# Composable Masking Schemes in the Presence of Physical Defaults and the Robust Probing Model

Sebastian Faust[1], Vincent Grosso[1,2], Santos Merino Del Pozo[3], Clara Paglialonga[1], François-Xavier Standaert[3]

[1] University of Bochum, Fakultät für Mathematik, Germany.
[2] Radboud University Nijmegen, Digital Security Group, The Netherlands.
[3] Université catholique de Louvain, ICTEAM/ELEN/Crypto Group, Belgium.

**Abstract.** Composability and robustness against physical defaults (e.g., glitches) are two highly desirable properties for secure implementations of masking schemes. While tools exist to guarantee them separately, no current formalism enables their joint investigation. In this paper, we solve this issue by introducing a new model, the robust probing model, that is naturally suited to capture the combination of these properties. We first motivate this formalism by analyzing the excellent robustness and low randomness requirements of first-order threshold implementations, and highlighting the difficulty to extend them to higher orders. Next, and most importantly, we use our theory to design higher-order secure, robust and composable multiplication gadgets. While admittedly inspired by existing approaches to masking (e.g., Ishai-Sahai-Wagner-like, threshold, domain-oriented), these gadgets exhibit subtle implementation differences with these state-of-the-art solutions (none of which being provably composable and robust). Hence, our results illustrate how sound theoretical models can guide practically-relevant implementations.

## 1 Introduction

**State-of-the-art.** Protecting hardware and software implementations against side-channel attacks is an important challenge in cryptographic engineering. The masking countermeasure is among the most popular solutions for this purpose, due to the good understanding of its security requirements [13, 32, 41, 22, 23]. Intuitively, masked implementations can be viewed as implementations performing computations on secret-shared data. Under the fundamental conditions that $(i)$ the leakages of the shares are independent of each other, and $(ii)$ the leakages of the shares are sufficiently noisy, masking guarantees that the measurement complexity of any side-channel attack grows exponentially in the number of shares. Since the implementation cost of a masking scheme only grows (roughly) quadratically in the number of shares, it therefore provides a theoretically sound principle to prevent side-channel attacks for any cryptographic primitive.

Unfortunately, ensuring the two main (independence and noise) requirements of secure masking is also well known to be a non-trivial task:

First, a lack of composability (typically caused by an insufficient refreshing of the shares) can reduce the security order in the *probing model* of Ishai et al. [32]. Such a default is illustrated by the FSE 2013 attack by Coron et al. [20].

Considering a masking scheme with $d$ shares, it means that a combination of $d' < d$ shares is sufficient to extract sensitive information. The natural solution to avoid such a default is to use composable (e.g., SNI [3]) gadgets, and/or to test the (hardware or software) description codes of the implementations (i.e., all the operations manipulating the shares) thanks to formal methods [2].

Second, even if a combination of $d$ shares is required to extract sensitive information (i.e., if probing security is guaranteed), physical defaults can reduce the security order in the *bounded moment model* of Barthe et al. [4]. It then means that the lowest key-dependent statistical moment of the leakage distribution is lower than the optimal $d$. Concretely, such reductions happen because the leakage function recombines the shares to some extent. Typical examples of physical defaults include glitches (i.e., combinatorial recombinations) [33, 34], transition-based leakages (i.e., memory recombinations) [18, 1] and potentially couplings (i.e., routing recombinations) [15].[1] In practice, the security order in the bounded moment model can be determined using the TVLA methodology [17, 27] or variations thereof [35, 46, 24]: see [47] for a recent discussion.

Third, and even if security is guaranteed in the probing and bounded moment model, the noise condition may be challenging to achieve too, possibly leading to an insufficient security in the (most practically relevant) *noisy leakage model* of Prouff and Rivain [41]. Concretely, this noise condition depends on two main parameters. On the one hand, the physical noise of the exploitable operations (e.g., the operations that depend on an enumerable part of the key if one considers standard "divide-and-conquer" side-channel attacks), which is generally assumed more or less equal for all the operations. On the other hand, the number of exploitable operations. In this respect, two recent works showed how an adversary can efficiently exploit these multiple leakages thanks to multivariate (aka horizontal) attacks [5, 30]. The core intuition behind these works is that as the number of shares in a masking schemes increases, the number of exploitable operations does too, implying that a larger physical noise is necessary for masking to deliver security.[2] In practice, the global noise level necessary for secure masking (which roughly corresponds to the ratio between the physical noise and the number of exploitable operations in the implementation) can be determined/estimated thanks to an information theoretic analysis [30].

**Motivation & goals.** Based on this state-of-the-art, it appears that the key remaining challenge in the design of secure masking schemes is to minimize the amount of (bad) surprises at implementation time. In this respect, our starting observation is that existing models provide a principled path for this purpose. That is, and as illustrated in Figure 1, this challenge can be split in three separate parts: first obtain security in the (most abstract) probing model (which guarantees composability), second ensure that physical leakages do not (completely) recombine the shares in the bounded moment model, finally evaluate the concrete security level in the noisy leakage model. Since the evaluation of

[1] Although their exploitability in concrete attacks is not yet fully demonstrated.

[2] Masked implementations based on table lookups make this issue even more critical because of the larger performance overheads they imply [48, 12].

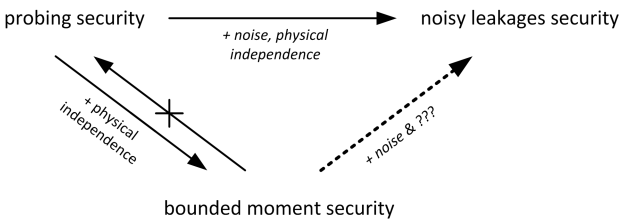


**Fig. 1.** Reductions between leakage security models (borrowed from [4]).

the noise condition is specifically discussed in [30], our focus in this paper is on the first two conditions. That is, how to guarantee security in the probing and bounded moment models, and what are the interactions between them?

Admittedly, specific answers to solve these two issues separately already exist. As previously mentioned, security in the probing model can be ensured thanks to composable gadgets and/or formal methods. As for security in the bounded moment model, it is well known that certain algorithmic features offer excellent ways to mitigate the shares' recombinations. The most famous example is the case of threshold implementations [38], where a non-completeness property is used to prevent that any combinatorial logic has access to all the shares of an encoded sensitive variable. The lazy engineering solution in [1] is another example where increasing the number of shares allows ruling out their full (memory) recombination. But open questions remain regarding whether these two threats (and possibly couplings) can be captured jointly? Also, the generalization of threshold implementations to higher-orders in [9] has been shown to suffer from refreshing issues in [43], and the heuristic solution in [44] does not yet provide a systematic way to evaluate randomness requirements – something probing security is very good at. So in general, a model allowing to capture both composability issues and to mitigate physical defaults would be very handy. These examples also question why first-order threshold implementations do not suffer from refreshing issues (despite low randomness requirements [40])?

**Our contribution.** We start with the case of first-order threshold implementations and show that their low randomness requirements can be explained thanks to (a slight variation of) the notion of Strong Non Interference (SNI) introduced in [3]. We then discuss the "number of shares vs. cycle count" tradeoff for such implementations. For this purpose, we observe that the correctness property of threshold implementations is in fact not needed for their intermediate results (i.e., we only want the final result to be correct). It allows us to exhibit examples of 4-bit S-boxes that globally match the definition of first-order threshold implementations in two shares and two cycles (a similar example is given in [14] for the Simon S-box). We then exploit this observation in order to (slightly) refine the exhaustive decomposition of Bilgin et al. in [10], for certain S-boxes. We conclude by discussing the additional challenges raised by higher-order glitch-free implementations, and use them to motivate the need of a new model.

We follow with our main contribution, which is to provide a formal tool to analyze such higher-order masked implementations. For this purpose, we introduce a new *robust probing model* which tweaks the original probing model in order to capture a wide class of physical defaults and can naturally be combined with existing notions of composability. Thanks to this model, we first discuss (and sometimes conjecture) simple lemmas regarding the combination of physical defaults. We then study concrete constructions of masked and threshold implementations. The main (admittedly consolidating) conclusion of this investigation is that the original multiplication algorithm of Ishai et al. [32] (or the parallel multiplication algorithm in [4]) naturally offer good robustness against physical defaults, while also offering good composability by design. We conclude the paper by discussing implementation guidelines for these algorithms in hardware, and confirm the relevance of our results with FPGA case studies.

## 2 Background

### 2.1 Circuit model

For our circuit model, we borrow the solution of [32] and represent a deterministic circuit $\mathsf{C}$ as a directed acyclic graph whose vertices are *combinatorial gates* and edges are wires carrying elements from a finite field $\mathbb{F}$. The simplest case is when $\mathbb{F}$ is the binary field so that wires carry bits and gates are Boolean operations AND and XOR. Yet, the addition and multiplication algorithms of secure masking schemes (e.g., described in Section 2.3) can actually run in larger fields $\mathbb{F}_{2^n}$: we then consider arithmetic circuits and gates rather than Boolean ones. In all cases, we denote the field addition by $\oplus$ and the field multiplication by $\odot$.

Since masking gadgets are randomized circuits, the model in [32] augments the previous deterministic circuits with *random gates* with fan-in 0: they produce a uniformly random element of the considered field. Eventually, robust masking requires circuits to be stateful (e.g., threshold implementations cannot maintain the non-completeness property discussed in the next sections otherwise [38]). For this purpose, we use *memory gates* which, on every invocation of the circuit, output the previous input to the gate and stores the current input for the next invocation. We note that these abstractions can be reasonably instantiated in practice, using true- or pseudo-random number generators for the random gates and registers (synchronized by a clock signal) for the memory gates.

