Threshold Implementations in the Robust Probing Model

Siemen Dhooghe
Svetla Nikova
Vincent Rijmen
firstname.lastname@esat.kuleuven.be
imec-COSIC, KU Leuven, Belgium

ABSTRACT

Threshold Implementations (TI) are provably secure algorithmic countermeasures against side-channel attacks in the form of differential power analysis. The strength of TI lies in its minimal algorithmic requirements. These requirements have been studied over more than 10 years and many efficient implementations for symmetric primitives have been proposed. Thus, over the years the practice of protecting implementations matured, however, the theory behind threshold implementations remained the same. In this work, we revise this theory by looking at the properties of correctness, non-completeness, and uniformity as a composable security model. We prove that this model provides first-order and higher-order univariate security in the glitch-robust probing model which lets us expand the theoretic framework of TI. We first provide a link between uniformity and the notion of non-interference, a known composable security notion building out the probing model. We then relax the notion of non-completeness which helps the design of secure expansion and compression functions. Lastly, we provide generalisations of the threshold notions to allow for general secret sharing schemes and provide examples of how different sharing schemes affect the security and efficiency of the countermeasure.

KEYWORDS

DPA; Masking; Security Proofs; Threshold Implementations

ACM Reference Format:
Siemen Dhooghe, Svetla Nikova, and Vincent Rijmen. 2019. Threshold Implementations in the Robust Probing Model. In *Theory of Implementation Security Workshop (TIS'19), November 11, 2019, London, United Kingdom.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3338467.3358949

1 INTRODUCTION

In 2006, Nikova *et al.* published the work on threshold implementations [16]. These implementations consist of a cascaded circuit where each stage represents a Boolean function which works over secret shared values. The work shows that if these Boolean functions attain three core properties, *correctness, non-completeness, and uniformity*, the hardware implementation is secured against any first-order Differential Power Analysis (DPA) attack [14]. Threshold implementations have become a popular countermeasure as it is easy to apply for designers and yet requires minimal overhead on the algorithm, for example it does not require an online random number generator. As a result, the methodology was used to secure several symmetric primitives [5, 12, 15, 18, 19].

In this work, we revise the security arguments of the original work on threshold implementations and expand its theoretical framework. We provide the following four contributions in the order they appear in the work.

- **Robust Probing Security.** We show that a cascaded circuit consisting of correct, non-complete, and uniform functions is secure against a first-order glitch-robust probing adversary placing the threshold circuit model as a composable security notion which builds out the probing model. We continue by showing that the higher-order non-completeness requirement from [2] leads to higher-order univariate probing security. Additionally, we discuss how the injection of fresh randomness affects the security of a threshold design.
- **Linking Non-interference.** We take the composable security notion Non-Interference (NI) from Barthe *et al.* [1] and show that its first-order variant implies uniformity. As a result, functions secure in the first-order NI model can be composed with threshold functions and be guaranteed of their security.
- **Relaxing Non-completeness.** The proof of probing security shows that we can relax the property of non-completeness to first-order robust probing security where the notion of uniformity takes the role of ensuring the composition of multiple probing secure functions remains secure. This relaxation in turns allows for more efficient secure designs.
- **Allowing General Sharing Schemes.** We generalise the notions of correctness, non-completeness, and uniformity by allowing for multiple inputs/outputs and general secret sharing schemes. We then discuss how to evaluate the uniformity of a sharing which has an embedded linear code to protect against fault attacks and we give an example of a more efficient sharing scheme than Boolean masking when evaluating a bricklayer of S-boxes.

2 NOTATION AND PRELIMINARIES

2.1 Notation

We denote stochastic variables with upper case characters and share vectors in bold. We denote $Sh(x)$ as the set of all valid share vectors of an element x. Additionally, $\mathbf{x}_{\bar{i}}$ denotes a share vector without the i^{th} share. Finally, we denote s_x as the number of shares $(\mathbf{x}_1, ..., \mathbf{x}_{s_x})$ of the secret x.

TIS'19, November 11, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6827-8/19/11…$15.00
https://doi.org/10.1145/3338467.3358949

Throughout this work, we use the words "information", "learn", and "entropy" to more easily explain notions in an informal way. In formal definitions and proofs, probabilistic notions such as independence are used for their mathematical simplicity.

2.2 Boolean Masking

In order to defend algorithms against side-channel attacks a sound and widely deployed approach is the masking countermeasure which was introduced at the same time by Chari *et al.* [4] and by Goubin and Patarin [11]. The technique splits each key-dependent variable x in the algorithm into shares x_i such that $x = \sum_i x_i$ over a finite field $\mathbb{F}$. In case this field is binary, this masking method is referred to as Boolean masking. If no d shares give information on the secret we say that the masking scheme has passive threshold d.

2.3 Threshold Implementations

We go over the basic properties of threshold implementations as shown in [16].

The first property pertains to the secret sharing of a variable. We require that a sharing provides the expected threshold level, meaning that with $d+1$ Boolean shares the adversary does not get information on the sharing's secret with less than $d+1$ shares. This leads to the property of a uniform sharing.

Definition 2.1 (Uniform Masking). A masking $\mathbf{X}$ in s_x shares is uniform if for all $x \in \mathbb{F}$ we have

$$P(\mathbf{X} = \mathbf{x} \,|\, X = x) = \begin{cases} |\mathbb{F}|^{-s_x+1} & \text{if } \mathbf{x} \in Sh(x)\,, \\ 0 & \text{else}\,. \end{cases}$$

In words, each share vector has an equal chance to occur.

