Higher-Order Glitches Free Implementation of the AES using Secure Multi-Party Computation Protocols

– Extended Version –

Thomas Roche[1⋆] and Emmanuel Prouff[2]

[1] ANSSI, 51 boulevard de La Tour-Maubourg 75700 Paris, France
thomas.roche(at)ssi(dot)gouv(dot)fr
[2] Oberthur Technologies, 71-73, rue des Hautes Pâtures 92726 Nanterre, France
e.prouff(at)oberthur(dot)com

Abstract. Higher-order side channel attacks (HO-SCA) is a powerful technique against cryptographic implementations and the design of appropriate countermeasures is nowadays an important topic. In parallel, another class of attacks, called *glitches attacks*, have been investigated which exploit the hardware glitches phenomena occurring during the physical execution of algorithms. Some solutions have been proposed to counteract HO-SCA at any order or to defeat glitches attacks, but no work has until now focussed on the definition of a sound countermeasure thwarting both attacks. We introduce in this paper a circuit model in which side-channel resistance in presence of glitches effects can be characterized. This allows us to construct the first glitches free HO-SCA countermeasure. The new construction can be built from any Secure Multi-Party Computation protocol and, as an illustration, we propose to apply the protocol introduced by Ben'Or *et al.* at STOC in 1988. The adaptation of the latter protocol to the context of side-channel analysis results in a completely new higher-order masking scheme, particularly interesting when addressing resistance in the presence of glitches. An application of our scheme to the AES block cipher is detailed, as well as an information theoretic evaluation of the new masking function that we call *polynomial masking*.

1 Introduction

Higher-0rder Side-Channel Analysis is a class of *physical cryptanalyses* against cryptosystems. They generalize the seminal attacks introduced in the late nineties by Kocher *et al.* [15]. Contrary to the latter ones that only exploit instantaneous leakages, HO-SCA attacks mix observations

⋆ This work has been conducted when the author was Post-doc at the University of Paris 8, Département de mathématiques, 2, rue de la Liberté; 93526 Saint-Denis, France

of several leakages to recover information about the secret parameters. The number of different leakages (*e.g.* corresponding to different times during the processing or to different locations in a circuit implementing the algorithm) defines the attack *order*.

A very common countermeasure to protect block cipher implementations against HO-SCA is to randomize the key-dependent variables by *masking* (*a.k.a. sharing*) techniques [4, 12]. The masking can be characterized by both the number n of random shares and the smallest number $d+1$ of them that are required to re-construct the variable. It is usually called a (n, d)-*sharing*. In most of masking techniques proposed until now, the parameters verify $n = d + 1$ but they can sometimes differ as in [24] and this paper. A scheme specifying how to apply (n, d)-sharing to protect an operation, or more generally an algorithm implementation, is called d^{th}-*order masking scheme*. It aims at defining a *modus operandi* that thwarts any SCA attack of order lower than or equal to d. Since the complexity of mounting a HO-SCA increases exponentially with the order (see *e.g.* [3]), a d^{th}-order masking scheme will eventually bring satisfying security when d grows. It certainly explains why they received so much attention from the SCA community and this popularity led to the construction of several d^{th}-order masking schemes with formal security proof [13, 28–30]. However proving resistance against d^{th}-order SCA is not sufficient alone to assess the practical security of an implementation. In [17], Mangard *et al.* have indeed pointed out the presence of so-called *hardware glitches* — common in CMOS technology — that deteriorate the effect of masking and imply more information leakage than the simple value of the variables specified by the implementation. Since the seminal work [17], several papers have successfully applied so-called *glitches attacks* against implementations that were secure in the classical HO-SCA adversary model. This has raised the need for implementations with proven security in the presence of glitches, but up to now, a unique masking scheme has been presented with such a proof [24]. This masking scheme is an interesting step forward but it is only resistant against 1^{st}-order SCA, which leaves open the problem of specifying cryptographic implementations secure against higher-order SCA attacks in presence of glitches.

1.1 Related Works

The problematic of specifying cryptosystem implementations thwarting d^{th}-order SCA attacks in presence of glitches is recent. Actually, to the best of our knowledge, most of the work on this subject have been done

by Nikova *et al.* [22–24]. In those papers, a scheme is proposed with proven 1^{st}-order SCA resistance in presence of glitches. Unfortunately, adapting it to also counteract SCA of order greater than 1 seems difficult while keeping the efficiency and performance on acceptable level. On the other hand, two implementations proven secure against d^{th}-order SCA attacks for any d have been proposed by Ishai *et al.* [13] and Rivain *et al.* [29]. Ishai *et al.* 's solution is dedicated to hardware implementations. It clearly does not thwart glitches attacks and modifying it accordingly is an (open) issue. Rivain *et al.* 's solution is dedicated to software contexts. When embedded in a physical device, there is no guaranty that no glitches effects will occur during the processing. Providing such a guaranty would impose that all the elementary operations are performed sequentially on a circuit which is re-initialized between each operation. However, such a process, if possible, would induce a prohibitive computational overhead.

1.2 Our Contribution

Our contribution is threefold. First, we introduce a generic framework for the design of hardware or software implementations that counteract HO-SCA in presence of glitches. This framework is built around the notion of *multi-party circuit* which is defined as the composition of several sub-circuits whose respective side-channel leakages are strictly independent. We argue that this kind of circuit can be designed thanks to temporal or spatial separation of the sub-circuits. Since the cost of such a separation is very important when assessing the implementation efficiency on standard CMOS platforms, our analysis raises the need for circuit separations that minimize the number of sub-circuits. In a second part of the paper, we draw a parallel between the construction of a HO-SCA resistant implementation inside our new framework and the classical problem of building Secure Multi-Party Computation (SMC) protocols in the context of *semi-honest* adversaries. As a matter of fact, starting from the seminal work of Ben-Or *et al.* [1], we show how to design a multi-party circuit that can implement any function with a minimum number of sub-circuits. As an example, we specify such a circuit for the AES-128 algorithm. To the best of our knowledge, our proposal is the first implementation that claims to thwart d^{th}-order SCA attacks in presence of glitches. To protect the manipulation of sensitive data, the method proposed in this paper involves a new kind of masking which is straightforwardly deduced from Shamir's secret sharing scheme [31]. This masking seems to be a good alternative to the additive masking, which was, up to now, the only solution

to achieve d^{th}-order security [13, 29] (or glitches freeness at the first order [24]). Our last contribution is the comparison of the two masking schemes in an information theoretic context (following the same outlines as in [9, 26]). We show that the algebraic complexity of the new masking function makes it much stronger against SCA attacks.

1.3 Paper Organization

The paper is organized as follows: in Section 2 are formally introduced the definitions of a d^{th}-order secure implementation in both the classical adversary model, and the adversary model in presence of glitches. Hence, a framework is introduced inside which a cryptographic circuit ruled by a SMC protocol can be proved to be *glitches free*. The SMC protocol proposed in [1] is recalled in Section 3, as well as the transition elements that make it a concrete d^{th}-order masking scheme in the context of SCA. In Section 4, a full description of the new masking scheme applied to the AES-128 block cipher is exhibited. In Section 5, the so called *polynomial masking function* is extracted from our scheme and compared through an information theoretic evaluation to the classical additive masking function. Finally we conclude and propose future work.

2 Models and Multi-Party Circuits

In this section, we introduce a framework in which the resistance of a (hardware or software) implementation against HO-SCA in presence of glitches attacks can be stated. First, we give a formal definition of the attacks. Then, in Sects. 2.2 and 2.3 we exhibit sufficient conditions and general principles to thwart the attacks.

2.1 Computation and Adversary Models

SCA attacks exploit a dependency between a subpart of the secret parameter and the variations of a physical leakage as function of a known input. This dependency results from the manipulation of some variables, called *sensitive*, by the implementation. We say that a variable Z is *sensitive* if it depends on both a known variable and a secret parameter. The physical leakage on such a variable will be viewed as a *noisy leakage function* $L(Z)$ which returns a noisy information about Z . The noisy property of $L(\cdot)$ is captured by assuming that the bias introduced in the distribution of Z given the leakage $L(Z)$ is bounded.

In this paper, we shall see the implementation targeted by a SCA attack as a *circuit*, whose formal definition is given hereafter:

Definition 1 (Circuit $\mathcal{C}_f$). *Let f be a function and let $\mathcal{O}$ be a set of elementary operations. A* circuit $\mathcal{C}_f$ implementing f thanks to operations in $\mathcal{O}$ *is an oriented graph where each cell c_i defines an elementary operation and each edge bears an intermediate variable V_{ij} which is an output of the operation c_i and an input of the operation c_j.*

Remark 1. The above notion of circuit encompasses both hardware and software implementations. In the hardware context, the set $\mathcal{O}$ may only contain the logical binary operations `XOR` *and* `AND`*. In the software context, the set $\mathcal{O}$ may only contain field operations $\oplus$ and $\otimes$ in* $\mathrm{GF}(2^m)$*, where m is the architecture size.*

Several adversary models have been proposed in the literature to capture a practical SCA attacker against a circuit $\mathcal{C}_f$. Among them, the *probing adversary model* is the most popular one. We give hereafter a formal definition of this adversary in our framework.

Definition 2 (d^{th}-order Probing Adversary Model). *Let $\mathcal{C}_f$ be a circuit with $(V_{ij})_{(i,j)\in I}$ as edges and let d be a positive integer. Let $\mathcal{L}$ be a set of noisy leakage functions. A* d^{th}-order Probing Adversary *against $\mathcal{C}_f$ is an adversary that can choose a subset J of I with $\#J = d$ and can observe the random variable $(L_{ij}(V_{ij}))_{(i,j)\in J}$, where $(L_{ij}(\cdot))_{ij}$ is a d-tuple of functions in $\mathcal{L}$.*

Remark 2. In the hardware context (when the circuit $\mathcal{C}_f$ is defined with respect to operations `XOR` *and* `AND`*), the Probing Adversary Model with $\mathcal{L}$ reduced to the identity function exactly fits with the definition given by Ishai et al. in [13]. In the software context (when the circuit $\mathcal{C}_f$ is defined with respect to operations on m-bit words with m being the architecture size), our definition corresponds to the classical notion of d^{th}-order SCA [2, 14, 19, 25]. In this case, $\mathcal{L}$ is usually defined as the set of noisy leakage functions $L(\cdot) = HW(\cdot) + \mathcal{B}(\mu, \sigma)$, where $\mathcal{B}$ is a gaussian independent noise with mean μ and standard deviation σ and where HW denotes the Hamming weight function.*

Notation. An attack performed by the adversary in Definition 2 is called *d^{th}-order probing attack*. A circuit secure against those attacks is said to be *d-probing secure*.

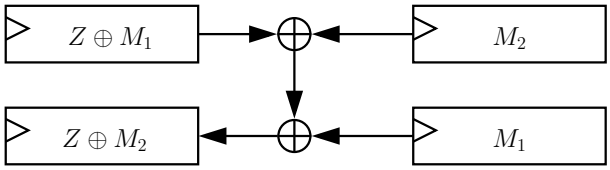
In 2003, Ishai *et al.* have exhibited in [13] a way to transform any circuit with elementary Boolean operations ($\{\texttt{XOR}, \texttt{AND}\}$) into a circuit se-

cured w.r.t. d^{th}-order Probing Adversary, for any d. This work has been extended in [29] by Rivain and Prouff to transform any circuit with operations in any finite field. The two works [13] and [29] show that achieving perfect security in the Probing Adversary Model is possible. In parallel however, several publications have shown that schemes secure in this model can still be broken in practice (see *e.g.* [17] and [18]). The reason for this is that in physical implementations (*e.g.* in CMOS), a lot of unintended switching activities occur. These unintended switching activities are usually referred to as dynamic hazards or *glitches*.

