}, (Z_i)_{0\le i<n})$    $\triangleright y \leftarrow f_0^{254}$
10: return $(Y_0, \dots, Y_{n-1})$
```

6 Side-Channel and Fault Resistant AES Implementation

The AES block cipher consists of multiple rounds of operations on its state. The iterations include three linear layers: `MixColumns`, `ShiftRows`, and `AddRoundKey` and one non-linear layer named `SubBytes`. In order to protect these functions from leaking information about the data they are processing, they must be designed to work on shares of the secret variables. These operations are known as *secure* or *SMC* addition(`Add`), multiplication(`EPMult`), squaring($\texttt{Sqr}_k$), and affine transformation(`Affine`); furthermore, the secure operations are composed of simpler field addition, multiplication, and squaring. The details of the simpler operations can be found in the appendices. Also, even though it is not an operation itself, a reliable source of randomness is fundamental. Thus, our implementation is built bottom-up, the field operations and randomness represent the building blocks, and more complex functions are layered on top of them

6.1 SMC Operations

The linear layers can be implemented in a straightforward manner with computations done locally. The MPC implementation of `SubBytes` consists of squarings and multiplications [RP12]. As explained in Section 3, faults injected during this part remain undetected in the previous scheme [RP12], which makes the `SubBytes` vulnerable. The `SubBytes` layer consists of two main stages.

- The power function $x \to x^{254}$ over GF(2^8), denoted by $Exp254(x)$, can be calculated using the Algorithm 3. Using Theorem 1 and Theorem 2, we can prove the t-SNI$_d^n$ security of the $Exp254(x)$ operation, as already proven in [BBD$^+$15].

  Theorem 5 (t-SNI$_d^n$ of $Exp254$)**.** *Let $(F_i)_{0\le i<n}$ be the input shares of $Exp254$, and let $(Y_i)_{0\le i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \le t_2$ of output shares such that $t_1 + t_2 \le d$, there exist two subsets I and J of indices with $|I| \le t_1$, such that those t_1 intermediate variables as well as the output shares $Y_{|\mathcal{O}}$ can be perfectly simulated from $F_{|I}$.*

- The second part of the `SubBytes` operation is the GF(2)-affine transformation and it is denoted by $\tau(y)$ [RP12]:

$$\begin{aligned}\tau_A(y) = 0x63 \quad &\oplus \quad (0x05 \cdot y) \oplus (0x09 \cdot y^2) \oplus (0xf9 \cdot y^4) \oplus (0x25 \cdot y^8)\\ &\oplus \quad (0xf4 \cdot y^{16}) \oplus (0x01 \cdot y^{32}) \oplus (0xb5 \cdot y^{64}) \oplus (0x8f \cdot y^{128}).\end{aligned}$$

Table 5: The number of SMC operations in one round of AES.

	$Exp254$	$\tau(y)$	`MixColumns`	`AddRoundKey`	`ShiftRows`
`EPMult`	16×4	-	-	-	-
`Sqr`$_1$	16×7	16×7	-	-	-
`Add`	-	16×7	12	16×1	-
`Affine`	-	16×8	16	-	-
`RefreshM`	16×2	-	-	-	-

Table 6: Total number of operations for different (n, d)-scenarios for one round of AES-128.

	$\varepsilon = 0$		$\varepsilon = 1$		$\varepsilon = 2$	$\varepsilon = 3$
	(3,1) [GRR98]	(3,1)	(4,1)	(6,2)	(5,1)	(6,1)
Field Mul.	2448	2640	4288	10656	6320	8736
Field Add	1428	2196	3696	10152	5580	7848
Randomness	192	192	256	768	320	384

Using the `EPMult`, we are able to compute the output of `SubBytes` securely while the probability of generating undetectable faults is 2^{-12} in the worst case for a $(4, 1)$-MPC where all the shares are faulty. To further break down the SMC design into its fundamental components, Table 5 shows the total number of SMC operations in one round of AES-128.

Based on these results, we provide the performance analysis and cost of different (n, d)-SMC schemes. The analysis is performed by using the total number of field multiplications, additions, and randomness requirements for one round of AES-128 as seen in Table 5 and Table 2. Results are shown in Table 6.

Next, we analyze first-order side-channel resistant AES-128 implementations. Using (4,1)-SMC, we are able to extend the first-order side-channel implementation of Roche and Prouff [RP12] to a combined first-order side-channel and fault resistant implementation. The extension increases the number of field multiplications by 62%, additions by 68%, and randomness requirements by 33%. Since the error detection coefficients are used for error propagation, our scheme is more efficient than simple duplication. Moreover, we can increase the side-channel resistance of the system to second order by using (6,2)-SMC. The cost of this implementation requires 148% more field multiplications and also 164% more additions, since it heavily depends on n and ε. On the other hand, the randomness requirement increases by 200%, because the cost of it is proportional to n and d. Also, as Table 6 shows, (4,1), (5,1), and (6,1)-SMCs have the same side-channel resistance and have the first, second, and third order fault resistance, respectively. The number of field multiplications and additions is nearly proportional to half of the fault resistance order. Therefore, we can conclude that increasing the order of fault resistance costs less than the increase of the side-channel resistance.

6.2 Software implementation

Up to this point, we have only discussed the theoretical performance results; the next section describes the performance results in terms of execution time of the whole encryption and its building blocks. The overall encryption execution timings for the (3,1) and (5,2) schemes are shown in Table 7. As a reference, we consider the 32-bit C implementation of AES in OpenSSL 1.0.1g, compiled for the ARM Cortex-M0+ and run at core clock frequency of 4 MHz. The execution time for this unmasked encryption is 481.5 μs. Table 7 shows its corresponding code and data size. Even though full unrolling is disabled, the code and data size is significantly larger, however, the execution time is 1090X faster than the fastest masked encryption in Table 7.

