Extending Glitch-Free Multiparty Protocols to Resist Fault Injection Attacks

Okan Seker[1], Thomas Eisenbarth[1], and Rainer Steinwandt[2]

[1] Worcester Polytechnic Institute, Worcester, MA, USA
{oseker,teisenbarth}@wpi.edu
[2] Florida Atlantic University, USA
rsteinwa@fau.edu

Abstract. Side channel analysis and fault attacks are two powerful methods to analyze and break cryptographic implementations. Recently, secure multiparty computation has been applied to prevent side channel attacks. While multiparty computation is known to be fault resistant as well, the particular schemes popular for side channel protection do not currently offer this feature. In this paper we introduce a new secure multiparty circuit to prevent both fault attacks and side channel analysis. The new scheme builds on an existing side channel countermeasure and extends it to preserve errors and propagate them until the end of the circuit. A new recombination operation ensures randomization of the output in the case of an error, ensuring that nothing can be learned from the faulty output. After introducing the new secure multiparty circuit, we show how it can be applied to AES and present the performance and security analysis.

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 have 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 [6, 1]. Similarly, data on power [19], sound [11] or electromagnetic emanation [9] of a target implementation is measured in a side channel analysis to learn information about the secret state. Another studied physical attack are *probing attacks*, where Ishai et al. [17] construct a generic countermeasure, which has also been analyzed in the context of side channels.

Due to the effectiveness of physical attacks, countermeasures to both fault and side channel attacks have been studied extensively. A common technique to counter a fault induction is error detection through adding redundancy. Often, reliable error detection requires duplication of computation in space or time [1], especially for symmetric ciphers, to achieve the desired error detection ratio. For asymmetric ciphers, lower overheads are often possible [32, 14]. Another direction of fault countermeasures are *infective* countermeasures, which aim at randomizing the secret state if an error occurs. Lomné et al. [20] introduced an infective countermeasure using multiplicative random masking. Gierlichs et al. [13] present the idea of dummy rounds. However, these countermeasures are broken by Bastellini et al. [2] by using bias on the multiplicative mask and the use of dummy rounds respectively. The second is improved by Tupsamudre et al. in [33]. This updated scheme has been analyzed in [3], showing that getting the countermeasure right and efficient is difficult.

For the prevention of side channel leakage, one popular and effective countermeasure is *masking* [7], which splits the secret state into several shares in a randomized fashion, as done in secret sharing. Computations are then performed on the shares, as done in secure multiparty computation (SMC). Examples include Threshold implementation [24] and its generalization [5, 27] or a proposal by Goubin et al. [15] and another one by Roche and Prouff [29]. Usually, the shares are not redundant, as the goal of masking is to provide as little information about the shared secret from a subset of the shares as possible.

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 detail, combining ad-hoc

methods can have adverse effects [26, 21]. Furthermore, overheads are huge and become larger for combined methods. Nevertheless, resistance against both attacks is important, even more since attacks can be combined to have greater effect on partially protected implementations [28, 20]. So far only limited prior work exists. Schneider et al. [30] recently proposed a combined side-channel and fault countermeasure using threshold implementations and error detecting codes [22]. Their proposal, while efficient, fault coverage depends on the fault distribution. While it detects faults, it does not by itself ensure the randomness of the output. Another countermeasure is introduced by De Cnudde et al. [8] which combines threshold implementations with error cascading. The idea is to forwarding the input unless a faulty encoding is detected.

Our contribution. In this paper we examine fault resistance of the glitch-free secure multiparty circuits proposed in [29] and propose a new combined protection scheme for both side channel and fault attacks. After introducing our fault models, we analyze 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 are explained. It is observed that using previous multiplication schemes, the errors do not propagate and become undetectable, making the circuit vulnerable to fault attacks.

We propose a new multiplication scheme in which errors are propagated by the algebraic operations and thus will propagate until the end of the circuit. With our new recombination operation, the output is randomized 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 operation. In the presence of faults, computation spreads the error to all shares and infects them for the recombination operation. Therefore, the attacker gets random outputs at the end of the cryptographic operation. 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.

The rest of the paper is structured as follows. In Section 2 we introduce the necessary background on Shamir's secret sharing, multiparty computation, and the scheme proposed in [29]. Using these protocols we explain the fault behavior on secret sharing schemes and give our main claim in Section 3 and explain the fault propagation of each operation. In Section 4 a new error preserving multiplication scheme is introduced and the preservation of errors as well as recombination operation is explained. Also we give the security analysis in Section 4.3. In Section 5 we show how this scheme can be applied to AES. Finally we present the performance analysis and experimental results.

2 Preliminaries

We start with introducing the concepts of Shamir's secret sharing and secure multiparty computation (SMC). Also we explain their analysis with respect to side channel resistance.

2.1 Shamir's Secret Sharing

Secret sharing schemes allow n participants to share a secret so that any $d + 1$ out of the n participants can reconstruct the secret, while any d or fewer participants learn nothing about the secret. In a prominent construction, due to Shamir [31], the secret is shared by a degree-d polynomial with coefficients in a finite field. Sharing and reconstruction process can be summarized as follows:

Secret Sharing: In order to share a secret a_0, a trusted dealer generates a random polynomial $p(x) = a_0 + a_1x + \ldots + a_dx^d$ of degree d and sets the constant term to the secret a_0. Then the dealer evaluates $p(x)$ at public coordinates α_i and provides the i^{th} player with its *share* $p(\alpha_i)$.

Secret Reconstruction: All coefficients of the secret polynomial $p(x)$ can, e. g., be found using an $n \times n$ Vandermonde matrix $V = (\alpha_j^i)_{i,j=0,\ldots,n-1}$. If $(\lambda_0^j, \ldots, \lambda_{n-1}^j)$ corresponds to the $j+1^{\text{th}}$ row of V^{-1} then

$$a_j = \sum_{i=0}^{n-1} p(\alpha_j)\lambda_j^i, \tag{1}$$

where $a_{d+1} = a_{d+2} = \ldots = a_{n-1} = 0$. With less than $d+1$ shares, all possible values of a_0 are equiprobable.

2.2 Secure Multiparty Computation

Secure Multiparty Computation (SMC) enables the distributed computation on shared data and can be implemented by means of sharing schemes [4]. Suppose we have two secret values f_0 and g_0 and each player gets two shares of the form $F(\alpha_i)$ and $G(\alpha_i)$, respectively, where $F(x) = f_0 \oplus f_1 x \oplus \ldots \oplus f_d x^d \in \mathrm{GF}(2^8)[x]$ and $G(x) = g_0 \oplus g_1 x \oplus \ldots \oplus g_d x^d \in \mathrm{GF}(2^8)[x]$. Here and throughout we assume a polynomial basis representation of $\mathrm{GF}(2^8)$ using the irreducible degree-8 polynomial from the AES specification. The basic operations needed below can be defined as follows:

Affine transformation of a secret: An affine transformation $\mathcal{L}(x) = ax \oplus b$ can be computed on a secret value by applying the transformation on the shares locally; $\mathcal{L}(F(\alpha_i))$ for $i = 1, \ldots, n$.