The effect of glitches on the side-channel resistance of masked circuits has first been analyzed in [17]. The same year, a technique to model this effect has been published in [33]. Hereafter, we apply the latter technique to model an adversary which performs d^{th}-order probing attacks against a circuit susceptible to glitches. For such a purpose, we transform our *static* definition of a circuit into a *dynamic* one. Actually, this simply amounts to assume that the random variable V_{ij} not only relies on the pair (i, j) but also on a third time parameter t. We denote by $V_{ij}(t)$ the dynamic version of V_{ij} and call *dynamic* the corresponding circuit. The main difference between the two models is that, in the probing model, the value taken by V_{ij} is assumed to be constant, whereas its dynamic version $V_{ij}(t)$ can change over the time, even when the circuit input is fixed. Example 1 illustrates a side effect of glitches (*i.e.* several switchings of a single gate during a clock cycle).

Example 1 (Example of Glitches effects). The figure hereafter represents a Boolean mask refreshing on a sensitive variable Z. The expected behaviour is the following sequential evaluation of the gates: $Z \oplus M_1 \longrightarrow Z \oplus M_1 \oplus M_2 \longrightarrow Z \oplus M_2$. However if, for some propagation delay reasons, the mask M_2 is late to arrive at the first XOR gate, the value $Z \oplus M_1$ will be XORed with a constant (let say 0) value before being directed to the second XOR gate. This results in the following sequence: $Z \oplus M_1 \longrightarrow Z \oplus M_1 \longrightarrow Z$. In this case, it can be noted that each XOR gate switches twice during a clock cycle (before and after the arrival of M_2). Moreover there exists a period of time, inside the clock cycle, where the manipulation of Z is not protected[3].

[3] The example is a simplified description of a device behaviour where a propagation delay results in data unmasking. In practice, less obvious glitches effects can be exploited. For instance, in [17, 33] attacks only exploit dependency between the number of gate switchings and sensitive data. Since our concern is not to study specific attacks but to build a model encompassing all glitches-based attacks, we



Notation. In the following, the *internal state transition at time t' of a circuit $\mathcal{C}_f$* with $(V_{ij})_{(i,j)\in I}$ as edges refers to all the non-zero transitions of the value taken by the $(V_{ij}(t))_{(i,j)\in I}$ at time $t = t'$. It is denoted by $\mathcal{C}_f(t')$.

Definition 3 (d^{th}-order Glitches Adversary Model). *Let $\mathcal{C}_f$ be a circuit and let d be a positive integer. Let $\mathcal{L}$ be a set of leakage functions. A* d^{th}-order Glitches Adversary against *$\mathcal{C}_f$ is an adversary that can choose d times t_1, t_2, ..., t_d and can observe the internal state transition at the d selected times $(L_i(\mathcal{C}_f(t_i))_{i\leq d}$, where, for each $i \leq d$, $L_i(\cdot)$ is a function in $\mathcal{L}$.*

Remark 3. Definition 3 implicitly makes the classical assumption that only computation leaks information*. This assumption introduced in 2004 independently by Blömer et al. [2] and Mikali and Reyzin [20] states that the leakage at time t only leaks information on the internal state transition $\mathcal{C}_f(t)$. It is moreover assumed that the targeted circuit is implemented on a single Hardware platform. If the circuit is executed over several platforms leaking independently, then the glitches adversary must not only choose a time t but must also select the platforms to spy at this time. In this case, the order can be defined as the number of different times in the attack, each time being itself weighted by the number of different platforms spied at this time.*

Remark 4. Definition 3 implies that, in presence of glitches, a unique side-channel observation at time t can potentially give information about the whole current computations. This adversary category is very powerful and is actually much stronger than the current attackers of embedded systems. By analyzing the security with respect to such kind of strong adversaries, the purpose is to build a set of countermeasures with the minimum of assumptions on the adversary capabilities.

---

chose to define a generic glitches Adversary as in Definition 3, namely w.r.t. to a very favourable situation.

Notation. An attack performed by the adversary defined in Definition 3 is called d^{th}*-order glitches attack*. A circuit secure against those attacks is said to be *d-glitches free*.

Security in the glitches adversary model implies security in the probing adversary model whereas the converse is obviously false. In the following sections, we introduce the notions of *d-probing security* and of *d-glitches freeness*. For both security notions, we exhibit generic construction principles that enable to design circuits achieving them.

2.2 Security in the Probing Adversary Model

The following definition formalizes the notion of security w.r.t. d^{th}-order probing adversaries. Note that this definition corresponds to that involved in numerous papers (*e.g.* [2, 5, 14, 19, 25, 26]).

Definition 4 (*d*-probing Security). *Let d be a positive integer. A circuit $\mathcal{C}_f$ with family of edges $\mathcal{V}$ is* d-probing secure *if and only if no family of at most d elements in $\mathcal{V}$ is sensitive.*

Remark 5. Definition 4 substantially states that a cryptographic circuit $\mathcal{C}_f$ is secure against d^{th}-order probing attacks if, for any sensitive variable Z and any subset $\mathcal{V}' \subseteq \mathcal{V}$ of size d, we have $\Pr(Z \mid \mathcal{V}') = \Pr(Z)$. It implicitly refers to the most powerful d^{th}-order probing adversary who works with a family of leakage functions reduced to the identity.

To achieve d-probing security, the most widely used approach is to split each sensitive variable appearing in the algorithmic description of the cryptosystem into several shares and to replace operations on the sensitive data by operations on the shares. We give hereafter a formal description of this technique called (n, d)*-sharing* in the sequel.

Definition 5 ((n, d)-sharing). *Let n and d be two positive integers such that $n > d$. A (n, d)-sharing of a variable $Z \in \mathrm{GF}(2^m)$ is a family of n variables $(Z_i)_{1 \leq i \leq n}$ such that:*

1. *there exists a deterministic function F from $\mathrm{GF}(2^m)^n$ into $\mathrm{GF}(2^m)$ for which;*
$$F(Z_1, \cdots, Z_n) = Z \ ,$$
2. *for every subset $I \subset [1; n]$ with cardinality lower than or equal to d;*
$$\Pr(Z \mid (Z_i)_{i \in I}) = \Pr(Z) \ .$$

In relation with the notion of (n, d)-sharing we will sometimes use the notion of independent sharings. A formal definition is given hereafter.

Definition 6 (Independence of (n, d)-sharings). *Let n and d be two positive integers such that $n > d$. Two (n, d)-sharings $(Z_i)_{i \leq n}$ and $(Z'_i)_{i \leq n}$ of two (not necessarily distinct) variables Z and Z' are said to be* independent *if for every pair of subsets (I, I') in $[1; n]$, each of cardinality lower than or equal to d, we have* $\Pr((Z_i)_{i \in I} \mid (Z'_i)_{i \in I'}) = \Pr((Z_i)_{i \in I})$.

*Remark 6. Two (n, d)-sharings are independent if they involve independent masking materials (*i.e. *different random values). The replacement of a (n, d)-sharing of Z by a new independent one is sometimes called* re-randomization *or* masks refreshing *in the literature [1, 29].*

In the SCA literature, the family $(Z_i)_{1 \leq i \leq n}$ is usually called a *masked representation at order d* of Z. The function F is usually simply defined as the sum of the Z_i and n is chosen equal to $(d+1)$. Several ways have been proposed to apply such a d-sharing to protect a circuit against probing attacks. To the best of our knowledge, only the circuit implementations proposed by Ishai *et al.* [13] and Rivain and Prouff [29] are d-probing secure for any fixed value d. Unfortunately, those schemes are not by construction secure in the Glitches Adversary Model (*i.e.* without further assumption, such a d-probing secure implementation might be susceptible to an adversary performing first-order glitches attacks)[4]

2.3 Security in The Gliches Adversary Model

To achieve security in the Glitches Adversary Model, we develop hereafter a strategy that consists in *hermetically* separating some parts of the computation. The idea is to split a circuit $\mathcal{C}_f$ implementing a function f into several sub-circuits $\mathcal{C}_{f_i}$ such that the observation of d or fewer sub-circuits gives no information on the original circuit input. The security/pertinence of this approach is essentially based on the following simple observation: if n sub-circuits $\mathcal{C}_{f_i}$ operate each on a single element of a (n, d)-sharing and if their processing leaks independently, then the circuit composed of the n sub-circuits is d-glitches free. Of course, this observation alone does not directly permit to design a d-glitches free circuit implementing any function f. Indeed, the above construction implies that each sub-circuit is input with a single share of a sharing and cannot access the other shares.

[4] Indeed, the construction of [13, 29] assume that some sequences of operations are executed in a fixed order (otherwise the security is broken). In a glitches context such assumption can be unvalidated as seen in Example 1.

By consequence, only a certain type of function f (homomorphic with respect to the sharing) can be split into n independent computations, each of them operating only a single share of f's input and returning a sharing of f's output (see Figure 1).

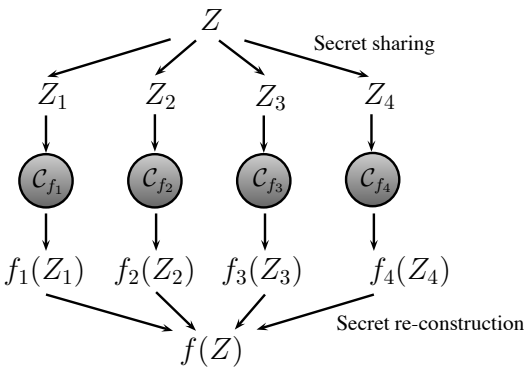


Fig. 1. Multi-party Circuit for a homomorphic function f and $n = 4$

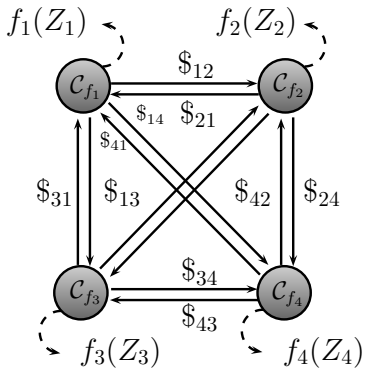


Fig. 2. Multi-party Circuit for any function f and $n = 4$

Secure Multi-Party Computation protocols extend the above idea (illustrated in Fig. 1) to any function f (*i.e.* not only homomorphic function for which, as we have seen, the solution is straightforward). For such an extension to be possible, each sub-circuit $\mathcal{C}_{f_i}$ is provided with the new ability to send information about its intermediate results to the other sub-circuits. To ensure that this has no impact on the d-glitches freeness of the overall circuit (composed of the n sub-circuits), the sent information must itself be shared between all the n sub-circuits. Under this constraint, each sub-circuit can only be given a single share of a (n, d)-sharing of the intermediate result of another sub-circuit. Moreover, to build a sound proof of security we also must ensure that all the shares accessed by a sub-circuit come from distinct and independent (n, d)-sharings. The hence obtained construction can be formalized by extending each sub-circuit $\mathcal{C}_{f_i}$ with a family of $n-1$ channels $(\$_{ij})_{j\neq i}$, the channel $\$_{ij}$ being dedicated to the communication between $\mathcal{C}_{f_i}$ and $\mathcal{C}_{f_j}$ and being not accessible by another sub-circuit $\mathcal{C}_{f_k}$, $k \neq i, j$. Through those channels, each $\mathcal{C}_{f_i}$ can access a share of any intermediate result of another sub-circuit, each new accessed share being related to a (n, d)-sharing independent of the previous ones. The family of extended circuits $(\mathcal{C}_{f_i}, (\$_{ij})_{i\neq j})_i$ is called a (n, d)-*multi-party circuit*. We hereafter give a formal definition of it.

Definition 7. *Let f be a function and let Z denote its input. Let $(Z_i)_i$ be a (n,d)-sharing of Z. A circuit $\mathcal{C}_f$ composed of (a minimum of[5]) n extended sub-circuits $(\mathcal{C}_{f1}, (\$_{1j})_{j\neq 1})$, ..., $(\mathcal{C}_{fN}, (\$_{Nj})_{j\neq N})$ is* a (n,d)-multi-party circuit *iff:*

- *every $\mathcal{C}_{f_i}$ is input with a share Z_i only,*
- *each $\mathcal{C}_{f_i}$ can access, through $\$_{ij}$, to a share of an (n,d)-sharing of an intermediate result of the sub-circuit $\mathcal{C}_{f_j}$,*
- *all the shares accessed by a sub-circuit $\mathcal{C}_{f_i}$ relate to mutually independent (n,d)-sharings,*
- *the $\mathcal{C}_{f_i}$ outputs form a (n,d)-sharing of $f(Z)$.*
- *for $i \neq j$, $\mathcal{C}_{f_i}$ leaks independently to $\mathcal{C}_{f_j}$.*

We give in Fig. 2 an example of a (n,d)-multi-party circuit for $n = 4$.