We aim to create an algorithm which works over shared variables instead of its secrets. As such, we denote $\mathbf{N}(\mathbf{x}) = \mathbf{y}$ as the shared function of $N(x) = y$. This shared function takes in the shares of x and gives back a sharing of N's outputs. We can thus consider a sharing of N as the component functions $\mathbf{f}_i$ taking in shares of x and giving a share of y as output. We first require that the shared function $\mathbf{N}$ gives a correct sharing of the output y, meaning that the sum of the output shares $\sum_{i=1}^{s_y} \mathbf{y}_i$ is equal to y. This leads to the correctness property.

Definition 2.2 (Correctness). Given x and $y = N(x)$, for each sharing $\mathbf{x} \in Sh(x)$ we have that the reconstruction of $\mathbf{y} = \mathbf{N}(\mathbf{x})$ is equal to y.

Due to the effect of glitches, a sample of a power trace can contain more information than of just one share and instead contains joint information on all gates viewing a glitch. More information and a descriptive example on the effect of glitches is found in [16]. To this effect harming the privacy of a secret shared algorithm, we need to carefully place registers and demand that each computation is made using "non-complete" information of a secret. This leads to the definition of non-completeness on the level of component functions.

Definition 2.3 (Non-completeness). A shared function $\mathbf{N}(\mathbf{x})$ is non-complete if each of its component functions $\mathbf{f}_i$ uses at most $s_x - 1$ input shares.

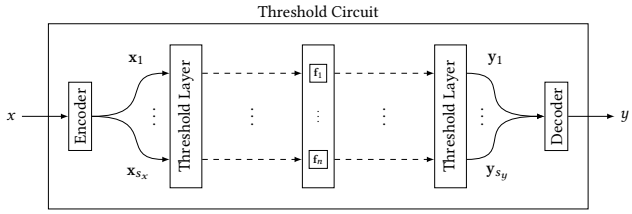

Figure 1: Representation of a threshold circuit.

Lastly, we require that all sharings in an algorithm contain enough entropy, meaning that each sharing is uniformly distributed as in Definition 2.1. Since we assume that the initial sharing of secrets introduces enough entropy such that the sharing is uniform, we require that all subsequent functions computing on a uniform sharing give back an output which is again uniform. This property is given in the following combinatorial form.

Definition 2.4 (Uniformity). A shared function $\mathbf{N}(\mathbf{x}) = \mathbf{y}$ is uniform if $\forall x \in \mathbb{F}$, $\forall \mathbf{y} \in Sh(N(x))$:

$$\left| \left\{ \mathbf{x} \in Sh(x) \,\middle|\, \mathbf{N}(\mathbf{x}) = \mathbf{y} \right\} \right| = \frac{|\mathbb{F}|^{s_x-1}}{|\mathbb{F}|^{s_y-1}}\,.$$

We note that if the number of inputs and outputs of the shared function are equal, the above property is equivalent to the shared function being a permutation.

A uniform function maps a uniform input to a uniform outputs, this is proven in [3].

Lemma 2.5. *If a shared function* $\mathbf{N}(\mathbf{x}) = \mathbf{y}$ *is uniform then the masking* $\mathbf{Y}$ *is uniform provided the masking* $\mathbf{X}$ *is uniform.*

We finalise this section by giving definitions of a threshold layer and a threshold circuit. These definitions are meant to capture the mathematical properties of a threshold implementation.

Definition 2.6 (Threshold Layer). A threshold layer is a shared function $\mathbf{N}$ where its input and output are synchronised. A threshold layer consists of single output functions $\mathbf{f}_i$, called component functions.

We then give a definition of a threshold layer where its input is first shared and its output is reconstructed. To capture pipelined implementations, an implementation technique where a logical circuit is divided in two divided by a register stage to allow for higher clock frequencies, we allow for the circuit to be divided over several threshold layers.

Definition 2.7 (Threshold Circuit). The composition of an input encoder, a shared realisation $\mathbf{N}$, and an output decoder is a threshold circuit if $\mathbf{N}$ consists of the serial composition of threshold layers $\mathbf{N}_i$.

We give a graphic representation of a threshold circuit in Figure 1.

3 PROBING SECURE THRESHOLD CIRCUITS

In this section, we discuss the first-order and higher-order univariate probing security of threshold circuits composed of correct, non-complete, and uniform layers. We end the section by discussing how to model fresh randomness in a threshold circuit.

3.1 First-order Probing Security

In this section we show that a threshold circuit composed of correct, non-complete, and uniform layers is first-order secure. To give security proofs, we need to have a security model. Since we are interested in the security of hardware implementations, we make use of the *glitch-robust probing model* from [10]. A probing adversary chooses up to a threshold number of wires of the algorithm's circuit representation to read the value from (probing). When considering the effect of glitches, with each probe the adversary can read all the input values which flow to that wire until a register is reached. The interested reader is encouraged to learn more about this model in [10], a comparison of the probing model with other leakage models is given in [13], and the proof of the reduction of the noisy leakage model to the probing model is found in [9]. In threshold circuits, the glitch-robust adversary transforms into an adversary reading the circuit's component functions due to the component functions being walled-off by registers.

Definition 3.1 (First-order Robust Probing Security). A threshold circuit is first-order robust probing secure if for any component function in the circuit, its input values are jointly independent of the circuit's secrets.

We now prove that a threshold circuit is first-order secure by providing lemmas to increase the proof's transparency.

The first lemma tells us that the notion of uniformity is composable. Meaning that the composition of two uniform functions is again uniform.