Addition of two secrets: Players can compute the addition of two secret values $f_0 \oplus g_0$ by $H(\alpha_i) = F(\alpha_i) \oplus G(\alpha_i)$ on each share separately for $i = 0, \ldots, n-1$.

Squaring operation: Players can calculate the square operation $\eta_k(y) = y^{2^k}$ without leaking any information, if shares satisfies the following conditions [25];

1. $\alpha_i \neq 0$ for $i = 1, \ldots, n$.
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 $\eta_k(F(\alpha_i)) = F'(\alpha_i)$ where $F'(x)$ is the polynomial whose coefficient are calculated by applying the same operation to the coefficients of $F(x)$. The family of shares $\eta_k(F(\alpha_i))$ for $i = 0, \ldots, n-1$ is a valid secret shares of $f_0^{2^k}$. However, communication between players is needed to do the reordering of the secret shares.

Multiplication of two secrets: If players want to compute the multiplication of two secret values $f_0 \cdot g_0$, then the algorithm is more complex than the previous operations. The following algorithm is based on the work by Gennaro et al. in [12] and it is a simplified version of the original multiplication first proposed by Ben-Or et al. [4]:

1. Each player P_i will compute $H(\alpha_i) = F(\alpha_i) \cdot G(\alpha_i)$,
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.
3. Each player P_i computes $\mathbf{Q}(\alpha_i) = \sum_{j=0}^{n-1} \lambda_j^0 \mathcal{Q}_j(\alpha_i)$ where $(\lambda_0^0, \ldots, \lambda_{n-1}^0)$ represents the first row of the inverse Vandermonde matrix.

The shares calculated in Step 1 belong to a degree $2d$ polynomial H with $H(0) = f_0 g_0$. Therefore the shares $(H(\alpha_0), \ldots, H(\alpha_{n-1}))$ do not constitute an $(n, d)-$ sharing. In step 2 and 3 degree reduction and randomization is done in order to generate a proper (n, d)-sharing representation of $f_0 g_0$. The final relation between shares $\mathbf{Q}(\alpha_i)$ and the secret value $f_0 \cdot g_0$ can be seen as follows:

$$\begin{aligned} \mathbf{Q}(0) &= \lambda_1 H(\alpha_1) \oplus \ldots \oplus \lambda_n H(\alpha_n) \\ &= \sum_{j=1}^{n} H(\alpha_j)\lambda_j = f_0 g_0. \end{aligned}$$

Hence, $\mathbf{Q}(\alpha_i)$ for $i = 0, \ldots, n-1$ will be a correct shared representation of $f_0 \cdot g_0$.

2.3 Secure Multiparty Computations as a Side Channel Countermeasure

Secure multiparty computations and secret sharing enable to split the information into shares in such a way that neither the shares nor the computations on them reveals any critical information. Roche and Prouff proposed to use the above described SMC as *multiparty circuit* (MPC) to counteract higher-order side channel analysis, and showed how to apply it to AES [29]. The main idea is using Shamir's secret sharing to share the sensitive variable to protect the circuit. The computations on these shares are done by using SMC methods from Ben-Or et al. [4] and Gennaro et al. [12]. They introduce a *d-th order glitches adversary model* and an (n, d)-*SMC*. Also they show that an (n, d)-SMC is secure against a d-th order side channel adversary. That is, an (n, d)-SMC resists d-th order side channel attacks, even in the presence of glitches. An (n, d)-SMC consists of sub-circuits, which only ever process one share of any variable. Moreover, output of these sub-circuits can be used as an input to other sub-circuits only if they constitute a proper sharing of intermediate values.

In order to construct a secure AES implementation, (n, d)-SMCs for addition, affine transform, and multiplication over GF(2^8) are presented. AES' `MixColumns`, `ShiftRows` and `AddRoundKey` only need addition and affine transform, which are fairly efficient. The `SubBytes` operation, however, requires several multiplications and squarings, making its implementation fairly expensive. Also AES' `KeyScheduling` can be seen as an affine transformation and S-box combination itself. Using these operations Roche and Prouff described a $d-$glitch free AES-128 implementation. Details on the AES implementation can also be found in Section 5.

Moradi et al. [23] provide a first implementation of this scheme in hardware, as well as a practical side channel analysis of their implementation. Similarly, Grosso et al. [16] examine the performance of existing masking schemes in software for low-power microcontrollers. Both works conclude that the scheme comes with a significant overhead, even when compared to other side channel protection schemes. The latter work proposes the usage of packed secret sharing to make the scheme more efficient for higher protection orders.

3 Fault Behavior of Secure Multiparty Computations

Another important feature of secure multiparty computations is that it can be used as a fault injection countermeasure. Clearly the fault resistance properties of the system depend on the number of shares. Depending on the number of altered shares, the error is *detectable*, *undetectable* or *correctable*. To be able to *correct* faults on d shares, previously proposed schemes require at least $3d + 1$ shares [4]. Furthermore, *robust* multiplication requires cheater detection at the input and output of every multiplication, which is a very costly operation [12].

To be able to introduce a fault detection mechanism, we need to observe the fault behavior of certain operations. In this section we examine the fault propagation of the SMC operations. We start with introducing the fault model.

The general fault models are discussed in [20] and the classification is explained using the invariance in the fault models:

1. *Fault attacks based on a fixed fault diffusion pattern:* The attacker can inject a random fault on a specific part of an intermediate variable and using the diffusion pattern of the cryptographic algorithm, can gain information.
2. *Fault attacks based on a fixed fault logical effect:* The second type of fault attacks is based on the fact that the logical effect of the error is fixed, i.e., the logical effect of the fault to the system is $x' = x \star e$, where x is the intermediate variable, e is the error and $\star$ is a logical operation.

In our model the intermediate values are proper shares. The attacker can induce additive faults to these shares and the logical effect of error e to the i^{th} share will be $p(x_i) \oplus e$. The additive model describes a wide class of errors that can be observed in practice.

3.1 Fault Behavior

The additive error model allows us to observe the effect of the faults to the system easily. The main idea is detecting the faults using the changes in the secret sharing polynomial.

We capture the effect of faults with the error polynomial, denoted by $\Delta(x)$. Fault detection is simply done by checking the degree of $\Delta(x)$. For an (n, d) secret sharing scheme with $n = d + \varepsilon + 1$, faults are undetectable if the degree of $\Delta(x)$ is d, i.e., the coefficients of the terms of degree $d+1, \ldots, n-1$ should be zero. We refer to these terms as *error detection terms*.