SMC multiplication is the bottleneck of the algorithm and in turn it relies on the field multiplication. The execution time can also be reduced by running at higher frequencies

Table 7: AES-128 encryption execution time, code and RW-data size depending on the GF(2^8) *operations variations. I:Instruction Only, M:Mixed, L:LUT, E:Exp-Log.*

	(3,1)			(5,2)			unmasked
GF(2^8) mult.	I	M	E	I	M	E	-
GF(2^8) sqr.	I	L	L	I	L	L	-
Encryption [GRR98]	1.45M	1.11M	0.52M	8.04M	6.48M	2.90M	-
Our scheme	1.75M	1.37M	0.64M	9.21M	7.47M	3.4M	-
Code Size (kB)	3.4	3.3	3.3	3.6	3.4	3.5	7.2
RW-data (B)	12	524	780	32	544	800	12
RO-data (B)	224	224	224	224	224	224	870

Table 8: Execution time for GF(2^8) *and SMC operations in μs with the CPU running at 4 MHz. I:Instruction Only, M:Mixed, E: Exp-Log*

	(3,1)			(4,1)		(5,1)		(5,2)			(6,2)	
	I	M	E	I	M	I	M	I	M	E	I	M
GF(2^8) mult.	54.5	44.5	17.5	54.5	44.5	54.5	44.5	54.5	44.5	17.5	54.5	44.5
GF(2^8) sqr.	13.8	1.5	1.5	13.8	1.5	13.8	1.5	13.8	1.5	1.5	13.8	1.5
`getrn()`	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8	3.8
SMC add.	15.2	15.2	15.2	18.2	18.2	21.2	21.2	21.2	21.2	21.2	24.2	24.2
Mult [GRR98]	1.2k	1.0k	0.48k	2.1k	1.7k	3.2k	2.7k	9.1k	7.5k	3.4k	13k	10.8k
`EPMult`	1.4k	1.1k	0.54k	2.4k	2k	3.8k	3.2k	9.8k	8.1k	3.7k	13.5k	11.2k

but the performance would remain the same, however, power consumption would increase. Table 7 details the amount of code and RW-data according to selected combinations of field operations, noted that other combinations are also possible to produce different code and data sizes.

Table 8 summarizes the execution timings corresponding to the different versions of field operations and the SMC multiplication. Based on Table 5, these building block operations represent the key elements to boost the performance of the masking scheme, that is the reason to look for faster methods to perform field arithmetic.

The only comparable implementation was presented in [GSF14] and features, according to its Figure 2, an approximate number is 4.5 million cycles for a (5,2) scheme. Our fastest second order implementation takes 11.6 million cycles which is nearly 2.6X slower. The comparison is based on the graphs in Figure 2 of [GSF14]. We surmise that part of the performance degradation is due to different platform features and the fact that our implementation is of constant time and performs on-the-fly mask generation. It also suggests that significant performance gains can be achieved through further optimizations of our proof-of-concept implementation.

Hardware Implementation The proposed scheme is also well-suited for hardware implementation, due to its glitch-resistance. A proof-of-concept implementation of the Roche and Prouff scheme was analyzed by Moradi and Mischke [MM13]. Their reference implementation introduces a rather high overhead, in area but also in lost performance. Our scheme will increase this overhead due to the fault resistance, as quantified in Section 5.4, mainly due to the increased number of shares. It should be noted that the reference implementation in [MM13] has parallel hardware, but still performs serialized processing of all shares. However, parallel processing of shares can be secure [BDF+17] and would provide a significant performance boost over the fully serialized implementation in [MM13].

6.3 Side Channel Analysis

Leakage Detection. The test vector leakage assessment (TVLA) test was proposed by Gilbert et al. [GGJR+11] and has become a widely used method to assess the leakage

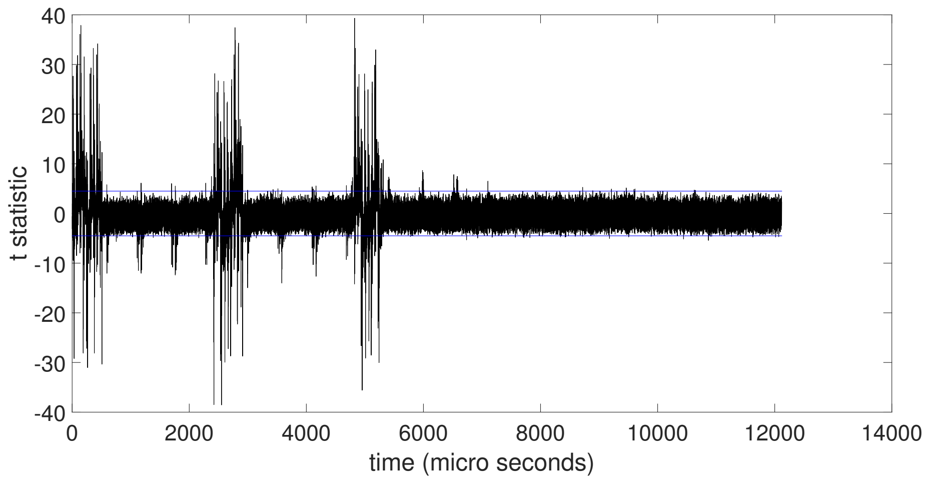

Figure 2: Leakage analysis with disabled masking after 12,000 traces.

resistance of embedded implementations [BGN⁺14, LMW14, STE17]. The test checks whether side-channel traces depend on the sensitive variables and it is designed only for side-channel leakage detection instead of fully recovering the secret key.

The leakage tests are conducted by generating two different sets of side-channel traces. The sets can be collected by making the device-under-test (DUT) process either a fixed input or a random input under the same conditions. To prevent false-positive leakage detection, it is recommended to collect the traces following a random pattern in which either a random input or a fixed input is fed to the DUT. After collecting the traces, means (μ_f, μ_r) and standard deviations (σ_f, σ_r) for two sets are calculated. For ease of computation, we apply the moving average technique introduced in [DCE16], which uses moving average instead of central average. Therefore, it is faster, more reliable, and more robust to environmental noise. Welch's t-test is executed as in Equation (6) where n_f and n_r denote the number of traces for fixed and random sets respectively.

$$t = \frac{\mu_f - \mu_r}{\sqrt{(\sigma_f^2/n_f) + (\sigma_r^2/n_r)}}. \tag{6}$$

The goal of this test is to show that the trace of a secret data is statistically indistinguishable from the trace of a random data. Remark that the test assumes no other information and thus it can be used to detect the leakage through the entire algorithm.