Lemma 3.2. *The composition of two uniform functions is again uniform.*

Proof. We recall the definition of a uniform function $\mathbf{N}$,

$$\forall x,\ \forall \mathbf{y} \in Sh(N(x)) : \left|\left\{\mathbf{x} \in Sh(x) \,\middle|\, \mathbf{N}(\mathbf{x}) = \mathbf{y}\right\}\right| = \frac{|\mathbb{F}|^{s_x-1}}{|\mathbb{F}|^{s_y-1}}\,.$$

Let $\mathbf{N}$ and $\mathbf{M}$ be two uniform functions, we want to prove that $\mathbf{N} \circ \mathbf{M}$ is uniform.

Fix an input secret x and an output vector $\mathbf{z} \in Sh(N(M(x)))$, we are interested in $|\{\mathbf{x} \in Sh(x) \mid \mathbf{N}(\mathbf{M}(\mathbf{x})) = \mathbf{z}\}|$. From $\mathbf{N}$ being a uniform function we know that

$$\left|\left\{\mathbf{y} \in Sh(M(x)) \,\middle|\, \mathbf{N}(\mathbf{y}) = \mathbf{z}\right\}\right| = \frac{|\mathbb{F}|^{s_y-1}}{|\mathbb{F}|^{s_z-1}}\,.$$

From $\mathbf{M}$ being a uniform function we find that for each $\mathbf{y} \in Sh(M(x))$

$$\left|\left\{\mathbf{x} \in Sh(x) \,\middle|\, \mathbf{M}(\mathbf{x}) = \mathbf{y}\right\}\right| = \frac{|\mathbb{F}|^{s_x-1}}{|\mathbb{F}|^{s_y-1}}\,.$$

Thus by looping through each $\mathbf{y}$ such that $\mathbf{N}(\mathbf{y}) = \mathbf{z}$, we find that

$$\left|\left\{\mathbf{x} \in Sh(x) \,\middle|\, \mathbf{N}(\mathbf{M}(\mathbf{x})) = \mathbf{z}\right\}\right| = \frac{|\mathbb{F}|^{s_x-1}}{|\mathbb{F}|^{s_y-1}} \frac{|\mathbb{F}|^{s_y-1}}{|\mathbb{F}|^{s_z-1}} = \frac{|\mathbb{F}|^{s_x-1}}{|\mathbb{F}|^{s_z-1}}\,,$$

which is what we needed to prove. □

We then move on to show that a uniform function has some interesting properties pertaining to the distribution of its input with respect to its output and vice versa. These independence properties are the shared function's equivalent of the following property of a uniform sharing proven in [3].

Lemma 3.3. *If the masking $\mathbf{X}$ of X is uniform, then $\mathbf{X}_{\bar{i}}$ and X are independent for every i.*

Thus the secret of a uniform sharing is independent of every set of $s_x - 1$ shares. We now show that there is a similar property uniform functions attain but then between its input and output. The first one says that all sets of $s_x - 1$ input shares of a function $\mathbf{N}(\mathbf{x})$ is independent of its output secret.

Lemma 3.4. *If a shared function $\mathbf{N}(\mathbf{x}) = \mathbf{y}$ has uniform inputs, then $\mathbf{X}_{\bar{i}}$ and Y are independent for every $i \in [s_x]$.*

Proof. We need to prove that

$$P(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}},\ Y = y) = P(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}})\, P(Y = y)\,.$$

We manipulate the equation's left side to equal its right one.

$$\begin{aligned} P(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}},\ Y = y) &= \sum_{N(x)=y} P(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}},\ X = x) && (1)\\ &= P(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) \sum_{N(x)=y} P(X = x) && (2)\\ &= P(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}})\, P(Y = y) \end{aligned}$$

Where (1) holds due to $\mathbf{N}$ being a function and (2) holds due to Lemma 3.3 since $\mathbf{X}$ is uniform. □

This independence property also holds in the opposite direction. Thus all sets of $s_y - 1$ of output shares of $\mathbf{N}$ are independent of the function's input secret.

Lemma 3.5. *If a uniform function $\mathbf{N}(\mathbf{x}) = \mathbf{y}$ has uniform inputs, then $\mathbf{Y}_{\bar{i}}$ and X with $N(X) = Y$ are independent for every $i \in [s_y]$.*

Proof. We recall the definition of a uniform function,

$$\forall x,\ \forall \mathbf{y} \in Sh(N(x)) : \left|\left\{\mathbf{x} \in Sh(x) \,\middle|\, \mathbf{N}(\mathbf{x}) = \mathbf{y}\right\}\right| = \frac{|\mathbb{F}|^{s_x-1}}{|\mathbb{F}|^{s_y-1}}\,.$$

We need to prove that

$$P(\mathbf{Y}_{\bar{i}} = \mathbf{y}_{\bar{i}},\ X = x) = P(\mathbf{Y}_{\bar{i}} = \mathbf{y}_{\bar{i}})\, P(X = x)\,.$$

We start from the left equation and manipulate it to the right one.

$$\begin{aligned} P(\mathbf{Y}_{\bar{i}} = \mathbf{y}_{\bar{i}},\ X = x) &= P(X = x)\, P(\mathbf{Y}_{\bar{i}} = \mathbf{y}_{\bar{i}} \mid X = x)\\ &= P(X = x)\, P(\mathbf{Y} = \mathbf{y} \mid X = x)\\ &= P(X = x)\, \frac{|\mathbb{F}|^{s_x-1}}{|\mathbb{F}|^{s_y-1}} P(\mathbf{X} = \mathbf{x} \mid X = x) && (3)\\ &= P(X = x)\, \frac{|\mathbb{F}|^{s_x-1}}{|\mathbb{F}|^{s_y-1}} |\mathbb{F}|^{-s_x+1} && (4)\\ &= P(X = x)\, |\mathbb{F}|^{-s_y+1}\\ &= P(X = x)\, P(\mathbf{Y}_{\bar{i}} = \mathbf{y}_{\bar{i}}) && (5) \end{aligned}$$

