Monomial Evaluation of Polynomial Functions Protected by Threshold Implementations

Simon Landry, Yanis Linge, Emmanuel Prouff

Monomial Evaluation of Polynomial Functions Protected by Threshold Implementations

–With an Illustration on AES –

Simon Landry[1,2], Yanis Linge[1], and Emmanuel Prouff[2,3]

[1] STMicroelectronics, Zone Industrielle, 190 Avenue Coq, 13106 Rousset, France,
{simon.landry,yanis.linge}@st.com
[2] Sorbonne Universités, UPMC Univ Paris 06, CNRS, INRIA, Laboratoire d'Informatique de Paris 6 (LIP6), Équipe PolSys, 4 place Jussieu, 75252 Paris, France
[3] ANSSI, France, emmanuel.prouff@ssi.gouv.fr

Abstract. In the context of side-channel countermeasures, threshold implementations (TI) have been introduced in 2006 by Nikova *et al.* to defeat attacks in presence of hardware effects called *glitches*. On several aspects, TI may be seen as an extension of another classical side-channel countermeasure, called *masking*, which is essentially based on the sharing of any internal state of the processing into independent parts (or shares). Among the properties of TI, uniform distribution of input and output shares is generally the most complicated to satisfy. Usually, this property is achieved by generating fresh randomness throughout the execution of the protected algorithm (*e.g.* the AES block cipher). In this paper, we combine the *changing of the guards* technique published by Daemen at CHES 2017 (which reduces the need for fresh randomness) with the work of Genelle *et al.* at CHES 2011 (which combines Boolean masking and multiplicative one) to propose a new TI without fresh randomness well suited to Substitution-Permutation Networks. As an illustration, we develop our proposal for the AES block cipher, and more specifically its non-linear part implemented thanks to a field inversion. In this particular context, we argue that our proposal is a valuable alternative to the state of the art solutions. More generally, it has the advantage of being easily applicable to the evaluation of any polynomial function, which was usually not the case of previous solutions.

Keywords: SCA · Threshold implementations · AES · Masking · Sharing · Secure Polynomial Evaluation.

1 Introduction

1.1 Problematic and State Of the Art

Introduced by Kocher [23] in the late nineties, *Side Channel Analysis (SCA)* is a serious threat in our connected world. Cryptography indeed evolves in a hostile environment, where an attacker can obtain information that leak from algorithms running on embedded devices. This information relate to *intermediate variables* and some of them are *sensitive* in the sense that they are linked to the secret key and can be exploited to mount a key-recovery attack [5].

The SCA threat [23] led to the increasing need for countermeasures. A solution is provided by the so-called *secret sharing* ([34], [9], [20]). In the context of SCA, we talk about *masking* [23].

The idea is to randomly split the sensitive variable into $(d+1)$ shares. The number d is refered as the *masking order*. A d^{th}-order masking can be broken by a $(d+1)^{th}$-order SCA, namely an adversary that targets $d+1$ intermediate variables at the same time [30] [27] [36]. However, masking countermeasure is vulnerable to *glitches* when the algorithm is embedded on a device. Glitches refer here switching variations of logic gates that are caused by interconnection delays between two consecutive register updates. Due to glitches, it has been shown in [26] that an attack can be mounted by exploiting side-channel information leakage at the output of some logic gates. In 2006, Nikova *et. al.* [29] proposed a special case of masking, called *threshold implementations (TI)* such that the security holds even in the presence of glitches. This countermeasure combines *secret sharing* and *multi-party computation (MPC)*. MPC creates methods for parties to jointly compute a public function over their inputs while keeping those inputs private (see [40] and [2] for more details).

By property of threshold implementations, it is known that at least 3 shares per internal state element are required to achieve security against first-order attacks in the presence of glitches [29]. The construction of such an implementation (with 3 shares only) for the AES algorithm has been an important research topic [28], [3], [10], [21]. The first difficulty is to achieve uniformity efficiently. The common technique to fix this issue is called *remasking*. It consists in adding fresh randomness to the implementation during the execution. However, remasking is very time and space consuming since the random bits have to be generated and stored in registers. To fix this constraint, Daemen proposed in [12] a technique called *changing of the guards* to assure uniformity of a function without fresh randomness (see its description in the extended version of this paper [24]). Based on this technique, Sugawara presented in [37] a provably 3-sharing uniform TI of the Canright's AES Sbox [6] without remasking. Then, the second challenge is the efficiency, and more precisely the number of cycles that are required to implement a threshold implementation of the AES Sbox resistant against first-order SCA attacks.

1.2 Our Contributions

In this paper, we propose a 3-sharing TI scheme that can be applied to secure the processing of any monomial. Our technique can be extended to secure the evaluation of polynomial functions in a generic manner and the proposed scheme is proved to be resistant against first-order SCA in the presence of glitches. In the latter case, the number of cycles taken by the secure evaluation is independent of the support of the polynomial while the number of gates and registers essentially increases linearly. Eventually, the design principles can be extended to higher-order security. As an illustration, we apply our method on the AES algorithm to design a 3-sharing TI of the AES Sbox without fresh randomness. It is an alternative to Sugawara's work [37].[4]

1.3 Overview of the paper

In Section 2 we first introduce notations related to the masking schemes in finite fields that we use in our paper. We present then the threshold implementations, the related security models and their properties. Based on [12], we also describe in the extended version of this paper [24] another technique than remasking to get uniformity in a construction. In Section 3 we present our generic

[4] In the extended version of this paper [24], we give a second possible scheme of a 3-sharing TI of the AES Sbox without fresh randomness. It takes two more cycles than our main proposal but it does not require pre-processing and additionnal memory space.

TI for secure evaluation of polynomial functions in $GF(2^n)$ and we argue that it is resistant against first-order SCA in the presence of glitches. Our proposal is built upon a primitive that must securely process the function mapping $x \in GF(2^n)$ into 1 if $x = 0$ and into 0 otherwise. In Section 4, we describe a 3-TI implementation of this primitive called here *Dirac function*. We argue that it is secure against first-order SCA in presence of glitches. For completeness, we present in the extended version of this paper [24] another original TI construction of the Dirac function, based on Sugawara's work [37] and on Ishaï *et. al.* multiplications [22]. The proofs of our lemmas and properties are given in the extended version of this paper [24]. We give an application of our technique to the AES algorithm in Section 5. Our first-order TI-masked 3-sharing implementation of the AES SBox is argued to be secure against SCA attacks in the presence of glitches. Finally, in Section 6, we discuss the performances of our construction compared to the prior state of the art.