Higher-Order t Test. To demonstrate the effectiveness of our implementation, an initial t-test was performed with a switched-off masking on a small set of samples and later with the masking enabled on a much larger set of measurements. To disable the masking scheme, during the sharing of the input operands, the highest-degree coefficient was hardcoded to `0x1`.

Figure 2 shows the intense leakage spread around three different points in time. They correspond to the initial three field multiplications that are done within the SMC multiplication. The peaks are generated because there is an immediate relationship between the operands and their corresponding shares. The shares for the fixed operands are always the same and thus consume an approximately equal amount of power on every execution so, when compared to the power consumption of random operands, a huge difference is revealed after just 12,000 traces.

After demonstrating the effectiveness of the t-test, results corresponding to (3,1)-`EPMult` and (5,2)-`EPMult` are displayed in Figure 3. While (3,1)-`EPMult` executed with a 4 MHz clock, (5,2)-`EPMult`'s clock was switched to 16 MHz, due to its execution length with 4 MHz clock would turn the trace collection impractical.

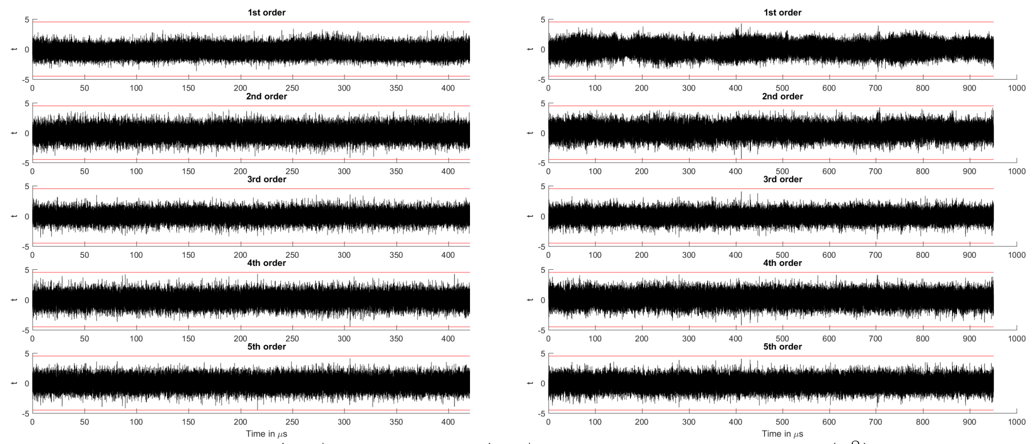

Figure 3: HO t-test for (3,1)-`EPMult` and (5,2)-`EPMult` with Exp-Log GF(2^8) *multiplication using 250.000 traces.*

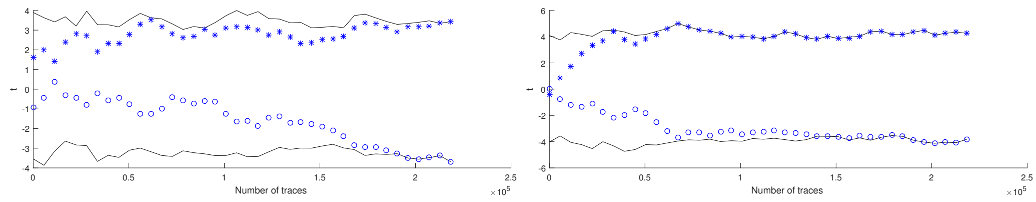

Figure 4: First order t growth for (3,1)-`EPMult` and (5,2)-`EPMult` with Exp-Log GF(2^8) *multiplication. The black lines show the evolution of the maximum and minimum first-order t values over the number of traces. The stars mark how the index of the last maximum value grew over the number of traces. The circles mark the last minimum values.*

For all subplots in Figure 3 are showing analysis results for 1st through 5th order. As the figure shows, the level of leakage is contained within the acceptable boundaries. Also, Figure 4 shows the t growth over the number of samples for the first-order t-test.

Multivariate t Test. SMC hardware implementations process their shares in parallel, therefore, the power consumption reflects the processing demand of all of them simultaneously. In our single-threaded software implementation, the operations on every share or pair of shares are performed sequentially. As a result, the power consumption at certain intervals may only be related to a single share or pair of shares being processed [SM15]. The multivariate t-test combines a sample from a particular point in time to other samples at different intervals of time. The objective is to identify if there is a relationship between the processing of the sets of shares that occur at different points in time.

Figure 5 shows the results of the multivariate analysis on relevant sections of the (3,1)-`EPMult`. Each of the plots belongs to the combination of the points in the section where the first pair of shares is processed and those of the sections where the remaining pairs are processed. Although the Exp-Log field multiplication uses table look-ups, the result in Figure 5 does not show any evidence of leakage derived from the memory accesses.

The multivariate analysis is a useful tool to reveal potential sources of interdependent side-channel leakage that otherwise would be hidden from the regular t-test. However, the time execution and memory constraints are significant factors to constrain the extension of the analysis to certain sections. Remark that for all of the multivariate analysis subplots, the horizontal axis does not represent time since the analysis itself requires the combination of traces at different points.

