Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols

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

[1] Oberthur Technologies, 71-73, rue des Hautes Pâtures 92726 Nanterre, France
e.prouff@oberthur.com

[2] ANSSI, 51 boulevard de La Tour-Maubourg 75700 Paris 07 SP, France
thomas.roche@ssi.gouv.fr

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. We introduce in this paper a circuit model that encompasses sufficient conditions to resist glitches effects. This allows us to construct the first countermeasure thwarting both glitches and HO-SCA attacks. Our new construction requires Secure Multi-Party Computation protocols and we propose to apply the one 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.

1 Introduction

Higher-0rder Side-Channel Analysis (HO-SCA for short) is a class of *physical cryptanalyses* against cryptosystems. They generalize the seminal side-channel attacks introduced in the late nineties by Kocher *et al.* [11]. Contrary to the latter attacks that only exploit instantaneous leakages, HO-SCA attacks mix observations of several leakages to recover information about the secret parameters of the targeted algorithm. The number of different leakages (*e.g.* related to different times during the processing or to different locations in a circuit) 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.*

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

B. Preneel and T. Takagi (Eds.): CHES 2011, LNCS 6917, pp. 63–78, 2011.

sharing) techniques [3, 8]. 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. In this case, it is called a *(n, d)-sharing.* A scheme specifying how to apply (n, d)-sharing to protect 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. Even though a $(d+1)^{\text{th}}$-order SCA exploiting the leakage of $d+1$ shares can always theoretically be successfully performed, it has been shown in [3] that the complexity of such an attack increases exponentially with the order due to noises effects. Hence a d^{th}-order masking scheme is a sound countermeasure against HO-SCA attacks when d is high enough w.r.t. the noise. Some d^{th}-order masking schemes have been proposed with formal security proof [9, 23–25]. However in 2004 Mangard *et al.* [12] pointed out a weakness in the adversary model involved to construct these masking schemes: when an implementation is processed, so-called *hardware glitches* — common in CMOS technology — occur that deteriorate the effect of masking and leak more information than the simple value of the variables specified by the implementation. Since the seminal work of Mangard *et al.* , several papers have successfully applied so-called *glitches attacks* against implementations that were secure in the classical HO-SCA adversary model (see *e.g.* [12]). Up to now, a unique masking scheme has been presented as secure in presence of glitches [16–19]. This masking scheme 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.* [16–19]. In those papers, a masking scheme is proposed that is claimed to resist 1^{st}-order SCA in presence of glitches. Unfortunately, adapting the proposed schemes to also counteract SCA attacks 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.* [9] and Rivain *et al.* [24]. Ishai *et al.* 's solution is dedicated to hardware implementations. It clearly does not thwart glitches attacks and modifying it to achieve resistance against the latter ones is an (open) issue. Rivain *et al.* 's solution is dedicated to software context. When embedded in a physical device there is no guaranty that no glitches effects will occur during the processing. Providing such a guaranty may be possible by ensuring 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

In this paper, we introduce a generic framework for the design of Hardware or Software implementations that counteract HO-SCA in presence of glitches (Sect. 2). 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. Secondly, we establish 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.* on *non-cryptographic* SMC protocols [1], we show how to design a multi-party circuit that can implement any function with a minimum number of sub-circuits (Sect. 3). As an example, we specify such a circuit for the AES-128 algorithm (Sect. 4 and [21] for a description of the scheme for the full AES-128). 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.

2 Preliminaries 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 Sect(s). 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 $\mathbf{Z}$ 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* $L(\mathbf{Z})$ which returns a noisy information about it. The noisy property of $L(\cdot)$ is captured by assuming that the bias introduced in the distribution of $\mathbf{Z}$ given the leakage $L(\mathbf{Z})$ 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 were 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 ($d^{\mathbf{th}}$-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 [9]. 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, 10, 14]. In this case, $\mathcal{L}$ is usually defined as the set of noisy leakage functions $L(\cdot) = \mathrm{HW}(\cdot) + \mathcal{B}(\mu, \sigma)$, 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.*

The two works of Ishai *et al.* [9] and Rivain and Prouff [24] 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.* [12] and [13]). 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 [12]. The same year, a technique to model the effect of glitches on the side-channel resistance of circuits has been published in [27]. Hereafter we model an adversary which performs glitches attacks against a circuit. For such a purpose, and following the same approach as in [27], 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 depends 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. 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 $V_{ij}(t)$ 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 $\mathcal{L}$.*

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 false. In the following sections, we introduce the notions of *d-probing security* and of *d-glitches freeness*. In both cases, we exhibit generic constructions 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.* [2,4,10,14,20]).