2 Preliminaries

2.1 Basics on Sharing

In this paper, we discuss on the sharing of both field elements defined in $GF(2^n)$ for some n and (n, m)-functions defined from $GF(2^n)$ into $GF(2^m)$ for some n and m.

Sharing of Internal Variables. When masking is applied to secure block cipher implementations, each sensitive variable x occurring during the computation is split into d random shares following a chosen group operation. We define the tuple $\mathbf{x} = (x_1, \ldots, x_d)$ as the d-sharing of x while the x_i's are called the masks. Also, the value of d plays the role of a secure parameter. We also introduce the function $s()$ that recovers x from its d-sharing $\mathbf{x}$ and will be refered as the *reconstruction function*. It is defined by $s(\mathbf{x}) = x$. For example, if the group operation is the addition $\oplus$ in $GF(2^n)$, we have:

$$x = x_1 \oplus x_2 \oplus ... \oplus x_d \qquad \textit{and} \qquad s(\mathbf{x}) = \bigoplus_{i=1}^{d} x_i = x.$$

In this paper, we shall work with three types of masking for $d = 3$. We shall refer to the above example as a 3-*Boolean sharing*. If the group operation is the multiplication $\otimes$ in $GF(2^n)^\star$, we shall refer to it as a 3-*multiplicative sharing* [1], [4]. To mask additive functions we usually use a Boolean masking. Since the propagation of Boolean masks through a multiplication is tricky to deal with, the multiplicative masking is more suitable to mask multiplicative functions (as *e.g.* power functions). Eventually, by combining a Boolean with a multiplicative one, we obtain a 3-*affine sharing* that we describe in the following definition.

Definition 1. ***(3-affine sharing) [15].*** *Let $x, \tilde{x}, \beta \in GF(2^n)$ and $\alpha \in GF(2^n)^\star$ be four elements such that $x = (\tilde{x} \otimes \alpha^{-1}) \oplus \beta$. Then, $\mathbf{x} = (\tilde{x}, \alpha, \beta)$ is called 3-affine sharing of x, and β (resp. α) is called the Boolean mask (resp. multiplicative mask).*

Sharing of Functions. In the sequel, we shall express the input of a function F by a small letter x and its corresponding output by a large letter X. A mapping from x to X can be defined by $F(x) = X$.

A mask realization of an (n, m)-function F is a vector $\mathbf{F} = (F_1, ..., F_d)$ of (dn, m)-functions. The notion of reconstruction function can then be extended to mask realizations by setting:

$$s(\mathbf{F(x)}) = F(x) = X$$

2.2 Basic Notions

To formally describe the security notions involved in this paper and to explain the different countermeasures, it is classical to use an abstraction of the implementation called *circuit*. We recall hereafter the definition of circuit given in [35].

Definition 2. *[35]* ***(Circuit** C_F & **wire.)*** *Let F be a function and let $\mathcal{O}$ be a set of elementary operations. An ideal circuit C_F implementing F thanks to operations in $\mathcal{O}$ is an oriented graph where each cell c_i defines an element of $\mathcal{O}$ and each edge corresponds to an intermediate value V_i that is called a wire. V_i corresponds to an output to the operation c_i and an input of the operation c_j.*

The performance of a circuit can be measured in terms of *registers* and *clock cycles* that we define as follows.

Definition 3. ***(Register** & **clock cycle).*** *A register is a circuit component which can store one wire V_i. A set of more than one register is called a registers layer. A clock cycle corresponds to the longest path between a state register A to the next state register B.*

In our context, the set $\mathcal{O}$ contains field operations $\oplus, \otimes$ and the inverse multiplication $\otimes^{-1}$ in field of characteristic 2, and more generally any power function. The capabilities of a SCA attacker against a circuit C_F are usually defined by adversary models. Among them, the *Probing Adversary Model* [22] is the most popular one. We give its formal definition as follows.

Definition 4. ***(t^{th}-order Probing Adversary Model).*** *[35] Let C_F be a circuit composed with $(V_i)_{i\in I}$ wires and let t be a positive integer. Let $\mathcal{L}$ be a set of noisy leakage functions. A t^{th}-order Probing Adversary against C_F is an adversary that can choose a subset J of I with $\#J = t$ and can observe the random variable $(\mathcal{L}(V_j))_{j\in J}$ where $(\mathcal{L}_j(.))_j$ is a t-tuple of functions in $\mathcal{L}$.*

An attack performed by the adversary in Definition 4 conducts to the notion of t^{th}*-order probing security* of a circuit. We give formal definition hereafter.

Definition 5. ***(t^{th}-order probing security).*** *[8] A circuit C_F is said to be t^{th}-order probing secure if for a d-sharing $(x_1, x_2, ..., x_d)$ of some variable x in input, a t^{th}-order Probing Adversary against C_F does not observe a dependence between t (or less) wires and x.*

An algorithm achieving t-probing security is resistant to the class of t^{th}-order side-channel attacks. Some masking schemes have been proposed with formal security proof in the Probing Adversary Model [22], [33], [32]. However, Mangard *et. al.* [25] showed the vulnerability of this model when the circuit evolves in the presence of glitches. Glitches occur because the signals of a combinational circuit can switch more than once per clock cycle if an input changes. The reason why most masking schemes can be attacked is that they process on the same wire masks and masked values. Since there is a link between them, the power consumption is not independent of the masks and masked values. Then the t^{th}-Probing Adversary Model is no longer suffisant for security.

The first provably secure masking scheme with resistance to glitches was the threshold implementation scheme of Nikova *et. al.* [29]. Then, based on the work in [38], Roche and Prouff [35] introduced the t^{th}-order Glitches Adversary Model to formalize the security of masked hardware implementations in the presence of glitches. We call this new adversary model the t^{th}*-order TI*

Adversary Model and describe it in Definition 6.

In the presence of glitches, the ideal circuit without signal propagation delay (Definition 2) can be extended to a more realistic circuit wherein a transient hazard is generated due to the delay ΔT between two elementary operations (*i.e.* logic gate transitions during a cycle between two registers updates). For a circuit C_F the internal state at *a time* T for a circuit C_F refers to all the values taken by the V_i's at time T. It is denoted by $C_F(T)$.