Where (3) holds due to Definition 2.4, (4) due to Definition 2.1 and (5) due to Lemma 2.5. □

These last two lemmas give a very intuitive proof of security against a probing adversary. Namely, if the adversary probes a component function of a threshold circuit consisting of correct, non-complete and uniform threshold layers, it learns (due to the component function's non-completeness) at most $s_x - 1$ shares of a secret x. Due to the input being a uniform sharing, the probed information is independent of x and since the threshold layer is

a function, this information is also independent of that layer's output secret. Similarly, due to each layer being a uniform function, the adversary learns nothing from the preceding threshold layer's input secret. Since the property of uniformity is composable among layers, the previous arguments hold with any composition of layers resulting in the probed information being independent of any of the threshold circuit's secrets.

The formal theorem of first-order security along with its proof is given as follows.

Theorem 3.6. *A threshold circuit composed of layers which are correct, non-complete, and uniform is first-order robust probing secure.*

Proof. We take an arbitrary probed component function $\mathbf{f}_i$ from the threshold circuit and we denote the probed input shares of $\mathbf{f}_i$ by $\mathcal{I}$.

We know that $\mathbf{f}_i$ lies in exactly one threshold layer $\mathbf{N}(\mathbf{x}) = \mathbf{y}$. This layer complies to the non-completeness property, thus from Definition 2.3 we know that $\mathcal{I}$ is independent of at least one input share $\mathbf{x}_i$. From Lemma 2.5 we know that $\mathbf{X}$ is uniform and thus from Lemma 3.3 follows that $\mathcal{I}$ is independent of x.

We then know from Lemma 3.4 that $\mathcal{I}$ is independent of the output secret y. By taking together multiple threshold layers and using Lemma 3.2, we know that $\mathcal{I}$ is independent of any output secrets of the layers after $\mathbf{N}$. Identically, $\mathcal{I}$ is independent of the input secrets of the layers before $\mathbf{N}$ due to Lemma 3.5.

Thus the probed information is independent of the input or output secrets of each threshold layer. Since these are the only sensitive variables in the threshold circuit and since the probed component function was taken arbitrarily, the theorem is proven. □

3.2 Higher-order Univariate Security

A generalisation on the security notions of threshold implementations has been given in the work of higher-order threshold implementations from [2] where there is the following definition higher-order non-completeness.

Definition 3.7 (d^{th}-order Non-completeness [2]). A shared function $\mathbf{N}(\mathbf{x})$ is d^{th}-order non-complete if any combination of d component functions $\mathbf{f}_i$ uses at most $s_x - 1$ input shares.

The above definition together with uniformity is sufficient for higher-order univariate security which signifies security against an adversary who can only read component functions in one threshold layer but up to d of them.

From the d^{th}-order non-completeness we see that the adversary can not view all the shares of an input to the probed threshold layer. As a result, if the input sharing was uniform, the probed information would be independent of the secret input to that layer. We thus find the following theorem whose proof is parallel to the one for first-order security.

Theorem 3.8. *A threshold circuit composed of layers which are correct, d^{th}-order non-complete, and uniform is secure against a d^{th}-order univariate adversary.*

3.3 On Randomness in Threshold Circuits

In our proof of probing security we assumed that each function was deterministic meaning that its input completely determined its output. However, this assumption could be invalidated as in practice designers inject fresh randomness in a shared function which is a technique to make functions uniform in case a proper uniform sharing of the function can not be found or is expensive. In threshold circuits we model the injected randomness as an input to the requesting layer where the sharing is created by the encoder (see Definition 2.7). Thus randomness is seen as an input sharing and since these shares are not used for correctness purposes, we define that the secret of this sharing is zero. Injecting extra randomness in a layer thus corresponds to adding extra inputs and outputs to the shared function in order to make it uniform. As a result, our security proofs still hold when extra randomness is injected as this extra sharing corresponds to embedding the non-uniform function into a larger potentially uniform function. This embedding can for example be done by using a Feistel network which gives the "Changing of the Guards" methods by Joan Daemen [7]. In this case, the shared function also recycles all extra randomness given to it so that it can be used for other functions in the circuit.

We note that recycling randomness not only works for inside a cipher but also for modes of operation as for example we can recycle the randomness used when reconstructing the ciphertext. Thus when reconstructing the ciphertext from its shares $\{\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2\}$, we give back the value $\mathbf{C}_0 + \mathbf{C}_1 + \mathbf{C}_2$ while keeping the shares $\mathbf{C}_1$ and $\mathbf{C}_2$ so that when we start a new cipher call, we share the plaintext M into the shares $\{M + \mathbf{C}_1 + \mathbf{C}_2, \mathbf{C}_1, \mathbf{C}_2\}$. This trick further reduces randomness costs of a protocol.

4 NON-INTERFERENCE IMPLIES UNIFORMITY

So far we have shown that threshold circuits build out the probing model to a serially composable security notion which allows for the easy design of secure functions. A threshold circuit is not the only model which allows for such composability, another well-known notion is "non-interference" introduced by Barthe *et al.* [1]. The notion of non-interference and strong non-interference not only allows for serial composition but also the parallel composition of functions (more specifically, functions working on dependent inputs). Additionally, it allows for higher-order multivariate passive protection whereas threshold circuits are specialised for first-order or higher-order univariate security. More information on the notions of non-interference can be found in the original work. In this section we consider first-order non-interference (1-NI) and show it implies uniformity which opens the possibility to find more efficient threshold designs compared to first-order non-interferent designs.