The following proposition states on the security of our construction against glitches attacks.

Proposition 1. *A* (n,d)-multi-party circuit *is secure in the d^{th}-order Glitches Adversary Model.*

Sketch of proof. The most powerful d^{th}-order glitches attack an adversary can perform against a circuit can only allow him to recover the entire input of the circuit. By definition of a (n,d)-sharing, each sub-circuit operates on a single element Z_i of a (n,d)-sharing of Z. Then the adversary needs to attack at least $d+1$ sub-circuits to recover information on a shared variable Z. The other variables manipulated by a circuit come from channels $\$_{ij}$ and (by definition) they rely on independent (n,d)-sharings. By consequence, the adversary cannot combine them to recover more information than a single share of the input Z_j of another circuit. Eventually, since the sub-circuits are assumed to leak independently, we deduce that the adversary has to perform at least a $(d+1)^{\text{th}}$-order glitches attack to break the scheme. ◇

Remark 7. To define a multi-party circuit we implicitly assumed that it was possible to separate the sub-circuits executions. It is out of the scope of this paper to propose such execution separation but we give in Appendix A two separation models. The cost of the sub-circuit separation may seem expensive with respect to a block cipher implementation, but we believe that it is mandatory in order to develop sound resistance against glitches effects (without relying on ad-hoc *logic styles). The construction we propose in the sequel aims at minimizing the sub-circuits number.*

[5] By definition the number of sub-circuits in a (n,d)-multi-party circuit is lower-bounded by n but is not limited.

In the next section we recall the basics about Secure Multi-party Computation and, in particular, a protocol introduced by Ben Or *et al.* in [1]. Then, we argue that this protocol can be adapted to our context in order to design (n, d)-multi-party circuits as long as n is greater than $2d$. Eventually, in Section 4, the construction is applied to define, for any $d > 1$, d-glitches free implementations of the AES.

3 Secure Multi-Party Computation

Secure Multi-Party Computation represents a rich area of research initiated by the seminal work of Yao in 1986 [34]. For a n-ary function f and a family of n players $(I_i)_{i \leq n}$, each holding a private value Z_i, a *secure multi-party computation* is a joint protocol enabling the players I_i to compute $f(Z_1, \cdots, Z_n)$ while under attack by an external adversary and/or by a subset of malicious players (also called the *colluding players*). The attack purpose is to learn the private information of the – non-colluding – honest players or to cause the computation to be incorrect. To study the resistance of a protocol against those threats, two kinds of adversaries are usually introduced. The first one, usually called *active*, is allowed to let the malicious parties deviate from the protocol in arbitrary ways. It is out of the scope of this paper. The second kind of adversary, called *passive* or *semi-honest*, is only allowed to create collusions of players to gain information about the secret. The corrupted players still follow the protocol and never forge wrong data. A security threshold parameter $d \leq n$ is used to indicate the maximum number of players the adversary is allowed to corrupt. A SMC protocol secure against a passive or active adversary with threshold d is called a *d-private protocol*. We show in this section how the problem of designing SMC protocols secure for this adversary model is related to the problem of designing multi-party circuits secure in the Glitches Adversary Model. Before that, we recall in the next section the main aspects of the SMC protocol introduced by Ben Or *et al.* in [1] on the basis of an idea proposed by Shamir in 1979.

3.1 Shamir's Secret Sharing Scheme and BGW's protocol

In a seminal paper [31], Shamir has introduced a simple and elegant way to share a secret Z (considered here in a field $K \equiv \mathrm{GF}(2^m)$) between $n < 2^m$ players such that no collusion of $d < n$ players can retrieve information about Z. In Shamir's protocol, an entity called *the Dealer* generates a degree-d polynomial $P_Z(X) \in K[X]$ with constant

term Z and secret coefficients a_i (*i.e.* $P_Z(X) = Z + \sum_{i=1}^{d} a_i X^i$). Then, he chooses n distinct non-zero elements $\alpha_1, \cdots, \alpha_n$ in K, makes them publicly available and *distributes* to each player I_i the value $Z_i = P_Z(\alpha_i)$. To *re-construct* the secret Z, the players publish their private values Z_i, reconstruct P_Z by polynomial interpolation (always possible since $n > d$) and evaluate $P_Z(X)$ in 0 (we have $Z = P_Z(0)$). It can be easily checked that Shamir's sharing fits with the notion of (n, d)-sharing given in Definition 5 with reconstruction function F defined s.t. $F(Z_1, \cdots, Z_n) = \sum_{i=1}^{n} Z_i \prod_{k=1, k\neq i}^{n} -\alpha_k(\alpha_i - \alpha_k)^{-1}$. Indeed, by definition of the Z_i and due to Lagrange's Interpolation, the value $F(Z_1, \cdots, Z_n)$ equals $P_Z(0)$ that is Z.

Remark 8. The products $\prod_{k=1, k\neq i}^{n} -\alpha_k(\alpha_i - \alpha_k)^{-1}$ for all i can be precomputed once for all. They actually correspond to the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the Vandermonde $(n \times n)$-matrix $(\alpha_i^j)_{1 \leq i,j \leq n}$. We hence have $F(Z_1, \cdots, Z_n) = \sum_{i=1}^{n} \lambda_i Z_i$.

Remark 9. An example of algorithm specifying how to generate the (n, d)-sharing of an element along with an example of algorithm specifying how to reconstruct it are given in Sect. 4 (Algorithms 9 and 10 respectively).

Starting from Shamir's secret sharing, Ben Or *et al.* have defined in [1] a d-private SMC protocol in the case where the number of players n satisfies $n > 2d$. This construction, called *BGW's protocol* in the following, is in fact a constructive proof of the following theorem (see [1, Theorem 1]):

Theorem 1. *For every (probabilistic) function f and $n > 2d$, there exists a d-private protocol.*

In BGW's protocol, the input $(Z_1, \cdots, Z_n)$ of the function f whose computation must be made d-private is assumed to correspond to Shamir's sharing of a secret variable Z. Namely, they correspond to the evaluation of a degree-d secret polynomial $P_Z(X)$ in n distinct non-zero public points α_1, ..., α_n. It is moreover assumed that each player I_i has been initially provided with a share Z_i which is unknown to the others. Then, the function f is modelled as a sequence of computations operating either an affine transformation on an intermediate state V or additions/multiplications between two intermediate states V and V'. Let us denote by `C` such a (univariate or bivariate) computation. BGW's protocol ensures that the intermediate states V and V' at input of a bivariate operation `C` have been shared w.r.t. two random polynomials P_V and $P_{V'}$ which are independent

but evaluated in the same public points α_1, ..., α_n (*i.e.* V_i and V_i' satisfy $V_i = P_V(\alpha_i)$ and $V_i' = P'_{V'}(\alpha_i)$ respectively). Moreover, for each $\mathtt{C}$ to process, BGW's protocol is designed such that each player I_i has either a single share V_i (if $\mathtt{C}$ is univariate) or a single pair of shares (V_i, V_i') (if $\mathtt{C}$ is bivariate). Eventually, BGW's protocol describes a d-private multi-party computation for each kind of field operation $\mathtt{C}$ depending on its nature. We recall them hereafter, where it is assumed that the polynomials P_V and P'_V are defined over $K[X]$.

If $\mathtt{C}$ is an affine transformation over K applied on a shared variable V, then the protocol consists in asking each player I_i to apply $\mathtt{C}$ on its private share V_i. After this step, each player owns a new share $\mathtt{C}(V_i)$ and the family $(\mathtt{C}(V_i))_i$ is a (n, d)-sharing of $\mathtt{C}(V)$. Indeed, since $\mathtt{C}$ is affine, $\mathtt{C}(P_V(X))$ is a degree-d polynomial $P_{\mathtt{C}(V)}(X)$ such that $P_{\mathtt{C}(V)}(0) = \mathtt{C}(V)$ and each $\mathtt{C}(V_i)$ corresponds to the evaluation of $P_{\mathtt{C}(V)}$ in α_i (*i.e.* $P_{\mathtt{C}(V)}(\alpha_i) = \mathtt{C}(V_i)$).

If $\mathtt{C}$ is the addition operation $\oplus$ over K applied on two shared intermediate states V and V', then the protocol consists in asking each player I_i to compute $\mathtt{C}(V_i, V_i') = V_i \oplus V_i'$. After this step, each player owns a new share $\mathtt{C}(V_i, V_i')$ and the family of shares $(\mathtt{C}(V_i, V_i'))_i$ is a (n, d)-sharing of $\mathtt{C}(V, V')$. Indeed, by construction of the V_i and V_i', we have $\mathtt{C}(V_i, V_i') = P_V(\alpha_i) \oplus P_{V'}(\alpha_i)$ which implies that $\mathtt{C}(V_i, V_i')$ corresponds to the evaluation of the polynomial $P_V(X) \oplus P_{V'}(X)$ in α_i. This polynomial is of degree at most d and satisfies $(P_V(0) \oplus P_{V'}(0)) = V \oplus V' = \mathtt{C}(V, V')$.

If $\mathtt{C}$ is the multiplication operation $\otimes$ over K applied on two shared intermediate states V and V', then the protocol is more complex than the previous ones. Actually, since this operation is not homomorphic with respect to sharing law, the protocol must involve communication between the players. Also, it will involve the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the Vandermonde $(n \times n)$-matrix $(\alpha_i^j)_{1 \leq i,j \leq n}$ and it is composed of three steps[6].

1. Each player I_i computes $\mathtt{C}(V_i, V_i') = V_i \otimes V_i' = P_V(\alpha_i) \otimes P_{V'}(\alpha_i)$.
2. Each player I_i randomly generates a degree-d polynomial Q_i such that $Q_i(0) = \mathtt{C}(V_i, V_i')$ and for every $j \neq i$, sends the value $Q_i(\alpha_j)$ to player I_j.
3. Each player I_i computes the linear combination $\mathbf{Q}(\alpha_i) = \sum_{j=1}^{n} \lambda_j Q_j(\alpha_i)$.

The shares $\mathtt{C}(V_i, V_i')$ computed by the players at Step 1 correspond to the degree-$2d$ polynomial $P_V(X) \times P_{V'}(X)$. As desired, the constant term

[6] The protocol described in this paper is an improved version of the protocol originally proposed by Ben-Or *et al.* [1]. It has been introduced by Gennero *et al.* in [10].

of this polynomial is $\mathtt{C}(V, V')$. However, the family $(\mathtt{C}(V_i, V_i'))_i$ built by Step 1 is not a (n, d)-sharing since the corresponding polynomial is firstly not of degree d and secondly, is not a random polynomial (its distribution over the set of degree-$2d$ polynomials with constant term $\mathtt{C}(V, V)$ is not uniform). To overcome this issue, Steps 2 and 3 perform both a degree reduction and a re-randomization of the shares. More precisely, Step 2 allows player I_i to compute the (n, d)-sharing $(Q_i(\alpha_j))_j$ of its share $\mathtt{C}(V_i, V_i')$ thanks to a random polynomial Q_i, and to send those shares to the other players. Then, in Step 3 each player I_i computes $\mathbf{Q}(\alpha_i) = \sum_{j=1}^{n} \lambda_j Q_j(\alpha_i)$. The family $(\mathbf{Q}(\alpha_i))_i$ corresponds to the evaluation in $(\alpha_i)_i$ of the polynomial $\mathbf{Q}(X) = \sum_j \lambda_j Q_j(X)$, which, by construction, is of degree d and admits $\mathtt{C}(V, V')$ as constant term. It is therefore a (n, d)-sharing of $\mathtt{C}(V, V')$ (see [10] for more details).