### 2.2 Probing security and (Strong) Non Interference

In order to formalize the security of a masking scheme, Ishai et al. introduced in [32] the *q-probing model*, in which an attacker is allowed to read up to $q$ intermediate wires of a target circuit. In order to protect a circuit in this model, every sensitive value $k$ is split into at least $q+1$ values, called *shares*, such that their sum gives $k$. The security of a randomized circuit modeled as in the previous paragraph (which transforms a randomly encoded input into a randomly encoded

output) can then be expressed in various ways. Since our following discussions will consider both composable and non-composable gadgets, we next provide three different definitions. The first one, which is limited to non-composable security, was given by Rivain and Prouff in a CHES 2010 work that initiated the use of the probing model for analyzing concrete masking schemes:

**Definition 1 ($q-$probing security [45]).** *A circuit gadget* $\mathsf{G}$ *is* $q-$probing secure *(or secure at order $q$ in the probing model) if and only if every $q$-tuple of its intermediate variables is independent of any sensitive variable.*

In the case of block ciphers, sensitive variables typically correspond to partial computation results depending on the plaintext and key [19]. Alternatively, security in the probing model can also be expressed with the existence of a *simulator*, which can mimic the adversary's view using only a black-box access to $\mathsf{G}$, i.e., without having the knowledge of any internal wire but only $q$ shares of each secret input. We use the definition from Barthe et al. for this purpose:

**Definition 2 ($q-$Non Interference [3]).** *A circuit gadget* $\mathsf{G}$ *is* $q-$Non Interfering *($q-NI$) if and only if for any set of $q_1$ probes on its intermediate values and every set of $q_2$ probes on its output shares with $q_1 + q_2 \leq q$, the totality of the probes can be simulated with only $q_1 + q_2$ shares of each input.*

In other words, a circuit gadget is called NI if no *distinguisher* is able to tell apart the adversary's view from the simulation. In this respect, one important technical clarification is that in the definition of Barthe et al., the distinguisher can access the joint distribution of the (simulated) probes and input shares. As a result, NI is a stronger notion than the previous probing security.

Eventually, when gadgets are composed for producing a more complex circuit, it is needed to take into account that using an output of a gadget as input of another one can give additional information to the attacker. In this case, the definition of $q$-NI is not sufficient anymore to ensure global security of the circuit. A stronger property, called $q-$*Strong Non Interference* (or $q-$SNI), was also introduced by Barthe et al. and is recalled in the following:

**Definition 3 ($q-$Strong Non Interference [3]).** *A circuit gadget* $\mathsf{G}$ *is* $q-$Strong Non Interfering *($q-$SNI) if and only if for any set of $q_1$ probes on its intermediate values and every set of $q_2$ probes on its output shares with $q_1 + q_2 \leq q$, the totality of the probes can be simulated with $q_1$ shares of each input.*

Intuitively, this property not only requires that the adversary's view can be simulated with $q$ secret shares as for $q-$NI security, but also that the number of shares needed for the simulation to succeed is independent from the number of output wires that are probed (i.e., only depends on the internal probes).[3]

[3] The relevance of the SNI definition to composability also relies crucially on the fact that the distinguisher can access the joint distribution of the (simulated) probes and input shares, just as is was previously clarified after the NI definition.

### 2.3 The ISW multiplication algorithm

The first $q$-probing secure multiplication algorithm was introduced in the seminal work of Ishai et al. [32], and has later been proved to be also $q$–SNI in [3].[4] In the following, we will use a slight variation of this algorithm, depicted in Algorithm 1. It is functionally equivalent and also takes as input the shares of two values $a$ and $b$ and gives as output the shares of a value $c$, such that $c = a \odot b$. The only difference is in the way we organize the intermediate results, which is better suited to prevent physical defaults, as will be discussed in Section 5.2.

**Algorithm 1** Modified ISW multiplication algorithm with $d \geq 2$ shares.

**Input:** shares $(a_i)_{1 \leq i \leq d}$ and $(b_i)_{1 \leq i \leq d}$, such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$.
**Output:** shares $(c_i)_{1 \leq i \leq d}$, such that $\bigoplus_i c_i = a \odot b$.

```
for i = 1 to d do
  for j = i + 1 to d do
    r_{i,j} ←$ F_{2^n};
    u_{j,i} ← r_{i,j} ⊕ a_j ⊙ b_i;
    u_{i,j} ← r_{i,j} ⊕ a_i ⊙ b_j;
  end for
end for
for i = 1 to d do
  c_i ← a_i ⊙ b_i ⊕ ⨁_{j=1,j≠i}^{d} u_{i,j};
end for
```

## 3 The special case of 1st-order TIs

From the performance point-of-view, one important feature of the multiplication algorithm in the previous subsection is that it requires fresh randomness for every multiplication in the circuit to protect. Yet, the Threshold Implementations' (TIs) literature shows that it is sometimes possible to protect a full block cipher execution with very minimum randomness (i.e., the block size, typically) [40]. This suggests that such implementations benefit from some sort of composability.[5] In this section, we investigate this interesting property of 1st-order TIs.

For this purpose, we first recall that TIs are a type of masking scheme aimed at counteracting power (or electromagnetic) analysis attacks in the presence of glitches. In the 1st-order case we consider in this section, a TI takes a function $\mathsf{f}(x)$ with a uniform sharing of the input $x$, next denoted as $\boldsymbol{x} = (x_1, \ldots, x_m)$ such that $x = x_1 \oplus \cdots \oplus x_m$. The function $\mathsf{f}(.)$ is then shared in a vector of functions $(\mathsf{f}_1, \ldots, \mathsf{f}_m)$ called component functions, which needs to satisfy:

[4] As shown in [45], such a multiplication algorithm easily generalizes to larger fields, given that the refreshing is adapted to make it composable as proposed in [20].
[5] Which, as will be clarified next, cannot be strictly defined as composability.

1. **Correctness**: $y = \mathsf{f}(x) = \bigoplus_{i=1}^{m} \mathsf{f}_i(\boldsymbol{x})$.
2. **Non-Completeness**: any component functions $\mathsf{f}_i$ of $\mathsf{f}$ must be independent of at least one input share. (Which was generalized to the higher-order case in [9] as the requirement that any combination of up to $q$ component functions $\mathsf{f}_i$ of $\mathsf{f}$ must be independent of at least one input share).
3. **Uniformity**: denoting the vector of the output shares as $\boldsymbol{c} = (\mathsf{f}_1(\boldsymbol{x}), \ldots, \mathsf{f}_m(\boldsymbol{x}))$, the probability $\Pr(\boldsymbol{C} = \boldsymbol{c} | c = \bigoplus_{i=1}^{m} c_i)$ must be equal to $\Pr[C = c]$.

Note that the non-completeness property is not related to the refreshing (composability) issues that we discuss in this section and rather relates to the modeling of physical defaults that will be carefully discussed in Section 4.1

### 3.1 Pseudo−NI and pseudo−SNI security

Let us now consider the $3 \times 1$-bit function $\mathsf{f}(x, y, z) = (x \odot y) \oplus z$ which is at the core of many efficient S-box decompositions for TIs. In this case, it is easy to find a 1st-order TI with only 3 shares, given by the following equations:

$$\begin{aligned} c_1 &= (x_2 \odot y_2) \oplus (x_2 \odot y_3) \oplus (x_3 \odot y_2) \oplus z_2, \\ c_2 &= (x_3 \odot y_3) \oplus (x_3 \odot y_1) \oplus (x_1 \odot y_3) \oplus z_3, \\ c_3 &= (x_1 \odot y_1) \oplus (x_1 \odot y_2) \oplus (x_2 \odot y_1) \oplus z_1. \end{aligned} \tag{1}$$

Interestingly, the addition of the third variable $z$ to the non-linear part $x \odot y$ guarantees the uniformity of the outputs. However, even if this gadget is "ideally implemented" in a single clock cycle (i.e., does not allow probes on intermediate computations such as $x_2 \odot y_2$, $x_2 \odot y_3$, ...), it is not 1−SNI nor even 1−NI.[6] For example, a single probe on $c_1$ (meaning $q_1 = 0$ and $q_2 = 1$) cannot be simulated with a single share per input. This is because the computation of $c_1$ requires two shares of $x$ and two shares of $y$, and there is no internal randomness in the gadget that can help the simulation. Note that by contrast, this gadget is 1−probing secure (since the $c_i$'s are independent of $x, y$ and $z$). So the standard notions of NI and SNI security cannot directly capture the low randomness requirements of 1st-order TIs. This is in fact natural since the main idea behind the security of 1st-order TIs is to leverage the uniformity of the shares. Therefore, and in order to exhibit a connection between TIs and composable masking schemes, we propose the following (slight) variation of existing NI/SNI definitions:

**Definition 4 (Pseudo−randomized gadgets).** *The pseudo−randomization* $\mathsf{G}'$ *of a circuit gadget* $\mathsf{G}$ *is defined as the gadget* $\mathsf{G}$ *modified such that any input share coming from a uniform encoding and appearing only once and as a monomial of degree one in the algebraic circuit description of the gadget* $\mathsf{G}$ *is removed from the gadget inputs and replaced by internal uniform randomness in* $\mathsf{G}'$*. We further denote these replaced monomials as pseudo−randomized monomials.*

[6] Actual implementations are admittedly not ideal, as will be discussed in Section 4.

**Definition 5 (Pseudo−$q$−NI and pseudo−$q$−SNI security).** *A circuit gadget* $\mathsf{G}$ *is pseudo−$q$−NI (respectively, pseudo−$q$−SNI) if and only if the pseudo-randomization* $\mathsf{G}'$ *of this gadget* $\mathsf{G}$ *is $q$−NI (respectively, $q$−SNI).*

Based on these definitions, we now have that the gadget of Equation 1 is pseudo−1−SNI, since the outputs $c_i$'s can be simulated thanks to uniform randomness. (We will show in Section 5.1 that this gadget is even pseudo−2−SNI). Of course, pseudo−SNI is a strictly weaker notion than SNI and it does not guarantee composability: it rather guarantees "pseudo−composability" in case the pseudo−randomized monomials are manipulated with care, which is in fact exactly what state-of-the-art (first-order) TIs exploit in a clever way.

We illustrate this fact based on the excellent survey of TIs given in [8]. Again ignoring glitches for now, one can observe that the gadget of Equation 1 fulfills the uniformity requirement by looking at Table 1. This is done by checking that each non-zero entry of the table equals $\frac{2^{n\cdot(d-1)}}{2^{m\cdot(d-1)}}$ with $n = 3$ and $m = 1$ the function's input and output bit-sizes, respectively, and $d$ the number of shares.