In fact, we consider the error detection terms as a system of linear equations whose unknowns are faults and the attacker has ε degrees of freedom. If the adversary is able to inject k faults to the system, then for $\varepsilon \geq k$, the system has only the trivial solution, i.e., all additive errors will be detected. For $\varepsilon < k$, the system will, usually, have many solutions. However, the detection probability for random faults is still very high. Using this motivation, we introduce the following claim which constitute a basis of our error detection method.

Claim 1. If there are $k(\leq \varepsilon)$ faulty shares in an (n, d) secret sharing scheme with $n = d + \varepsilon + 1$ then the corresponding error polynomial $\Delta(x) = \delta_0 \oplus \delta_1 x \oplus \ldots \oplus \delta_{d+1} x^{d+1} \oplus \ldots \oplus \delta_{d+\varepsilon} x^{d+\varepsilon}$ has at most $k-1$ zero coefficients in the *error detection terms* $\delta_{d+1}, \ldots, \delta_{d+\varepsilon}$. Hence $\deg(\Delta(x)) \geq d+1$ and the error is detectable.

Proof. Let $\alpha_0, \ldots, \alpha_{n-1}$ be public evaluation points and $F(x) = f_0 \oplus f_1 x \oplus \ldots \oplus f_d x^d$ be the secret sharing polynomial. Without loss of generality, assume there exist k errors in the first k shares of a secret sharing. From Equation ((1)), the relation between faulty shares and coefficients of $F(x)$ and $\Delta(x)$ can be seen as follows:

$$V^{-1} \underbrace{\begin{pmatrix} F(\alpha_0) \oplus \sigma_0 \\ \vdots \\ F(\alpha_{k-1}) \oplus \sigma_{k-1} \\ F(\alpha_k) \\ \vdots \\ F(\alpha_{n-1}) \end{pmatrix}}_{\text{Faulty shares}} = V^{-1} \underbrace{\begin{pmatrix} F(\alpha_0) \\ \vdots \\ F(\alpha_{k-1}) \\ F(\alpha_k) \\ \vdots \\ F(\alpha_{n-1}) \end{pmatrix}}_{\text{Points of } F(x)} \oplus V^{-1} \underbrace{\begin{pmatrix} \sigma_0 \\ \vdots \\ \sigma_{k-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{\text{Points of } \Delta(x)} = \begin{pmatrix} f_0 \\ \vdots \\ f_d \\ 0 \\ \vdots \\ 0 \end{pmatrix} \oplus \begin{pmatrix} \delta_0 \\ \vdots \\ \delta_d \\ \delta_{d+1} \\ \vdots \\ \delta_{n-1} \end{pmatrix}$$

where $V = (\alpha_j^i)$ is an $n \times n$ Vandermonde matrix. Using $V^{-1}_{i,j} = \lambda_j^i$ we can form a system of linear equations for error detection terms of $\Delta(x)$:

$$\begin{aligned}
\sigma_0 \lambda_0^{n-1} \oplus \ldots \oplus \sigma_{k-1} \lambda_{k-1}^{n-1} &= \delta_{d+\varepsilon} \\
\sigma_0 \lambda_0^{n-2} \oplus \ldots \oplus \sigma_{k-1} \lambda_{k-1}^{n-2} &= \delta_{d+\varepsilon-1} \\
\sigma_0 \lambda_0^{n-3} \oplus \ldots \oplus \sigma_{k-1} \lambda_{k-1}^{n-3} &= \delta_{d+\varepsilon-2} \\
&\vdots \\
\sigma_0 \lambda_0^{n-\varepsilon} \oplus \ldots \oplus \sigma_{k-1} \lambda_{k-1}^{n-\varepsilon} &= \delta_{d+1}
\end{aligned}$$

As each λ_i^j belongs to the inverse Vandermonde matrix, the above equations are linearly independent. Assume that $\delta_i = 0$ for $i \in I$ and $I \subset \{d+1, d+2, \ldots, d+\varepsilon\}$ with $|I| = k$. The system becomes a homogeneous system with k unknown and k equations. So the only solution will be trivial one. However, if $|I| = k-1$ then, we will get non-trivial solutions also.

Therefore using k errors, at most $k-1$ error detection terms are set to zero. Hence, at least one of the error detection coefficients of $\Delta(x)$ becomes non-zero, $\deg(\Delta(x)) \geq d+1$, and the error can be detected. □

The last part of this claim, i.e., $\deg(\Delta(x)) \geq d+1$ and hence the error being detectable, was already proven in [34, Sec. 4.1].

3.2 Propagation of Errors

Fault injection countermeasures are less concerned with the correction of errors. Main goal is to *detect* the fault and to ensure nothing can be learned from a faulty output. Therefore our aim is to preserve faults and to detect them only once when the output is produced. To be able to detect the faults using Claim 1, we need to observe the error propagation for each SMC component. In this section we discuss the preservation of faults and show the vulnerabilities of the computations. For the descriptions we use the same notation as in Section 2.2.

Affine transformation of a share: Assume an affine operation $\mathcal{L}(x) = ax \oplus b$ is used to calculate the secret $\mathcal{L}(f_0)$. $\mathcal{L}(x)$ changes the faults only in magnitude while the localization of the faults are preserved. Using Vandermonde representation we can see the propagation as follows;

$$V^{-1}\begin{pmatrix} \mathcal{L}(F(\alpha_0)\oplus\sigma_0) \\ \vdots \\ \mathcal{L}(F(\alpha_{k-1})\oplus\sigma_{k-1}) \\ \mathcal{L}(F(\alpha_k))) \\ \vdots \\ \mathcal{L}(F(\alpha_{n-1})) \end{pmatrix} = V^{-1}\left[\underbrace{\begin{pmatrix} aF(\alpha_0)\oplus b \\ \vdots \\ aF(\alpha_{k-1})\oplus b \\ aF(\alpha_k)\oplus b \\ \vdots \\ aF(\alpha_{n-1})\oplus b \end{pmatrix}}_{\text{Degree} = d} \oplus \underbrace{\begin{pmatrix} a\sigma_0 \\ \vdots \\ a\sigma_{k-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{\text{Degree} > d}\right].$$

$(aF(\alpha_0)\oplus b, \ldots, aF(\alpha_{n-1})\oplus b)$ is a valid secret sharing of $af_0 \oplus b$ and its polynomial is of degree d. So faults are preserved by $(a\sigma_0, \ldots, a\sigma_{k-1}, 0, \ldots, 0)$. Moreover, the behavior of the faulty shares is the same if they are injected during the computation of the affine transformation.