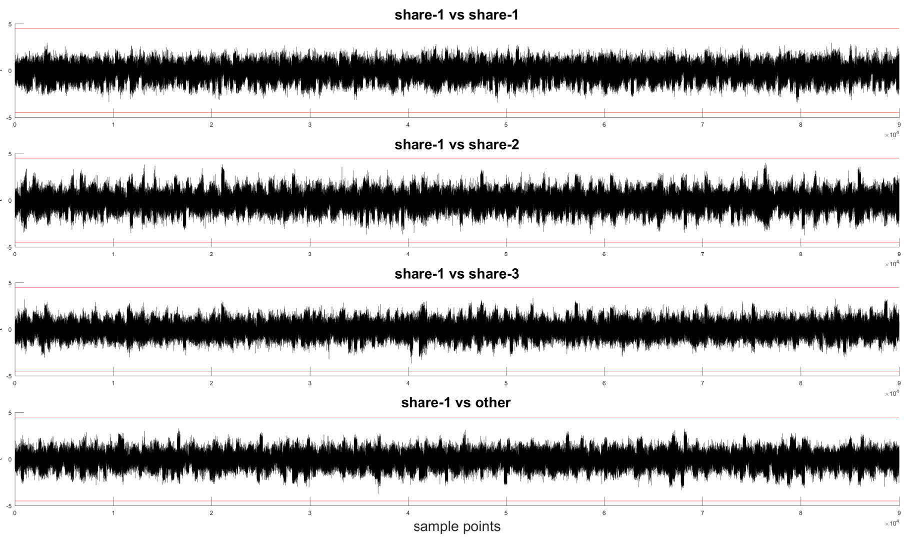

Figure 5: Multivariate t-test for sections of the (3,1)-`EPMult` based on Exp-Log GF(2^8) *multiplication. Share-1 vs Share-1 shows the multivariate t-test results of all combinations of points during the first* GF(2^8) *multiplication. Share-1 vs Share-2 shows the results of the multivariate t-test for the combination of points from the first field multiplication to the second one. Share-1 vs Share-3 corresponds to the multivariate t-test analog to the previous case. Share-1 vs Other shows the results of multivariate t-test of the points during the first field multiplication combined with all the points of a section close to the end of `EPMult`.*

Table 9: Probabilities of generating undetectable faults for `EPMult`.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$
(4,1)	1.54×10^{-5}	1.54×10^{-5}	1.53×10^{-5}	1.53×10^{-5}
(5,1)	1.54×10^{-5}	6.03×10^{-8}	6.01×10^{-8}	5.98×10^{-8}
(6,1)	1.54×10^{-5}	6.03×10^{-8}	2.37×10^{-10}	2.35×10^{-10}
(6,2)	6.03×10^{-8}	6.03×10^{-8}	6.01×10^{-8}	6.01×10^{-8}

6.4 Fault Analysis

Next, we present experimental results of fault injection on the proposed scheme on the simulation in `SAGE`. As given in Section 5, faults can be undetectable in a sequence of operations. As the only non-linear operation of AES, we focused on `SubBytes` operation.

We start with the $Exp254$ operation to our analyses. First, we do the theoretical analyses on `EPMult` and $Exp254$. In Table 9 and in Table 10, one can see the probabilities of generating undetectable faults for `EPMult` and $Exp254$, respectively. For these analyses, we assumed that inputs polynomials are faulty. That is, for `EPmult`, the faults are detectable using $(F_i)_{(0 \leq i < n)}$ and $(G_i)_{(0 \leq i < n)}$ in Algorithm 1. And for $Exp254$, the faults are detectable using $(F_i)_{(0 \leq i < n)}$ in Algorithm 3. Note that, k is defined as the number of faulty shares as in Corollary 1. As seen both tables, the probabilities slightly changes, depending on the conditions listed in Corollary 1. Also, even if we used a sequence of operations, the generating undetectable faults for $Exp254$ mostly depend on the last multiplication (line 9 in Algorithm 3). Notice that, the faults in the initial input shares spread to first $\varepsilon + d$ shares of both inputs of `EPMult` operations within $Exp254$. Therefore, the number of faulty shares after the first `EPMult` is calculated as $max(n, 2(\varepsilon + d))$.

In the second part, we verify the theoretical analyses with the experimental results. We look the fault detection capabilities of $Exp254$ and $\tau_a \circ Exp254$. The experimental setup can be summarized as follows:

1. Select a secret variable $x \in$ GF(2^8) and create an (n, d)-sharing of x as $(F_i)_{(0 \leq i < n)}$.

Table 10: Probabilities of generating undetectable faults for $Exp254$.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$
(4,1)	1.53×10^{-5}	1.53×10^{-5}	1.53×10^{-5}	1.53×10^{-5}
(5,1)	5.98×10^{-8}	5.98×10^{-8}	5.98×10^{-8}	5.98×10^{-8}
(6,1)	2.34×10^{-10}	2.34×10^{-10}	2.34×10^{-10}	2.34×10^{-10}
(6,2)	6.03×10^{-8}	6.01×10^{-8}	6.01×10^{-8}	6.01×10^{-8}

Table 11: Probabilities of generating undetectable faults for $Exp254$ and $\tau_A \circ Exp254$ using `SAGE` simulation.

	# Secret Shares / **# Fault Injections**			
	$k = 1$	$k = 2$	$k = 3$	$k = 4$
	2^{16} / $\mathbf{2^8}$	2^8 / $\mathbf{2^{16}}$	2^8 / $\mathbf{2^{16}}$	2^8 / $\mathbf{2^{16}}$
(4,1)	1.45×10^{-5}	1.52×10^{-5}	3.04×10^{-7}	2.88×10^{-5}
(5,1)	2.40×10^{-7}	6.01×10^{-8}	6.01×10^{-8}	1.20×10^{-7}
(6,1)	0	0	0	0
(6,2)	0	0	1.20×10^{-7}	1.26×10^{-5}

2. Select k faults $\sigma_i \in \mathrm{GF}(2^8) \setminus \{0\}$ and inject the faults to the first k shares of x.

3. Do fault detections on the output of $Exp254((F_i)_{0 \leq i < n})$ and the output of $\tau_a \circ Exp254((F_i)_{0 \leq i < n})$.

For example in the $(4, 1)$ case, even if faults spread to all shares, the probability of generating undetectable errors at most 2^{-12}, as expected. In each multiplication, faults are spread to $k' \leq \varepsilon + d$ shares and these shares become input for another multiplication. Therefore, $Propogation_\varepsilon$ changes for each multiplication. As seen in Table 11, if we increase n, the probability of undetectable faults decreases with respect to the conditions in Corollary 1. In these experiments, we maximize the attackers capabilities to simulate the real-world settings, and efficiently analyze the fault model and detection capabilities of our scheme. Remark that, for an (n, d)-scheme with k faults, the total number of secret shares is $(2^8)^{d+1}$ and the total number of all possible faults is $(2^8)^k$.

The number of undetectable faults are same in $Exp254$ and $\tau_a \circ Exp254$. Therefore, we can conclude that τ_a does not produce undetectable faults. And if the fault is detectable using the output of $Exp254$, the attacker should inject another fault to τ_A to generate undetectable faults. Moreover, in some experiments, all faults become detectable even if the propagation probability is not 1. Although we perform the experiments with maximized number of faults, randomness is added in the nature of multiplication. Therefore, the numbers are not exact values, but rather upper bounds.