Definition 6. ***(t^{th}-order TI Adversary Model).*** *[35] Let C_F be a circuit and let t be a positive integer. Let $\mathcal{L}$ be a set of leakage functions. A t^{th}-order TI Adversary against C_F is an adversary that can choose t times $T_1, T_2, ..., T_t$ and can observe the internal state transition at the t selected times $(L_i(C_F(T_i)))_{i\leq t}$ where, for each $i \leq t, L_i(.)$ is a function in $\mathcal{L}$.*

For example, in Figure 1, let C_F be a circuit implementing a function F from a state register A to a state register B, and let β, β' be two Boolean masks. We hereafter focus on the capabilities of an adversary looking at the wire V_3 and we denote by br a Gaussian independent noise. For the 1^{st}-order Probing Adversary Model of Definition 4, the attacker can observe noisy leakage functions $L_3(V_3) = x \oplus \beta' + br$ that gives no information on x. In comparison, for the 1^{st}-order TI Adversary Model of Definition 6, due to glitches between the two previous gates $\oplus$, it might exist a time T such that $L_3(V_3(T)) = x + br$. This can happen if the first Xor gate is transiently evaluated with $x \oplus \beta$ at first input and nothing at second input (for instance because the signal on β' takes more time to be delivered).

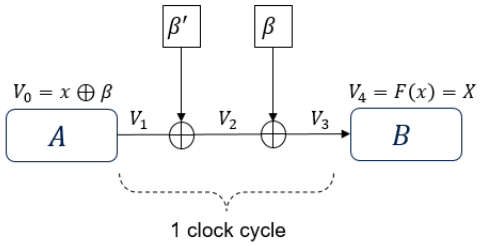

Fig. 1: Example of circuit for a 1^{st}-order Probing (or TI) Adversary Model

Finally, an attack performed by the adversary in Definition 6 leads to the notion of *t^{th}-order TI security* of a circuit. We give a formal definition hereafter.

Definition 7. ***(t^{th}-order TI security).*** *A circuit C_F is said to be t^{th}-order TI secure in the presence of glitches if for a d-sharing $(x_1, x_2, ..., x_d)$ of some variable x in input, a t^{th}-order TI Adversary against C_F does not observe a dependence between any set t wires and x.*

An algorithm achieving t-TI security implies a t-probing security (the converse is false). In the presence of glitches, a circuit C_F implementing a function F is split into several parts C_{F_i} such

that the observation of t or fewer parts gives no information on the original circuit input. Nikova *et. al.* [29] achieved this goal by combining secret sharing technique and secure MPC protocols. Their countermeasure, wich can be applied to any F, is called *threshold implementations* and was originally secure in the first-order TI Adversary Model.

To be first-order TI-secure, a TI must satisfy the three following properties.

Property 1. **(Correctness.)** Let a masked function $F : GF(2^{dn}) \mapsto GF(2^{dm})$ be a TI $\mathbf{F} = (F_1, ..., F_d)$. The TI $\mathbf{F}$ is correct if and only if $x = s(\mathbf{x})$ implies that $X = F(x) = s(\mathbf{F(x)})$.

This property ensures that the obtained output $\mathbf{F}(\mathbf{x})$ effectively corresponds to the sharing of the ouput of the initial input x by the function F. The reconstruction function s of the output X does not need to be necessarily the same than for the input x.

Property 2. **(Non-completness.)** A TI $\mathbf{F} = (F_1, ..., F_d)$ mapping a sharing $(x_1, ..., x_d)$ into a new sharing $(y_1, ..., y_d)$ is *non-complete* if for every $j \leqslant d$, there exists $i \neq j$ such that F_j is functionally independent of x_i (*i.e.* $F(x_1, ..., x_i, ..., x_d) = F(x_1, ..., x_{i-1}, x_{i+1}, ..., x_d)$).

Since we have insufficient knowledge from each F_j to reconstruct the secret x, this property guarantees security in the first-order TI Adversary Model.

Property 3. **(Uniformity.)** [7] For every $\mathbf{b} = (b_1, b_2, ..., b_d)$ in $GF(2^{dm})$, the number of $\mathbf{x}$ in $GF(2^{dn})$ for which $\mathbf{F}(\mathbf{x}) = \mathbf{b}$ is equal to $2^{(d-1)(n-m)}$ times the number of x in $GF(2^n)$ for which $F(x) = s(\mathbf{b})$.

This property ensures that, if the masking of the input to $\mathbf{F}$ is uniform, then the output of $\mathbf{F}$ is also a uniform masking of the output of F. It is important when the output of the TI is the input of another function. The design of TI achieving this property has been the core of many works and most of them were based on the idea of *remasking* where fresh randomness is used to uniformize the output sharing. In the extended version of this paper [24], we describe another technique to get uniformity within an implementation.

3 Our TI Generic Evaluation Technique

We describe in this section our main contribution, that is a TI-masked generic evaluation method that can be applied to any polynomial function $f(x) = \sum_{i=0}^{2^n-2} a_i x^i$ in $GF(2^n)$, contrary to the state of the art. According to our security model (see Definition 6), it is assumed that an attacker cannot see more than one intermediate result during the processing, but glitches are possible. Our construction is inspired from the ideas of affine masking and multiplicative masking, developped in [18] and [15]. However, to gain in execution time compared with [18], we try to minimize the number of conversions from Boolean masking to multiplicative masking and *vice-versa*. Through each cycle, the sensitive value is either masked thanks to a Boolean mask or a multiplicative mask, or both. Our construction is first-order TI-secure (*i.e.* resistant against first-order SCA in the presence of glitches). All the following lemmas and properties are proved in the extended version of this paper [24].

3.1 Our First-Order TI-secure Monomial Evaluation

To construct a first-order TI for any polynomial function defined in $GF(2^n)$, our generic proposal is essentially built upon a same masking scheme which will be separately applied to the evaluation of each monomial of f with non-zero coefficient. Let Q_{power} denotes one of those monomials and let us assume that is defined by $Q_{power}(x) = x^q$. In our proposed-scheme, the processing of Q_{power} is multiplicatively masked. Multiplicative masking needs to be implemented carefully not to be vulnerable to first-order SCA with a zero value power model, namely the *zero-problem* [19] [16]. In the litterature [14], it has been proposed to map the zero value in $GF(2^n)$ to a non-zero value in $GF(2^n)^\star$ using the *Dirac* function in order to take care of the zero-problem.