Addition of two shares: Assume the sum of two shared secrets $f_0 \oplus g_0$ is wanted to be calculated. Each party computes the sum of its shares locally as $H(\alpha_i) = F(\alpha_i) \oplus G(\alpha_i)$ and clearly if one of the shares is faulty then the corresponding output share will also be faulty.

Attacker can inject faults to both polynomials in different or in same shares, however the overall effect to the system will be similar. In this case however there is a probability that faults become undetectable. The error detection coefficients can be zero if corresponding coefficients of F and G are equal that is,

$$f_{d+1} \oplus g_{d+1} = \ldots = f_{n-1} \oplus g_{n-1} = 0. \tag{2}$$

As the nature of secret sharing both F and G are random polynomials. Therefore the probability of Equation (2) is $(1/|\mathbb{F}|)^{d+\varepsilon}$.

Squaring operation: As in the affine transformation, each player applies the transformation on its share. The operation changes the magnitude of faults. The propagation of errors can be summarized as follows, where $(i_1, \ldots, i_n)$ is a permutation of indexes $(1, \ldots, n)$.

$$V^{-1}\begin{pmatrix} [F(\alpha_1)\oplus\sigma_1]^{2^t} \\ \vdots \\ [F(\alpha_k)\oplus\sigma_k]^{2^t} \\ [F(\alpha_{k+1})]^{2^t} \\ \vdots \\ [F(\alpha_n)]^{2^t} \end{pmatrix} = V^{-1}\left[\underbrace{\begin{pmatrix} [F'(\alpha_{i_1})]^{2^t} \\ \vdots \\ [F'(\alpha_{i_k})]^{2^t} \\ [F'(\alpha_{i_{k+1}})]^{2^t} \\ \vdots \\ [F'(\alpha_{i_n})]^{2^t} \end{pmatrix}}_{\text{Degree} = d} \oplus \underbrace{\begin{pmatrix} \sigma_1^{2^t} \\ \vdots \\ \sigma_k^{2^t} \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{\text{Degree} > d}\right].$$

Hence, the points of the degree-d polynomial correspond to valid shares of secret $f_0^{2^t}$ and the error is propagated.

Multiplication of two shares: Now we examine the propagation of faults in the multiplication scheme introduced in Section 2.2 step by step.

1. Players whose shares are faulty calculate $H(\alpha_i) = F(\alpha_i)G(\alpha_i) \oplus \sigma_i$ where the magnitude of the fault depends on the polynomial on which it is injected. Assume σ_i^F and σ_i^G represent the faults of the i^{th} shares of F and G respectively. The general form of a share can be summarized as

$$H(\alpha_i) = F(\alpha_i)G(\alpha_i) \oplus F(\alpha_i)\sigma_i^G \oplus G(\alpha_i)\sigma_i^F \oplus \sigma_i^F\sigma_i^G. \tag{3}$$

   Equation (3) is still valid if only one polynomial is faulty. For example if only $F(\alpha_i)$ is faulty then the resulting share will be $H(\alpha_i) = F(\alpha_i)G(\alpha_i) \oplus G(\alpha_i)\sigma_i^F$.
2. Each player generates a random degree d polynomial $\mathcal{Q}_i(x)$ with $\mathcal{Q}_i(0) = H(\alpha_i)$ and sends the corresponding values to the other players. In this step all faults in H spread to shares and hence they become undetectable.
3. Each player calculates its share by adding all corresponding values $\mathcal{Q}_i(x)$ and gets a valid sharing of the faulty secret value.

   As a result, the final secret sharing polynomial will be a valid (n, d) secret sharing scheme and an adversary is able to inject faults without detection.

We analyzed the error propagation for each SMC operation and observed the vulnerability of the multiplication scheme. In order to generate a fault resistant implementation we need error-preserving SMC addition and affine transform and most importantly an error-preserving SMC multiplication. Therefore, in the next section we extend the multiplication scheme by Gennaro et al. [12] to an error preserving multiplication scheme.

4 Error Preserving Multiparty Computation

The error-preserving multiparty computation scheme below differs significantly from other proposals, such as robust SMC. Unlike, e. g., [12, 10], 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 not 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.
- **Error-Preserving Computation** Once errors occur, the error will spread through the state and remain part of the state. The advantage of this is that error detection only needs to be done once, when 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 polynomial 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 [29] in a straightforward manner. However, the *multiplication* is difficult to construct in a way such 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 an additional error detection coefficient. The error detection coefficients have several advantages: 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, which is applied at the output and ensures that outputs are random if an error occurred. SMC squaring and affine transform can still be used as before, as they do not influence the the fault propagation in a negative way.

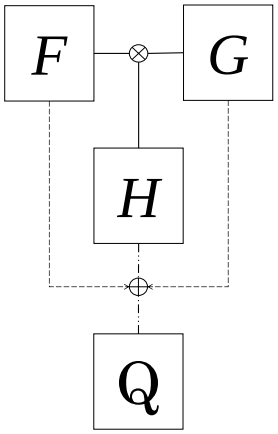

Fig. 1. Framework of the multiplication scheme: The error detection terms of H, F and G are fed to shares of $\mathbf{Q}$ during the randomization step.

4.1 Error Preserving Multiplication

Multiplication is the most critical SMC operation. Even without error detection, the multiplication is the reason to require $n > 2d$, since the product of two degree-d polynomials is of degree-$2d$. To achieve error detection, more shares are needed. In fact, we show that to detect ε errors, a total of $n > 2d + \varepsilon$ shares are needed. A brief representation of the error preserving multiplication can be seen in Figure 1.

In the new scheme, the error propagation and infective computation is achieved by including the *error detection terms* of input polynomials and the intermediate polynomial $H(x)$ in the calculation of the output. Recall that the error detection terms are the terms whose degrees are between $d+1, \ldots, n-1$. Clearly, if there are no faults injected to the system, they remain unchanged at zero. Also side channel leakages are eliminated by using $\mathcal{Q}_i$ which is a new random polynomial. All players only use their local information to construct their part $\mathcal{Q}'_i$. A step-by-step description of our new multiplication scheme that can resist ε faults follows:

1. Each player P_i locally computes $H(\alpha_i) = F(\alpha_i) \cdot G(\alpha_i)$.
2. Each player P_i generates a degree d polynomial $\mathcal{Q}_i(x)$ such that $\mathcal{Q}(0) = H(\alpha_i)$. To propagate faults from the input, the player P_i also multiplies its i^{th} share of $H(x)$ and of $G(x) \oplus F(x)$ the corresponding Vandermonde element. The result is added to a specific share of $\mathcal{Q}_i$ to propagate the error information to the output of the multiplication. Then P_i sends the $\mathcal{Q}'_i(\alpha_j)$ values to player P_j for $j = 0, \ldots, n-1$ and $j \neq i$.