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 *iff no family of at most d elements in $\mathcal{V}$ is sensitive.*

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 F from* $\mathrm{GF}(2^m)^n$ *into* $\mathrm{GF}(2^m)$ s.t. $F(Z_1, \cdots, Z_n) = Z$,
2. *for every $I \subset [1; n]$* s.t. $\#I \leq d$, *we have* $\Pr(Z \mid (Z_i)_{i\in I}) = \Pr(Z)$.

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 3. 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, 24].

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.* [9] and Rivain and Prouff [24] are d-probing secure for any fixed value d. Unfortunately, those schemes are not by construction secure in the Glitches Adversary Model. In the following, we introduce a new strategy to directly obtain circuits secure in the Glitches Adversary Model.

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 the processings leak 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. By consequence, only a certain type of function f (homomorphic with respect to the sharing) can be split into n independent computations, each operating only a single share of f's input and returning a sharing of $\mathcal{C}_f$'s output.

Secure Multi-Party Computation protocols, and in particular the protocol in [1] recalled in next section, are methods that extend the above idea to any function f (*i.e.* not only homomorphic function for which, as we have seen, the solution is straightforward). This extension however requires to provide each sub-circuit $\mathcal{C}_{f_i}$ with the ability to send to the other ones information about some of its intermediate results. To ensure that this new ability does not impact on the d-glitches freeness of the circuit composed of the n sub-circuits, this sent information must itself be shared between all the other n sub-circuits. This implies that each sub-circuit must only be able to access a single share of a (n, d)-sharing of the intermediate result of another sub-circuit. To ensure that a sub-circuit cannot receive several shares of a same (n, d)-sharing or cannot access several shares related to dependent (n, d)-sharings, we set the condition that all the shares accessed by a sub-circuit come from distinct and independent (n, d)-sharings. To formalize our construction, each sub-circuit $\mathcal{C}_{f_i}$ is extended 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. 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 give a formal definition of it hereafter.

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 n extended sub-circuits $(\mathcal{C}_{f_1}, (\$_{1j})_{j\neq 1})$, ..., $(\mathcal{C}_{f_n}, (\$_{nj})_{j\neq n})$ 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 $\mathcal{C}_{f_i}'s$ outputs form a (n,d)-sharing of $f(Z)$.*
- *for $i \neq j$, $\mathcal{C}_{f_i}$ leaks independently to $\mathcal{C}_{f_j}$.*

The following proposition states on the security of our construction against glitches attacks[1].

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

In the next section we recall the basics about Secure Multi-party Computation and, in particular, a SMC 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 $2d$.

3 Secure Multi-party Computation

Secure Multi-Party Computation represents a rich area of research initiated by the seminal work of Yao in 1986 [28]. 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 purpose of the attack is to learn the private information of the – non-colluding – honest players or to cause the computation to be incorrect. As a result, there are two important requirements of a multi-party computation protocol: correctness and privacy. Those two requirements relate to two different kinds of adversaries. 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 adversary kind, called *passive* or *semi-honest*, is only allowed to create collusion of players to gain

[1] A sketch of proof can be found in the extended version of this paper [21].

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 1988.

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

In a seminal paper [26], Shamir has introduced a simple and elegant way to share a secret Z (considered here in $\mathrm{GF}(2^m)$) 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 $P_Z(X) \in \mathrm{GF}(2^m)[X]$ 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 $\mathrm{GF}(2^m)$, 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. $F(Z_1, \cdots, Z_n) = \sum_{i=1}^{n} Z_i \prod_{k=1,k\neq i}^{n} -\alpha_k(\alpha_i - \alpha_k)^{-1}$ (due to Lagrange's Interpolation, $F(Z_1, \cdots, Z_n)$ equals $P_Z(0)$ that is Z).

Remark 4. 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 $F(Z_1, \cdots, Z_n) = \sum_{i=1}^{n} \lambda_i Z_i$.

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 Theorem 1 (see [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 modeled 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 $\texttt{C}$ such a (univariate or bivariate) computation. BGW's protocol ensures that the intermediate states V and V' at input of a bivariate operation $\texttt{C}$ 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 $\texttt{C}$ to process, BGW's protocol is designed such that each player I_i has either a single share V_i (if $\texttt{C}$ is univariate) or a single pair of shares (V_i, V'_i) (if $\texttt{C}$ is bivariate). Eventually, BGW's protocol describes a d-private multi-party computation for each kind of operation $\texttt{C}$ depending on its nature. We recall them hereafter.