All the previous operations are defined over the field K. In a block cipher algorithm, they are often combined with GF(2)-affine transformations. The latter ones are rarely affine over K since they do not satisfy $\mathtt{C}(\lambda V) = \lambda \mathtt{C}(V)$ for every $\lambda \in K$. Of course, any such GF(2)-affine function can be represented as a field transformation f taking the form $y \in K \mapsto f(y) = b_{-1} + \sum_{k=0}^{m-1} b_k y^{2^k}$ for some $b_k \in K$. If one denotes by ψ the natural mapping $\mathrm{GF}(2)^m \mapsto \mathrm{GF}(2^m)$, we have $\mathtt{C}(V) = \psi^{-1} f(\psi(V))$. Thanks to this representation, any GF(2)-affine transformation can be securely evaluated thanks to the BGW protocol recalled previously. However, such a processing implies numerous secure multiplications. In the following, we propose an alternative scheme which allows us to process any GF(2)-affine transformation without secure multiplications. For this purpose we introduce a new condition on the public points α_i involved in Shamir and BGW schemes. Then, the idea is to perform the evaluation of each monomial $b_k y^{2^k}$ separately. Thanks to the new condition on the α_i, we show that this processing can be done without secure multiplication and just needs a subsequent re-indexing of the players.

Improved processing for a transformation $\mathtt{C}$ in the form $V \mapsto b_k V^{2^k}$ (GF(2)-affine but not linear over K). For such a processing, a special attention must be paid in order to avoid the need for costly multiplication operations. To this end, we need the public points α_i to satisfy the following conditions:

- *Distinct and Non-zero Condition.* The elements α_i are all distinct and non-zero.
- *Stability Over Frobenius Automorphism.* For every α_i, there exists α_j such that $\alpha_j = \alpha_i^2$.

The first constraint is already a prerequisite for Shamir's sharing scheme. As the security of the latter scheme, as well as that of Ben-Or *et al.* scheme, stands for any choice of distinct and non-zero α_i, the second condition induces no flaw. Moreover, we prove in Appendix D that it is always possible to find such a set of $\{\alpha_i\}$ in $\mathrm{GF}(2^8)$ and we exhibit a construction (an analysis for any m is let for further research).

Similarly to the other operations, we assume that each player I_i owns the i^{th} element of the (n, d)-sharing of V. The only additional assumption is that the α_i satisfy the stability condition. To securely execute the transformation $\mathtt{C} : V \mapsto b_k V^{2^k}$, we propose that each player processes separately the transformation $P_V(\alpha_i) \mapsto \mathtt{C}(P_V(\alpha_i)) = b_k(P_V(\alpha_i))^{2^k}$ to its share. Let us denote by P' the degree-d polynomial built from P_V by applying the transformation $\mathtt{C}$ to its coefficients. By construction, we have $P'(\alpha_i^{2^k}) = \mathtt{C}(P_V(\alpha_i))$ and the set of shares $\{P'(\alpha_i^{2^k})\}_{i\leq n}$ is a (n, d)-sharing of $\mathtt{C}(V)$. From now, we can hence change the notation P' for $P_{\mathtt{C}(V)}$. If the α_i do not satisfy the stability condition, then the (n, d)-sharings $\{P_V(\alpha_i)\}_{i\leq n}$ and $\{P_{\mathtt{C}(V)}(\alpha_i^{2^k})\}_{i\leq n}$ correspond to different sets of public points (one being the image of the other by Frobenius transformation). Such a situation is not favourable since it implies a supplementary processing to update the set of public points. On the opposite, when the α_i do satisfy the stability condition, then the sets $\{\alpha_i^{2^k}\}_i$ and $\{\alpha_i\}_i$ are the same and we get $\left\{P_{\mathtt{C}(V)}(\alpha_i^{2^k})\right\}_{i\leq n} = \left\{P_{\mathtt{C}(V)}(\alpha_i)\right\}_{i\leq n}$. In other terms, the multi-party computation proposed above enables the definition of a (n, d)-sharing of $\mathtt{C}(V)$ with unchanged set of public points. Moreover, since the processing $\mathtt{C} : y \mapsto b_k y^{2^k}$ can be tabulated, the overall computation only involves n look-up table accesses (one for each player) and no costly multiplication.

Remark 10. The Frobenius operation in $\mathrm{GF}(2^m)$ *may be seen as a special case of the transformation* $\mathtt{C}(V) = b_k V^{2^k}$ *and shall be treated similarly. A detailed description of the first-order masking scheme for the Frobenius operation and for the* $\mathrm{GF}(2)$*-affine transformation of the AES is provided in Appendix E.*

Remark 11. Contrary to the other methods listed in this section (e.g. the secure computations of field additions and multiplications), the improved processing proposed for $GF(2)$*-affine transformations does not directly benefit from the security proofs given in [1]. To make those latter proofs still applicable for the improved processing, the following supplementary hypothesis must be verified by the players:* each player must have the ability to erase some part of its memory*. This assumption is not compatible*

with the traditional Secure Multi-Party Computation context but is acceptable in the context of multi-party circuits.

3.2 SMC protocol and Multi-Party Circuits

The design of a (n, d)-multi-party circuit from BGW's protocol is merely based on the following remark: the d-privacy for a set of n semi-honest players evaluating a function f coincides with the d-probing security for a set of n circuits implementing f. Hence, if each player I_i in BGW's protocol is replaced by an extended sub-circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$, then the previous description specifies a (n, d)-multi-party circuit. Moreover, such a design can be specified for any function f as long as n and d satisfy $n > 2d$. If f is defined over the finite field $\mathrm{GF}(2^m)$ with addition and multiplication laws $\oplus$ and $\otimes$, the sub-circuits $\mathcal{C}_{f_i}$ are defined with respect to the elementary operations $\{\mathcal{A}(m), \oplus, \otimes\}$, where $\mathcal{A}(m)$ denotes the set of affine functions over $\mathrm{GF}(2^m)$. By construction, each extended sub-circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$ always operates on a single share of a sensitive variable and has never access to the other shares nor a function of them (cf. definition 7). Consequently, the observation of an extended sub-circuit cannot give more information than a single share of a (n, d)-sharing. Hence, since the sub-circuit executions do not overlap, and by definition of a (n, d)-sharing, a glitches adversary must observe the behavior of at least $d + 1$ extended sub-circuits $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$ to recover sensitive information.

Eventually, to fully specify how to put BGW's protocol into practice for a (n, d)-multi-party circuit, it just remains to clarify the following practical points:

- **Messages exchange between sub-circuits (a.k.a. players) during Step 2 of the secure processing of $\otimes$.** The exchange of messages is done thanks to the channels $\$_{ij}$. In software, each channel $\$_{ij}$ between a pair of sub-circuits $(\mathcal{C}_{f_i}, \mathcal{C}_{f_j})$ may simply consist in a RAM space which is not accessed by another circuits $\mathcal{C}_{f_k}$ with $k \neq i, j$. In hardware, the designer can code a unique communication channel for each pair of circuits running sequentially. Another solution could be to run each sub-circuit in a different environment (*e.g.* different platforms) and to implement a channel between each pair of them (see Appendix A).
- **Messages exchange with erasure during processing of $\mathrm{GF}(2)$-affine function (non-linear over K).** A second routine must be specified to enable two extended sub-circuits to exchange data followed by partial erasure of their memory. Actually, each sub-circuit

$\mathcal{C}_{f_i}$ is provided with a procedure that enables it to write data on $\$_{ij}$ and to erase those data from its own memory. This procedure is called `write-then-erase` in the following.

– **The initial shares distribution by a honest entity (the Dealer).** In our context, the role of the Dealer is played by a special procedure run before processing the multi-party circuit. This procedure shares the sensitive variable as usually done in the literature to counteract d^{th}-order probing attacks. To also achieve security in the d^{th}-order Glitches Adversary Model, the computation is split into elementary operations that are processed sequentially. Although expensive this strategy can always be followed to go from probing attack security to glitches attacks security (see next section on the comparison with state of the art solutions). The actual algorithm is presented in next Section (Algorithm 9).

Because it is the most tricky part in BGW's protocol, we develop hereafter the algorithm processed by each sub-circuit $\mathcal{C}_{f_i}$ when computing the (n, d)-sharing of the product of two shared values over a field $\text{GF}(2^m)$. It exactly corresponds to the multiplication operation discussed in the previous section when developed in the context of extended sub-circuits instead of players.

Notation. Instruction $\mathbf{read}(X, \$_{ij})$ reads the content of $\$_{ij}$ (viewed as a channel or a memory address) and update X with it. Instruction $\mathbf{write}(X, \$_{ij})$ writes the value X on $\$_{ij}$.

Algorithm 1 Secure Multiplication Part Dedicated to a Sub-Circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$

INPUT: the i^{th} element $P_V(\alpha_i)$ of a (n, d)-sharing of V, the i^{th}element $P_{V'}(\alpha_i)$ of a (n, d)-sharing of V' and a set of channels $(\$_{ij})_{j\neq i}$.

OUTPUT: the i^{th} element $\mathbf{Q}_{V\otimes V'}(\alpha_i)$ of a (n, d)-sharing of $V \otimes V'$.

PUBLIC: the n distinct points α_i, the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the matrix $(\alpha_i^j)_{1\leq i,j\leq n}$

---

1. **do** $W_i \leftarrow P_V(\alpha_i) \otimes P_{V'}(\alpha_i)$

**** Randomly generate a d-tuple (a_j) of coefficients in* $\text{GF}(2^m)$

2. **for** $j = 1$ **to** d
   do $a_j \leftarrow \texttt{rand}(\text{GF}(2^m))$

**** Compute a (n, d)-sharing $(Q_i(\alpha_1), \cdots, Q_i(\alpha_n))$ of W_i.*

3. **for** $j = 1$ **to** n
   do $Q_i(\alpha_j) \leftarrow W_i \oplus \bigoplus_{k=1}^{d} a_k \alpha_i^k$

**** Send the shares of W_i to the other sub-circuits $\mathcal{C}_{f_j}$ through $\$_{ij}$.*

4. **for** $j = 1$ **to** n, $j \neq i$

write$(Q_i(\alpha_j), \$_{ij})$
**** Receive a share $Q_j(\alpha_i)$ from each sub-circuit C_j through $\$_{ji}$.*
5. **for** $j = 1$ **to** n, $j \neq i$
read$(Q_j(\alpha_i), \$_{ij})$
**** Compute the share $\mathbf{Q}_{V\otimes V'}(\alpha_i)$*
6. **do** $\mathbf{Q}_{V\otimes V'}(\alpha_i) \leftarrow \bigoplus_{j=1}^{n} \lambda_j Q_j(\alpha_i)$.

Steps 2 enables to randomly generate a degree-d polynomial $Q_i(X) = W_i + \bigoplus_{j=1}^{d} a_j X^j$. Step 3 evaluates $Q_i(X)$ in each public point α_j to construct a (n, d)-sharing $(Q_i(\alpha_j))_j$ of W_i. Step 4 sends those shares to the other sub-circuits and Step 5 enables $\mathcal{C}_{f_i}$ to receive the shares $Q_j(\alpha_i)$ computed by the other sub-circuits $\mathcal{C}_{f_j}$. Eventually, Step 6 computes a share of $V \otimes V'$ for sub-circuit $\mathcal{C}_{f_i}$.

3.3 Complexity of the Scheme and Comparison

Complexity Evaluation Except the multiplication, the proposed scheme replaces each operation of a given function by n similar operations (see Section 4 for an application of the scheme to AES-128 block cipher). Concerning the multiplication $\otimes$ over GF(2^m), it is performed by asking each of the sub-circuits to run Algorithm 1. As a consequence, the multiplication $\otimes$ is replaced by $n^2(d+1)+n$ multiplications, $n^2(d+1)-n$ additions and $2(n-1)$ **read**/**write** operations. In the following table, we develop this complexity for $n = 2d+1$ (which is the smallest value n allowed in BGW's protocol). For comparison purpose, we also give the complexity of the secure multiplication proposed in [29]. We recall that when applied with $n = 2d+1$, the multiplication algorithm proposed in Algorithm 1 offers the same (perfect) resistance against d-probing attacks than the method proposed in [29].