$$\mathcal{Q}'_i(\alpha_j) = \begin{cases} \mathcal{Q}_i(\alpha_j) \oplus \frac{\lambda_i^{n-(j-1)}}{\lambda_i^0} H(\alpha_i) & \text{if } 0 \leq j < \varepsilon \\ \mathcal{Q}_i(\alpha_j) \oplus \frac{\lambda_i^{n-(j-1)}}{\lambda_i^0} (F(\alpha_i) \oplus G(\alpha_i)) & \text{if } \varepsilon \leq j < \varepsilon + d \cdot \\ \mathcal{Q}_i(\alpha_j) & \text{if } \varepsilon \leq j \leq n-1 \end{cases}$$

3. In the third step each player calculates its new share $\mathbf{Q}(\alpha_j)$ by the same summation $\mathbf{Q}(\alpha_j) = \sum_{i=0}^{n-1} \lambda_i^0 \mathcal{Q}'_i(\alpha_j)$. However, the first $\varepsilon + d$ players also get an error detection coefficient of $H(x)$ or $F(x) \oplus G(x)$. Since these values are masked, these operations do not reveal any sensitive information. For example, player P_j $(0 \leq j < \varepsilon)$ calculates its share $\mathbf{Q}(\alpha_j)$ by:

$$\begin{aligned}\mathbf{Q}(\alpha_j) &= \sum_{i=0}^{n-1} \lambda_i^1 \mathcal{Q}_i'(\alpha_j) \\ &= \left[\lambda_0^0 \mathcal{Q}_1(\alpha_j) \oplus \ldots \oplus \lambda_{n-1}^0 \mathcal{Q}_{n-1}(\alpha_j)\right] \oplus \left[\lambda_0^{n-j} H(\alpha_0) \oplus \ldots \oplus \lambda_n^{n-j} H(\alpha_{n-1})\right] \\ &= \left[\lambda_0^0 \mathcal{Q}_1(\alpha_j) \oplus \ldots \oplus \lambda_{n-1}^0 \mathcal{Q}_n(\alpha_j)\right] \oplus h_{n-j-1}.\end{aligned}$$

The output shares can be summarized as in Equation 4 where h_i, g_i and f_i represent the coefficients of i^{th} degree term of H, G and F respectively.

$$\mathbf{Q}(\alpha_i) = \begin{cases} \mathbf{Q}(\alpha_i) \oplus h_{n-i-1} & \text{if } 0 \le j < \varepsilon \\ \mathbf{Q}(\alpha_i) \oplus g_{n-i-1} \oplus f_{n-i-1} & \text{if } \varepsilon \le j < \varepsilon + d \\ \mathbf{Q}(\alpha_i) & \text{if } \varepsilon + d \le j < n \end{cases}. \tag{4}$$

Clearly, if only one of the polynomials is faulty, i.e. $\deg(G) > d$ or $\deg(F) > d$ then $\deg(H) > 2d$. If both polynomials are faulty, then $\deg(H)$ can be arranged as $2d$. However, faults are still propagated using the terms of G and F as explained in Step 2 with a very high likelyhood. A more detailed security analysis is provided in Section 4.3.

Algorithm 4.1: MULTIPLICATION$((F(\alpha_i), G(\alpha_i))_{1 \le i \le n})$

for $i = 0$ to $n-1$
do
- $H(\alpha_i) = F(\alpha_i) \cdot G(\alpha_i)$
- Generate a random degree d polynomial $\mathcal{Q}_i(x)$ with $\mathcal{Q}_i(0) = H(\alpha_i)$
- **for** $i = 0$ to $n-1$ **do**
  - **if** $(0 \le j < \varepsilon)$: $P_j \leftarrow \mathcal{Q}_i(\alpha_j) \oplus \frac{\lambda_i^{n-j}}{\lambda_i^0} H(\alpha_i)$
  - **else if** $(\varepsilon \le j < \varepsilon + d)$: $P_j \leftarrow \mathcal{Q}_i(\alpha_j) \oplus \frac{\lambda_i^{n-j}}{\lambda_i^0} (F(\alpha_i) \oplus G(\alpha_i)$
  - **else**: $P_j \leftarrow \mathcal{Q}_i(\alpha_j)$

for $i = 0$ to $n-1$
do $\mathbf{Q}(\alpha_i) = \sum_{j=0}^{n-1} \lambda_j^0 \mathcal{Q}_j'(\alpha_i)$

return $((\mathbf{Q}(\alpha_i))_{0 \le i \le n-1})$

4.2 Recombination Operation and Infective Computation

The advantage of using error preserving multiplication is that only one error detection is sufficient for fault detection. As explained before, our aim is to randomize the output in case of injected faults, so that the attacker cannot learn any information. Next we explain the *infectiousness* of the faults, while introducing the fault detection and recombination algorithm.

The recombination operation is composed of two main steps *recombination*, and *reconstruction*. The inputs of the operation are shares of the secret value $F(\alpha_i)$ for $0 \le i \le n-1$ and a non-zero random vector $(r_0, \ldots, r_{\varepsilon+d-1})$ where $r_i \in \mathrm{GF}(2^8) \setminus 0$ and the outputs are the secret value f_0 and fault decision.

1. ***Recombination:*** In this step we reshare the input polynomial and a non-zero random multiple of error detection coefficients are added to shares.
   The importance of this step is, error detection terms are actually generated by faults, therefore the adversary is still able to get information from the output. Using random values we ensure the randomization of the secret value.
   (a) Each player P_i generates a degree d polynomial $\mathcal{Q}_i(x)$ such that $\mathcal{Q}_i(0) = F(\alpha_i)$. Then P_i sends the $\mathcal{Q}'_i(\alpha_j)$ values to player P_j and error detection coefficients are fed to final shares by using the same method as in multiplication.

$$\mathcal{Q}'_i(\alpha_j) = \begin{cases} \mathcal{Q}_i(\alpha_j) \oplus r_i \frac{\lambda_i^{n-(j-1)}}{\lambda_i^0} F(\alpha_i) & \text{if } 0 \leq j < \varepsilon + d \\ \mathcal{Q}_i(\alpha_j) & \text{if } \varepsilon \leq j \leq n-1 \end{cases}.$$

   (b) Each player calculates its new share $\mathbf{Q}(\alpha_j)$ by $\sum_{i=0}^{n-1} \lambda_i^0 \mathcal{Q}_i(\alpha_j)$. However, the first $\varepsilon + d$ players also generate a non-zero random value and add it to the shares with the corresponding coefficient.

$$\mathbf{Q}(\alpha_i) = \begin{cases} \mathbf{Q}(\alpha_i) \oplus r_i f_{n-i-1} & \text{if } 0 \leq j < \varepsilon + d \\ \mathbf{Q}(\alpha_i) & \text{if } \varepsilon + d \leq j < n \end{cases}. \tag{5}$$

2. ***Reconstruction:*** In the last step Secret value and error detection coefficients are reconstructed by using following formula :

$$f_j = \sum_{i=0}^{n-1} \lambda_i^j \mathbf{Q}(\alpha_i) \text{ for } j = 0, n-1, \ldots, d+1$$

   Clearly, if F is faulty, then at least one of the error detection terms is non-zero. In the second step secret value is randomized by using these terms therefore, infective computation is achieved.