Definition 8. ***(Dirac function.)*** *The Dirac function is defined such that* $\delta(x) = 1$ *if* $x = 0$ *and* $\delta(x) = 0$ *otherwise.*

In order to compute $Q_{power}(x)$, we use the following property:

Property 4. For every integer q, we have:

$$(x \oplus \delta(x))^q = x^q \oplus \delta(x).$$

If we consider an input x of 8 bits, the Dirac function can be calculated by processing 7 AND gates on the bit-wise complemented bits of x such that:

$$\delta(x) = \overline{x_1} \otimes \overline{x_2} \otimes ... \otimes \overline{x_8}. \tag{1}$$

We propose another secure implementations of the Dirac function based on a lookup-table (LUT) in Section 4. We obtain a 3-Boolean sharing $(\delta_1, \delta_2, \delta_3)$ of $\delta(x)$. The advantage of this second implementation is that it can be executed in a single cycle (see Definition 3).

The full processing of our first-order TI-secure monomial evaluation can be done in 4 cycles. At input, the value x is assumed to be split into three parts following Definition 1. We denote the 3-affine sharing of x by $\mathbf{x} = ((x \oplus \beta) \otimes \alpha, \alpha, \beta)$ such that $s(\mathbf{x}) = (x \oplus \beta) \otimes \alpha \otimes \alpha^{-1} \oplus \beta = x$. Then, to get a multiplicative masking of $Q_{power}(x) = x^q$ we first remove the Boolean mask β of x and replace it by $\delta(x)$, while satisfying all the properties of TI. It can be achieved in two cycles. Due to Property 4, we obtain a new 2-multiplicative TI sharing of $x^q \oplus \delta(x)$: $((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q)$. We extend this sharing by adding a third share $(\beta \otimes \alpha^q)$ that will be used later. Finally, we obtain $\mathbf{x}_\delta = ((x^q \oplus \delta(x)) \otimes \alpha^q, \alpha^q, \beta \otimes \alpha^q)$.

Lemma 1. *The implementation of* $\mathbf{x}_\delta$ *is correct, non-complete and uniform. It is first-order TI-secure against SCA.*

Finally, the last two cycles allow us to replace in a secure way the Boolean mask $\delta(x)$ of x^q by a new one β'. We obtain at the end a new TI 3-affine sharing $\mathbf{xq} = ((x^q \oplus \beta') \otimes \alpha^q, \alpha^q, \beta')$ of x^q.

Lemma 2. *The implementation of* $\mathbf{xq}$ *is correct, non-complete and uniform. It is first-order TI-secure against SCA.*

The generalized scheme of our monomial evaluation is illustrated in Figure 2.

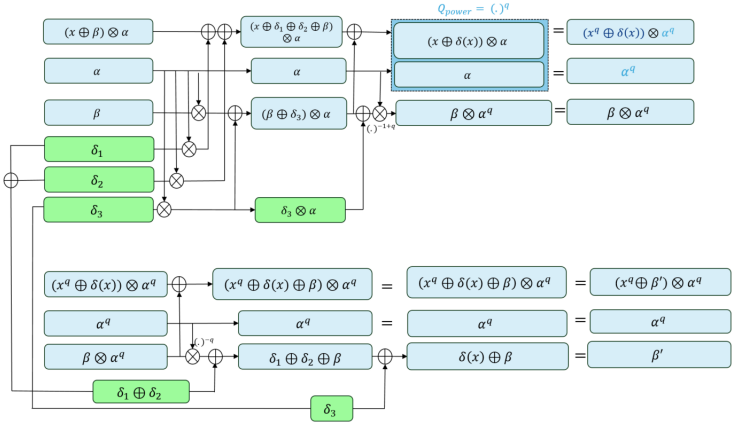

Fig. 2: First-order TI-masked Monomial Evaluation of x^q in 4 cycles

3.2 Extension of Our Technique for Any Polynomial Function

The concurrent application of the secure monomial evaluation described in previous section straightforwardly leads to a first-order secure TI for any polynomial function f. We describe hereafter the main steps of our proposal which, from a 3-sharing $(\tilde{x}, \alpha, \beta)$ of x, outputs a 3-sharing $(\tilde{f(x)}, \alpha_f, \beta_f)$ of $f(x)$ in 6 cycles:

- Step 1: firstly, a 3-Boolean sharing of the Dirac $\delta(x)$ is computed in 1 cycle from the 3-affine sharing of x (for details on the construction see Section 4). Note that this computation is done only once to secure the whole polynomial function evaluation.

- Step 2: secondly, each monomial of f can be evaluated thanks to our TI depicted in Figure 2. This step takes 4 cycles.

- Step 3: finally, the last step consists in combining the 3-affine sharings of all the monomial evaluations to get the 3-affine sharing $((f(x) \oplus \beta_f) \otimes \alpha_f, \alpha_f, \beta_f)$ of $f(x)$. It takes 1 cycle.

For example, let us assume that we want to securely process the 3-affine sharing of the function $f(x) = x^5 \oplus x^{33}$ from the 3-affine sharing $(\tilde{x}, \alpha, \beta)$ of x. With the (LUT) method described in Section 4, we first securely compute a 3-Boolean sharing of $\delta(x)$ in 1 cycle. Then we apply our scheme in Figure 2 to the monomials x^5 and x^{33}. These two computations can be done in parallel in 4 cycles and give two new 3-affine sharings of **x5** and **x33**. Finally, the last step takes 1 cycle and is split as follows:

- the multiplication of the second shares of both **x5** and **x33** to construct a new multiplicative mask $\alpha_f = \alpha^5 \otimes \alpha^{33}$.
- the xor between the third shares of **x5** and **x33** to get $\beta_f = \beta_{x^5} \oplus \beta_{x^{33}}$.
- the xor between the first share of **x5** multiplied by α^{33} and the first share of **x33** multiplied by α^5 to get $(x^5 \oplus x^{33} \oplus \beta_{x^5} \oplus \beta_{x^{33}}) \otimes (\alpha^5 \otimes \alpha^{33})$ and thus the new 3-affine sharing **f** $= ((f(x) \oplus \beta_f) \otimes \alpha_f, \alpha_f, \beta_f)$ of f.