**Table 1.** Number of times that the output shares $(c_1, c_2, c_3)$ occur for a given input $(x, y, z)$ in a 3-share Toffoli TI gate (as per [8], Section 3.3, Figure 3.1).

| | $(c_1, c_2, c_3)$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $(x, y, z)$ | 000 | 011 | 101 | 110 | 001 | 010 | 100 | 111 |
| 000 | 16 | 16 | 16 | 16 | 0 | 0 | 0 | 0 |
| 010 | 16 | 16 | 16 | 16 | 0 | 0 | 0 | 0 |
| 100 | 16 | 16 | 16 | 16 | 0 | 0 | 0 | 0 |
| 111 | 16 | 16 | 16 | 16 | 0 | 0 | 0 | 0 |
| 001 | 0 | 0 | 0 | 0 | 16 | 16 | 16 | 16 |
| 011 | 0 | 0 | 0 | 0 | 16 | 16 | 16 | 16 |
| 101 | 0 | 0 | 0 | 0 | 16 | 16 | 16 | 16 |
| 110 | 0 | 0 | 0 | 0 | 16 | 16 | 16 | 16 |

Now imagine that we want to build a $3 \times 3$-bit function based on this Toffoli gate. In a first case, we use $d = (x \odot y) \oplus z$, $e = x$, $f = y$. In a second case we use $d = (x \odot y) \oplus z$, $e = x$, $f = z$. By computing tables similar to Table 1 for these two functions (that we do not reproduce for brevity), we find out that the non-zero entries equal 1 (as required by the uniformity property) in the first case, and 2 on the second case. This indicates that the first function's output shares can be directly re-used as a uniform input sharing for another function. By contrast, the second function's output shares are not guaranteed to be re-usable, which is intuitively explained by observing that it forwards the pseudo–randomized monomial $z$ (therefore potentially enabling a careless manipulation).

We insist that our motivation for defining pseudo−SNI is explanatory. Namely, this definition allows us to justify the low randomness requirements of first-order TIs based on the probing model, while also putting forward their important conceptual differences with standard composable masking schemes (which we will

use next to motivate the need of a new model). That is, as such the pseudo–composability of a single gadget such as the Toffoli gate of Equation 1 is not sufficient to ensure composability. But by combining this pseudo–composability with a more global condition (namely the uniformity of all the shares to be re-used), first-order TIs can strongly limit their randomness requirements.

### 3.2 The number of shares vs. cycle count tradeoff

In general, obtaining function decompositions that guarantee non-completeness and uniformity is a non-trivial task [10]. For most TIs, this comes at the cost of additional shares (e.g., in the previous instance 1st-order security is obtained with three shares rather than the minimal two). In this subsection, we show that a natural tradeoff exists for TIs, between their number of shares and cycle count. For this purpose, we start from the two main observations given next:

1. The threshold implementation of complex circuits (e.g., S-boxes) generally results from a composition of simpler stages of gadgets, where memory elements separate the stages in order to "block" the propagation of glitches. But nothing prevents trying to split the gadgets more than what is strictly needed for glitch-freeness (e.g., the Toffoli gate in the previous section was implemented in one cycle, but one could also do it in two cycles).
2. In general, the correctness property is not needed for the intermediate stages of the computation: it is sufficient that the final result is correct.

Based on these observations, it is easy to see that one possible solution to implement $\mathsf{f}(x, y, z) = (x \odot y) \oplus z$ in only two shares is given by:

$$\begin{aligned} c_1 &= \Big[\big[(x_1 \odot y_1) \oplus z_1\big] \oplus (x_1 \odot y_2)\Big], \\ c_2 &= \Big[\big[(x_2 \odot y_2) \oplus z_2\big] \oplus (x_2 \odot y_1)\Big], \end{aligned} \tag{2}$$

where the [.] parentheses are used to denote the clock cycles. Functionally, the multiplication is similar to the ISW one, but it again exploits the XOR with $z$ in order to make the gadget pseudo−composable. Such an implementation is illustrated in Figure 2, where the circled boxes are functions and the darker rectangles are memory elements. We now have that only the result in the second stage is correct. By contrast, the intermediate stage is not (it is not even a deterministic function of the unmasked inputs). Yet, each stage of this decomposition is non-complete and uniform (w.r.t. their inputs). As in the previous subsection, this can be explained by observing that each stage of the decomposition is pseudo−1−SNI, and that the pseudo-randomized monomials are only used once, which provides a probing-based explanation to the recent results in [14].

### 3.3 Generic decomposition for unbalanced Feistel networks

To conclude this section, we observe that the Toffoli gate in the previous subsection can be viewed as an unbalanced Feistel network with 3 branches and a degree 2 function. We next systematize it to unbalanced Feistel networks with

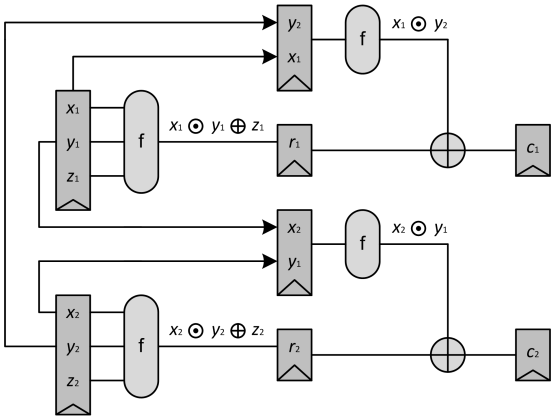


**Fig. 2.** Example of 2-share/2-cycle decomposition of a Toffoli gate.

$\lambda$ inputs in the left branch (entering the function $\mathsf{f}$) and $\rho$ inputs in the right branch. More precisely, the function we want to protect can be written as:

$$\mathsf{f}(\underbrace{u, v, \ldots}_{\lambda \text{ inputs}}) \oplus \underbrace{x \oplus y \oplus z \oplus \ldots}_{\rho \text{ inputs}}.$$

As illustrated in Appendix A, Figure 10, a TI can be obtained for such a function with 2 shares in at most $\frac{2^\lambda}{2\rho} + 1$ stages, which we detail as follows. First observe that if we use two shares, we have $2^\lambda$ different "non-complete sets" of shares, containing only one share of each secret input. On each of these non-complete sets, we may need to compute a non-complete component function $\mathsf{f}_i$ (and strictly have to when $\mathsf{f}$ is of degree $\lambda$). Thus we have $2^\lambda$ partial results that we need to add to the right part of the input (that does not go through $\mathsf{f}$). Next observe that in a single stage it is possible to add the output of $2\rho$ component functions to the $2\rho$ (untouched) shares of the right branch (which play the same role as the $z$ bit in the previous subsection). This implies that we (roughly) need $\frac{2^\lambda}{2\rho}$ stages to implement the full function. Note that the generalized Feistel structure ensures that each stage is a bijection of the shares, which guarantees the uniformity property, as mentioned in [11]. Eventually, we need one more stage to compress the right branch of the network (i.e., to add the first shares together).

Concretely, this generic composition allows us to slightly refine the exhaustive search of Bilgin et al. in [10], by exhibiting different tradeoffs between the number of shares and the number of cycles in the TIs of 4-bit S-boxes. Keeping this previous work's notations where $\mathcal{Q}^4_{\mathsf{xxx}}$ denotes the the quadratic class indexed $\mathsf{xxx}$, and $\mathcal{C}^4_{\mathsf{xxx}}$ denotes the cubic class indexed $\mathsf{xxx}$, we first remark that some classes can be written as an unbalanced Feistel network (see Appendix B). By checking the uniformity of various compositions of such networks, we found that $\mathcal{Q}^4_4, \mathcal{Q}^4_{12}, \mathcal{Q}^4_{293}, \mathcal{Q}^4_{294}$, and $\mathcal{Q}^4_{299}$ can be masked with two shares in two stages

(rather than three shares and one stage in [10]). We also found that $C_1^4$ and $C_{13}^4$ can be masked with two shares and four stages (rather than four shares and one stage in [10]). The descriptions of our decompositions are in Appendix C.

## 4 Robust and composable probing security

In order to motivate our new model, we now argue that higher-order secure gadgets combining resistance against physical defaults and composability are not straightforward to design with existing tools. For this purpose, we once more start by ignoring physical defaults and consider the next 3-share gadget:

$$\begin{aligned} c_1 &= (x_1 \odot y_1) \oplus (x_1 \odot y_2) \oplus (x_1 \odot y_3) \oplus r_1, \\ c_2 &= (x_2 \odot y_1) \oplus (x_2 \odot y_2) \oplus (x_2 \odot y_3) \oplus r_2, \\ c_3 &= (x_3 \odot y_1) \oplus (x_3 \odot y_2) \oplus (x_3 \odot y_3) \oplus r_3. \end{aligned} \tag{3}$$

This is actually a straightforward implementation of Algorithm 1 in one cycle, with a simplified refreshing that just sums a share of 0 to the partial products. Clearly, if one assumes that no information is leaked about (i.e., no probes are given on) the internal values $x_i \odot y_j$ and their intermediate sums, this implementation is $2-$SNI (the proof is identical to the one given in Section 5.1, Lemma 3 for the gadget of Equation 1). The problem here is that such a model is known to be unrealistic. More precisely, a concrete hardware implementation may actually leak something about the intermediate values via glitches (or other physical defaults). In this respect, note that in the TI gadget of Equation 1, such an issue is mitigated by ensuring an additional property of non-completeness, which is not ensured by Equation 3. This shows that composability alone is not enough to reason about higher-order masked implementations in hardware.

Taking the opposite side of the problem, it has been shown that while non-completeness and uniformity are sufficient conditions for the composability of first-order glitch-resistant circuits, it does not easily scale to higher security orders. More precisely, while so-called higher-order TIs maintain security against glitches [9], they suffer from composability issues [43]. This shows that these two properties alone are not enough to reason about higher-order masked implementations in hardware. As later discussed in [44], the addition of resfreshing gadgets is needed for this purpose. Intuitively, this is easily understood based on the discussion of pseudo–composability in the previous section. Namely, the relevance of the uniformity property is actually related to the fact that in the context of first-order threshold implementations, one only has to prevent univariate attacks (i.e., attacks exploiting a single probe or targeting a single point in time of the leakage traces). By contrast, higher-order security requires considering multivariate attacks (i.e., attacks exploiting multiple probes or targeting multiple points in time of the leakage traces), which are not captured by the (original) definition of uniformity. In this respect, one option could naturally be to try generalizing the notion of uniformity. But this would imply imposing a

more global (and hard to assess) condition to the implementations as $q$ increases (i.e., to get away from the concept of composability pursued in this work).