7 Conclusion

Fault and side-channel attacks have become a real threat to cryptographic systems if the adversary can observe and interact with the physical implementation. In this work, we propose a new secure multiparty computation to achieve both fault and side-channel resistance. It is shown that the proposed schemes can be used to perform addition, affine transformation, multiplication, and squaring while resisting both well-defined fault and side-channel adversaries. One advantage of the proposed scheme is a reduced overhead, as only an extra operation within the error preserving multiplication is needed.

We define a new multiplication engine in such a way that, once a fault occurs, information about the error remains as a part of the shares. The error propagates through the algebraic operations with high probability. It will be detectable even after further computations on the shares. This gives implementers the choice to perform error detection regularly (for higher detection rates at a higher overhead) or only implicitly during recombination at the end of the circuit. The error detection method is based on the degree of the secret sharing polynomial, which increases when errors occur. After the initial increase, additional levels of logic operations can result in a loss of degree for faulty states, leaving a small probability of undetected errors. A secrecy-preserving fault detection operation is defined to perform the detection. Also the idea of forwarding faults allows us to delay any error detection as late as the final recombination step. We introduce a recombination gate which is used for both fault detection and reconstruction of the secret. Hence, fault detection can be carried out when the output is produced. Moreover, the recombination gate features another desired property: Infective Computation. If an error occurs, our scheme ensures that attackers cannot learn anything since the output is random.

Security properties of our scheme are given using ISW probing model and a formal analysis of fault resistance. Every scheme used in the paper, including fault detection operation and recombination operation is proven to be secure in ISW probing model using the reformulated t-SNI security notion. Also, the first formal security proof of the multiplication scheme [RP12] is proven within this work, since the previous scheme can be seen as a subset of our scheme. Fault detection of our scheme is examined using notion of *Propagation*. The error-detection capacities of each operations are formally given by analysing the undetectable faults for each operation.

We propose a practical C implementation AES-128, tested on a popular ultra-low power architecture, the ARM Cortex M0+ core. We also measure its performance and demonstrate its level of side-channel resistance by addressing a full leakage analysis including higher order moments on the SMC multiplication. Also, to show the fault resistance capabilities of the proposed scheme, we perform the experiments on the `SubBytes` operation which can be considered as the most to vulnerable part of AES. The implementation provides multiple masking schemes with different types of field operations and is easily portable to higher orders. Different masking orders with different field operations executed in constant time. The code provides a *fully constant execution flow with constant memory accesses.*

Acknowledgments

This work is supported by the National Science Foundation, under grant CNS-1618837.

References

[BBD$^+$15] 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. Cryptology ePrint Archive, Report 2015/506, 2015.

[BDF$^+$17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. *Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model*, pages 535–566. Springer International Publishing, Cham, 2017.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In Walter Fumy, editor, *Advances in Cryptology EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer Berlin Heidelberg, 1997.

[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.

[Ben88] Ben-Or, Michael and Goldwasser, Shafi and Wigderson, Avi. Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[Ber68] E.R. Berlekamp. *Algebraic coding theory.* McGraw-Hill series in systems science. McGraw-Hill, 1968.

[BG13] Alberto Battistello and Christophe Giraud. Fault analysis of infective AES computations. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on*, pages 101–107. IEEE, 2013.

[BG16] Alberto Battistello and Christophe Giraud. A note on the security of CHES 2014 symmetric infective countermeasure. In *Constructive Side-Channel Analysis and Secure Design – COSADE 2016*, 2016.

[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*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer Berlin Heidelberg, 2014.

[CB08] D. Canright and Lejla Batina. *A Very Compact "Perfectly Masked" S-Box for AES*, pages 446–459. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[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.

[CGPZ16] Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. *Faster Evaluation of SBoxes via Common Shares*, pages 498–514. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer Berlin Heidelberg, 1999.

[CPR13] Jean-Sébastien Coron, Emmanuel Prouff, and Thomas Roche. On the use of shamir's secret sharing against side-channel analysis. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications*, pages 77–90, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 742–763, 2015.

[CRZ13] Guilhem Castagnos, Soline Renner, and Gilles Zémor. *High-order Masking by Using Coding Theory and Its Application to AES*, pages 193–212. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[CT05] Hamid Choukri and Michael Tunstall. Round reduction using faults. *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 5:13–24, 2005.

[DCE16] A. Adam Ding, Cong Chen, and Thomas Eisenbarth. *Simpler, Faster, and More Robust T-Test Based Leakage Detection*, pages 163–183. Springer International Publishing, Cham, 2016.

[DCN16] Thomas De Cnudde and Svetla Nikova. More efficient private circuits ii through threshold implementations. In *International Workshop on Fault Diagnosis and Tolerance in Cryptography 2016*. IEEE, 2016.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. *Unifying Leakage Models: From Probing Attacks to Noisy Leakage.*, pages 423–440. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[DMN+12] J. M. Dutertre, A. P. Mirbaha, D. Naccache, A. L. Ribotta, A. Tria, and T. Vaschalde. Fault round modification analysis of the advanced encryption standard. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 140–145, June 2012.

[GGJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj M. Prabhakaran, Amit Sahai, and Eran Tromer. Circuits Resilient to Additive Attacks with Applications to Secure Computation. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 495–504, New York, NY, USA, 2014. ACM.

[Gir06] Christophe Giraud. An RSA implementation resistant to fault attacks and to simple power analysis. *Computers, IEEE Transactions on*, 55(9):1116–1120, 2006.

[GM11] Louis Goubin and Ange Martinelli. Protecting AES with Shamir's secret sharing scheme. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 79–94. Springer, 2011.

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

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems CHES 2001*, pages 251–261. Springer, 2001.

[GR16] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? Cryptology ePrint Archive, Report 2016/264, 2016.

[GRR98] Rosario Gennaro, Michael O Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM, 1998.

[GSF14] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: How large is the gap for AES? *Journal of Cryptographic Engineering*, 4(1):47–57, 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 *Progress in Cryptology–LATINCRYPT 2012*, pages 305–321. Springer, 2012.