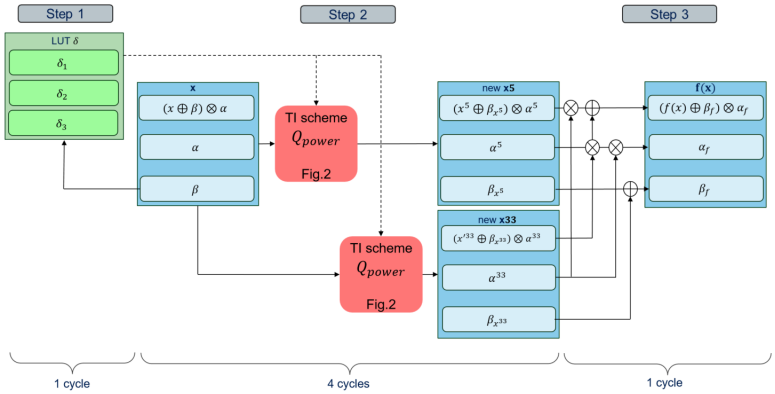

Fig. 3: First-order TI-masked Evaluation of $f(x) = x^5 \oplus x^{33}$

The whole example is depicted in Figure 3.

Remark. The secure monomials' evaluation is done in parallel, leading to an increase of the implementation area which is roughly linear in the number of monomials to evaluate. Using the ideas developed in [11], it is possible to improve the area complexity by using cyclotomic classes and the fact that some powers are linear (those in the form $x \mapsto x^{2^j}$).

The next section explains the implementation of the Dirac function used in step 1 of Figure 3.

4 Construction of the Dirac Function as a lookup Table (LUT)

In our first-order TI-secure evaluation of polynomial functions of the form $f(x) = \sum_{i=0}^{2^n-2} a_i x^i$, we need to securely compute a 3-Boolean sharing of the Dirac of x from the 3-affine sharing of x (see Step 1 in Figure 3 in Sect. 3). For such a purpose, we propose below to represent the Dirac function as a lookup table. We also propose an alternative secure construction in the extended version of this paper [24].

We recall that, for each 3-affine sharing $\mathbf{x}$ of x, we know the value of its corresponding multiplicative and Boolean masks α and β, respectively. A possible implementation for the 3-Boolean sharing $(\delta_1, \delta_2, \delta_3)$ of the Dirac $\delta(x)$ is to precompute a table T of 256 bits from a 3-affine sharing $(\tilde{x}, \alpha, \beta)$ of x. We first have to know the value of the mask β and the masked value $(x \oplus \beta)$, which is given by processing $\tilde{x} \otimes \alpha^{-1}$. Then, thanks to a random bit $r \in GF(2)$, the table is constructed such that $T[x \oplus \beta]$ takes the value $r + 1$ if $x = 0$ and r otherwise. During the execution of the AES algorithm, the mask r stays unchanged while the mask β evolves. In order to be in coherence with our second construction of the Dirac function in the extended version of this paper [24], we choose to split r into two parts such that $r = r_1 \oplus r_2$. Algorithm 1 describes the way how the LUT is created. This LUT is fully computed once for the first Dirac request and is then modified at each execution of the Dirac such that the values at positions β and β' are xored with 1, where β' denotes the next Boolean mask of the next value x' for which we would like to securely evaluate the Dirac function.

The modification of the table is explained in Algorithm 2. This processing, which takes one cycle can be done during other operations between two Q_{power} layers.

Hence, to obtain a 3-Boolean sharing of the Dirac value of x, we will store in our construction three bits $\delta_1 = \delta(x) \oplus r_1 \oplus r_2, \delta_2 = r_2$ and $\delta_3 = r_1$ (see Figure 2) and 256 bits for the LUT. To conclude, this computation takes one cycle and store 259 bits.

Algorithm 1 Compute once a 3-Boolean sharing of $\delta(x)$

Require: $r = r_1 \oplus r_2 \in GF(2), (x) \oplus \beta, \beta$
Ensure: $(\delta_1, \delta_2, \delta_3)$ s.t. $\delta_1 \oplus \delta_2 \oplus \delta_3 = \delta(x)$
  $T \leftarrow [0] * 256$
  $T[\beta] \leftarrow T[\beta] \oplus 1$
  $T \leftarrow T \oplus [r, ..., r]$
  return $T[(x) \oplus \beta], r_1, r_2$

Algorithm 2 Updating of the LUT T

Require: T, β, β'
Ensure: modified table T
  $T[\beta] \leftarrow T[\beta] \oplus 1$
  $T[\beta'] \leftarrow T[\beta'] \oplus 1$
  return T

To conclude, Sections 3 and 4 describe our generic first-order TI-secure polynomial evaluation. To be able to compare this proposal with the TI of the state of the art, we give an illustration of it on the well-studied AES algorithm in the next section.

5 An Illustration on the AES Algorithm

5.1 AES Algorithm

The AES block cipher [13] operates on an 4×4 array of 16 bytes called a *state*. An AES plaintext value is modified thanks to additive and multiplicative operations in order to obtain the corresponding ciphertext at the end of the encryption. Each round of the AES is composed of four stages: *AddRoundKey*, *SubBytes*, *ShiftRows* and *MixColumns*. The AES SBox is defined as the composition of an affine transformation called AT over $GF(2^8)$ and the multiplicative inverse Inv over the field $GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$. The *SubBytes* operation consists in applying the SBox on each byte of the state. Moreover, the last round ommits the MixColumns operation and add a final AddRoundKey stage. The processus is composed of either 10, 12 or 14 rounds, depending on the key size. Based on the secret key, a key expansion process defines all the round keys that are involved in the different rounds.

5.2 Strategy

Substitutions-Permutation Network (SPN) based block ciphers (as AES) design involve additive and multiplicative operations defined on a finite field. To efficiently mask the sensitive value through each operation, Genelle *et. al.* proposed a scheme to securely transform an additive sharing into a multiplicative one [18] (Boolean masking being dedicated to Boolean operations while multiplicative masking being dedicated to multiplicative operation). In order to reduce the number of cycles for our implementation, we improved their work by using a 3-affine sharing which is constructed such that it contains both Boolean and multiplicative masks. As the existing papers [28], [3], [10], [21] proposed a masked design of the AES Sbox processing, we describe in the followed sections a TI-masked implementation of this SPN algorithm.