Algorithm 4.2: RECOMBINATION$((F(\alpha_i))_{0 \leq i < n}, (r_0, \ldots, r_{\varepsilon+d-1}))$

Recombination:
for $i = 0$ to $n-1$
do {
  Generate a random degree d polynomial $\mathcal{Q}_i(x)$
  with $\mathcal{Q}_i(0) = F(\alpha_i)$
  for $i = 0$ to $n-1$
  do {
    if $(0 \leq j < \varepsilon + d)$
      $P_j \leftarrow \mathcal{Q}_i(\alpha_j) \oplus r_j \frac{\lambda_i^{n-j}}{\lambda_i^0} F(\alpha_i)$
    else
      $P_j \leftarrow \mathcal{Q}_i(\alpha_j)$
  }
}
for $i = 0$ to $n-1$
do $\mathbf{Q}(\alpha_i) = \sum_{j=0}^{n-1} \mathcal{Q}'_j(\alpha_i)$
Reconstruction:
$f_j = \sum_{i=0}^{n-1} \lambda_i^j \mathbf{Q}(\alpha_i)$ for $j = 0, n-1, \ldots, d+1$
for $i = d+1$ to $n-1$
do {
  if $(f_i ! = 0)$
    Fault decision$\leftarrow$Fault is detected
}
return (f_0, Fault decision)

4.3 Security Analysis

Up to now, we explained three main properties of our multiplication scheme: *error detection*, *error-preservation* and *infective computation*. We now discuss the security features of our combined countermeasure under specific attack models.

First we introduce security features on side channel analysis. In the model, an adversary can observe power traces of the circuit during a finite number of executions to perform a d^{th}th order side channel attack. Although (n, d)-SMC can be attacked by a $d + 1^{th}$ order side channel analysis, attacks become increasingly impractical as d increases [7]. We use the same approach to split information between n players as done by Roche et al. [29]. The sensitive variables are masked by an (n, d)-secret sharing scheme and SMC enables us to do the distributed computations securely. Also in the proposed multiplication scheme, error detection coefficients are calculated in a distributed fashion. Each player only uses local informations which are produced share and a random polynomial. Therefore, produced share which contains sensitive informations is masked by random polynomial and the communication between players can be done securely. Finally, in the last step, players uses the same equation and some shares will get a error detection coefficient. All calculations are done locally and all sensitive variables are masked by a random polynomial.

Moreover, in the *recombination* we use the same arguments as in multiplication. In the first step error detection coefficients are masked by a random polynomial and all calculations are done locally. Therefore, there will be no information leakages in the communication step. As a result, the secret value is re-shared and masked with a random polynomial. In the *reconstruction* step the calculations are done using these masked shares. Hence, we are able to reconstruct the fault decision the secret value securely. Note that in case of a detectable error, the the secret value is randomized so that no information can be learned from it.

Next we state the fault resistance features of the proposed scheme. Clearly, the level of fault resistance depends on ε. As given in Claim 1, ε is deduced as the maximum number of faults that *definitely* increases the degree of the secret sharing polynomial. Using this discussion and the notation given by Schneider et al. [30] 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;

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

Since we do not have any restriction on faults, we assume that faults are selected from GF(2^8) with a uniform distribution. Assume the number of injected faults to the system is k, then using the number of solutions for the system of linear equations in Claim 1, the probability of *fault propagation* will be;

$$Pr[deg(\Delta(x)) > d] = 1 - \frac{(2^8)^{\max(k,\varepsilon)-\varepsilon} - 1}{(2^8)^k - 1}. \tag{6}$$

Therefore, in first multiplication faults are propagated with the probably 1, if $k \leq \varepsilon$. However we cannot use the Equation (6) as the fault coverage directly, since faults can be injected in different instances or one fault can spread to large number of shares. As a result faults become unstable and undetectable.

In a sequence of operations faults can become undetectable after a SMC addition or multiplication. In Section 3.2 we give the probability of undetectable faults in SMC addition. In the following, we do the security analysis.

As given previously our main idea is to propagate faults using error detection coefficients. Therefore faults become undetectable if and only if all of these terms become zero. First of all both input polynomials should be faulty and two conditions are required to generate undetectable faults.

1. Error detection coefficients between $d + 1, \ldots, 2d$ should be equal;

$$f_{d+1} \oplus g_{d+1} = \ldots = f_{2d} \oplus g_{2d} = 0.$$

2. Faults should be in the same shares and for every faulty share i, denoted by $F'(\alpha_i) = F(\alpha_i) + \sigma_{i_F}$ $G'(\alpha_i) = G(\alpha_i) + \sigma_{i_G}$, the following equation should hold:

$$\begin{aligned} F'(\alpha_i) \cdot G'(\alpha_i) &= [F(\alpha_i) + \sigma_{i_F}] \cdot [G(\alpha_i) + \sigma_{i_G}] \\ &= F(\alpha_i)G(\alpha_i) + \underbrace{F(\alpha_i)\sigma_{i_G} + G(\alpha_i)\sigma_{i_F} + \sigma_{i_F}\sigma_{i_G}}_{=0}. \end{aligned}$$

As the consequence of two conditions, the output polynomial will be a degree d polynomial and faults become undetectable. As given in Section 3.2 the probability of the first condition is $1/|\mathbb{F}|$. Similarly, we can calculate the probability of the second condition easily and it is close to $1/|\mathbb{F}|$[3]. Then the probability can be calculated as follows;

$$\begin{aligned} Propagation_{\varepsilon} &= 1 - \left(Pr[deg(\Delta(x)) \leq d] + \left(\frac{1}{\mathbb{F}}\right)^{\varepsilon+d} \right) \\ &= 1 - \left(\frac{(2^8)^{\max(k,\varepsilon)-\varepsilon} - 1}{(2^8)^k - 1} + \left(\frac{1}{2^8}\right)^{\varepsilon+d} \right). \end{aligned} \tag{7}$$

As a result Equation (7) corresponds to propagation probability of SMC multiplication. Also using same discussion it can be clearly seen that Equation (7) is valid for SMC addition.

In this section we analyzed the security features of the new multiplication scheme by means of side-channel and fault injection resistance. Also we emphasized the fault propagation probabilities for SMC addition and multiplication. Therefore, we can say that if both input polynomial are faulty in a SMC addition or multiplication then, output polynomial is also faulty with high probability.