[GST14] Daniel Genkin, Adi Shamir, and Eran Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *Advances in Cryptology–CRYPTO 2014*, pages 444–461. Springer Berlin Heidelberg, 2014.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 308–327, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. *Private Circuits: Securing Hardware against Probing Attacks*, pages 463–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[KJJR11] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[LFZD14] Pei Luo, Yunsi Fei, Liwei Zhang, and A Adam Ding. Side-channel power analysis of different protection schemes against fault attacks on AES. In *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[LMW14] Andrew J. Leiserson, Mark E. Marson, and Megan A. Wachs. *Gate-Level Masking under a Path-Based Leakage Metric*, pages 580–597. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[LRT12] Victor Lomné, Thomas Roche, and Adrian Thillard. On the Need of Randomness in Fault Attack Countermeasures-Application to AES. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012 Workshop on*, pages 85–94. IEEE, 2012.

[MM13] Amir Moradi and Oliver Mischke. On the simplicity of converting leakages from multivariate to univariate. In *Cryptographic Hardware and Embedded Systems-CHES 2013*, pages 1–20. Springer, 2013.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. *Correlation-Enhanced Power Analysis Collision Attack*, pages 125–139. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[MMSS18] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited - or why proofs in the robust probing model are needed. Cryptology ePrint Archive, Report 2018/490, 2018.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. *Successfully Attacking Masked AES Hardware Implementations*, pages 157–171. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes.* Elsevier, 1977.

[MS81] R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Commun. ACM*, 24(9):583–584, September 1981.

[NRS09] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of non-linear functions in the presence of glitches. In *Information Security and Cryptology–ICISC 2008*, pages 218–234. Springer, 2009.

[OMPR05] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A Side-channel Analysis Resistant Description of the AES S-box. In *Proceedings of the 12th International Conference on Fast Software Encryption*, FSE'05, pages 413–423, Berlin, Heidelberg, 2005. Springer-Verlag.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Advances in Cryptology–CRYPTO 2015*, pages 764–783. Springer LNCS, 2015.

[RDP08] Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. *Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis*, pages 127–143. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[REB$^+$08] Francesco Regazzoni, Thomas Eisenbarth, Luca Breveglieri, Paolo Ienne, and Israel Koren. Can knowledge regarding the presence of countermeasures against fault attacks simplify power attacks on cryptographic devices? In *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*, pages 202–210. IEEE, 2008.

[RLK11] Thomas Roche, Victor Lomné, and Karim Khalfallah. Combined Fault and Side-Channel Attack on Protected Implementations of AES. In Emmanuel Prouff, editor, *Smart Card Research and Advanced Applications*, volume 7079 of *Lecture Notes in Computer Science*, pages 65–83. Springer Berlin Heidelberg, 2011.

[RMB$^+$17] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel Smart. Capa: The spirit of beaver against physical attacks. Cryptology ePrint Archive, Report 2017/1195, 2017.

[RP10] Matthieu Rivain and Emmanuel Prouff. *Provably Secure Higher-Order Masking of AES*, pages 413–427. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[RP11] Thomas Roche and Emmanuel Prouff. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 63–78. Springer, 2011.

[RP12] Thomas Roche and Emmanuel Prouff. Higher-order glitch free implementation of the AES using secure multi-party computation protocols. *Journal of Cryptographic Engineering*, 2(2):111–127, 2012.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sha99] Adi Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, November 23 1999. US Patent 5,991,415.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - a clear roadmap for side-channel evaluations. Cryptology ePrint Archive, Report 2015/207, 2015.

Algorithm 4 Fault Detection Operation

```
Input: Shares of $f_0$ as $(F_i)_{0\le i<n}$.
Output: Fault Decision.
1: $(r_0,\dots,r_d) \leftarrow \mathbb{F}_{2^m}$                ▷ Coefficients of the random polynomial.
2: for $i = 0$ to $n-1$ do                                       ▷ Randomization.
3:     $F_{\mathcal{R}_i} \leftarrow F_i$
4:     for $k = 0$ to $d$ do                                      ▷ Evaluate the polynomial.
5:         $\mathcal{R}_i \leftarrow \mathcal{R}_i \oplus r_k\alpha_j^k$          ▷ Referred to as $\mathcal{R}_i^k$
6:     end for
7:     $F_{\mathcal{R}_i} \leftarrow F_{\mathcal{R}_i} \oplus \mathcal{R}_i$
8: end for
9: Fault Detection using the set of secret shares: $(F_{\mathcal{R}_i})_{0\le i<n}$     ▷ Detection.
```

[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*, pages 302–332. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[SP06] Kai Schramm and Christof Paar. *Higher Order Masking of the AES*, pages 208–225. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[STE17] A. Shahverdi, M. Taha, and T. Eisenbarth. Lightweight side channel resistance: Threshold implementations of simon. *IEEE Transactions on Computers*, 66(4):661–671, April 2017.

[TBM14] Harshal Tupsamudre, Shikha Bisht, and Debdeep Mukhopadhyay. Destroying Fault Invariant with Randomization. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 93–111. Springer Berlin Heidelberg, 2014.

[YO13] Maki Yoshida and Satoshi Obana. Detection of Cheaters in Non-interactive Polynomial Evaluation. Cryptology ePrint Archive, Report 2013/032, 2013.

A Security Proofs

Before going into the proofs of fault detection operation and recombination operation, we need to clarify that the *detection* parts are excluded from the proofs to make definitions compatible. As given in Equation 2, the detection mechanism requires all shares. However, during fault detection we already mask the sensitive variable.

A.1 Fault Detection Operation

Theorem 6 (t-SNI$_d^n$ of Fault Detection Operation)**.** *Let $(F_i)_{0\le i<n}$ be the input shares of Fault Detection Operation and let $(F_{\mathcal{R}_i})_{0\le i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \le t_2$ of output shares such that $t_1 + t_2 < d+1$, there exists a subset I of indices with $|I| \le t_1$, such that those t_1 intermediate variables as well as the output shares $F_{\mathcal{R}_{|\mathcal{O}}}$ can be perfectly simulated from $F_{|I}$.*

Proof. The proof is very similar to the proof of Theorem 2. For every probed variable F_i, $F_{\mathcal{R}_i}$, or $\mathcal{R}_i^d$ add i to I. Clearly, we add at most one index to I and, therefore, $|I| \le t_1$.