Based on these observations, we can summarize the state-of-the-art higher-order masking schemes as follows. On the one hand TIs maintain good security against shares' recombinations due to glitches but do not provide a systematic way to determine the type and amount of refreshings needed to guarantee composability. On the other hand, the probing model provides a way to reason about composability thanks to the notion of SNI but, in its original description, this model does not capture physical defaults such as glitches. In the following, we show that there is a natural generalization of the probing model that allows combining the best of these two worlds, i.e., to analyze masking gadgets that are both composable and robust against a wide class of physical defaults.[7]

### 4.1 Modeling physical defaults

As a starting point, we recall that the analysis of physical security properties always requires a description of the target. This is in fact already true in the (most abstract) probing model, where one generally captures implementations as lists of (leaking) operations. Quite naturally, this requirement becomes more critical if one wishes to obtain some robustness against physical defaults. Since our goal is to incorporate a possibly large set of such defaults in our abstractions, we therefore need to start by describing them in a more detailed manner.

For this purpose, we use the example of threshold implementation in Figure 3 where the three types of physical defaults listed in introduction are illustrated. First, combinatorial recombinations (e.g., glitches) potentially mix (and therefore recombine) the inputs of the component functions $\mathsf{f}_i$. Second, memory recombinations (e.g., transitions) potentially mix (and therefore recombine) the content of the memory elements in consecutive invocations/cycles. In Figure 3, this would typically happen if the same memory gate is used to store the $y_i$'s by erasing the $x_i$'s. Third, routing recombinations (i.e., couplings) potentially mix (and therefore recombine) the shares manipulated by adjacent wires.

In order to capture physical defaults, we propose to use a natural tweak of the probing model where probes are specifically or generically *$\epsilon$-extended*. Generic extensions mean that the model is independent of the circuit topology, specific extensions are dependent on it. More precisely, we consider the following:

**Specific model for glitches.** *For any $\epsilon$-input gadget $\mathsf{G}$, combinatorial recombinations can be modeled with specifically $\epsilon$-extended probes so that probing any output of the gadget allows the adversary to observe all its $\epsilon$ inputs.*

[7] Robust and composable gadgets are sufficient for designing higher-order secure masked implementations. Yet, it is not always needed that all gadgets in an implementations satisfy both properties. So our results leave ample room for performance optimizations, by taking advantage of formal methods to analyze full codes [2], exploiting pseudo–randomized monomials in the higher-order setting, or observing that composability or physical defaults may be too small for being exploitable [23, 21].

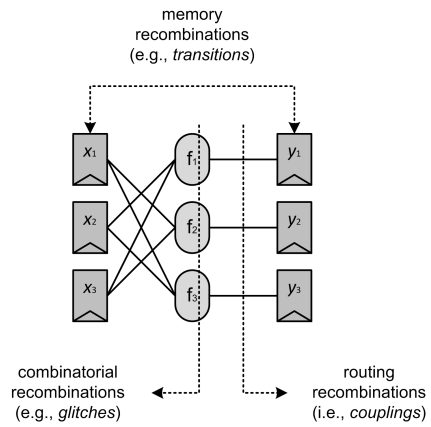


**Fig. 3.** Physical defaults for an exemplary TI (borrowed from [4]).

Note that, as first detailed in [25] and recently revisited in [7], such a (worst-case) recombination actually happens in most cases for standard CMOS circuits. As a result, it directly imposes a natural restriction on the topology of (robust) masked circuits. Namely, defining the shares fan-in of a gadget as the number of shares of a sensitive variable at its inputs, we generally need that *the shares fan-in of each gadget in a masked circuit should be limited* (for example, the shares fan-in of the 1st-order TIs in Section 3.1 is 2), and *any composition of gadgets with limited shares fan-in should be separated by memory elements*. The latter requirement directly comes from the fact that composing gadgets without adding memory elements in between may further increase the shares fan-in (as well known in the TI literature [8]). In the rest of the paper, we will therefore consider masked circuits topologies that follow these minimum guidelines.

**Specific model for transitions.** *For a memory cell* $\mathsf{m}$*, memory recombinations can be modeled with specifically* 2*-extended probes so that probing* $\mathsf{m}$ *allows the adversary to observe any pair of values stored in 2 of its consecutive invocations.*

Note that this model exactly corresponds to the transition-based leakages in [1] which have been shown a good abstraction of memory recombinations.

**Specific model for couplings.** *For any set of adjacent wires* $\mathcal{W} = (w_1, \ldots, w_d)$*, routing recombinations can be modeled with specifically* $c$*-extended probes so that probing one wire* $w_i$ *allows the adversary to observe* $c$ *wires adjacent to* $w_i$*.*

Note that $c = 1$ means no couplings. Admittedly, this last physical effect is the most prospective one. In general, we insist that the previous models are not expected to perfectly reflect physical defaults, but to capture them sufficiently well to guide algorithmic designs with better robustness against them.

The previous models can be changed into their generic version by extending the probes without link to the circuit topology (excepted its maximum shares fan-in). For a circuit with maximum shares fan-in $f$, generic glitches then "trans-

late" any probe in $f$ probes (independent of whether they correspond to the same gadget), transitions translate any probe in two probes (independent of whether they correspond to the same memory cell), and generic couplings translate any probe into $c$ probes (independent of whether they observe adjacent wires).

We next define $(g, t, c)$–robust $q$–probing secure (or $q$–NI/SNI) circuits as circuits that are secure in the $q$–probing model (or $q$–NI/SNI) with an adversary whose probes are (specifically or generically) extended with glitches if $g = 1$ (if $g = 0$ combinatorial recombinations are assumed avoided at the implementation level), with transitions if $t = 1$ (if $t = 0$ memory recombinations are assumed avoided at the implementation level) and with couplings if $c > 1$ (if $c = 1$ routing recombinations are assumed avoided at the implementation level).

*Remark.* It is important to emphasize that robust probing security does not guarantee security against physical defaults. It only does it if the previous models capture the physical reality to a sufficient extent. In this sense, robust probing security should really be seen as analogous to (and an extension of) the non-completeness property in TIs. For example, by guaranteeing $(1, 0, 1)$–robust $q$–probing security, we ensure that glitches in masked circuits are not able to recombine shares exactly as guaranteed by the non-completeness property (yet with the advantage of allowing reasoning about composability – see next). This is also explicitly implied by the fact that the (abstract) robust probing model does not allow testing physical recombinations (the bounded moment or noisy leakage models are needed for that). As illustrated in Figure 1, probing security is only the first (necessary) step on the way towards noisy leakage security.

### 4.2 Worst-case generic lemma

The previous model directly implies the following worst-case lemma (which corresponds to a careless implementation where all physical defaults occur):

**Lemma 1.** *Any $2fcq$–probing secure circuit with maximum shares fan-in $f$ is $(1, 1, c)$–robust $q$–probing secure with generically extended probes.*

Note that this lemma only holds for probing security (not for NI/SNI) for a similar reason as in Section 3.1. As will be clear in Section 5.2, arguing about robust composability requires a more subtle discussion of the extended probes' positions. The proof is obvious: it simply exploits the fact that any probe is then "multiplied" by $f$ (because of glitches) $\times$ 2 (because of transitions) $\times$ c (because of couplings). It directly implies that one needs $2fcq+1$ shares to obtain robust $q$–probing secure circuits. Naturally, one may expect that exploiting an appropriate circuit topology leads to better results, which we will discuss in Section 5. Beforehand, we discuss physical defaults' combinations and whether a more specific physical model may already improve the previous lemma.

### 4.3 Physical defaults combination

Looking back at Figure 3, its is clear that some types of physical defaults' combinations are unavoidable. In particular, there is no physical argument allowing one to rule out the fact that couplings can be combined with transitions if the adver-

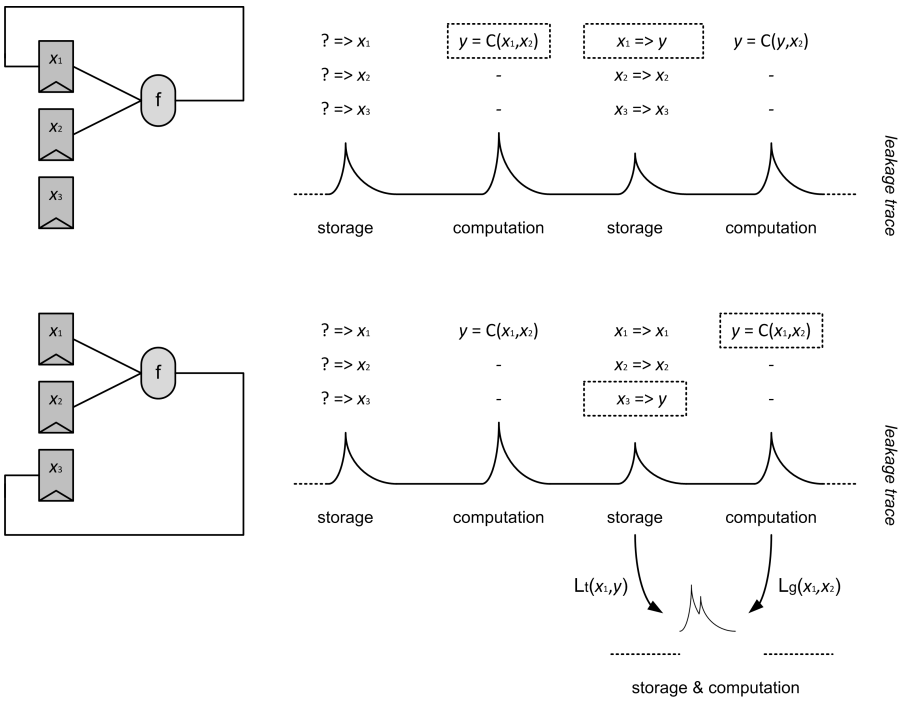


**Fig. 4.** Exemplary combinations of transitions and glitches.

sary probes adjacent memory cells. And similarly, one could combine couplings and glitches: take for example an adversary probing $y_1$ with a glitch-extended probe (allowing him to observe $x_2$ and $x_3$) and assume that the wire carrying $x_2$ is coupled with $x_1$ in Figure 3. So for $f = 2$, a loss by a factor $2c$ in the generic Lemma 1 seems founded, and the main question is whether the additional factor 2 corresponding to the combination of transitions and glitches is too.