5 Side-Channel and Fault Resistant AES Implementation

In the previous sections, we introduced our combined countermeasure against side channel analysis and fault attacks. Properties and security features are explained and in this section we will apply the scheme to design a secure multi-party circuit implementation of AES-128. We use the fundamental operations stated in Section 2 and the new error preserving multiplication. Our implementation uses an (n, d)-SMC and the security levels are dependent on the parameters. First, we explain the design of AES-128 then we give the performance analysis and experimental results of the implementation.

5.1 Implementation Details

AES-128 consists of 10-rounds of operations on its 16-byte state. The operations consist of two affine transformations `MixColumns`, `AddRoundKey`, one permutation layer `ShiftRows` and one non-linear layer the `SubBytes` operation. Since it is the only non-linear layer of AES, the MPC implementation of `SubBytes` consist of squaring and multiplications [29] and most importantly, faults injected during this part remains undetected in previous schemes. So that it is vulnerable to fault attacks.

Clearly, the algorithms for `AddRoundKey` and `MixColumns` are error preserving as given detailed in Section 3 and can be implemented in a straightforward manner. Therefore, we focus on The `SubBytes` operation which is divided into two parts:

- The power function $y \rightarrow y^{254}$ over GF(2^8), denoted by $Exp254(y)$. This part is the hardest part of the AES to protect. In order to find the output, a sequence of 4 multiplications and 3 squarings are implemented [18].

$$y \rightarrow y^2 \rightarrow y(y^2) \rightarrow (y^3)^4 \rightarrow y^3 y^{12} \rightarrow (y^{15})^{16} \rightarrow y^{12} y^{240} \rightarrow y^2 y^{252} \rightarrow y^{254}.$$

- The second part of the `SubBytes` operation is the GF(2)-affine transformation and it is denoted by $\tau(y)$. It can be efficiently implemented using the following sequence of squarings and affine transforms in GF(2^8) [29]:

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

[3] Assume $|\mathbb{F}^*|$ is denoted by q The exact probability can be calculated as follows: $\frac{q^2(q-1)+5q^2+4q}{(q+1)^4}$.

Using error preserving multiplication algorithm, 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 $(4, 1)$ secret sharing scheme.

In the similar manner, we can introduce the `KeyScheduling` since it is a combination of `SubBytes` operations and affine transformations. Finally, the `Shiftrows` operations is a rotation of states, therefore it can be done by a permutation which is obviously error preserving. Moreover, the output of these operations are also valid secret shares, therefore they constitute a proper sub-circuit as defined in Section 2.3. Hence, using the algorithms we can generate a sequence of sub-circuits and finally build our side-channel and fault-injection resistant AES-128 implementation.

5.2 Performance Analysis

First, we compare our error-preserving multiplication with the SMC multiplication by Gennaro et al. [12]. Table 1 compares the two operations 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.

Table 1. Number of operations in Gennaro at al. [12] and error preserving multiplication in Section 4.1.

	Gennaro at al. [12]			Error Preserving Multiplication		
	step 1	step 2	step 3	step 1	step 2	step 3
Field Mul.	n	n^2d	n^2	n	$n^2d + n(\varepsilon + d)$	n^2
Field Add	-	n^2d	$(n-1)n$	-	$n^2d + n(\varepsilon + d + 1)$	$(n-1)n$
Randomness	-	nd	-	-	nd	-

The additional cost of this step is $n(\varepsilon + d)$ field multiplications and $n(\varepsilon + d + 1)$ additions. On the other hand, each player generates a random degree-d polynomial and sends the corresponding values to other players as in the previous scheme. This step requires d random values and n^2 polynomial evaluations, where each evaluation costs d field multiplications and d additions.

Next, we compute the total number of SMC operations in one round of AES-128. As shown in Table 2, four main parts of AES; `SubBytes`(as a composition of $Exp254(y)$ and $\tau(y)$), `MixColumns`, `AddRoundKey` and `ShiftRows` are defined as a composition of SMCs.

Table 2. The number of SMC operations in one round of AES.

	$Exp254$	$\tau(y)$	`MixColumns`	`AddRoundKey`	`ShiftRows`
SMC Multiplication	16×4	-	-	-	-
SMC Squaring	16×7	16×7	-	-	-
SMC Addition	-	16×7	12	16×1	-
SMC Affine Transform	-	16×8	16	-	-

In order to compare the performance of the different parameters, we determine the total number of field multiplications, additions and randomness requirements of each SMC operation. The numbers for error preserving multiplications can be derived from Table 1. Similarly, using the operation descriptions in Section 2.2 we can derive the numbers for other operations. Squaring requires n^2 field multiplication and addition can be seen as n XOR. Also affine transformation requires a multiplication and an XOR for each share Using these observations we can find the total number of basic operations for SMC addition, multiplication and affine transform, as given in Table 3.

Based on these results, we provide the performance analysis and cost of different (n, d)-SMC schemes. The analysis is done by using the total number of field multiplications, additions and randomness requirements for one round of AES-128. Results are shown in Table 4.

Table 3. Number of field multiplications, additions and randomness requirements for the SMC operations.

SMC operation	Multiplication	Squaring	Addition	Affine Transform
Field Mul.	$n^2(d+1)+n(e+d+1)$	n	-	n
Field Add	$n^2(d+1)+n(e+d)$	-	n	n
Randomness	nd	-	-	-

Table 4. Total number of operations for different (n, d) scenarios.

	$\varepsilon=0$		$\varepsilon=1$		$\varepsilon=2$	$\varepsilon=3$
	(3,1) [12]	(3,1)	(4,1)	(6,2)	(5,1)	(6,1)
Field Mul.	2448	2640	4288	10656	6320	8736
Field Add	1428	2196	3696	9768	5580	7848
Randomness	192	192	256	768	320	384

First, we compare the original (3,1)-SMC implementation based on the multiplication by Gennaro at al. [12] to a $(3, 1)$-SMC implementation using the error preserving multiplication. The new scheme only requires %8 more field multiplications. The 53% more addition operations which probably impact performance less, as XORs are cheap on most platforms. Observe that, since $\varepsilon = 0$ for this parameter set, the error preserving multiplication cannot detect errors during parts of the multiplication.

Secondly, 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 [29] 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 is 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 proportional to n and d.

Finally, we analyze the number of operations while increasing the fault resistance order. As seen from the table (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 are nearly proportional to half of the fault resistance order. Therefore we can conclude that, increasing the order of fault resistance costs less than the increasing the side channel resistance.

Another overhead of our scheme is the recombination operation. We introduce this operation for fault detection, reconstruction and infective computation. As seen in the Table 5, a single recombination costs us more than an error preserving multiplication, but the number of recombination operation is exactly the same as the number of secret values. Therefore, the overall cost of this operation still enables us to do the secret reconstruction efficiently.