We define first-order (strong) non-interference using the probabilistic notions given in the work of De Meyer *et al.* [8]. These notions consider the case where a non-interferent component is given a uniform sharing. The goal of this section is to prove that if a non-interferent function is given a uniform input, it will give back a uniform output.

We further generalise first-order non-interference to work over an arbitrary number of Boolean masked shares (previously only 2

shares were considered). We require a simulator who can reproduce each possible probe or sets of $s_y - 1$ output shares given a set of $s_x - 1$ input shares.[1] Intuitively, this generalisation of non-interference grants composable security since a probed value in a function can be simulated with all-but-one input shares, which in its turn forms the output of a previous function. In case the latter function is also non-interferent, these output values can again be simulated with all-but-one of its input shares. This chains until we reach the initial sharing function. Due to the passive threshold of the Boolean masking scheme, knowing all-but-one shares of the encoder's output implies that the adversary does not learn the secret of the shares.

We thus look at first-order non-interference in case the considered function is given uniform inputs.

Definition 4.1 (First-order Non-interference). A shared function $\mathbf{N}(\mathbf{x}) = \mathbf{y}$ with uniform input shares is 1-NI if, for any glitch-robust probe q and any $\mathbf{x}$, the following condition holds:

$$\exists i \in [s_x] : P(Q = q \,|\, \mathbf{X} = \mathbf{x}) = P(Q = q \,|\, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}),$$

together with:

$$\forall j \in [s_y],\ \exists i \in [s_x] : P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}} \,|\, \mathbf{X} = \mathbf{x}) = P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}} \,|\, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}).$$

While the above notion secures the serial composition of functions, we need a more strict property if we want to secure the parallel composition as well. This leads to the notion of strong non-interference where we add the condition that the output does not reveal any information on the input.

Definition 4.2 (First-order Strong Non-interference). A shared function $\mathbf{N}(\mathbf{x}) = \mathbf{y}$ with uniform input shares is 1-SNI if it is 1-NI and the following condition holds for any $\mathbf{x}$:

$$\forall j \in [s_y] : P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}} \,|\, \mathbf{X} = \mathbf{x}) = P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}}).$$

Essentially, non-interference tells us that each set of all-but-one output shares is independent of the function's input secret which is a property shared with uniform functions (see Lemma 3.5).

Theorem 4.3. *If a shared function* $\mathbf{N}(\mathbf{x}) = \mathbf{y}$ *is* 1*-NI (Def 4.1) it is uniform.*

Proof. We take an arbitrary secret x with $N(x) = y$ and an arbitrary $\mathbf{y} \in Sh(y)$, and we assume that $\mathbf{X}$ is uniform. For a uniform input sharing $\mathbf{X}$, it follows from Definition 4.1 that every set of all-but-one output shares $\mathbf{Y}_{\bar{j}}$ is independent of the input secret X since for an arbitrary x we have

$$\begin{aligned} P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}},\, X = x) &= \sum_{\mathbf{x} \in Sh(x)} P(\mathbf{X} = \mathbf{x})\, P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}} \,|\, \mathbf{X} = \mathbf{x}) \\ &= \sum_{\mathbf{x} \in Sh(x)} P(\mathbf{X} = \mathbf{x})\, P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}} \,|\, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) \\ &= \sum_{\mathbf{x} \in Sh(x)} P(\mathbf{X}_i = \mathbf{x}_i)\, P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}},\, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) \\ &= \sum_{\mathbf{x}_{\bar{i}}} P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}},\, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}), \end{aligned}$$

which is independent of the secret X.

[1] For a formal definition of probing simulation, the reader is referred to [8].

We prove that the output of $\mathbf{N}$ is uniform. We find the following equalities for an arbitrary j.

$$\begin{aligned} P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}},\, Y = y) &= \sum_{N(x)=y} P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}},\, X = x) \\ &= \sum_{N(x)=y} P(X = x)\, P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}} \,|\, X = x) \\ &= \sum_{N(x)=y} P(X = x)\, P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}}) \\ &= P(Y = y)\, P(\mathbf{Y}_{\bar{j}} = \mathbf{y}_{\bar{j}}) \end{aligned}$$

□

Oppositely, we find a function which is non-complete and uniform but which is not first-order non-interferent. Take the multiplication $T(a, b, c) = ab + c$ in three shares.

$$\begin{aligned} \mathbf{d}_1 &= \mathbf{a}_1\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_2 + \mathbf{a}_2\mathbf{b}_1 + \mathbf{c}_1 \\ \mathbf{d}_2 &= \mathbf{a}_2\mathbf{b}_2 + \mathbf{a}_2\mathbf{b}_3 + \mathbf{a}_3\mathbf{b}_2 + \mathbf{c}_2 \\ \mathbf{d}_3 &= \mathbf{a}_3\mathbf{b}_3 + \mathbf{a}_3\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_3 + \mathbf{c}_3 \end{aligned}$$

Since $\mathbf{c}$ is seen as an input and not as unique randomness, two output shares can not be simulated with only two shares of each input.

As a result of the connection between uniformity and non-interference, we find that it is possible to securely compose (sequentially) NI-secure functions with uniform and non-complete threshold layers. This also allows designers to secure parallel composed functions working on dependent inputs as we can use strong non-interference for those components in case no efficient uniform function is found.

5 RELAXING NON-COMPLETENESS

We see from the proof of probing security that we can relax the notion of non-completeness as we only require that the information viewed by an adversary is independent of a function's input secret given a uniform input sharing. We thus relax non-completeness as first-order glitch-robust probing security. Effectively, this means that we can consider multiple staged functions where intermediate sharings need not be uniform but given a uniform input the block gives back a uniform output and where all intermediate computations are first-order robust probing secure. This is typically useful when considering functions which first expand and then compress their number of shares. We give a simple example for the function $T(a, b, c, d) = abc + d$ where each value is shared with two shares. We find the following way of calculating $\mathbf{T}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$ in multiple stages where each stage registers the outcome of its calculation.