In order to illustrate our discussions, we will take the two simple circuit examples given in Figure 4, which allow two main observations:

First, certain transitions can be simulated by glitches. Take for example the upper circuit of the figure: in the second storage cycle, the top memory cell witnesses a transition $x_1 \Rightarrow y$. But a glitch-extended probe on $y$ allowing an adversary to observe $x_1$ and $x_2$ can simulate this transition, since $y = \mathsf{f}(x_1, x_2)$. So there is no combination of transitions and glitches possible in this case. Yet, this positive result does not always hold since, for example, the $x_3 \Rightarrow y$ transition in the lower circuit cannot be simulated by a glitch-extended probe.

Second, and in general, transitions and glitches cannot be combined if the leakage samples corresponding to the storage and computation in a circuit are independent of each other. Such independent leakage samples would correspond to the oversimplified model of the top figure, where the leakage traces have

independent parts corresponding to the storage and computation of intermediate values. If that model was perfect (which is admittedly not expected in practice), the adversary would have to choose between a glitch-extension and a transition-extension of his probes (leading to a factor $2c$ rather than $4c$ in Lemma 1).

Based on these observations, we can conclude that the main question regarding the combination of transitions and glitches in masked implementations relates to their dependencies, which leads to another pair of important facts:

First, in practice computations within gadgets occur extremely fast after the storage, leading these two steps to overlap, as at the bottom of Figure 4.

Second, such an overlap can be viewed as a type of parallel implementation (since the leakage samples due to the combinatorial gates are combined with those of the memory gates), which are known to be difficult to capture with the probing model and are better reflected by the bounded moment model [4].

In this context, we first note that whether the combination of transition-based leakages and glitch-based leakages, denoted as $\mathsf{L}_\mathrm{t}(.)$ and $\mathsf{L}_\mathrm{g}(.)$ in Figure 4, reduce the security order in the bounded moment model essentially depends on the algebraic degree of the combination function. As shown in [4], Lemma 1, a linear combination of $\mathsf{L}_\mathrm{t}(.)$ and $\mathsf{L}_\mathrm{g}(.)$ (e.g., a sum in $\mathbb{R}$) will not reduce this security order. By contrast, a non-linear one will. We then just observe that such a non-linear combination of $\mathsf{L}_\mathrm{t}(.)$ and $\mathsf{L}_\mathrm{g}(.)$ in fact exactly corresponds to the couplings of Section 4.1. Namely, couplings typically imply that the leakage of adjacent wires (or combinatorial gadgets, memory gates) are combined non-linearly, which is reflected by the extension factor $c$ in the probing model, and is captured by an algebraic degree $c$ for the combination function in the bounded moment model. This reasoning leads us to the following informal lemma:

**Lemma 2.** *Any* $\max(2, f)q$*–probing secure circuit with maximum shares fan-in* $f$ *is secure of order* $q$ *in the bounded moment model if it has transitions & glitches but no non-linear combinations of transitions & glitches (i.e., couplings).*

Admittedly, this lemma depends on a hardware assumption which is not expected to perfectly hold in practice. Yet, we believe it leads to interesting guidelines for cryptographic hardware designers. It suggests that if couplings can be kept negligible within an implementation (which depends on the noise level: see [22], Section 4.2), then any combination of glitches and transitions that occurs should not be detrimental to its concrete security level. We next describe experiments confirming that there are contexts in which this assumption holds.

### 4.4 Experimental validation

We implemented a first-order TI of the PRESENT S-box using two stages similar to the one pictured in Figure 3 and following the guidelines in [36], Figure 3, in a Xilinx Spartan-6 FPGA that we measured on the SAKURA-G board.[8] It provides built-in attack points to measure the voltage drop over a $1\Omega$ shunt

[8] http://satoh.cs.uec.ac.jp/SAKURA/index.html

resistor placed in the Vdd path of the target FPGA that, by means of the corresponding voltage regulator, was supplied at 1.2 V. We ran our device at 3MHz and performed measurements by means of a Teledyne Lecroy HRO66Zi WaveRunner 12-bit digital oscilloscope (DSO) at a sampling rate of 500 MS/s and a bandwidth limit of 20 MHz to reduce the environmental noise. Besides, a passive probe (i.e., a SMA-to-BNC coaxial cable) that avoids the additional noise induced by, e.g., active components in differential probes, was used in our experiments. This allowed us to first reproduce the previous results of Moradi and Wild. We then tweaked the design in two different manners.

First, rather than using six different registers to store the input and output shares $x_1, x_2, x_3, y_1, y_2, y_3$, we used the same register to store $x_1$ and $y_1$. Concretely, this change is expected to lead to first-order leakages due to transitions. Second, we additionally refreshed the output of $\mathsf{f}_1$ with uniform randomness before storing it in the re-used register storing $x_1$. Interestingly, this refreshing should not improve the security order in case glitches and transitions are combined (since a glitch-extended probe on $y_1$ should then give this additional randomness to the adversary for free), and it should improve it if glitches and transitions are not combined (since the adversary should then choose between a glitch-extended probe on $y_1$ before it has been stored in the register, and a transition-extended probe on $y_1$ after it has been stored in the register).

Based on these implementations, and since only interested in the security order of our designs, we launched CRI's non-specific T-test to detect differences between the traces corresponding to fixed and random inputs [27, 17]. The results of these experiments are reported in Figures 5 and 6. For completeness, an exemplary trace is given in Appendix A, Figure 11. Figure 5 exhibits a first-order leakage (presumably due to transitions) when no refreshing is used. Figures 6 suggests the cancellation of this first-order leakage when the refreshing is activated. More precisely it shows a reduction of this first-order leakage to negligible for the noise level in our measurements, since an easier-to-detect second-order leakage is detected (so the best adversarial strategy is then to estimate a second-order moment, as per [22], Section 4.2). The latter confirms that there are certain types of transitions that do not combine detrimentally with glitches. The further investigation of these combinations is an interesting research direction.

## 5 Concrete constructions

We now consider the case of a couple of popular constructions from the literature and discuss if/how they differ from the previous worst-case predictions.

### 5.1 Equation 1 is pseudo$-(1,0,1)-$robust $1-$probing secure

As a first example of application of our lemma, we can consider the 1st-order TI gadget discussed in Section 3.1, which is a typical basis for the first-order TI of block cipher S-boxes. In our hardware implementation case, registers are

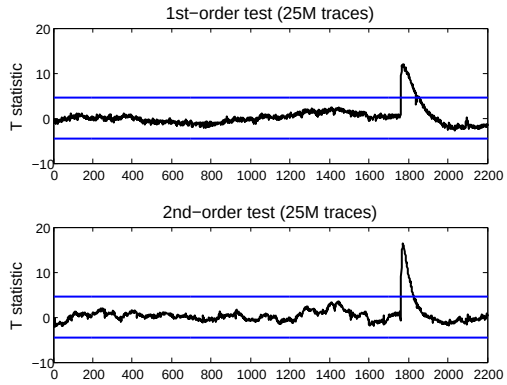


**Fig. 5.** Non-specific T-test results for a first-order TI of the PRESENT S-box tweaked so that the register storing $x_1$ and $y_1$ in Figure 3 is re-used without refreshing.

selected in order to avoid transition issues so that $t = 0$ in the robust probing model. That is, the Toffoli gadget is computed in one cycle and its outputs are stored in memory gates. The scheme is pseudo–2–SNI, as it is shown next.

**Lemma 3.** *The TI gadget in Equation 1 is pseudo–2–SNI.*

*Proof.* According to Definition 5, in order to prove that the gadget in Equation 1 is pseudo–2–SNI, we need to prove that its pseudo-randomization, let it be $\mathsf{G}$, is 2–SNI. The algorithm $\mathsf{G}$ corresponds to Equation 1, with the difference that the inputs are only the shares $x_1, x_2, x_3, y_1, y_2, y_3$ and the values $z_1, z_2, z_3$ are assigned uniformly at random. Let $\Omega = \{w_1, w_2\}$ be a set of 2 adversarial observations on the pseudo–randomized gadget $\mathsf{G}$. Since the implementation of the scheme is only in one cycle, the adversary does not have internal probes. Therefore the probes can only lie in one of the following two groups:

(1) the input shares $x_i$ and $y_j$ with $i, j \in \{1, 2, 3\}$;
(2) the output shares $c_1, c_2, c_3$.

Let $q_1$ (resp., $q_2$) be the number of observations on the input (resp., output) values (with $q_1 + q_2 \leq 2$). We first define two sets of indices $I$ and $J$ such that $|I| \leq q_1$ and $|J| \leq q_1$ and the values of the probes can be perfectly simulated given only the knowledge of $(x_i)_{i \in I}$ and $(y_j)_{j \in J}$. The sets are constructed as:

- Initially $I$ and $J$ are empty.
- For every probe as in group (1), add $i$ to $I$ and $j$ to $J$.

Since the adversary is allowed to make at most $q_1$ probes on the input values, it holds that $|I| \leq q_1$ and $|J| \leq q_1$. In order to prove the SNI property, we now show the simulation phase, by distinguishing three different cases.

1. If $q_1 = 2$, then the probes $w_1$ and $w_2$ are both in group (1) and, by definition of the set $I$, the simulator has access to the observed shares $x_i$ and $y_i$.

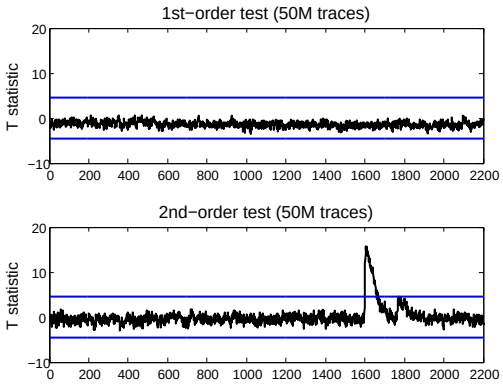


**Fig. 6.** Non-specific T-test results for a first-order TI of the PRESENT S-box tweaked so that the register storing $x_1$ and $y_1$ in Figure 3 is re-used with refreshing.