Method	multiplications	additions	random bytes
This paper	$4d^3 + 8d^2 + 3d$	$4d^3 + 8d^2 + 7d + 2$	$d(2d+1)$
[29]	$2d^2 + 2d$	$d^2 + d + 1$	$d(d+1)/2$

Table 1. Complexity of the secure processing of a field multiplication.

Comparison With Other State Of The Art Solutions The construction proposed in the paper has similarities with the works [24], [13] and [29].

In [24], Nikova *et al.* have already attempted to apply the multi-party computation theory in the context of hardware implementations, with 1-glitches freeness in mind. Contrary to our proposal where data are shared thanks to Shamir's scheme, Nikova *et al.* 's construction relies on the classical additive sharing (namely the circuit's input is additively masked with several, say $n-1$, random variables). To secure the processing on the masked input and the masks, they propose to split the computation according to a set of security rules. The obtained circuit sharing differs from ours in the two following main points. First, the security is only proven against first-order attacks, which implies that Nikova *et al.* 's construction cannot be used to design (n, d)-multi-party circuits for $d > 1$. Secondly, the sharing is not explicit and involves an exhaustive search that becomes impossible when the size m of the circuit input is greater than 5. Moreover, there is no guaranty that the approach works for any circuit. In particular, Moradi *et al.* [21] discuss the difficulty of applying Nikova *et al.* 's scheme to the AES s-box. In [24], the scheme has been applied to the block cipher Noekeon [6]. Instead of taking the direction proposed in the present paper where the circuit is divided into several sub-circuits leaking independently, they take an opposite position where the different shares of a variable are manipulated *simultaneously* by the same circuit. Our scheme could also be implemented in such a way but the resulting security against d^{th}-order attacks would be significantly reduced. Indeed the manipulation of all the shares at the same time makes the overall leakage at this time dependent on all the shares altogether. This enables — at least theoretically — to mount a 1^{st}-order SCA attack. This fact is clearly pointed out in [24] but assumed to only lead to unpractical attacks. This has actually been validated in practice by Moradi *et al.* , who exhibit in [21] a first-order MIA on their AES implementation based on Nikova's scheme.

On the opposite side, the constructions proposed in [13] (operations in GF(2)) and in its extension [29] (operations in GF(2^m)) are d-probing secure but not 1-glitches free. The cost of the secure multiplication in BGW's protocol is greater than that of the d-probing secure multiplication proposed in [29] (see Tab. 1). The overhead between the two methods is essentially explained by the fact that BGW's multiplication is designed to achieve d-glitches freeness whereas Rivain-Prouff's one is not. As a matter of fact and as far as we know, the only sound way to induce glitches freeness in Rivain-Prouff's multiplication would be to implement it on a multi-party circuit such that each elementary operation is processed on a separate sub-circuit. This implies the use of $O(d^2)$ sub-circuits when

BGW's protocol, and then our scheme, was designed to minimize the number of players (thus the number of sub-circuits here) to $2d + 1$. Hence, even though the overall bit-complexity of our scheme is one order of magnitude more expensive than Rivain-Prouff's scheme, its limited cost in sub-circuits number makes it competitive when the design of sub-circuits is prohibitive (see remark 7 and Appendix A).

4 Glitches Free HO-Masking of the AES

We apply here the construction proposed in Section 3.2 to design a multi-party circuit implementing the AES-128 block cipher. The AES-128 block cipher iterates 10 times a round transformation on a 16-bytes internal state initially filled with the plaintext. The transformation is composed of a key addition `AddRoundKey`, a nonlinear layer `SubBytes` which applies the same *substitution-box* (s-box) to every byte of the internal state and a linear layer composed of the so-called transformations `ShiftRows` and `MixColumns`. The s-box S is defined as the left-composition of a transformation η_A affine over $\mathrm{GF}(2)^8$, with the power function $y \mapsto y^{254}$ over the field $\mathrm{GF}(256) \simeq \mathrm{GF}(2)[X]/(X^8 + X^4 + X^3 + X + 1)$. In the following, we propose a (n, d)-multi-party circuit $(\mathcal{C}_{f_1}, \cdots, \mathcal{C}_{f_n})$ implementing the AES-128 algorithm. First, for each of the four AES transformations listed above we detail in Algorithms 3 to 7 the processing done by each sub-circuit. To specify the algorithms, we exactly applied the protocols discussed in Section 3.1. Eventually, we combine them to specify the processing that must be done by each sub-circuit $\mathcal{C}_{f_i}$ in order to ensure that the family $(\mathcal{C}_{f_1}, \cdots, \mathcal{C}_{f_n})$ is a d-glitches free implementation of the AES-128.

Let us start our description with the exponentiation computation $y \mapsto y^{254}$ which is the most tricky part to protect. As shown in [29], it can be split in a sequence of raisings to powers in the form 2^j (which will be treated as operations in the form $V \mapsto b_k V^{2^k}$ – see Sect. 3.1 –) and of 4 field multiplications. For any j, let us denote by η_j the power function $y \mapsto y^{2^j}$, the exponentiation algorithm proposed in [29] is recalled hereafter:

Algorithm 2 Exponentiation to the 254

INPUT: V
OUTPUT: $Y = V^{254}$

1. $Z \leftarrow \eta_1(V)$ $[Z = V^2]$
2. $Y \leftarrow Z \otimes V$ $[Y = V^2 V = V^3]$
3. $W \leftarrow \eta_2(Y)$ $[W = (V^3)^4 = V^{12}]$

4. $Y \leftarrow Y \otimes W$ $[Y = V^3 V^{12} = V^{15}]$
5. $Y \leftarrow \eta_4(Y)$ $[Y = (V^{15})^{16} = V^{240}]$
6. $Y \leftarrow Y \otimes W$ $[Y = V^{240} V^{12} = V^{252}]$
7. $Y \leftarrow Y \otimes Z$ $[Y = V^{252} V^2 = V^{254}]$

Starting from Algorithm 2 and applying Algorithm 1 to securely process the multiplications $\otimes$, we develop hereafter the s-box computation routine processed by each extended sub-circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j \neq i})$. We split the description in two algorithms dedicated to the exponentiation $y \mapsto y^{254}$ and to the $GF(2)$-affine function η_A respectively.

Algorithm 3 Secure Exponentiation Routine Dedicated to an Extended Circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j \neq i})$

INPUT: the i^{th} element $P_V(\alpha_i)$ of a (n, d)-sharing of V and a family of channels $(\$_{ij})_{j \neq i}$.
OUTPUT: the i^{th} element $P_{V^{254}}(\alpha_i)$ of a (n, d)-sharing of V^{254}.
PUBLIC: the n distinct points α_i, the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the matrix $(\alpha_i^j)_{1 \leq i,j \leq n}$

1. **do** $P_Z(\alpha_j) \leftarrow \eta_1(P_V(\alpha_i))$ *(where j is* s.t. $\alpha_j = \eta_1(\alpha_i))$ $[Z = \eta_1(V)]$
   do `write-and-erasure` $(P_Z(\alpha_j), \$_{ij})$
2. **for** j s.t. $\alpha_i = \eta_1(\alpha_j)$ **do**
   `read` $(P_Z(\alpha_i), \$_{ji})$
3. **do** $P_Y(\alpha_i) \leftarrow$ Algorithm 1$(P_V(\alpha_i), P_Z(\alpha_i), (\$_{ij})_{j \neq i})$ $[Y = V \otimes Z]$
4. **do** $P_W(\alpha_j) \leftarrow \eta_2(P_Y(\alpha_i))$ *(where j is* s.t. $\alpha_j = \eta_2(\alpha_i))$ $[W = \eta_2(Y)]$
   do `write-and-erasure` $(P_W(\alpha_j), \$_{ij})$
5. **for** j s.t. $\alpha_i = \eta_2(\alpha_j)$ **do**
   `read` $(P_W(\alpha_i), \$_{ij})$
6. **do** $P_Y(\alpha_i) \leftarrow$ Algorithm 1$(P_Y(\alpha_i), P_W(\alpha_i), (\$_{ij})_{j \neq i})$ $[Y = Y \otimes W]$
7. **do** $P_Y(\alpha_j) \leftarrow \eta_4(P_Y(\alpha_i))$ *(where j is* s.t. $\alpha_j = \eta_4(\alpha_i))$ $[Y = \eta_4(Y)]$
   do `write-and-erasure` $(P_Y(\alpha_j), \$_{ij})$
8. **for** j s.t. $\alpha_i = \eta_4(\alpha_j)$ **do**
   `read` $(P_Y(\alpha_i), \$_{ij})$
9. **do** $P_Y(\alpha_i) \leftarrow$ Algorithm 1$(P_Y(\alpha_i), P_W(\alpha_i), (\$_{ij})_{j \neq i})$ $[Y = Y \otimes W]$
10. **do** $P_Y(\alpha_i) \leftarrow$ Algorithm 1$(P_Y(\alpha_i), P_Z(\alpha_i), (\$_{ij})_{j \neq i})$ $[Y = Y \otimes Z]$
11. **return** $P_Y(\alpha_i)$

Before presenting the secure processing of the GF(2)-affine part η_A of the AES s-box, we recall that for every $y \in \mathrm{GF}(2)[X]/(X^8 + X^4 + X^3 + X + 1)$ it is defined by:

$$\eta_A(y) = \mathrm{0x63} + \mathrm{0x05} \otimes y + \mathrm{0x09} \otimes y^2 + \mathrm{0xf9} \otimes y^4 + \mathrm{0x25} \otimes y^8 + \mathrm{0xf4} \otimes y^{16} + \mathrm{0x01} \otimes y^{32} + \mathrm{0xb5} \otimes y^{64} + \mathrm{0x8f} \otimes y^{128} \ . \quad (1)$$

In the rest of the paper, the constant terms 0x63, 0x05, ..., 0x8f are denoted by a_{-1}, a_0, ..., a_7 respectively.

For efficiency reasons, we apply hereafter the alternative scheme proposed in Sect. 4 to process GF(2)-affine functions. This implies that the public points α_i have been chosen to satisfy the stability condition stated in Sect. 4 (the construction proposed in Appendix D can for instance be used for this purpose).

Algorithm 4 Secure η_A Processing Routine Dedicated to an Extended Circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$

INPUT: the i^{th} element $P_V(\alpha_i)$ of a (n,d)-sharing of V and a family of channels $(\$_{ij})_{j\neq i}$.
OUTPUT: the i^{th} element $P_{\eta_A(V)}(\alpha_i)$ of a (n,d)-sharing of $\eta_A(V)$.
PUBLIC: the n distinct points α_i, the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the matrix $(\alpha_i^j)_{1\leq i,j\leq n}$

1. **do** $P_{a_0 V}(\alpha_i) \leftarrow a_0 \otimes P_V(\alpha_i)$
2. **for** $t = 1$ **to** 7 **do**
3. **do** $P_{a_t V^{2^t}}(\alpha_i^{2^t}) \leftarrow a_t \otimes P_V(\alpha_i)^{2^t}$
4. **for** j s.t. $\alpha_j = \alpha_i^{2^t}$ **do**
   `write-and-erasure` $(P_{a_t V^{2^t}}(\alpha_i^{2^t}), \$_{ij})$ $[P_{a_t V^{2^t}}(\alpha_i^{2^t}) = P_{a_t V^{2^t}}(\alpha_j)]$
5. **for** j s.t. $\alpha_j^{2^t} = \alpha_i$ **do**
   `read` $(P_{a_t V^{2^t}}(\alpha_j^{2^t}), \$_{ij})$ $[P_{a_t V^{2^t}}(\alpha_j^{2^t}) = P_{a_t V^{2^t}}(\alpha_i)]$
6. **do** $P_{\eta_A(V)}(\alpha_i) \leftarrow P_{a_{-1}}(\alpha_i) \oplus P_{a_0 V}(\alpha_i) \oplus ... \oplus P_{a_7 V^{2^7}}(\alpha_i)$
7. **return** $P_{\eta_A(V)}(\alpha_i)$

We now develop hereafter the routines `AddRoundKey`, `ShiftRows` and `MixColumns` processed by each extended sub-circuit $(\mathcal{C}_{f_i}, (\$_{ij})_j)$.