Stage 1	Stage 2	Stage 3	Stage 4
$\mathbf{e}_0 = \mathbf{a}_0\mathbf{b}_0\mathbf{c}_0 + \mathbf{d}_0$	$\mathbf{f}_0 = \mathbf{e}_0 + \mathbf{e}_1$	$\mathbf{g}_0 = \mathbf{f}_0 + \mathbf{f}_1$	$\mathbf{h}_0 = \mathbf{g}_0 + \mathbf{g}_1$
$\mathbf{e}_1 = \mathbf{a}_0\mathbf{b}_0\mathbf{c}_1$	$\mathbf{f}_1 = \mathbf{e}_2$	$\mathbf{g}_1 = \mathbf{f}_2$	$\mathbf{h}_1 = \mathbf{g}_2 + \mathbf{g}_3$
$\mathbf{e}_2 = \mathbf{a}_0\mathbf{b}_1\mathbf{c}_0$	$\mathbf{f}_2 = \mathbf{e}_3$	$\mathbf{g}_2 = \mathbf{f}_3$	
$\mathbf{e}_3 = \mathbf{a}_0\mathbf{b}_1\mathbf{c}_1$	$\mathbf{f}_3 = \mathbf{e}_4$	$\mathbf{g}_3 = \mathbf{f}_4 + \mathbf{f}_5$	
$\mathbf{e}_4 = \mathbf{a}_1\mathbf{b}_0\mathbf{c}_0$	$\mathbf{f}_4 = \mathbf{e}_5$		
$\mathbf{e}_5 = \mathbf{a}_1\mathbf{b}_0\mathbf{c}_1$	$\mathbf{f}_5 = \mathbf{e}_6 + \mathbf{e}_7$		
$\mathbf{e}_6 = \mathbf{a}_1\mathbf{b}_1\mathbf{c}_0$			
$\mathbf{e}_7 = \mathbf{a}_1\mathbf{b}_1\mathbf{c}_1 + \mathbf{d}_1$			

The above sharing is robust probing secure as a glitch-extended probe can either only view one share of $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ or a value masked by one share of $\mathbf{d}$, in either case the information is independent of the input secrets. Additionally, the above sharing gives a uniform sharing $\mathbf{h}$ given that $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and $\mathbf{d}$ is uniform. The result is a sharing which has high latency but minimal randomness requirements.

6 THRESHOLD CIRCUITS WITH DIFFERENT SHARING SCHEMES

The definitions from Section 2 only considered single input/output functions with Boolean masking. We now give more general definitions considering multiple inputs and outputs as in [17]. We also note that from a security perspective, threshold circuits easily handle inputs from different sharings, for example, a shared function taking in both a polynomial masked variable and a Boolean masked variable. Thus we extend the definitions of correctness, non-completeness, and uniformity to account for different sharing schemes. For this extension, we use the notation $Sh_{(i)}$ to denote the masking scheme for the i^{th} share vector.

We start by defining the passive threshold of a secret sharing scheme.

Definition 6.1 (Passive Threshold). A secret sharing scheme has a passive threshold d if all sets of up to d shares are independent of the secret.

This definitions tells us how many shares we need to view in order to get information on the secret. With Boolean masking this threshold is equal to $s_x - 1$, meaning that you need to view all shares in order to know the secret.

We again look at the uniformity property of a sharing.

Definition 6.2 (Uniform Masking). $(\mathbf{X}_{(1)}, ..., \mathbf{X}_{(n)})$ is uniform if there exists a constant c such that for all $(x_{(1)}, ..., x_{(n)})$ we have

$$P((\mathbf{X}_{(1)}, ..., \mathbf{X}_{(n)}) = (\mathbf{x}_{(1)}, ..., \mathbf{x}_{(n)}) \big| (X_{(1)}, ..., X_{(n)}) = (x_{(1)}, ..., x_{(n)})) = \begin{cases} c & \text{if } \forall i \in [n],\ \mathbf{x}_{(i)} \in Sh_{(i)}(x_{(i)}), \\ 0 & \text{else}. \end{cases}$$

Notice that the uniformity of a sharing is a joint distribution. As a result, we see that different share vectors of a uniform sharing are by definition independent of each other. This has raised some confusion as for example in the work of Reparaz *et al.* [19] there is the extra requirement on "$d + 1$" sharings that each input is shared independently. However, this requirement was already included in the original definition of uniformity and should thus also hold for general "$td + 1$" sharings.

We give the general version of the three core properties. The first one tells us that all the outputs of a shared function can be reconstructed to their corresponding secret outputs.

Definition 6.3 (Correctness). Given $(x_{(1)}, ..., x_{(n)})$, for each sharing $\mathbf{x}_{(i)} \in Sh_{(i)}(x)$, we have that the reconstruction of $(\mathbf{y}_{(1)}, ..., \mathbf{y}_{(m)}) = \mathbf{N}(\mathbf{x}_{(1)}, ..., \mathbf{x}_{(n)})$ is equal to $(y_{(1)}, ..., y_{(m)}) = N(x_{(1)}, ..., x_{(n)})$.

The multiple input/output definition of non-completeness considers the passive threshold d_i of each sharing separately. A component function can then take in d_i shares of the corresponding secret. We note, however, that when we are encoding shares to counter fault attacks, we are no longer working with threshold secret sharing schemes. We thus relax the notion of non-completeness by allowing each component function to take in linear-dependent shares, for example a component function can take in all replicas of a share. The number of linear-independent shares a component function can input is bounded by the sharing scheme's passive threshold.