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

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

If $\texttt{C}$ is the multiplication operation $\otimes$ applied on two shared intermediate states V and V', then the protocol is more complex than the previous ones. It involves the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the Vandermonde $(n \times n)$-matrix $(\alpha_i^j)_{1 \le i,j \le n}$ and it is composed of three steps[2]. Each player I_i:

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

The shares $\texttt{C}(V_i, V'_i)$ 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 of this polynomial is $\texttt{C}(V, V')$. However, the family $(\texttt{C}(V_i, V'_i))_i$ 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 $\texttt{C}(V, V)$ 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 $\texttt{C}(V_i, V'_i)$ 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

[2] 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 [6].

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 [6] for more details).

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. 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:

(a) 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.

(b) 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.

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 $\mathrm{GF}(2^m)$.

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

Algorithm1 . Secure Multiplication Part Dedicated to a Sub-Circuit $\mathcal{C}_{f_i}$, $(\$_{ij})$

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 points α_i, the first row $(\lambda_1, \cdots, \lambda_n)$ of the inverse of the matrix (α_i^j) with i and j lower than 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 $\mathrm{GF}(2^m)$
2. for $j = 1$ to $d$   do $a_j \leftarrow \mathtt{rand}(\mathrm{GF}(2^m))$
*** 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. Concerning the multiplication $\otimes$ over $\mathrm{GF}(2^m)$, 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 [24]. 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 [24].

Comparison with Other State of the Art Solutions. In [18], 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

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

Method	multiplications	additions	random bytes
This paper	$4d^3 + 8d^2 + 3d$	$4d^3 + 8d^2 + 7d + 2$	$d(2d + 1)$
[24]	$2d^2 + 2d$	$d^2 + d + 1$	$d(d + 1)/2$

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 can not 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.* [15] discuss the difficulty of applying Nikova *et al.* 's scheme to the AES s-box. In [19], the scheme has been applied to Noekeon. 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.

On the opposite side, the constructions proposed in [9] (operations in GF(2)) and in its extension [24] (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 [24] (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.

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 nonlinear layer `SubBytes` which applies the same *substitution-box* (s-box) to every byte of the internal state. The s-box S

is defined as the left-composition of an affine transformation Γ_A over GF(256) with the power function $x \mapsto x^{254}$ over the field GF(256). In the following, we propose a (n,d)-multi-party circuit $(\mathcal{C}_{f_1}, \cdots, \mathcal{C}_{f_n})$ implementing the power function. The secure implementation of the full AES-128 is not detailed here, but it can be straightforwardly deduced from the algorithms presented in this section and in the previous section.

As shown in [24], the exponentiation $x \mapsto x^{254}$ can be processed thanks to a chain of operations composed of raisings to powers in the form 2^j (which are linear over GF(256)) and 4 field multiplications. For any j, let us denote by η_j the power function $x \mapsto x^{2^j}$, the exponentiation algorithm proposed in [24] is recalled hereafter:

Algorithm2 . Exponentiation to the 254

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

```
1. $Z \leftarrow \eta_1(V)$                 $[Z = V^2]$
2. $Y \leftarrow Z \otimes V$               $[Y = V^2V = V^3]$
3. $W \leftarrow \eta_2(Y)$                 $[W = (V^3)^4 = V^{12}]$
4. $Y \leftarrow Y \otimes W$               $[Y = V^3V^{12} = V^{15}]$
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 $(\mathcal{C}_{f_i}, (\$_{ij})_{j \neq i})$.

Algorithm3 . Secure S-box 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 $P_V(\alpha_i)$ 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 (α_i^j) with i and j lower than n.