Algorithm 5 Recombination Operation

```
Input: Shares of $f_0$ as $(F_i)_{0\leq i<n}$ and non-zero random values $(\mathsf{r}_0,\ldots,\mathsf{r}_{\varepsilon+d-1})$.
Output: $f_0$ and Fault Decision.
 1: for $i = 0$ to $n-1$ do                                          ▷ Re-Sharing
 2:     $(r_{i,1},\ldots,r_{i,d}) \leftarrow \mathbb{F}_{2^m}$        ▷ Coefficients of the random polynomial.
 3:     for $j = 0$ to $n-1$ do
 4:         $\mathcal{Q}_{i,j} \leftarrow F_i$                         ▷ referred by $\mathcal{Q}^0_{j,i}$.
 5:         for $k = 1$ to $d$ do                                    ▷ Evaluate the polynomial.
 6:             $\mathcal{Q}_{i,j} \leftarrow \mathcal{Q}_{i,j} \oplus r_k\alpha_j^k$      ▷ referred to as $\mathcal{Q}^k_{j,i}$.
 7:         end for
 8:         $\mathcal{Q}_{i,j} \leftarrow \mathcal{Q}_{i,j} \oplus \mathtt{E}^R_{i,j}$    ▷ Add a share of an error detection term.
 9:         $\mathbf{Q}_j \leftarrow \mathbf{Q}_j \oplus \lambda_i^0\mathcal{Q}_{i,j}$    ▷ referred to as $\mathbf{Q}_{j,i}$.
10:     end for
11: end for
12: Reconstruction using the set of secret shares: $(\mathbf{Q}_i)_{0\leq i<n}$    ▷ Reconstruction.
```

1. If $\mathcal{R}_i^k$ is probed, we can perfectly simulate it with a random value, since it does not depend on any variable.
2. If $F_{\mathcal{R}}$ is probed, we can simulate it as $F_i \oplus \mathcal{R}_i$. Note that, $\mathcal{R}_i$ can be simulated as in the first step.

Therefore, we are able to simulate all the probed variables. Now, we consider the simulation of output variables. We need to show that $F_{\mathcal{R}_i}$ for $i \in \mathcal{O}$ can be simulated from $F_{|I}$. If $i \in I$, we can simulate $F_{\mathcal{R}_i}$ as explained above. We now examine the simulation of output variables $F_{\mathcal{R}_i}$, where $i \notin I$. That means, $\mathcal{R}_i^d$ is not probed and is not involved in computation of $F_{\mathcal{R}_i}$. Hence, we can perfectly simulate $F_{\mathcal{R}_i}$ by a random value.

□

A.2 Recombination Operation

Theorem 7 (t-SNI$_d^n$ of Recombination Operation)**.** *Let $(F_i)_{0\leq i<n}$ be the input shares of the Recombination Operation and let $(\boldsymbol{Q}_i)_{0\leq i<n}$ be the output shares. For any set of t_1 intermediate variables and any subset $|\mathcal{O}| \leq t_2$ of output shares such that $t_1 + t_2 < d+1$, there exists a subset I of indices with $|I| \leq t_1$, such that those t_1 intermediate variables as well as the output shares $\boldsymbol{Q}_{|\mathcal{O}}$ can be perfectly simulated from $F_{|I}$.*

Proof. As stated in Section 4.3, the recombination operation can be seen as a modified version of `EPMult`, therefore, the proof is built on the same structure as in the proof of Theorem 1 in Section 5.

In the first part of the proof, we construct the sets of input share indices I depending on the intermediate variables that are probed. If F_i, $r_{i,j}$, $F^k_{j,i}$ or $E^R_{i,j}$ is probed, add i to I

- **Group 1:** If F_i or $\mathcal{F}^0_{i,j}$ or $\mathtt{E}^R_{i,j}$ is probed, add i to I.
- **Group 2:** If $\mathtt{E}^R_{i,j}$ or $\mathsf{r_j}$ is probed, add i to I.
- **Group 3:** If $r_{i,j}$ or $\mathcal{Q}^k_{i,j}$ where $k \in \{1,\ldots,d\}$ is probed, add i to I and J.

According to our selection, we add at most one index to I and J for each probe and, therefore, $|I| \leq t_1$ and $|J| \leq t_1$.

1. The simulations of the probed variables in group 1 are straightforward. Since $i \in I$, we can perfectly simulate F_i. Similarly, $\mathcal{Q}^0_{i,j}$, since $i \in I$.

2. Since the elements λ_i of the inverse Vandermonde matrix are public variables, we can simulate $\mathsf{E}^R_{i,j}$ as defined in Equation (3). In fact we let r_j for $j \in \{0, \ldots, \varepsilon + d + 1\}$ as in the real circuit.

3. If $\mathcal{Q}^k_{i,j}$ is probed, we need to consider two cases:
   - If $r_{i,k}$ is also probed, we let $r_{i,k}$ as in the real circuit, therefore, we can simulate $\mathcal{Q}^k_{i,j}$ as $F_i \oplus r_{i,k}\alpha^k_j$ where α_j is a public value.
   - If $r_{i,k}$ is not probed, it does not enter into the computations of $\mathcal{Q}^k_{i,j}$, therefore, we can perfectly simulate $\mathcal{Q}^k_{i,j}$ with a random value.

4. If $\mathbf{Q}_{j,i}$ is probed, we need to consider two cases as in the previous step:
   - If all the values $r_{i,k}$ for $1 \leq k \leq d$ are probed, we can perfectly simulate the values $\mathcal{Q}^k_{i,j}$, and hence $\mathbf{Q}_{j,i}$ can be simulated. Note that, λ^0_i is an element of the inverse Vandermonde matrix so it is a public value.
   - If at least $r_{i,k}$ for $1 \leq k \leq d$ is not probed, that means $r_{i,k}$ does not enter into the computation of $\mathbf{Q}_{j,i}$, therefore, $\mathbf{Q}_{j,i}$ can be simulated by a random value.