Table 5. Recombination Operation

	Recombination	Reconstruction
Field Mul.	$n^2(d+1)+n(2\varepsilon+2d+1)$	$n(\varepsilon+d+1)$
Field Add	$n^2(d+1)+n(\varepsilon+d-1)$	$(n-1)(\varepsilon+d+1)$
Randomness	$\varepsilon+d+nd$	-

5.3 Experimental Results

Before concluding, we give experimental results of the side-channel and fault injection resistant AES-128 implementation. We do the simulations in `SAGE` and the aim is to experimentally analyze fault resistance of

different (n, d)-SMC schemes. Since the most vulnerable part of the implementation is `SubBytes` operation, faults are injected to this operation.

First, we start with $Exp254$ operation. The idea is to experimentally verify the unstable faults by worst case scenario, in which faults are injected to the beginning of $Exp254$ operation. To be able to perform the experiments exhaustively we work on GF(2^4), as checking all possible errors on all possible sharings of all possible values is doable in reasonable time. However we did not change the sequence of multiplication and squaring operations, since our aim is to check whether this sequence is generating undetectable faults and at what rate.

First of all we generate all possible valid secret sharings. Remark that; public shares $(\alpha_0, \ldots, \alpha_{n-1})$ are fixed to efficiently calculate the squaring. And all possible faults are injected to the first share of the input polynomial. Fault detection is done when the final output the operation ($Exp254$ or entire `SubBytes`) is produced.

For example in $(4, 1)$ 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 spreads to $\varepsilon + d$ shares and these shares are become input for another multiplication. Therefore Equation (7) changes for each multiplication. Unstable can occur in such conditions and in these experiments we search for unstable errors in `SubBytes` operation. As seen in the Table 6, if we increase the $\varepsilon + d$, the probability of undetectable faults decreases with respect to two conditions in Section 4.3. And most importantly, all undetectable faults satisfy these two conditions.

Remark that, in some experiments all faults become detectable even if the propagation probability is not one. Although we do the experiments on each possible polynomials and each possible faults randomness is added in the nature of multiplication. Therefore, the numbers are not the exact values, but rather upper bounds.

Table 6. Experimental results on the probability of a faulty input generating an output with undetectable faults.

	$Exp254$	$Exp254 + \tau(y)$
(4,1)	3.6×10^{-3}	9.4×10^{-3}
(5,1)	5.2×10^{-4}	1.9×10^{-3}
(6,1)	0	0
(6,2)	3.6×10^{-3}	3.5×10^{-3}

In the second experiment, we inject the faults in the beginning of the $Exp254$ and do the fault detection in each step of $\tau(y)$. As seen in the table, probabiliites of generating undetectable faults are close to the ones of the first experiment. The reason for this is that faults become undetectable after an addition operation if and only if error detection coefficients of input polynomials are the same. And the probability of this is $(1/\mathbb{F})^{\varepsilon+d}$.

Moreover, in Table 7 we can see the probability of generating undetectable faults in GF(2^8) for different secret sharing schemes. Some probabilities are equal to each other since, it does not depend on n as given in Section 4.3. Depending on the number of faulty shares, fault become undetectable. Therefore we can use this formulas to analyze an arbitrary functions.

Table 7. Probability of generating undetectable faults in GF(2^8) where k is the number of faulty shares.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$
(4,1)	1.5×10^{-5}	3.9×10^{-4}	3.9×10^{-4}	3.9×10^{-4}
(5,1)	5.9×10^{-8}	5.9×10^{-8}	1.5×10^{-5}	1.5×10^{-5}
(6,1)	2.3×10^{-10}	2.3×10^{-10}	2.3×10^{-10}	5.9×10^{-8}
(6,2)	5.9×10^{-8}	3.8×10^{-4}	3.9×10^{-4}	3.9×10^{-4}

6 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 proposed a new secure multiparty computation to achieve both fault and side channel resistance. It is shown that the proposed secure multiparty circuits can be used to perform addition, affine transform, multiplication and squaring while resisting both well-defined fault and side channel adversaries. The advantage of the proposed scheme is a reduced overhead, as only an extra operation within the multiplication is needed.

One main idea is not to correct or detect any errors inside the circuit. Instead, we define our operations in such a way that, once an error occurs, it will remain detectable even after further computations on the shares. Since the previously used multiplication scheme is not error preserving, we propose a new error preserving multiplication. This approach allows us to delay any error detection until the final recombination step. We introduce a recombination gate which is used for both error detection and reconstruction of the secret. Hence, error detection is done when the output is produced. 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 faults. The recombination gate features another desired property: Infective computation. If an error occurs, our scheme ensures that attacker cannot learn anything since the output is random.

Finally, we give a full description of fault and side channel resistant AES implementation. Using error preserving operation and recombination operation we are able to build an ε fault resistant and d-th order side channel resistant AES-128 implementation that detects faults with a very high probability. Note that faults injected in the `SubBytes` operation will be undetectable using the scheme by Roche and Prouff, if the inversion part of this operation is targeted. We performed experiments with fault injection of all possible faults in various instances and shares of AES-128 implementation. The novel recombination gate always ensures proper randomization of affected outputs.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. Dan Boneh, RichardA. DeMillo, and RichardJ. 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.
7. 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.
8. 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.
9. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded SystemsCHES 2001*, pages 251–261. Springer, 2001.
10. 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.

11. 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.
12. 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.
13. 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.
14. Christophe Giraud. An RSA implementation resistant to fault attacks and to simple power analysis. *Computers, IEEE Transactions on*, 55(9):1116–1120, 2006.
15. Louis Goubin and Ange Martinelli. Protecting AES with Shamirs secret sharing scheme. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 79–94. Springer, 2011.
16. 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.
17. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology-CRYPTO 2003*, pages 463–481. Springer, 2003.
18. Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in gf (2m) using normal bases. *Information and computation*, 78(3):171–177, 1988.
19. Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.
20. 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.
21. 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.
22. Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*. Elsevier, 1977.
23. 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.
24. 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.
25. Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the AES using secure multiparty computation protocols. In *Cryptographic Hardware and Embedded Systems–CHES 2011*, pages 63–78. Springer, 2011.
26. 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.
27. 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.
28. 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.
29. Thomas Roche and Emmanuel Prouff. Higher-order glitch free implementation of the AES using secure multiparty computation protocols. *Journal of Cryptographic Engineering*, 2(2):111–127, 2012.
30. 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.
31. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
32. Adi Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks, November 23 1999. US Patent 5,991,415.
33. 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.

34. Maki Yoshida and Satoshi Obana. Detection of Cheaters in Non-interactive Polynomial Evaluation. Cryptology ePrint Archive, Report 2013/032, 2013. http://eprint.iacr.org/.