Definition 6.4 (Non-completeness). A shared function $\mathbf{N}$ is non-complete if every of its component functions uses at most d_i linear-independent shares of the i^{th} input where d_i is that input sharing's passive threshold.

Similarly as said in the previous section, we can, instead of non-completeness, demand that the shared function is first-order glitch-robust probing secure.

Finally, the notion of uniformity of a shared function is extended similarly to uniformity of a sharing. A uniform function expects a joint uniform input and in return gives back a joint uniform output. The property is again given in its combinatorial form.

Definition 6.5 (Uniformity). A shared function $\mathbf{N}(\mathbf{x}_{(1)}, ..., \mathbf{x}_{(n)})$ is uniform if there is a c such that $\forall x_{(i)} \in \mathbb{F}, \forall \mathbf{y}_{(i)} \in Sh_{(i)}(y_{(j)})$, and $(y_{(1)}, ..., y_{(m)}) = N(x_{(1)}, ..., x_{(n)})$:

$$\left| \left\{ \cup_i \mathbf{x}_{(i)} \in Sh_{(i)}(x_{(i)}) \,\middle|\, \mathbf{N}(\mathbf{x}_{(1)}, ..., \mathbf{x}_{(n)}) = (\mathbf{y}_{(1)}, ..., \mathbf{y}_{(m)}) \right\} \right| = c\,.$$

Examples: We give some examples of threshold implementations which use different sharing schemes.

To gain fault protection one can duplicate shares and add an error detection mechanism to check whether a fault occurred on one of the two replicas. However, such a duplicated sharing is not uniform by Definition 2.1 which considers Boolean shares. Instead, to correctly verify the sharing, it needs to be seen as a "duplicated Boolean sharing", *i.e.*, $Sh(x) = \{(x_1, x_1, x_2, x_2) \,|\, x_1 + x_2 = x\}$. Using Definition 6.2 where $Sh(x)$ is seen as the set of all duplicated Boolean sharings of x, one can again verify a duplicated circuit as secure. This further generalises for sharings which have a linear code embedded in them. Examples of such sharings used in implementations are given in the work of Schneider *et al.* [20] where these sharings are used in order to detect injected faults. Generally, when considering a Boolean masked value where each share is encoded using a linear code C, we write our set of shares as $Sh(x) = \{\mathbf{x} \in C^{s_x} \mid \oplus_{i=1}^{s_x} \mathbf{x}_i = x\}$. Using this definition of shares, we can again define the designs of [20] as uniform.

Generalising uniformity and non-completeness to work with different sharing schemes also allows us to gain improved efficiency over the usual Boolean sharing schemes. We give an example where we jointly share inputs which are independently processed. This example is related to the work from Coron *et al.* [6] where the above definitions of non-completeness and uniformity form its theoretical basis.

We consider two parallel S-Boxes working on different inputs.

Taking the example in Figure 2, we can share $\mathbf{a}$ and $\mathbf{x}$ using usual Boolean masking, considering we use 2 shares we have a state of size $4|\mathbb{F}|$. However, we can also consider the sharing $Sh(a, x) = \{(a + r, x + r, r) | r \in \mathbb{F}\}$ which saves a field element in the state size. The passive threshold of the previous sharing scheme is equal to one, thus when we consider a function which operates on both

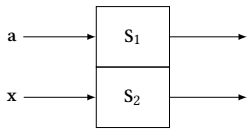

Figure 2: Two shared S-boxes (S_1, S_2) working on separate inputs $\mathbf{a}, \mathbf{x}$.

$\mathbf{a}$ and $\mathbf{x}$, the sharing will become expensive. In ciphers such a recombination happens at its diffusion layer, nevertheless in most ciphers we still stand to gain. Considering AES as an example, we see that its diffusion layer only combines four S-boxes, namely due to MixColumns. Thus by jointly sharing four bytes in a row of an AES state we can save in the shared state size and still work with the more restrictive passive threshold of the sharing.

We demonstrate the above sharing considering two parallel multiplications. Following Figure 2, we have $S_1(a, b, c) = ab + c$ and $S_2(x, y, z) = xy + z$ where we will jointly share the inputs of S_1 and S_2. This gives us the following shares $(\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$, $(\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$, $(\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3)$ such that $\mathbf{a}_1 + \mathbf{a}_3 = a$ and $\mathbf{a}_2 + \mathbf{a}_3 = x$ similar for $\mathbf{b}$ and $\mathbf{c}$. By introducing extra stages, we find the following sharing for the two multiplications $\mathbf{S}_1$ and $\mathbf{S}_2$.

Stage 1	Stage 2	Stage 3
$\mathbf{d}_1 = \mathbf{a}_1\mathbf{b}_1 + \mathbf{c}_1$	$\mathbf{e}_1 = \mathbf{d}_1 + \mathbf{d}_2$	$\mathbf{f}_1 = \mathbf{e}_1 + \mathbf{e}_2$
$\mathbf{d}_2 = \mathbf{a}_1\mathbf{b}_3$	$\mathbf{e}_2 = \mathbf{d}_3$	$\mathbf{f}_2 = \mathbf{e}_3 + \mathbf{e}_4$
$\mathbf{d}_3 = \mathbf{a}_3\mathbf{b}_1$	$\mathbf{e}_3 = \mathbf{d}_4 + \mathbf{d}_5$	$\mathbf{f}_3 = \mathbf{e}_5$
$\mathbf{d}_4 = \mathbf{a}_2\mathbf{b}_2 + \mathbf{c}_2$	$\mathbf{e}_4 = \mathbf{d}_6$	
$\mathbf{d}_5 = \mathbf{a}_2\mathbf{b}_3$	$\mathbf{e}_5 = \mathbf{d}_7$	
$\mathbf{d}_6 = \mathbf{a}_3\mathbf{b}_2$		
$\mathbf{d}_7 = \mathbf{a}_3\mathbf{b}_3 + \mathbf{c}_3$		

We can see that the output $\mathbf{f}$ is uniform given that $\mathbf{a}, \mathbf{b}$, and $\mathbf{c}$ is uniform and that the computation is first-order glitch-robust probing secure. Thus by changing our sharing scheme and trading off latency, we can reduce our shared state size. We also note that a clever designer can change the sharing scheme from operation to operation to potentially further reduce costs.