Through each AES operation, we denote the different 3-affine sharings S_i as in Figure 4. All of them are obtained by the meaning of one or more intermediate 3-sharing $S_{i,j}$. Each $S_{i,j}$ represents a computation done in one cycle as defined in Definition 3. Moreover, our scheme is first-order TI-secure. The demonstration of this security is based on lemmas that are proved in the extended version of this paper [24].

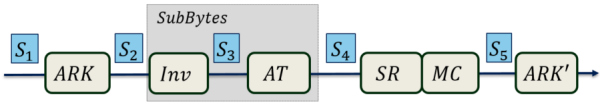

Fig. 4: 3-sharings S_i of our TI-masked AES

We first show how to apply our TI-masked monomial evaluation on the AES SBox. Then, we complete our presentation with a TI of all other steps of the AES.

5.3 Application of Our Monomial Evaluation Technique on the AES SBox

In this section we highlight our generic method by adapting it to the AES SBox processing. This step comes after the *AddRoundKey* operation and so takes the 3-affine sharing of $x \oplus k$, where x is the plaintext byte and k the round-key byte.

As discuss before, the SBox operation can be decomposed such that $SB(x) = AT(Inv(x))$, where AT represents the affine transformation and Inv the multiplicative inversion in $GF(2^8)$. In this case, this inversion is equivalent to our operation $Q_{power}(x)$ with $q = 254$ meaning $x^{254} = x^{-1}$ in $GF(2)[x]/(x^8 + x^4 + x^3 + x + 1)$.

To mask the *SubBytes* step, AT is masked additively whereas Inv denotes is multiplicatively protected. By implementing the Dirac function as a LUT, the masking processing of an AES SBox takes five cycles.[5] The scheme is illustrated in Figure 5. It gives the new following 3-multiplicative

[5] As an observation, the scheme can also be implemented in seven cycles if the Dirac function δ is computed in three cycles with TI ISW multiplications (see in the extended version of this paper [24].

sharing S_3 of $(x \oplus k)^{-1} \oplus \delta(x \oplus k)$ and the new 3-Boolean sharing S_4 of $SB(x \oplus k)$:

$$S_3 = ([(x \oplus k)^{-1} \oplus \delta(x \oplus k)] \otimes \alpha^{-1}, \alpha^{-1}, \beta \otimes \alpha^{-1}),$$
$$S_4 = (SB(x \oplus k) \oplus \beta', \alpha, \beta'),$$

with $\beta' = AT(\beta) \oplus AT(\delta(x \oplus k)) \oplus 0\text{x}63$.

Lemma 3. *The implementations of S_3 and S_4 are correct, non-complete and uniform. It is first-order TI-secure against SCA.*

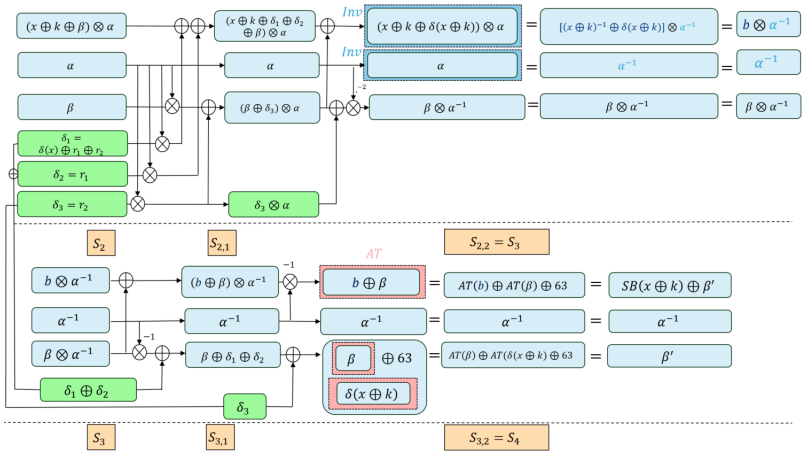

Fig. 5: First-order TI-Masked SubBytes Operation in 4 cycles

Remark. We made the choice to obtain a 3-Boolean sharing of $SB(x \oplus k)$ instead of a 3-affine sharing (as in our generic method in Figure 2) because it is more suitable for the input of the next AES operation, namely the *MixColumns* operation. This last is detailled in the next section with other AES steps.

5.4 Presentation of the Other TI-masked AES Operations

Step 1: AddRoundKey (ARK). At the beginning of the computation, the sixteen plaintext bytes x and the sixteen round keys k are splited into three parts following Definition 1. Let α be a multiplicative non-zero mask and let β_x and β_k be two Boolean masks for the plaintext and the round key, respectively. Without loss of generality, we define the first 3-affine sharing S_1 of one plaintext byte as:

$$(\tilde{x}, \alpha, \beta) = ((x \oplus \beta_x) \otimes \alpha, \alpha, \beta_x)$$

In the same way, the 3-affine sharing of one round key k is:

$$(\tilde{k}, \alpha, \beta) = ((k \oplus \beta_k) \otimes \alpha, \alpha, \beta_k)$$

Note that we choose the same multiplicative mask. Then, the ARK operation can be performed by xoring the first shares $\tilde{x}$ and $\tilde{k}$ in one hand and the third shares β_x and β_k on the other hand. We obtain in one cycle a new 3-sharing $S_2 = ((x \oplus k \oplus \beta) \otimes \alpha, \alpha, \beta = \beta_x \oplus \beta_k)$. This computation can be performed on all of the sixteen bytes in parallel.

Lemma 4. *The implementation of S_2 is correct, non-complete and uniform. It is first-order TI-secure against SCA.*

Step 2: SubBytes. This operation is detailled in Section 5.3.

Step 3: ShiftRows (SR). This operation is here combines with MixColumns operation and consists a shift on the left of $1, 2, 3$ bytes of the AES state. It is a reindexing of the state and it does not require any cycle.

Step 4: MixColumns (MC). Then, the MixColumns operation consit in a multiplication of each column of the AES state by the matrix MC (see $c(x)$ in Equation 3.12 in [13]). This step takes one cycle. The j^{th} line of MC is denoted by MC_j. Each byte x_i at input of the MC processing is represented as a 3-sharing $S_4 = (SB(x_i \oplus k) \oplus \beta'_i, \alpha^{-1}, \beta'_i)$. MixColumns operation will give new values y_i. Without loss of generality, we obtain the following new 3-sharing for the byte y_1: $S_5 = ((y_1 \oplus \beta'') \otimes \alpha', \alpha', \beta'')$, with $y_1 = MC_1 \otimes (SB(x_i \oplus k)_{1 \leq i \leq 4}), \alpha' = \alpha^{-1}$ and $\beta'' = MC_1 \otimes (\beta'_i)_{1 \leq i \leq 4}$.