Algorithm 5 `AddRoundKey` Processing Routine Dedicated to an Extended Circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$

INPUT: the i^{th} share $(P_{s_j}(\alpha_i))_{j\leq 16}$ of a (n,d)-sharing of the AES state $(s_j)_{j\leq 16}$ and the i^{th} share $(P_{k_j}(\alpha_i))_{k_j\leq 16}$ of a (n,d)-sharing of the round key $(k_j)_{j\leq 16}$.
OUTPUT: the i^{th} share $(P_{s_j\oplus k_j}(\alpha_i))_{j\leq 16}$ of a (n,d)-sharing of $(s_j \oplus k_j)_{j\leq 16}$.

1. **for** $i = 1$ **to** 16
   do $P_{s_j\oplus k_j}(\alpha_i) \leftarrow P_{s_j}(\alpha_i) \oplus P_{k_j}(\alpha_i)$
2. **return** $(P_{s_j\oplus k_j}(\alpha_i))_{j\leq 16}$

Algorithm 6 `ShiftRows` Processing Routine Dedicated to an Extended Circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$

INPUT: the i^{th} share $(P_{s_j}(\alpha_i))_{j\leq 16}$.

OUTPUT: the i^{th} share $(P_{s_j}(\alpha_i))_{j\leq 16}$ of a (n,d)-sharing of $(s_j)_{j\leq 16} = \texttt{ShiftRows}((s_j)_{j\leq 16})$.

1. $(P_{s_j}(\alpha_i))_{j\leq 16} \leftarrow \texttt{ShiftRows}((P_{s_j}(\alpha_i))_{j\leq 16})$
2. **return** $(P_{s_j}(\alpha_i))_{j\leq 16}$

The following `MixColumns` processing starts from the implementation recalled in Appendix B.

Algorithm 7 `MixColumns` Processing Routine Dedicated to an Extended Circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$

INPUT: the i^{th} share $(P_{s_j}(\alpha_i))_{j\leq 16}$.
OUTPUT: the i^{th} share $(P_{s_j}(\alpha_i))_{j\leq 16}$ of a (n,d)-sharing of $(s_j)_{j\leq 16} = \texttt{MixColumns}((s_j)_{j\leq 16})$.

1. **for** $\ell = 0$ **to** 3
   do $P_{tmp}(\alpha_i) \leftarrow P_{s_{1+\ell}}(\alpha_i) \oplus P_{s_{5+\ell}}(\alpha_i) \oplus P_{s_{9+\ell}}(\alpha_i) \oplus P_{s_{13+\ell}}(\alpha_i)$
   do $P_{s_{1+\ell}}(\alpha_i) \leftarrow \texttt{02} \otimes (P_{s_{1+\ell}}(\alpha_i) \oplus P_{s_{5+\ell}}(\alpha_i)) \oplus P_{tmp}(\alpha_i) \oplus P_{s_{1+\ell}}(\alpha_i)$
   do $P_{s_{5+\ell}}(\alpha_i) \leftarrow \texttt{02} \otimes (P_{s_{5+\ell}}(\alpha_i) \oplus P_{s_{9+\ell}}(\alpha_i)) \oplus P_{tmp}(\alpha_i) \oplus P_{s_{5+\ell}}(\alpha_i)$
   do $P_{s_{9+\ell}}(\alpha_i) \leftarrow \texttt{02} \otimes (P_{s_{9+\ell}}(\alpha_i) \oplus P_{s_{13+\ell}}(\alpha_i)) \oplus P_{tmp}(\alpha_i) \oplus P_{s_{9+\ell}}(\alpha_i)$
   do $P_{s_{13+\ell}}(\alpha_i) \leftarrow P_{s_{1+\ell}}(\alpha_i) \oplus P_{s_{5+\ell}}(\alpha_i)) \oplus P_{s_{9+\ell}}(\alpha_i) \oplus P_{tmp}(\alpha_i)$
2. **return** $(P_{s_j}(\alpha_i))_{j\leq 16}$

The AES key-scheduling function can itself be split into only affine transformations and s-box computations. Hence, the same strategy can be followed to split its processing between the $\mathcal{C}_{f_i}$ as we did with the AES round transformation. The `KeyScheduling` secure processing routine dedicated to the extended circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$ is denoted by `KeyScheduling_sec`.

We now propose an algorithmic description of the computations implemented by each extended sub-circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$ such that the family $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})_i$ is a (n,d)-multi-party circuit implementing the AES. We stress here that all these algorithms are a direct expression of BGW's secure protocol and that there is no need for a special treatment due to the context. The correctness/security of these algorithms is hence not discussed here and the reader can refer to Section 3 or directly to the original paper [1] to convince him/herself about it.

Algorithm 8 Secure AES Processing Routine Dedicated to an Extended Circuit $(\mathcal{C}_{f_i}, (\$_{ij})_{j\neq i})$

INPUT: the i^{th} share $(P_{s_j}(\alpha_i))_{j\leq 16}$ of a (n,d)-sharing of the AES state $(s_j)_{j\leq 16}$ and the i^{th} share $(P_{k_j}(\alpha_i))_{k_j\leq 16}$ of a (n,d)-sharing of the master key $(k_j)_{j\leq 16}$.
OUTPUT: the i^{th} element $P_V(\alpha_i)$ of a (n,d)-sharing of $S(V)$.
PUBLIC: the n distinct points α_i, the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the matrix $(\alpha_i^j)_{1\leq i,j\leq n}$

*** All but last rounds ***

1. **for** $r = 1$ **to** 10
2. **do** $(P_{s_j}(\alpha_i))_{j\leq 16} \leftarrow$ Algorithm-5 $((P_{s_j}(\alpha_i))_{j\leq 16}, (P_{k_j}(\alpha_i))_{j\leq 16})$[`AddRoundKey` Part]
3. **for** $j = 1$ **to** 16
   do $P_{s_j}(\alpha_i) \leftarrow$ Algorithm-3$(P_{s_j}(\alpha_i), (\$_{ij})_{j\neq i})$ [`SubBytes` Exp. Part]
   do $P_{s_j}(\alpha_i) \leftarrow$ Algorithm-4$(P_{s_j}(\alpha_i), (\$_{ij})_{j\neq i})$ [`SubBytes` η_A. Part]
4. **do** $(P_{s_j}(\alpha_i))_{j\leq 16} \leftarrow$ Algorithm-6$((P_{s_j}(\alpha_i))_{j\leq 16})$ [`ShiftRows` Part]
5. **do** $(P_{s_j}(\alpha_i))_{j\leq 16} \leftarrow$ Algorithm-7$((P_{s_j}(\alpha_i))_{j\leq 16})$ [`MixColumns` Part]
6. **do** $(P_{k_j}(\alpha_i))_{j\leq 16} \leftarrow$ `KeyScheduling_sec` $((P_{k_j}(\alpha_i))_{j\leq 16}, (\$_{ij})_{j\neq i})$[`KeyScheduling` Part]

*** Last round ***

7. **do** $(P_{s_j}(\alpha_i))_{j\leq 16} \leftarrow$ Algorithm-5 $((P_{s_j}(\alpha_i))_{j\leq 16}, (P_{k_j}(\alpha_i))_{j\leq 16})$[`AddRoundKey` Part]
8. **for** $j = 1$ **to** 16
   do $P_{s_j}(\alpha_i) \leftarrow$ Algorithm-3$(P_{s_j}(\alpha_i), (\$_{ij})_{j\neq i})$ [`SubBytes` Part]
9. **do** $(P_{s_j}(\alpha_i))_{j\leq 16} \leftarrow$ Algorithm-6$((P_{s_j}(\alpha_i))_{j\leq 16})$ [`ShiftRows` Part]
10. **do** $(P_{k_j}(\alpha_i))_{j\leq 16} \leftarrow$ `KeyScheduling_sec` $((P_{k_j}(\alpha_i))_{j\leq 16}, (\$_{ij})_{j\neq i})$[`KeyScheduling` Part]
11. **do** $(P_{s_j}(\alpha_i))_{j\leq 16} \leftarrow$ Algorithm-5 $((P_{s_j}(\alpha_i))_{j\leq 16}, (P_{k_j}(\alpha_i))_{j\leq 16})$[`AddRoundKey` Part]

12. **return** $(P_{s_j}(\alpha_i))_{j\leq 16}$

By construction, the family of extended sub-circuits $(\mathcal{C}_{f\,i}, (\$_{ij})_{j\neq i})_i$, each of them processing Algorithm 8, implements the AES-128 block cipher. It is moreover a (n, d)-multi-party circuit since it is an instance of BGW's protocol.

To construct the (n, d)-sharings of the plaintext and the master key which are distributed to the sub-circuits, the following algorithm is applied:

Algorithm 9 (n, d)-sharing Construction

INPUT: a 16-byte vector $(V_j)_{j\leq 16}$
OUTPUT: a (n, d)-sharing $((P_{V_j}(\alpha_1))_{j\leq 16}, \cdots, (P_{V_j}(\alpha_n))_{j\leq 16})$ of $(V_j)_{j\leq 16}$.
PUBLIC: n distinct points α_i.

---

1. **for** $j = 1$ **to** 16

Randomly generate a family $(a_i)_{i\leq d}$ of coefficients in GF(256).

2. **for** $t = 1$ **to** d
   do $a_t \leftarrow$ `rand`(GF(256))

Compute $P_{V_j}(\alpha_i) = m + \sum_{j\leq d} a_j \cdot \alpha_i^j$

3. **for** $i = 1$ **to** n
   do $P_{V_j}(\alpha_i) \leftarrow V_j$
4. **for** $t = 1$ **to** d
   do $P_{V_j}(\alpha_i) \leftarrow P_{V_j}(\alpha_i) + a_j \cdot \alpha_i^t$

5. **return** $((P_{V_j}(\alpha_1))_{j\leq 16}, \cdots, (P_{V_j}(\alpha_n))_{j\leq 16})$

Eventually, the following algorithm is applied to reconstruct the AES output from the (n, d)-sharing $((P_{s_j}(\alpha_1))_{j\leq 16}, \cdots, (P_{s_j}(\alpha_n))_{j\leq 16})$ returned by the sub-circuits $\mathcal{C}_{f_i}$ at the end of Algorithm 8.

Algorithm 10 Reconstruction

INPUT: a (n, d)-sharing $((P_{s_j}(\alpha_1))_{j\leq 16}, \cdots, (P_{s_j}(\alpha_n))_{j\leq 16})$
OUTPUT: the un-shared value $(s_j)_{j\leq 16}$.
PUBLIC: the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the matrix $(\alpha_i^j)_{1\leq i,j\leq n}$.

1. **for** $j = 1$ **to** 16
   Process Lagrange's Interpolation.
2. **for** $i = 1$ **to** n
   do $s_j \leftarrow \sum_{i=1}^{n} \lambda_i P_{s_j}(\alpha_i)$
3. **return** $(s_j)_{j\leq 16}$

This concludes the full algorithmic description of a d-glitches free AES-128 implementation. We will now focus on the study of Shamir's secret sharing scheme as a masking scheme.

5 The Polynomial Masking Function

In previous sections we have used Shamir's sharing scheme to secure the sensitive data manipulations and we have applied SMC protocols to enable secure elementary computations on the shared variables. This section focusses on Sharing's scheme and studies it as a masking technique, hereafter referred as *polynomial masking*. In particular, we compare its SCA resistance w.r.t. the classical Boolean masking [12] which is up to now the only masking which has been applied to define d-probing resistant implementations for any d [13, 29].