```
1. do $P_Z(\alpha_i) \leftarrow \eta_1(P_V(\alpha_i))$                                              $[Z = \eta_1(V)]$
2. do $P_Y(\alpha_i) \leftarrow$ Algorithm 1$(P_V(\alpha_i), P_Z(\alpha_i), (\$_{ij})_{j \neq i})$    $[Y = V \otimes Z]$
3. do $P_W(\alpha_i) \leftarrow \eta_2(P_Y(\alpha_i))$                                              $[W = \eta_2(Y)]$
4. do $P_Y(\alpha_i) \leftarrow$ Algorithm 1$(P_Y(\alpha_i), P_W(\alpha_i), (\$_{ij})_{j \neq i})$    $[Y = Y \otimes W]$
5. do $P_Y(\alpha_i) \leftarrow \eta_4(P_Y(\alpha_i))$                                              $[Y = \eta_4(Y)]$
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 $P_Y(\alpha_i) \leftarrow$ Algorithm 1$(P_Y(\alpha_i), P_Z(\alpha_i), (\$_{ij})_{j \neq i})$    $[Y = Y \otimes Z]$
8. do $P_V(\alpha_i) \leftarrow \Gamma_A(P_Y(\alpha_i))$                                             $[Y = \Gamma_A(Y)]$
```

5 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 and 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 build a d-glitches free AES-128 implementation. As a side effect of basing our security on 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 the proposed scheme when both active and passive attacks must be thwarted by the implementation. In addition, our work, together with the recent analysis [7] of Shamir's secret sharing scheme conducted in the context of SCA, shows that this sharing is a valuable alternative to the classical Boolean masking. It indeed not only enables to define glitches-free implementations, but it is also intrinsically more resistant against higher-order SCA (see [7,21] for an argumentation of this point). We based our study on very strong hypothesis 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 [5] that is based on efficient parallel computations.

References

1. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC 1988: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 1–10. ACM, New York (1988)
2. Blömer, J., Guajardo, J., Krummel, V.: Provably Secure Masking of AES. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 69–83. Springer, Heidelberg (2004)
3. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)
4. Coron, J.-S.: A New DPA Countermeasure Based on Permutation Tables. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 278–292. Springer, Heidelberg (2008)
5. Franklin, M., Yung, M.: Communication complexity of secure computation (extended abstract). In: STOC 1992: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing, pp. 699–710. ACM, New York (1992)

6. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In: PODC, pp. 101–111 (1998)
7. Goubin, L., Martinelli, A.: Protecting AES with Shamir's Secret Sharing Scheme. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 63–78. Springer, Heidelberg (2011)
8. Goubin, L., Patarin, J.: DES and Differential Power Analysis – The Duplication Method. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999)
9. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
10. Joye, M., Paillier, P., Schoenmakers, B.: On Second-order Differential Power Analysis. In: Rao and Sunar [22], pp. 293–308
11. Kocher, P., Jaffe, J., Jun, B.: Introduction to Differential Power Analysis and Related Attacks. Technical report, Cryptography Research Inc. (1998)
12. Mangard, S., Popp, T., Gammel, B.M.: Side-Channel Leakage of Masked CMOS Gates. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 351–365. Springer, Heidelberg (2005)
13. Mangard, S., Schramm, K.: Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 76–90. Springer, Heidelberg (2006)
14. Messerges, T.S.: Using Second-Order Power Analysis to Attack DPA Resistant Software. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000)
15. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the limits: A very compact and a threshold implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011)
16. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 529–545. Springer, Heidelberg (2006)
17. Nikova, S., Rijmen, V., Schläffer, M.: Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 218–234. Springer, Heidelberg (2009)
18. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of non-linear functions in the presence of glitches. Technical report, VAMPIRE II (2010)
19. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptology 24(2), 292–321 (2011)
20. Prouff, E., Roche, T.: Attack on a higher-order masking of the AES based on homographic functions. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 262–281. Springer, Heidelberg (2010)
21. Prouff, E., Roche, T.: Higher-Order Glitches Free Implementation of the AES using Secure Multi-Party Computation Protocols. Cryptology ePrint Archive (to appear, 2011), `http://eprint.iacr.org/`
22. Rao, J.R., Sunar, B. (eds.): CHES 2005. LNCS, vol. 3659. Springer, Heidelberg (2005)
23. Rivain, M., Dottax, E., Prouff, E.: Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. Cryptology ePrint Archive, Report 2008/021 (2008), `http://eprint.iacr.org/`

24. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
25. Schramm, K., Paar, C.: Higher Order Masking of the AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 208–225. Springer, Heidelberg (2006)
26. Shamir, A.: How to Share a Secret. Commun. ACM 22(11), 612–613 (1979)
27. Suzuki, D., Saeki, M., Ichikawa, T.: DPA Leakage Models for CMOS Logic Circuits. In: Rao and Sunar [22], pp. 366–382
28. Yao, A.C.-C.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science, pp. 162–167. IEEE Computer Society, Washington, DC, USA (1986)