2. If $q_1 = 1$ and $q_2 = 1$, then wlog $w_1$ is in group (1) and $w_2$ is in group (2). By definition of the sets $I$ and $J$, the simulator has again access to the observed shares, therefore $w_1$ can be perfectly simulated. As for $w_2$, thanks to the presence of the random value $z_h$ with $h \in \{1, 2, 3\}$, the probe can be simulated by assigning a random and independent value.
3. If $q_2 = 2$, then $w_1$ and $w_2$ are both in group (2) and they are of the form $x_i y_i + x_i y_j + x_j y_i + z_i$ with $i \in \{1, 2, 3\}$. Since the (pseudo–randomized) shares of $z$ appearing in their computation are different in each output share, the simulator can assign $w_1$ and $w_2$ to a random and independent value.

In all the cases listed above, the probes $w_1$ and $w_2$ can be perfectly simulated with $q_1$ shares of the input. We finally note that if $|\Omega| = 1$, then the simulation of the probe trivially follows the procedure of one of the previous cases. Therefore we conclude that the gadget G is 2–SNI, completing the proof. □

Combining this result and Lemma 1, it follows that this TI gadget is pseudo–(1,0,1)–robust 1–probing secure with 3 shares. In other words, it uses an additional share to prevent glitches, exactly following worst-case analysis. Note that (as mentioned in Section 4.2) the resulting gadget is not pseudo–(1,0,1)–robust 1–SNI (since glitch-extended probes cannot be simulated with one share per input). Yet, it is sufficient to argue about the security of TIs for full ciphers. That is, assuming the uniformity condition in Section 3.1 is fulfilled, such "full TIs" are pseudo–2–SNI without glitches. And then, by invoking Lemma 1 only once, we have that this full implementation is (1,0,1)–robust 1–probing secure.

### 5.2 ISW is $(1, 0, 1)$-robust $q-$SNI with $q + 1$ shares in 2 cycles

As a second example, we show that when moving to higher-orders the ISW multiplication actually beats our worst-case lemma, and therefore provides an excellent solution for robust and composable gadgets. More precisely, it is proven

in [3] that this algorithm is (0,0,1)–robust $q$–SNI, using $q+1$ shares. We next show formally that the scheme is additionally secure against glitches (i.e., it is (1,0,1)–robust $q$–SNI), if one optimally follows the guidelines of Section 4.1 and limits the shares fan-in to 1. For this purpose, we will consider an implementation of the ISW multiplication in two cycles illustrated in Figure 7 for the case with 3 shares and security order 2 (the proof is naturally given for all orders).

In this respect, it is first important to recall that compared to the previous section, we now use the specific model for glitches, that exploits this particular circuit topology. Concretely, it means that for the implementation illustrated in Figure 7, the adversary can access the following three types of probes:

- Internal (3-extended) probes $p_{i,j}$ on the $u_{i,j}$'s giving access to three shares: namely $a_i$, $b_j$ and the corresponding value of the randomness matrix.
- Internal (3-extended) probes $p_i$ on the $c_i$'s giving access to $u_{i,1}$, $u_{i,2}$, $u_{i,3}$.
- Output (non-extended) probes on the $c_i$'s giving only access to one share.

Note that in this model, an adversary willing to obtain a single value (e.g., an $r_{i,j}$) will simply use a (more informative) extended probe including this value. Note also that despite giving 3-extended probes to the adversary, we do not break the shares fan-in limit of 1. Besides, and quite importantly, the $c_i$ shares appear twice in the list: either as internal probes which can be glitch-extended, or as external probes which are not glitchy since stored in an additional memory element. Despite not being necessary for probing security, the additional output memory elements storing the $c_i$ shares are strictly necessary in order to obtain a robust and composable gadget, which we formalize as follows:

**Lemma 4.** *The multiplication gadget in Algorithm 1 implemented in two cycles (one for the $u_{i,j}$ values, one for the $c_i$ values) is (1,0,1)–robust q–SNI.*

The proof of the Lemma is in Appendix D. In order to provide some intuition, we illustrate it with the 3-share implementation of Figure 7. An adversary attacking the internal values of this scheme can observe at most two of the following extended probes, each of them allowing him to see between 2 and 3 shares:

- *1st stage:*

$$\begin{array}{lll} p_{1,1} := (a_1, b_1, 0), & p_{1,2} := (a_1, b_2, r_{1,2}), & p_{1,3} := (a_1, b_3, r_{1,3}), \\ p_{2,1} := (a_2, b_1, r_{1,2}), & p_{2,2} := (a_2, b_2, 0), & p_{2,3} := (a_2, b_3, r_{2,3}), \\ p_{3,1} := (a_3, b_1, r_{1,3}), & p_{3,2} := (a_3, b_2, r_{2,3}), & p_{3,3} := (a_3, b_3, 0). \end{array}$$

- *2nd stage:*

$$p_1 := (u_{1,1}, u_{1,2}, u_{1,3}), \qquad p_2 := (u_{2,1}, u_{2,2}, u_{2,3}), \qquad p_3 := (u_{3,1}, u_{3,2}, u_{3,3}).$$

Alternatively, attacking output shares allows the observation of shares $c_1, c_2, c_3$. In the simulation, we therefore distinguish the following possible cases:

1. Both probes are on the internal shares (e.g., $p_{1,2}, p_1$).

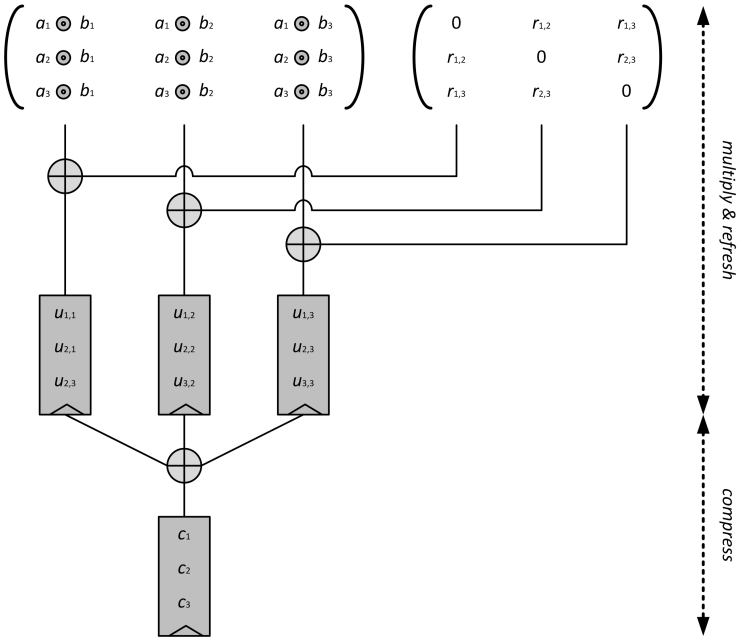


**Fig. 7.** $(1, 0, 1)$-robust 2-SNI implementation of ISW in 2 cycles.

2. One probe is internal, the other is on the output shares (e.g., $p_{1,2}, c_1$).
3. Both probes are on the output shares (e.g., $c_1, c_2$).

In the first case, and according to the proof, we construct the set of indices $I = \{1\}$ and $J = \{1, 2\}$. In the simulation phase we assign $r_{1,2}$ to a random value and we can perfectly compute $p_{1,2}$ by having access to $a_1$ and $b_2$. As for $p_1$, we perfectly simulate the first component $u_{1,1}$ by using $a_1$ and $b_1$; since $2 \in J$ and $2 \notin I$ we can use the components of the probed $p_{1,2}$ to simulate the second component $u_{1,2}$; and since $3 \notin J$ we can pick a uniform and random value for simulating the third component $u_{1,3}$. In the second case, we have $I = \{1\}$ and $J = \{1, 2\}$. We simulate $p_{1,2}$ as before and we assign a uniform and random value to $c_1$, thank to the presence of the random bit $r_{1,3}$. In the third case, since $c_1$ depends on the random bit $r_{1,3}$ which does not appear in the computation of $c_2$, and $c_2$ depends on the random bit $r_{2,3}$ which does not appear in the computation of $c_1$, we can simulate both shares as random and independent values.

We finally point out that the success of the simulation for the output values is due to the clever distribution of the random bits. Indeed each output share depends on a number of distinct random bits equal to the security order, and these random bits appear a second time in the computation of only one different output share each. This allows us to simulate the output probes with a random and independent value, and therefore to use the required number of input shares in order to satisfy the definition of SNI. In this respect, it is important to recall

that without the second register to store the $c_i$ shares, the output probes could be glitch-extended. In this case it would not be possible to satisfy the definition of SNI (i.e., the gadget would not be robust and composable simultaneously). It is an interesting open problem to determine whether such robust and composable gadgets could be obtained with less randomness (e.g., as discussed in [6]).

### 5.3 Glitch locality principle

The previous proof highlighted that in order to obtain robust and composable implementations of the ISW multiplication, it is necessary that their outputs $c_i$'s are stored in memory gates, in order to stop the propagation of glitches in the circuit. This leads to the following useful formalization:

**Lemma 5.** *If a gadget* $\mathsf{G}$ *storing its outputs in registers is both* $(1,0,1)-$*robust* $q-$*NI and* $q-$*SNI (without glitches), then it is also* $(1,0,1)-$*robust* $q$*-SNI.*

*Proof.* By separating the probes between $q_1$ internal and $q_2$ output ones, we have that: $(i)$ the internal probes can be simulated with $q_1$ shares per input since the gadget is $(1,0,1)-$robust $q_1-$probing secure (with $q_1 \leq q$), and $(ii)$ the $q_2$ probes can be simulated with $q_1$ input shares since the gadget is $q-$SNI without glitches (and the output probes cannot be extended). □

Intuitively, this last lemma suggests that the glitch issue is essentially "internal" to the masking gadgets. As long as registers are inserted after those gadgets, a designer can deal with glitch robustness (captured with the robust NI notion) and composability (captured with the SNI notion) independently. This observation may facilitate the automated search of robust and composable gadgets.

## 6 Practical security evaluation

We conclude this work by analyzing hardware implementations of the multiplication proven (1,0,1)–robust $q$–SNI with $q+1$ shares in the previous section.

### 6.1 Architectural guidelines & related works

Lemma 4 and Figure 7 directly suggest simple architectural guidelines for implementing an ISW multiplication gadget in two cycles. Before providing the results of practical security evaluations, we discuss the relations and differences between our proposal and previous works suggesting similar solutions.