Since the efficiency of a d^{th}-order SCA decreases exponentially when d grows (see Sect. 1), the amount of information leaking on a shared sensitive variable when targeting all the shares, is a sound way to compare two (n, d)-sharings [9, 32]. To quantify this amount, we need to model the relationship between the physical leakage and the value of the variable manipulated at the time of the leakage. For such a purpose, we associate each d-tuple of shares $(Z_0, \cdots, Z_d)$ with a $(d + 1)$-tuple of leakages $\mathbf{L} = (L_0, \cdots, L_d)$ s.t. $L_i = HW(Z_i) + \beta_i$, with β_i an independent Gaussian noise with mean 0 and standard deviation σ^2. We shall use the notation

$\mathbf{L} \leftarrow (Z_0, \cdots, Z_d)$ to refer to this association. Since we assumed that the variable Z and the masking material are uniformly distributed over their definition sets, the signal-to-noise ratio (SNR) of the leakage can be defined as $\mathrm{Var}\left[\mathrm{HW}(Z_i)\right] / \mathrm{Var}\left[\beta_i\right]$ (that is $2/\sigma^2$ if we assume that Z_i is 8-bit long).

To evaluate the information revealed by each tuple of shares for Boolean and polynomial masking techniques, we computed the mutual information[7] $I(Z, \mathbf{L})$ between the sensitive variable Z and $\mathbf{L}$. We list hereafter the leakages we considered and the underlying leaking variables:

$$\text{1st-order Boolean masking:} \quad \mathbf{L} \leftarrow (Z \oplus B, B) \ . \tag{2}$$

$$\text{2nd-order Boolean masking:} \quad \mathbf{L} \leftarrow (Z \oplus B_1 \oplus B_2, B_1, B_2) \ . \tag{3}$$

$$\text{1st-order Poly. masking:} \quad \mathbf{L} \leftarrow (P_Z(\alpha_1), P_Z(\alpha_2)) \ . \tag{4}$$

$$\text{2nd-order Poly. masking:} \quad \mathbf{L} \leftarrow (P_Z(\alpha_1), P_Z(\alpha_2), P_Z(\alpha_3)) \ . \tag{5}$$

Remark 12. The polynomial P_Z considered in (4) *and* (5) *is an otherwise random polynomial with constant coefficient Z and degree at most 1 and 2 respectively. The α_i are distinct public values.*

Figure 3 summarizes the information theoretic evaluation for each leakage (1) to (4). For d equal to 1 or 2, it can be observed that the amount of information revealed by the $d+1$ elements of the Boolean masking is greater than that revealed by the $d+1$ elements of the polynomial masking. Moreover, the gap between the two amount of information gets wider as the noise increases and, apparently, when the order d increases. This can be explained by the greater algebraic complexity of the polynomial masking function compared to the Boolean one. Indeed, the algebraic degree of polynomial masking increases with the masking order d whereas Boolean masking stays linear. Eventually, it can also be observed that, at a fixed order (1 or 2 here), the difference in the amount of information retrieved on the sensitive variable between Boolean and polynomial maskings is exponential in the square root of the noise standard deviation σ.

Following the same reasoning as in [9, 32], we deduce from the observations above that polynomial masking offers a better resistance to HO-SCA attacks. Our analysis also suggests that difference between the

[7] As shown in [32], the number of measurements required to achieve a given success-rate in a maximum likelihood attack can be expressed as a function of the mutual information evaluation and equals $c \times I(Z, \mathbf{L})^{-1}$, where c is a constant related to the chosen success-rate.

two resistance levels increases with the noise standard deviation and the order.

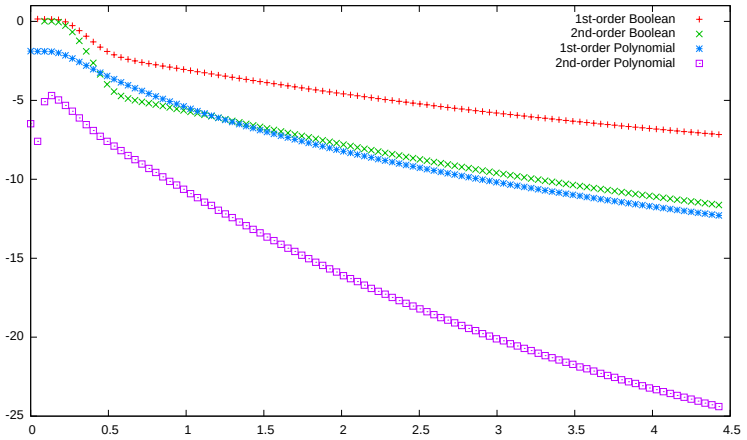


Fig. 3. Mutual information ($\log_{10}$) between the leakage and the sensitive variable over an increasing noise standard deviation (x-axis).

6 Conclusion

Thanks to the notion of multi-party circuit, we have shown in this paper that it is possible to prove, under realistic assumptions, the resistance of a d^{th}-order masking scheme in the presence of glitches. This new framework enables to convert any classical d^{th}-order secure scheme into an implementation immune to glitches effects. The complexity of the new implementation greatly depends on the number of sub-circuits in which the initial scheme has been shared. To reduce the complexity, the latter scheme must therefore be carefully chosen. Here, we have proposed to adapt the SMC protocol proposed in [1] to define a circuit sharing that is particularly well suited to our problematic. We have applied it to exhibit a full description of a d-glitches free AES-128 implementation. As a side effect of basing our security on a SMC scheme, the protocol is intrinsically immune against fault injection attacks when fewer than 1/3 of the sub-circuits are corrupted. This is a real asset of our proposal compared to the existing ones when both active and passive attacks must be thwarted by the implementation. Finally, we have formally studied the effectiveness

of Shamir's secret sharing which is a core primitive of [1]. In particular, we have argued that it is much more robust that the classical Boolean sharing.

The multi-party circuit model permits the systematic use of SMC protocols for building d-glitches free schemes. As a first approach, we based our study on very strong hypotheses on the attacker power. Even if such brutal approach allows us to develop sound proofs of security, the resulted secure implementation is costly. Future works could investigate more realistic (weaker) adversary models in order to build lighter secure implementations, or, to the same purpose, study alternative SMC protocols, less generic than BGW's protocol but more efficient. Another avenue could be to study some existing optimizations of BGW's protocol (*e.g.* the optimization based on Franklin and Yung's trick [8] that is based on efficient parallel computations).

References

1. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM.
2. J. Blömer, J. G. Merchan, and V. Krummel. Provably Secure Masking of AES. In M. Matsui and R. Zuccherato, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 69–83. Springer, 2004.
3. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. In *Second AES Candidate Conference – AES 2*, Mar. 1999.
4. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.
5. J.-S. Coron. A New DPA Countermeasure Based on Permutation Tables. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, *SCN 2008*, volume 5229 of *LNCS*, pages 278–292. Springer, 2008.
6. J. Daemen, M. Peeters, G. Assche, and V. Rijmen. The Noekeon Block Cipher. In *Proceedings of first NESSIE Workshop*, 2000. http://cryptonessie.org.
7. T. Eisenbarth, C. Paar, and B. Weghenkel. Building a side channel based disassembler. In M. Gavrilova, C. Tan, and E. Moreno, editors, *Transactions on Computational Science X*, volume 6340 of *Lecture Notes in Computer Science*, pages 78–99. Springer Berlin / Heidelberg, 2010.
8. M. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 699–710, New York, NY, USA, 1992. ACM.
9. G. Fumaroli, A. Martinelli, E. Prouff, and M. Rivain. Affine masking against higher-order side channel analysis. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010.

10. R. Gennaro, M. O. Rabin, and T. Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In *PODC*, pages 101–111, 1998.
11. Goldack. Side-channel based reverse engineering for microcontrollers, 2008. Master's thesis, Ruhr-Universität, Bochum, Germany.
12. L. Goubin and J. Patarin. DES and Differential Power Analysis – The Duplication Method. In Ç. Koç and C. Paar, editors, *CHES '99*, volume 1717 of *LNCS*, pages 158–172. Springer, 1999.
13. Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In D. Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
14. M. Joye, P. Paillier, and B. Schoenmakers. On Second-order Differential Power Analysis. In Rao and Sunar [27], pages 293–308.
15. P. Kocher, J. Jaffe, and B. Jun. Introduction to Differential Power Analysis and Related Attacks. Technical report, Cryptography Research Inc., 1998.
16. R. Lidl and H. Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, seconde edition, 1997. Avec une introduction de P. M. Cohn.
17. S. Mangard, T. Popp, and B. M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *LNCS*, pages 351–365. Springer, 2005.
18. S. Mangard and K. Schramm. Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In L. Goubin and M. Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 76–90. Springer, 2006.
19. T. Messerges. Using Second-order Power Analysis to Attack DPA Resistant Software. In Ç. Koç and C. Paar, editors, *CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer, 2000.
20. S. Micali and L. Reyzin. Physically Observable Cryptography (Extended Abstract). In M. Naor, editor, *Theory of Cryptography Conference – TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296. Springer, 2004.
21. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of aes. In K. G. Paterson, editor, *EUROCRYPT*, volume 6632 of *LNCS*, pages 69–88. Springer, 2011.
22. S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In P. Ning, S. Qing, and N. Li, editors, *IICICS'06*, volume 4307 of *LNCS*, pages 529–545. Springer, 2006.
23. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. In P. J. Lee and J. H. Cheon, editors, *ICISC 2008*, volume 5461 of *LNCS*, pages 218–234. Springer, 2008.
24. S. Nikova, V. Rijmen, and M. Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.
25. G. Piret and F.-X. Standaert. Security Analysis of Higher-Order Boolean Masking Schemes for Block Ciphers (with Conditions of Perfect Masking). *IET Information Security*, 2:1–11, 2008.
26. E. Prouff and T. Roche. Attack on a higher-order masking of the aes based on homographic functions. In G. Gong and K. Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010*, volume 6498 of *Lecture Notes in Computer Science*, pages 262–281. Springer Berlin / Heidelberg, 2010.
27. J. Rao and B. Sunar, editors. *CHES 2005*, volume 3659 of *LNCS*. Springer, 2005.

28. M. Rivain, E. Dottax, and E. Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. Cryptology ePrint Archive, Report 2008/021, 2008. http://eprint.iacr.org/.
29. M. Rivain and E. Prouff. Provably secure higher-order masking of aes. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
30. K. Schramm and C. Paar. Higher Order Masking of the AES. In D. Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 208–225. Springer, 2006.
31. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.
32. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The world is not enough: Another look on second-order dpa. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer Berlin / Heidelberg, 2010.
33. D. Suzuki, M. Saeki, and T. Ichikawa. DPA Leakage Models for CMOS Logic Circuits. In Rao and Sunar [27], pages 366–382.
34. A. C.-C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society.

A Sub-circuits Separation Models

- *Temporal Separation* (suited to sequential execution): the sub-circuits are processed sequentially without overlapping. Depending on the platform, it may need a state re-initialization step between each sub-circuit execution, such that an operation has no influence on the side-channel leakage of the next operation (owned by the next sub-circuit). Indeed, glitches attacks may be related to the notion of persistence of an operation. In Software for instance, when considering a pipelined processor, the power consumption of the current executed instruction can be influenced by previous or next instructions (see for instance Goldack Master Thesis on SCA based Retro-Engineering [11] and the recent paper of Eisenbarth *et al.* [7]).
- *Spacial Separation* (suited to distributed computations): the sub-circuits are processed on different hardware platforms. The platforms shall be connected through a communication channel but shall have completely independent side-channel leakage. In a Software context, one can for instance use a Multiprocessors System-on-Chip (MPSoC), where each processor executes a different sub-circuit and communicates through a bus with restricted read/write access. In a Hardware context, a similar distributed setup can be designed with n devices (*e.g.* FPGA) that implement the n different sub-circuits.

B `MixColumns` Processing

In the paper, we will assume that MixColumns is implemented as

$$\begin{cases} s'_{0,c} \leftarrow \texttt{02}(s_{0,c} \oplus s_{1,c}) \oplus tmp \oplus s_{0,c} \\ s'_{1,c} \leftarrow \texttt{02}(s_{1,c} \oplus s_{2,c}) \oplus tmp \oplus s_{1,c} \\ s'_{2,c} \leftarrow \texttt{02}(s_{2,c} \oplus s_{3,c}) \oplus tmp \oplus s_{2,c} \\ s'_{3,c} \leftarrow s'_{0,c} \oplus s'_{1,c} \oplus s'_{2,c} \oplus tmp \end{cases}$$

where $tmp = s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus s_{3,c}$ and where $\texttt{02}$ denotes a look-up table for the function $x \mapsto \texttt{02} \cdot x$.