7 CONCLUSION

We have proven that a function shared with the threshold implementation methodology is secure against a first-order glitch-robust probing adversary. From this proof of security, we were able to provide a link between non-interference and uniformity and to relax the notion of non-completeness. We also provided more general definitions of correctness, non-completeness, and uniformity in order to account for sharing schemes different from Boolean masking. The results are an assortment of techniques to potentially increase the efficiency of shared implementations.

Further work in this line consists of using the new understanding of non-completeness and uniformity to generalise the notions in order to provide security against stronger attackers such as higher-order multivariate passive attacks or fault attacks.

Acknowledgements. The authors would like to thank Michiel Van Beirendonck for the interesting discussions.

This work was supported in part by the Research Council KU Leuven: C16/18/004, by the NIST Research Grant 60NANB15D346, and by the EU H2020 project FENTEC. Siemen Dhooghe is supported by a Ph.D. Fellowship from the Research Foundation - Flanders (FWO). Svetla Nikova was partially supported by the Bulgarian National Science Fund, Contract No. 12/8.

REFERENCES

[1] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. 2016. Strong Non-Interference and Type-Directed Higher-Order Masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. 116–129. http://doi.acm.org/10.1145/2976749.2978427

[2] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2014. Higher-Order Threshold Implementations. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II (Lecture Notes in Computer Science)*, Palash Sarkar and Tetsu Iwata (Eds.), Vol. 8874. Springer, 326–343. https://doi.org/10.1007/978-3-662-45608-8_18

[3] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2015. Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Trans. on CAD of Integrated Circuits and Systems* 34, 7 (2015), 1188–1200. https://doi.org/10.1109/TCAD.2015.2419623

[4] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings (Lecture Notes in Computer Science)*, Michael J. Wiener (Ed.), Vol. 1666. Springer, 398–412. https://doi.org/10.1007/3-540-48405-1_26

[5] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2016. Masking AES with d+1 Shares in Hardware. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings (Lecture Notes in Computer Science)*, Benedikt Gierlichs and Axel Y. Poschmann (Eds.), Vol. 9813. Springer, 194–212. https://doi.org/10.1007/978-3-662-53140-2_10

[6] Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. 2016. Faster Evaluation of SBoxes via Common Shares. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings (Lecture Notes in Computer Science)*, Benedikt Gierlichs and Axel Y. Poschmann (Eds.), Vol. 9813. Springer, 498–514. https://doi.org/10.1007/978-3-662-53140-2_24

[7] Joan Daemen. 2017. Changing of the Guards: A Simple and Efficient Method for Achieving Uniformity in Threshold Sharing. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. 137–153. https://doi.org/10.1007/978-3-319-66787-4_7

[8] Lauren De Meyer, Begül Bilgin, and Oscar Reparaz. 2019. Consolidating Security Notions in Hardware Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019, 3 (2019), 119–147. https://doi.org/10.13154/tches.v2019.i3.119-147

[9] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. 2014. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*. 423–440. https://doi.org/10.1007/978-3-642-55220-5_24

[10] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. 2018. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018, 3 (2018), 89–120. https://doi.org/10.13154/tches.v2018.i3.89-120

[11] Louis Goubin and Jacques Patarin. 1999. DES and Differential Power Analysis (The "Duplication" Method). In *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings (Lecture Notes in Computer Science)*, Çetin Kaya Koç and Christof Paar (Eds.), Vol. 1717. Springer, 158–172. https://doi.org/10.1007/3-540-48059-5_15

[12] Hannes Groß, David Schaffenrath, and Stefan Mangard. 2017. Higher-Order Side-Channel Protected Implementations of KECCAK. In *Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017*. 205–212. https://doi.org/10.1109/DSD.2017.21

[13] Yael Tauman Kalai and Leonid Reyzin. 2019. A Survey of Leakage-Resilient Cryptography. *IACR Cryptology ePrint Archive* 2019 (2019), 302. https://eprint.iacr.org/2019/302

[14] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. 388–397. https://doi.org/10.1007/3-540-48405-1_25

[15] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. 2011. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*. 69–88. https://doi.org/10.1007/978-3-642-20465-4_6

[16] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. 2006. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*. 529–545. https://doi.org/10.1007/11935308_38

[17] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. 2011. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptology* 24, 2 (2011), 292–321. https://doi.org/10.1007/s00145-010-9085-7

[18] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. 2011. Side-Channel Resistant Crypto for Less than 2, 300 GE. *J. Cryptology* 24, 2 (2011), 322–345. https://doi.org/10.1007/s00145-010-9086-6

[19] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. 2015. Consolidating Masking Schemes. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*. 764–783. https://doi.org/10.1007/978-3-662-47989-6_37

[20] Tobias Schneider, Amir Moradi, and Tim Güneysu. 2016. ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*. 302–332. https://doi.org/10.1007/978-3-662-53008-5_11