Now we explain how to simulate output shares $\mathbf{Q}_i$ for all $i \in \mathcal{O}$ where $\mathcal{O}$ is an arbitrary subset of $[1, n]$ with t_2 elements such that $t_1 + t_2 < d + 1$. Clearly, using t_1 probes, we can observe at most t_1 intermediate variables of $\mathbf{Q}_i$, where $\mathbf{Q}_i$ can be written as: $\mathbf{Q}_i = \sum_{j=0}^{n-1} \lambda^0_i \mathcal{Q}_{j,i}$. Since $t_1 + t_2 < d + 1$, at least one intermediate variable of $\mathbf{Q}_i$ is not probed. Therefore, we can simulate $\mathcal{Q}_{j,i}$ with $j \notin U$ by generating a random degree d polynomial and evaluating it for α_i. Hence, we can simulate $\mathbf{Q}_i$ for each $i \in \mathcal{O}$.

□

A.3 `RefreshM` Algorithm

Proof. The proof is relatively straight forward. For every probed variable F_i, C_i, or $\mathcal{Q}_|$ add i to I. Clearly, we add at most one index to I and, therefore, $|I| \leq t_1$.

1. If r_k or $\mathcal{Q}^k_j$ probed, we let the variables as in the circuit and perfectly simulate them.

2. if C_i is probed, we can perfectly simulate the variable by $F_i \oplus \mathcal{Q}_i$, by letting $\mathcal{Q}_i$ as in the real circuit.

Therefore, we are able to simulate all the probed variables. Now, we consider the simulation of output variables. We need to show that C_i for $i \in \mathcal{O}$ can be simulated from $F_{|I}$. If $i \in I$, we can simulate C_i as explained above. We now examine the simulation of output variables C_i, where $i \notin I$. That means, $\mathcal{Q}_j$ is not probed and is not involved in the computation of C_i. Hence, we can perfectly simulate C_i by a random value.

□

B Implementation Details

B.1 Target Platform

Due to the rapid development environment and the omnipresence of ARM cores in embedded applications, we employed the NUCLEO-L053R8 board from STMicroelectronics to test our robust implementation. It features a 32-bit ARM Cortex-M0+ microcontroller labelled STM32L053R8T6. It can reach a clock frequency of up to 32 MHz and it is equipped with a hardware random-number generator (RNG) capable of generating one 32-bit random number every 40 cycles. The RNG must run at 48MHz. Internal Phase-Locked Loop circuits (PLLs) can be used to match this frequency.

Algorithm 6 GF(2^8) `Multiplication (instructions only).`

```
// $z = h \cdot v$
for ($i = 0$ ; $i < 8$ ; $i++$) do
    $mask = -((h >> i)\&1)$;                ▷ (1)
    $z = z$ ^(mask & $v$);                  ▷ (2)
    $mask = -((v >> 7)\&1)$;                ▷ (3)
    $v <<= 1$;                              ▷ (4)
    $v$ ^= mask & 0x1b;                     ▷ (5)
end for
return $z$
```

Algorithm 7 GF(2^8) `Multiplication (mixed).`

```
// $z = h \cdot v$
for ($i = 0$ ; $i < 8$ ; $i++$) do
    $mask = -((h >> i)\&1)$;                ▷ (1)
    $z = z$ ^(mask & $v$);                  ▷ (2)
    $v$ = secondOp[$v$];                    ▷ (3, 4, 5)
end for
return $z$
```

A particular feature of this development board is that it provides two contact points to measure the actual current consumption of the ARM chip. We took advantage of it to place a low-value resistor between the pins to measure the voltage drop for our side-channel analysis. The code was initially sketched in mbed and later migrated to ARM MDK-Lite (KEIL uVision 5.21), however, the code is architecture-independent.

B.2 Field Multiplication

This is a heavily used operation across the implementation. Even a small performance variation in this operation significantly affects the whole algorithm, thus it is very important to optimize this operation and consider different trade-offs. The following paragraphs briefly describe the four variations that are available in the implementation code. The slowest version of the multiplication is based on instructions only, with the minimum memory usage and in *constant time*. The result of the field multiplication is returned after 8 iterations, as given in the Algorithm 6.

A second version of this operation is a trade-off function that combines a precomputed 256-byte look-up table (LUT) and instructions. The only difference from the previous version is that the LUT contains all possible computations of v based on items 3, 4, and 5 from instruction only-multiplication Algorithm 6 as those three lines of code only depend on operand v.

The best time-memory trade-off field multiplication [GR16], known as the Exp-Log multiplication, is derived from the logarithm property $vh = g^{log_g(v)+log_g(h)}$. An appropriate generator g must be selected to precompute the logarithm and exponentiation tables so the multiplication is reduced to three table look-ups and logical and arithmetic operations, especially required to check if any of the operands is zero.

Ultimately, the fastest instance of this operation is based on two pre-computed 4-kB LUTs. To generate the tables, one of the operands is split into its most-significant nibble and least-significant nibble $v = v_m 2^4 + v_l$, then every possible permutation of each nibble is multiplied times all possible permutations of the other operand $vh = v_m h 2^4 + v_l h$ but only the most-significant nibble multiplication is reduced modulo the irreducible polynomial. To get the result of the field multiplication, only two look-ups and one addition are required, however, this method is very expensive in terms of memory usage and thus kept outside our performance analysis.

B.3 Field Squaring

The implementation code features two ways of performing field squaring. The first one, as in the multiplication case, is based on instructions only; the second one is simply a 256-byte LUT of all possible square values of the input.

The following pseudo-code describes in detail the algorithm of the code-only field squaring which can be conducted in a single line of C code.

$$\begin{aligned} y^2 &= (y \& \texttt{0x01}) \oplus ((y \& \texttt{0x02}) << 1) \oplus ((y \& \texttt{0x04}) << 2) \oplus ((y \& \texttt{0x08}) << 3) \\ &\oplus (-((y \& \texttt{0x10}) >> 4) \& \texttt{0x1b}) \oplus (-((y \& \texttt{0x20}) >> 5) \& \texttt{0x6c}) \\ &\oplus (-((y \& \texttt{0x40}) >> 6) \& \texttt{0xab}) \oplus (-((y \& \texttt{0x80}) >> 7) \& \texttt{0x9a}). \end{aligned}$$