Lemma 5. *The implementation of S_5 is correct, non-complete and uniform. It is first-order TI-secure against SCA.*

In this section, we obtain a first-order TI-secure version of the whole AES. In the next section, we compare performances of our SBox TI AES proposal regarding other TI propositions of the state of the art.

6 Performances of Our Proposition

6.1 Comparison of our proposal for AES SBox Regarding the Prior State of the Art

In order to compare our work with the state of the art regarding the number of cycle and the number of random bits per SBox, we will use [37]. Table 1 shows theses performances evaluation. Our proposal is better or equivalent in terms of number of cycles than [28], [3], [10], [21], and does not require fresh randomness. Compare to [37], our proposal takes one more cycle but it can be easily extended to evaluate any polynomial function as shown in Section 3. Moreover, our scheme is applicable to any cryptographic algorithm based on a power computation.

6.2 Performances of our proposal for the complete AES

As an illustration, our first-order TI-secure implementation can be adapted to the AES algorithm as shown in Section 5. In this case, the construction takes 70 cycles for each byte of the AES state. Table 2 gives the number of required cycles for each AES operation in bold characters in the last column. It also gives the number of cycles that are needed to compute a byte of the AES state through all the 10 rounds. Note that the *ShiftRows* operation is missing in the table because it is combined with the *MixColumns* operation. This number of cycles could be optimized by parallelizing some computation with respect to the non-completness property.

Design	Random bits/Sbox	Nb Cycles	generalizable
[28]	44	7	no
[3]	16	5	no
[10]	54	5	no
[21]	18	6	no
[37]	0	4	no
This work	0	5	yes

Table 1: Comparison of our proposal for AES SBox Regarding the Prior State of the Art

Operations	9 first rounds	Last round	Nb cycles
ARK	✓	✓(×2)	$\mathbf{1} \times 11$
SB	✓	✓	$\mathbf{5} \times 10$
MC	✓	×	$\mathbf{1} \times 9$
Total			70

Table 2: Number of cycles for a byte of an AES state during the whole TI-masked processing

7 Conclusion

In this paper, we have introduced a new threshold implementation allowing us to evaluate any power function in 6 cycles (while the area increases linearly with the number of powers which must be processed for the polynomial evaluation). It is provably first-order TI-secure against SCA in the presence of glitches. We have argued that this technique can be straightforwardly extended to securely process any polynomial function. Finally, we gave an illustration of our method to the AES SBox. In this case, we got a TI running in 5 cycles without fresh randomness. Our proposal is better or equivalent in terms of number of cycles than [28], [3], [10], [21], but does not require fresh randomness. Regarding [37], our proposal takes one more cycle but it can be easily extended to evaluate any polynomial function in a generic manner. For completeness, we also proposed a full TI-masked AES. This scheme is inspired by [17] and is based on affine masking. Our construction takes 1120 cycles for the whole AES computation and does not require fresh randomness.

As a future work, it will be interesting to see how our technique can be applied to secure other algorithms. For instance, we could evaluate our method on a sponge construction-based algorithm. Moreover, it will be also interesting to compare the area required to implement our technique and to experimentally validate our theoretical performance estimations. We could improve the AES scheme by parallelizing some computations with respect to the non-completness TI property.

References

1. Akkar, M., Giraud, C.: An implementation of DES and aes, secure against some attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2162, pp. 309–318. Springer (2001). https://doi.org/10.1007/3-540-44709-1_26, https://doi.org/10.1007/3-540-44709-1_26

2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 1–10. ACM (1988). https://doi.org/10.1145/62212.62213, https://doi.org/10.1145/62212.62213
3. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-offs for threshold implementations illustrated on AES. IEEE Trans. on CAD of Integrated Circuits and Systems **34**(7), 1188–1200 (2015). https://doi.org/10.1109/TCAD.2015.2419623, https://doi.org/10.1109/TCAD.2015.2419623
4. Blömer, J., Guajardo, J., Krummel, V.: Provably secure masking of AES. In: Handschuh, H., Hasan, M.A. (eds.) Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3357, pp. 69–83. Springer (2004). https://doi.org/10.1007/978-3-540-30564-4_5, https://doi.org/10.1007/978-3-540-30564-4_5
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004). https://doi.org/10.1007/978-3-540-28632-5_2, https://doi.org/10.1007/978-3-540-28632-5_2
6. Canright, D.: A very compact s-box for AES. In: Rao and Sunar [31], pp. 441–455. https://doi.org/10.1007/11545262_32, https://doi.org/10.1007/11545262_32
7. Carlet, C.: Boolean Functions for Cryptography and Error-Correcting Codes, p. 257–397. Encyclopedia of Mathematics and its Applications, Cambridge University Press (2010). https://doi.org/10.1017/CBO9780511780448.011
8. Carlet, C., Prouff, E., Rivain, M., Roche, T.: Algebraic decomposition for probing security. IACR Cryptology ePrint Archive **2016**, 321 (2016), http://eprint.iacr.org/2016/321
9. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener [39], pp. 398–412. https://doi.org/10.1007/3-540-48405-1_26, https://link.springer.com/content/pdf/10.1007%2F3-540-48405-1_26.pdf
10. Cnudde, T.D., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with d+1 shares in hardware. In: Bilgin, B., Nikova, S., Rijmen, V. (eds.) Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016. p. 43. ACM (2016). https://doi.org/10.1145/2996366.2996428, https://doi.org/10.1145/2996366.2996428
11. Coron, J., Roy, A., Vivek, S.: Fast evaluation of polynomials over binary finite fields and application to side-channel countermeasures. J. Cryptographic Engineering **5**(2), 73–83 (2015). https://doi.org/10.1007/s13389-015-0099-9, https://doi.org/10.1007/s13389-015-0099-9
12. Daemen, J.: Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 137–153. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_7, https://doi.org/10.1007/978-3-319-66787-4_7
13. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer Verlag, Berlin, Heidelberg, New York (2002)
14. Damgård, I., Keller, M.: Secure multiparty AES. In: Sion, R. (ed.) Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6052, pp. 367–374. Springer (2010). https://doi.org/10.1007/978-3-642-14577-3_31, https://doi.org/10.1007/978-3-642-14577-3_31
15. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine masking against higher-order side channel analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6544, pp. 262–280. Springer (2010). https://doi.org/10.1007/978-3-642-19574-7_18, https://doi.org/10.1007/978-3-642-19574-7_18