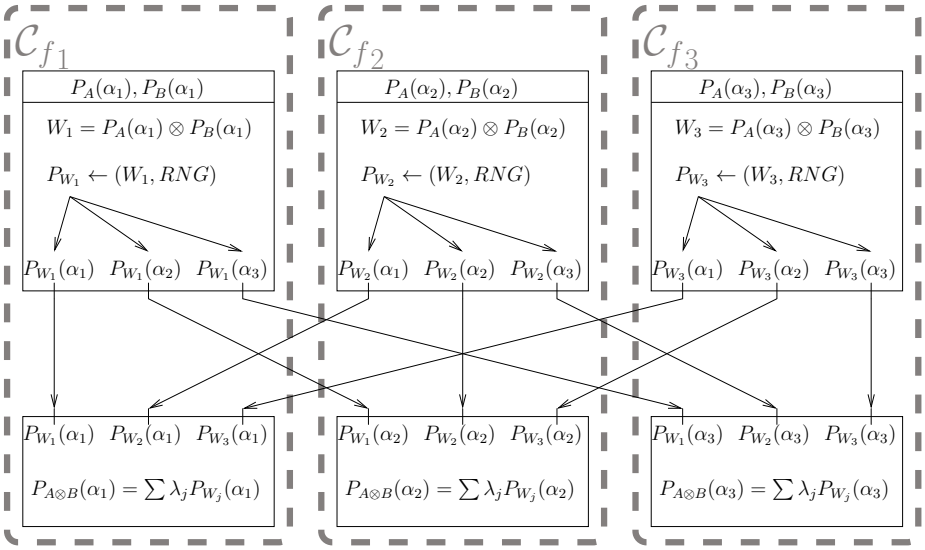
C First-Order Glitches Free Masking Scheme of The Product in GF(2^m)

In this annexe, we give a block diagram description of the execution of a product in GF(2^m) following the scheme presented in Section 3.2 for a masking order equal to 1. We denote by A and B the two values to be multiplied. The Shamir's secret sharing of A (resp. B) is denoted by $\{P_A(\alpha_1), P_A(\alpha_2), P_A(\alpha_3)\}$ (resp. $\{P_B(\alpha_1), P_B(\alpha_2), P_B(\alpha_2)\}$), where P_A (resp. P_B) is a random polynomial of degree at most 1 with constant coefficient equal to A (resp. B) and where $\{\alpha_1, \alpha_2, \alpha_3\}$ are public fixed elements of GF$(2^m)^\star$.
On Figure C, the $\{\lambda_i\}$ denote the elements of the first row of the inverse of the Vandermonde $(n \times n)$-matrix $(\alpha_i^j)_{1 \leq i,j \leq n}$, these values are fixed and considered pre-computed once for all. Moreover, the notation $P_X \leftarrow (X, RNG)$ means the generation of a polynomial P_X of degree at most 1 such that its constant coefficient is equal to X and the other one is random (*i.e.* generated by "RNG").

D Design of Sub-Sets in GF(2^8) Stable By Frobenius Transformation

Theorem 2. *For any integer $d < 2^8$, there exists a sub-set $S \subseteq GF(2^8)^\star$ with cardinality d and such that $x \in S$ implies $x^2 \in S$.*

Proof. In the following, we will denote by β a primitive element of GF(2^8). For some $s \in \{0, \cdots, 2^8 - 1\}$, let us consider the set $C_s = \{s, 2s, \cdots, 2^{m_s - 1}s\}$ where m_s is the smallest integer such that $2^{m_s}s \equiv s \bmod (2^8 - 1)$ (the set C_s is usually called 2-class cyclotomic of s modulo $2^8 - 1$). By construction, the set $\mathcal{C}_s = \{\beta^i\}_{i \in C_s}$ is stable by the Frobenius transformation (*i.e.*



squaring in GF(2^8)). Moreover, it is well known that $\mathcal{C}_s$ is the set of the roots of the β^s's minimal polynomial, which degree divides 8. Hence, the cardinal of $\mathcal{C}_s$, denoted N_s, also divides 8.

Using Moebius Inversion Formula, it is possible to enumerate all sets $\mathcal{C}_s$ of fixed cardinal N_s that divides 8 (see, for instance, [16, Theorem 3.25]), we then have:

- 2 sets of 1 element
- 1 set of 2 elements
- 3 sets of 4 elements
- 30 sets of 8 elements

Finally it can be checked that for any $d < 2^8$ it is possible to construct a set S of cardinality d by union of some of the sets enumerated above (minus the set containing 0). Such a set S would contain distinct and non-zero elements. Moreover it would be stable by the Frobenius operation.

E First-order Glitches Free Masking of the AES S-Box

Hereafter we detail the masking of the AES s-box for $d = 1$ (*i.e.* three shares). As recalled in Sect. 4, this function takes the form $\eta_A \circ \mathrm{Inv}$ where η_A is a GF(2)-affine function and Inv is the function $v \mapsto v^{254}$ where

multiplications are computed over $\mathrm{GF}(2)[X]/(X^8 + X^4 + X^3 + X + 1)$. We first focus on the field inversion computation (Algorithm 3) and then we detail how to process the affine part of the s-box. We assume that the secure s-box processing is done for a variable V that is split into $n = 3$ shares $V_0 = P_V(\alpha_0)$, $V_1 = P_V(\alpha_1)$ and $V_2 = P_V(\alpha_2)$ where $P_V(X)$ is a random degree-1 polynomial with constant term V and where α_0, α_1 and α_2 satisfy the two conditions given in Sect. 3.1. In the following, we assume that $P_V(X)$ equals $V + aX$ and we chose $\alpha_0 = 1$, $\alpha_1 = 0xbc$ and $\alpha_2 = 0xbd$.

- By definition of P_V, at the beginning of Algorithm 3, we have:

$$V_0 = V + a\alpha_0 \ , \tag{6}$$
$$V_1 = V + a\alpha_1 \ , \tag{7}$$
$$V_2 = V + a\alpha_2 \ . \tag{8}$$

- After the first squaring (Step 1 in Algorithm 3), the three sub-circuits $\mathcal{C}_{f_i}$ have a $(3, 1)$-sharing (Z_0, Z_1, Z_2) of the variable $Z = V^2$. By construction, the shares satisfy:

$$Z_0 = V^2 + (a\alpha_0)^2 = V^2 + a^2\alpha_0 \tag{9}$$
$$Z_1 = V^2 + (a\alpha_1)^2 = V^2 + a^2\alpha_2 \tag{10}$$
$$Z_2 = V^2 + (a\alpha_2)^2 = V^2 + a^2\alpha_1 \ , \tag{11}$$

  since $\alpha_1^2 = \alpha_2$. Reordering of the shares is needed since Z_1 relates to public points α_2 whereas Z_2 relates to α_1. Circuits $\mathcal{C}_{f_1}$ and $\mathcal{C}_{f_2}$ are hence up to now assumed to exchange their roles for the remaining of the protocol.
- After Algorithm 3 next step, the $\mathcal{C}_{f_i}$ have built shares Y_0, Y_1 and Y_2 such that:

$$Y_0 = V^3 + b\alpha_0 \tag{12}$$
$$Y_1 = V^3 + b\alpha_1 \tag{13}$$
$$Y_2 = V^3 + b\alpha_2 \ , \tag{14}$$

  where b is some new coefficient.
- Then comes the rising to the power 4 and the sub-circuits have three new shares W_0, W_1 and W_2 such that:

$$W_0 = V^{12} + (b\alpha_0)^4 = V^{12} + b^4\alpha_0 \tag{15}$$
$$W_1 = V^{12} + (b\alpha_1)^4 = V^{12} + b^4\alpha_1 \tag{16}$$
$$W_2 = V^{12} + (b\alpha_2)^4 = V^{12} + b^4\alpha_2 \tag{17}$$

In this case, no re-ordering is needed. In fact, for all (linear) exponentiations by a power in the form 2^i with i even, the re-ordering is not needed. It is only needed for exponentiations by a power in the form 2^i with i odd. As a consequence, all the remaining steps of Algorithm 3 (which correspond to the even case) can be performed as in the CHES paper, with no need for re-ordering.

Let us now consider the affine part η_A of the AES s-box. Since η_A is GF(2)-affine, then for any polynomial representation $\mathbb{F}_2[X]/p(X) \cong \mathbb{F}_{256}$ there exists a unique tuple of coefficients $(a_0, \cdots, a_7) \in (\mathbb{F}_2[X]/p(X))^8$ such that η_A can be defined for every $x \in \mathbb{F}_2[X]/p(X)$ by:

$$\eta_A(x) = 0\text{x}63 + \sum_{i=0}^{7} a_i \cdot x^{2^i} \quad . \tag{18}$$

Actually, for AES the function η_A has been designed such that it takes the simplest form $\eta_A(x) = a_0 + a_1 x$ when considered on a particular polynomial representation $\mathbb{F}_2[X]/p'(X)$ of $\mathbb{F}_{256}$. However, the latter representation differs from that used to describe the AES sbox inverse part. For the latter representation, which is $\mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$, the general form (18) becomes:

$$\begin{aligned}\eta_A(x) = 0\text{x}63 &+ 0\text{x}05 \cdot x + 0\text{x}09 \cdot x^2 + 0\text{xf}9 \cdot x^4 + 0\text{x}25 \cdot x^8 \\ &+ 0\text{xf}4 \cdot x^{16} + 0\text{x}01 \cdot x^{32} + 0\text{xb}5 \cdot x^{64} + 0\text{x}8\text{f} \cdot x^{128} \quad . \end{aligned} \tag{19}$$

From now, we do not consider the constant term 0x63 (with which it is easy to deal when applying the masking scheme) and we assume that a_0, a_1, ..., a_7 denote 0x05, 0x09, ... and 0x8f respectively. We have seen that the exponentiations by powers in the form 2^i (which are GF(2)-linear) imply a re-ordering of the shares when i is odd. Based on this remark, we split the computation of $\eta_A(x)$ as described in (19) into two sub-sums (one for the even i and one for the odd i): we get $\eta_A(x) = A_1(x) + A_2(x)$ with $A_1(x) = \sum_{i=0}^{3} a_{2i+1} x^{2^{2i+1}}$ and $A_2(x) = \sum_{i=0}^{3} a_{2i} x^{2^{2i}}$. The two sub-sums can be processed on each share separately (without taking care of shares re-ordering). Once this is done, we just need to exchange the roles of players 1 and 2 at the end of the first sub-sum processing and then to add the obtained three shares with the three shares at output of the second sub-sum. Eventually, the constant term 0x63 is added to each share. We sum-up the secure η_A processing hereafter when applied to the $(3, 1)$-sharing (Z_1, Z_2, Z_3) of Z (by construction, we recall that they satisfy $Z_0 = Z + a\alpha_0$, $Z_1 = Z + a\alpha_1$ and $Z_2 = Z + a\alpha_2$).

1. Compute the first sub-sum A_1:

$$Z'_0 = A_1(Z_0) \tag{20}$$
$$Z'_1 = A_1(Z_1) \tag{21}$$
$$Z'_2 = A_1(Z_2) \tag{22}$$

   Note that this processing can be done thanks to a look-up table.
2. Exchange the two last shares (*e.g.* thanks to an index change):

$$Z'_1 \leftrightarrow Z'_2 \tag{23}$$

3. Compute the second sub-sum A_2:

$$Z''_0 = A_2(Z_0) \tag{24}$$
$$Z''_1 = A_2(Z_1) \tag{25}$$
$$Z''_2 = A_2(Z_2) \tag{26}$$

   Note that this processing can be done thanks to a look-up table.
4. Final processing:

$$Z_0 \leftarrow Z'_0 + Z''_0 + 0\text{x}63 \tag{27}$$
$$Z_1 \leftarrow Z'_1 + Z''_1 + 0\text{x}63 \tag{28}$$
$$Z_2 \leftarrow Z'_2 + Z''_2 + 0\text{x}63 \tag{29}$$

The 3-tuple (Z_0, Z_1, Z_2) is a sharing of $A(Z^{-1})$ that is of AES-SBOX(Z).