16. Fumaroli, G., Mayer, E., Dubois, R.: First-order differential power analysis on the duplication method. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4859, pp. 210–223. Springer (2007). https://doi.org/10.1007/978-3-540-77026-8_16, https://doi.org/10.1007/978-3-540-77026-8_16
17. Genelle, L., Prouff, E., Quisquater, M.: Montgomery's trick and fast implementation of masked AES. In: Nitaj, A., Pointcheval, D. (eds.) Progress in Cryptology - AFRICACRYPT 2010 - 4th International Conference on Cryptology in Africa, Dakar, Senegal, July 5-7, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6737, pp. 153–169. Springer (2010). https://doi.org/10.1007/978-3-642-21969-6_10, https://doi.org/10.1007/978-3-642-21969-6_10
18. Genelle, L., Prouff, E., Quisquater, M.: Thwarting higher-order side channel analysis with additive and multiplicative maskings. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6917, pp. 240–255. Springer (2011). https://doi.org/10.1007/978-3-642-23951-9_16, https://doi.org/10.1007/978-3-642-23951-9_16
19. Golic, J.D., Tymen, C.: Multiplicative masking and power analysis of AES. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 198–212. Springer (2002). https://doi.org/10.1007/3-540-36400-5_16, https://doi.org/10.1007/3-540-36400-5_16
20. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1717, pp. 158–172. Springer (1999). https://doi.org/10.1007/3-540-48059-5_15, https://doi.org/10.1007/3-540-48059-5_15
21. Groß, H., Mangard, S., Korak, T.: An efficient side-channel protected AES implementation with arbitrary protection order. In: Handschuh, H. (ed.) Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10159, pp. 95–112. Springer (2017). https://doi.org/10.1007/978-3-319-52153-4_6, https://doi.org/10.1007/978-3-319-52153-4_6
22. Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_27, https://doi.org/10.1007/978-3-540-45146-4_27
23. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener [39], pp. 388–397. https://doi.org/10.1007/3-540-48405-1_25, https://doi.org/10.1007/3-540-48405-1_25
24. Landry, S., Linge, Y., Prouff, E.: Monomial Evaluation of Polynomial Functions Protected by Threshold Implementations – With an Illustration on AES – – Extended Version –, to appear.
25. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked CMOS gates. In: Menezes, A. (ed.) Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3376, pp. 351–365. Springer (2005). https://doi.org/10.1007/978-3-540-30574-3_24, https://doi.org/10.1007/978-3-540-30574-3_24
26. Mangard, S., Pramstaller, N., Oswald, E.: Successfully attacking masked AES hardware implementations. In: Rao and Sunar [31], pp. 157–171. https://doi.org/10.1007/11545262_12, https://doi.org/10.1007/11545262_12
27. Messerges, T.S.: Using second-order power analysis to attack DPA resistant software. In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings. Lecture Notes in Computer Science,

vol. 1965, pp. 238–251. Springer (2000). https://doi.org/10.1007/3-540-44499-8_19, https://doi.org/10.1007/3-540-44499-8_19
28. 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.) Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 69–88. Springer (2011). https://doi.org/10.1007/978-3-642-20465-4_6, https://doi.org/10.1007/978-3-642-20465-4_6
29. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptology **24**(2), 292–321 (2011). https://doi.org/10.1007/s00145-010-9085-7, https://doi.org/10.1007/s00145-010-9085-7
30. Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Trans. Computers **58**(6), 799–811 (2009). https://doi.org/10.1109/TC.2009.15, https://doi.org/10.1109/TC.2009.15
31. Rao, J.R., Sunar, B. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings, Lecture Notes in Computer Science, vol. 3659. Springer (2005). https://doi.org/10.1007/11545262, https://doi.org/10.1007/11545262
32. Rivain, M., Dottax, E., Prouff, E.: Block ciphers implementations provably secure against second order side channel analysis. In: Nyberg, K. (ed.) Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5086, pp. 127–143. Springer (2008). https://doi.org/10.1007/978-3-540-71039-4_8, https://doi.org/10.1007/978-3-540-71039-4_8
33. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6225, pp. 413–427. Springer (2010). https://doi.org/10.1007/978-3-642-15031-9_28, https://doi.org/10.1007/978-3-642-15031-9_28
34. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2248, pp. 552–565. Springer (2001). https://doi.org/10.1007/3-540-45682-1_32, https://doi.org/10.1007/3-540-45682-1_32
35. Roche, T., Prouff, E.: Higher-order glitch free implementation of the AES using secure multi-party computation protocols - extended version. J. Cryptographic Engineering **2**(2), 111–127 (2012). https://doi.org/10.1007/s13389-012-0033-3, https://doi.org/10.1007/s13389-012-0033-3
36. Schramm, K., Paar, C.: Higher order masking of the AES. In: Pointcheval, D. (ed.) Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3860, pp. 208–225. Springer (2006). https://doi.org/10.1007/11605805_14, https://doi.org/10.1007/11605805_14
37. Sugawara, T.: 3-share threshold implementation of AES s-box without fresh randomness. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 123–145 (2019). https://doi.org/10.13154/tches.v2019.i1.123-145, https://doi.org/10.13154/tches.v2019.i1.123-145
38. Suzuki, D., Saeki, M., Ichikawa, T.: DPA leakage models for CMOS logic circuits. In: Rao and Sunar [31], pp. 366–382. https://doi.org/10.1007/11545262_27, https://doi.org/10.1007/11545262_27
39. Wiener, M.J. (ed.): Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, Lecture Notes in Computer Science, vol. 1666. Springer (1999). https://doi.org/10.1007/3-540-48405-1, https://doi.org/10.1007/3-540-48405-1
40. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986. pp. 162–167. IEEE Computer Society (1986). https://doi.org/10.1109/SFCS.1986.25, https://doi.org/10.1109/SFCS.1986.25