First, the Consolidated Masking Scheme (CMS) of Reparaz et al. [44] implemented in [16] is exploiting an ISW-like multiplication with a simpler refreshing having only linear randomness requirements. As discussed in [4], such simple refreshing schemes are only guaranteed to be secure when used to refresh an immutable secret state (e.g., a block cipher master key) and are not composable. So while such implementations guarantee security against adversaries targeting a single tuple of shares, all bets are off against more powerful adversaries trying to combine all the exploitable leakage samples in a block cipher execution. Analyzing such a context is an interesting scope for further research.

Second, the Domain Oriented Masking (DOM) in [28, 29] exactly exploits the ISW multiplication with quadratic randomness requirements.[9] Besides the absence of security proof (which is of course our core contribution), the main difference with our proposal is that these implementations do not use an additional memory element to store the glitch-free $c_i$ values (which are directly used as inputs for other gadgets) and perform the multiplication in one cycle (see [28], Table 1). So while they are essentially SNI without glitches (up to Footnote 9), and glitch-resistant without composition, such implementations do not provide simultaneous guarantees of glitch-freeness and composability as we do.

Eventually, the parallel masking schemes recently introduced in [4] exploit the same "compute partial products – refresh – compress" structure as our implementation in Figure 7. So despite more specialized to software implementations, they can lead to similar 2-cycle hardware implementations as ours.

Summarizing, these related works show an appealing similarity between all existing attempts to reach higher-order security for masked implementations. In particular, they all compute the same partial products and mostly differ in the way they deal with composability and robustness thanks to refreshings and memory elements. In this respect, we believe the robust probing model brings three interesting features. First, it allows formally guiding implementation choices related to physical defaults that so far required engineering intuition. Second it can lead to implementations providing robustness against physical defaults and composability jointly. Third, it is versatile since by tuning the $g, t$ and $c$ parameters, we can ask more or less to hardware designers, therefore enabling trading risks of implementation surprises and performance overheads.

We insist that despite current masking gadgets have similarities, the proposals in this work already illustrate that the robust probing model allows guiding designs. For example, the implementation of Figure 7 has subtle differences with previously published ones. In this respect, we recall that not being robust and composable does not imply that an implementation is insecure. It only implies that its security evaluation is more complex, since one cannot leverage the local security order analysis of simple gadgets, and rather has to deal directly with the complexity of full implementations. So our main contribution is conceptual. Namely, the introduction of the robust probing model enables the systematic analysis of physical defaults in masking schemes, e.g., with formal methods such as [2]. The application of this model to more complex case studies (e.g., combinations of physical defaults) is an interesting topic for further investigations.

### 6.2 Experimental results

We implemented the 2-cycle architecture of Figure 7 in a Xilinx Spartan-6 FPGA for $d = 2$ and 3 shares, using exactly the same setup as in Section 4.4. Based on this setup, and since only interested in the security order of our designs, we

[9] Surprisingly, the authors did however use refreshings with linear randomness requirements for the re-sharing of dependent inputs (see [28], Table 1) which therefore raises similar composability questions as for the CMS implementations.

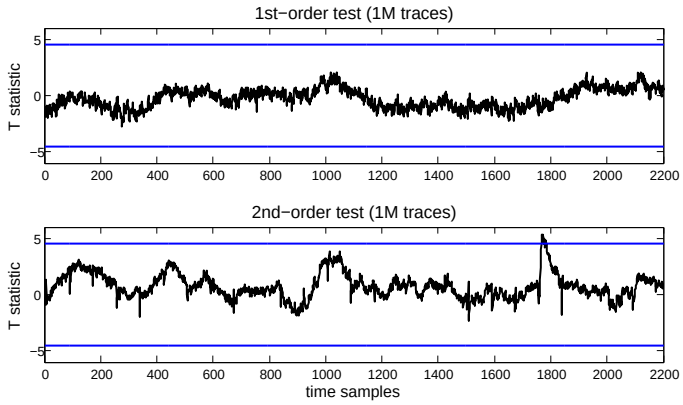


**Fig. 8.** Non-specific T-test results for $d = 2$ shares.

again launched CRI's non-specific T-test to detect differences between the traces corresponding to fixed inputs and random inputs [27, 17]. In the $d = 2$ case, we were able to spot second-order leakages with 1 million measurements (see Figure 11). In the $d = 3$ case, we used 10 millions measurements and exploited the tweak proposed in [47], Section 3.2, (i.e., we repeated 50 times the measurement of 250,000 traces and averaged them in order to mitigate the noise amplification due to masking and to speed up the detection). This allowed us to detect third order leakages (see Figure 8). None of our experiments suggested any lower-order leakage, confirming the results in [29]. Thanks to the composability of our implementations, we can therefore claim for the first time that a combination of such higher-order hardware gadgets will remain robust against glitches and maintain their security for full (e.g., block cipher) implementations, confirming the practical relevance of our model. We finally note that as all current higher-order and glitch-free implementations, our proposed architecture has significant memory requirements. Namely, we need $d^2$ registers to compute the multiplication in 2 cycles, which is similar to the 8-bit and 3-share case described in [16], Figure 3. Investigating the tradeoffs between memory requirements and cycle count (e.g., implementing the multiplication in $d+1$ cycles with $d+1$ registers), and more generally improving the cost and performances of higher-order masked implementations in hardware and software, are important open problems.

**Acknowledgments.** The authors are grateful to François Dupressoir, Tobias Schneider and the CHES 2017 reviewers for useful comments and feedback. Sebastian Faust is funded by the Emmy Noether Program FA 1320/1-1 of the German Research Foundation (DFG). François-Xavier Standaert is an associate researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in parts by European Commission through the ERC project 724725 (acronym SWORD) and the H2020 project REASSURE.

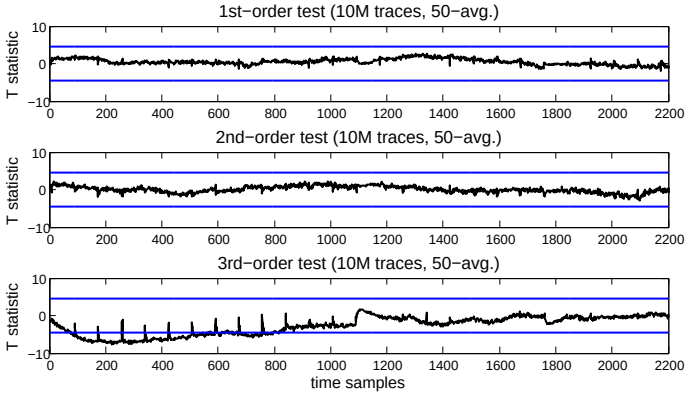


**Fig. 9.** Tweaked non-specific T-test results for $d = 3$ shares.

## References


1. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *CARDIS 2014*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
2. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Oswald and Fischlin [39], pages 457–485.
3. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM, 2016.
4. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017.
5. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Gierlichs and Poschmann [26], pages 23–39.
6. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.
7. Guido Bertoni and Marco Martinoli. A methodology for the characterisation of leakages in combinatorial logic. In *SPACE 2016*, pages 363–382, 2016.
8. Begül Bilgin. Threshold implementations: A countermeasure against higher-order differential power analysis. PhD Thesis, KU Leuven (Belgium) and U Twente (The Netherlands), May 2015.

9. Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014 , Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.
10. Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3 x 3 and 4 x 4 S-Boxes. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2012.
11. Erik Boss, Vincent Grosso, Tim Güneysu, Gregor Leander, Amir Moradi, and Tobias Schneider. Strong 8-bit S-boxes with efficient masking in hardware. In Gierlichs and Poschmann [26], pages 171–193.
12. Nicolas Bruneau, Sylvain Guilley, Zakaria Najm, and Yannick Teglia. Multi-variate high-order attacks of shuffled tables recomputation. In Güneysu and Handschuh [31], pages 475–494.
13. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO 1999*, volume 1666, pages 398–412. Springer, 1999.
14. Cong Chen, Mohammad Farmani, and Thomas Eisenbarth. A tale of two shares: Why two-share threshold implementation seems worthwhile - and why it is not. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 819–843, 2016.
15. Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? Cryptology ePrint Archive, Report 2016/1080, 2016. `http://eprint.iacr.org/2016/1080`.
16. Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In Gierlichs and Poschmann [26], pages 194–212.
17. Jeremy Cooper, Elke De Mulder, Gilbert Goodwill, Josh Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test vector leakage assessment (TVLA) methodology in practice (extended abstract). ICMC 2013. `http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf`.
18. Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *COSADE 2012*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.
19. Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007.
20. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Moriai [37], pages 410–424.
21. Joan Daemen. Spectral characterization of iterating lossy mappings. In *SPACE 2016*, pages 159–178, 2016.
22. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.

23. Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Oswald and Fischlin [39], pages 401–429.
24. François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 240–262. Springer, 2016.
25. Wieland Fischer and Berndt M. Gammel. Masking at gate level in the presence of glitches. In Rao and Sunar [42], pages 187–200.
26. Benedikt Gierlichs and Axel Y. Poschmann, editors. *CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*. Springer, 2016.
27. Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop, 2011. `http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf`.
28. Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. Cryptology ePrint Archive, Report 2016/486, 2016. `http://eprint.iacr.org/2016/486`.
29. Hannes Gross, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.
30. Vincent Grosso and François-Xavier Standaert. Masking proofs are tight (and how to exploit it in security evaluations). Cryptology ePrint Archive, Report 2017/116, 2017. `http://eprint.iacr.org/2017/116`.
31. Tim Güneysu and Helena Handschuh, editors. *CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*. Springer, 2015.
32. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
33. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
34. Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Rao and Sunar [42], pages 157–171.
35. Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does my device leak information? An a priori statistical power analysis of leakage detection tests. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 486–505. Springer, 2013.
36. Amir Moradi and Alexander Wild. Assessment of hiding the higher-order leakages in hardware - what are the achievements versus overheads? In Güneysu and Handschuh [31], pages 453–474.
37. Shiho Moriai, editor. *FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*. Springer, 2014.
38. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.
39. Elisabeth Oswald and Marc Fischlin, editors. *EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*. Springer, 2015.

40. Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-channel resistant crypto for less than 2, 300 GE. *J. Cryptology*, 24(2):322–345, 2011.
41. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.
42. Josyula R. Rao and Berk Sunar, editors. *CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*. Springer, 2005.
43. Oscar Reparaz. A note on the security of higher-order threshold implementations. Cryptology ePrint Archive, Report 2015/001, 2015. `http://eprint.iacr.org/2015/001`.
44. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.
45. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
46. Tobias Schneider and Amir Moradi. Leakage assessment methodology - extended version. *J. Cryptographic Engineering*, 6(2):85–99, 2016.
47. François-Xavier Standaert. How (not) to use welch's t-test in side-channel security evaluations. Cryptology ePrint Archive, Report 2017/138, 2017. `http://eprint.iacr.org/2017/138`.
48. Michael Tunstall, Carolyn Whitnall, and Elisabeth Oswald. Masking tables - an underestimated security risk. In Moriai [37], pages 425–444.

# A Additional figures

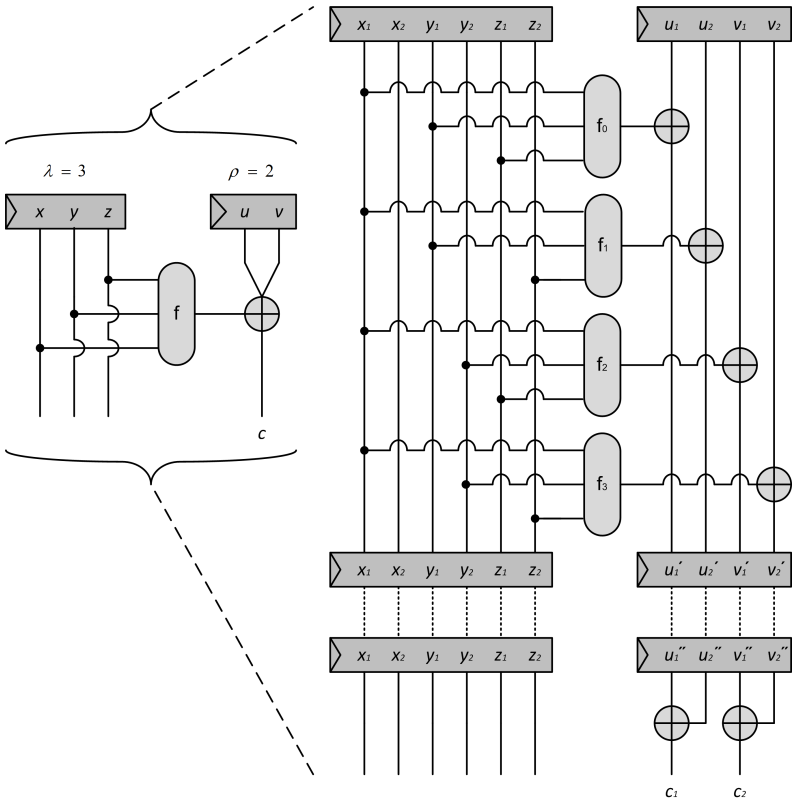


**Fig. 10.** Unbalanced Feistel network (left) and its TI (right).

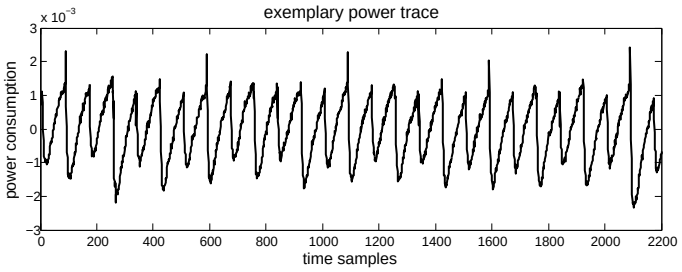


**Fig. 11.** Exemplary power trace for the implementation of Figure 7.

## B 4-bit functions with generalized Feistel representation

$$
\begin{aligned}
Q_4^4 &: (x,y,z,t) \mapsto (x \oplus zt, y, z, t)\\
Q_{12}^4 &: (x,y,z,t) \mapsto (x, y \oplus (y \oplus z)t, z \oplus yt, t)\\
Q_{293}^4 &: (x,y,z,t) \mapsto (x \oplus yz, y \oplus (y \oplus z)t, z \oplus yt, t)\\
Q_{294}^4 &: (x,y,z,t) \mapsto (x \oplus yt, y \oplus zt, z \oplus yt, t)\\
Q_{299}^4 &: (x,y,z,t) \mapsto (x \oplus (x \oplus z)t, y \oplus (x \oplus y \oplus z)t, z \oplus (y \oplus z)t, t)\\
C_1^4 &: (x,y,z,t) \mapsto (x \oplus yzt, y, z, t)\\
C_2^4 &: (x,y,z,t) \mapsto (x \oplus (x \oplus y)zt, y \oplus xzt, z, t)\\
C_3^4 &: (x,y,z,t) \mapsto (x \oplus zt, y \oplus xzt, z, t)\\
C_{13}^4 &: (x,y,z,t) \mapsto (x \oplus yzt, y \oplus (y \oplus z)t, z \oplus yt, t)\\
C_{243}^4 &: (x,y,z,t) \mapsto (x \oplus (x+z)(t+yt) \oplus yz, y \oplus (x \oplus y)t, z \oplus (y \oplus z)t, t)
\end{aligned}
$$

## C 4-bit quadratic shared functions

### C.1 $Q_4^4$

$$
\begin{array}{ll}
x_1 & \Big[\big[x_1 \oplus (z_1 \odot t_1)\big] \oplus (z_1 \odot t_2)\Big]\\
x_2 & \Big[\big[x_2 \oplus (z_2 \odot t_2)\big] \oplus (z_2 \odot t_1)\Big]\\
y_1 & y_1\\
y_2 & y_2\\
z_1 & z_1\\
z_2 & z_2\\
t_1 & t_1\\
t_2 & t_2
\end{array}
$$

### C.2 $Q_{12}^4$

$$
\begin{array}{ll}
x_1 & x_1 \\
x_2 & x_2 \\
y_1 & \Big[\big[y_1 \oplus ((z_1 \oplus y_1) \odot t_1)\big] \oplus ((z_1 \oplus y_1) \odot t_2)\Big] \\
y_2 & \Big[\big[y_2 \oplus ((z_2 \oplus y_2) \odot t_2)\big] \oplus ((z_2 \oplus y_2) \odot t_1)\Big] \\
z_1 & \Big[\big[z_1 \oplus (y_1 \odot t_1)\big] \oplus (y_1 \odot t_2)\Big] \\
z_2 & \Big[\big[z_2 \oplus (y_2 \odot t_2)\big] \oplus (y_2 \odot t_1)\Big] \\
t_1 & t_1 \\
t_2 & t_2
\end{array}
$$

### C.3 $Q_{293}^4$

$$
\begin{array}{ll}
x_1 & \Big[\big[x_1 \oplus (z_1 \odot y_1)\big] \oplus (z_2 \odot y_1)\Big] \\
x_2 & \Big[\big[x_2 \oplus (z_2 \odot y_2)\big] \oplus (z_1 \odot y_2)\Big] \\
y_1 & \Big[\big[y_1 \oplus ((z_1 \oplus y_1) \odot t_1)\big] \oplus ((z_1 \oplus y_1) \odot t_2)\Big] \\
y_2 & \Big[\big[y_2 \oplus ((z_2 \oplus y_2) \odot t_2)\big] \oplus ((z_2 \oplus y_2) \odot t_1)\Big] \\
z_1 & \Big[\big[z_1 \oplus (y_1 \odot t_1)\big] \oplus (y_1 \odot t_2)\Big] \\
z_2 & \Big[\big[z_2 \oplus (y_2 \odot t_2)\big] \oplus (y_2 \odot t_1)\Big] \\
t_1 & t_1 \\
t_2 & t_2
\end{array}
$$

### C.4 $Q^4_{294}$

$$
\begin{array}{ll}
x_1 & \Big[\big[x_1 \oplus (t_1 \odot y_1)\big] \oplus (t_2 \odot y_1)\Big] \\
x_2 & \Big[\big[x_2 \oplus (t_2 \odot y_2)\big] \oplus (t_1 \odot y_2)\Big] \\
y_1 & \Big[\big[y_1 \oplus (z_1 \odot t_1)\big] \oplus (z_2 \odot t_1)\Big] \\
y_2 & \Big[\big[y_2 \oplus (z_2 \odot t_2)\big] \oplus (z_1 \odot t_2)\Big] \\
z_1 & z_1 \\
z_2 & z_2 \\
t_1 & t_1 \\
t_2 & t_2
\end{array}
$$

### C.5 $Q^4_{299}$

$$
\begin{array}{ll}
x_1 & \Big[\big[x_1 \oplus (t_1 \odot (x_1 \oplus z_1))\big] \oplus (t_2 \odot (x_1 \oplus z_1))\Big] \\
x_2 & \Big[\big[x_2 \oplus (t_2 \odot (x_2 \oplus z_2))\big] \oplus (t_1 \odot (x_2 \oplus z_2))\Big] \\
y_1 & \Big[\big[y_1 \oplus ((x_1 \oplus y_1 \oplus z_1) \odot t_1)\big] \oplus ((x_1 \oplus y_1 \oplus z_1) \odot t_2)\Big] \\
y_2 & \Big[\big[y_2 \oplus ((x_2 \oplus y_2 \oplus z_2) \odot t_2)\big] \oplus ((x_2 \oplus y_2 \oplus z_2) \odot t_1)\Big] \\
z_1 & \Big[\big[z_1 \oplus ((y_1 \oplus z_1) \odot t_1)\big] \oplus ((y_1 \oplus z_1) \odot t_2)\Big] \\
z_2 & \Big[\big[z_2 \oplus ((y_2 \oplus z_2) \odot t_2)\big] \oplus ((y_2 \oplus z_2) \odot t_1)\Big] \\
t_1 & t_1 \\
t_2 & t_2
\end{array}
$$

### C.6 $C_1^4$

$$
\begin{array}{ll}
x_1 & \Big[\Big[\Big[\big[x_1 \oplus (y_1 \odot z_1 \odot t_1)\big] \oplus (y_1 \odot z_1 \odot t_2)\Big] \oplus (y_1 \odot z_2 \odot t_1)\Big] \oplus (y_2 \odot z_1 \odot t_1)\Big] \\
x_2 & \Big[\Big[\Big[\big[x_2 \oplus (y_2 \odot z_2 \odot t_2)\big] \oplus (y_2 \odot z_2 \odot t_1)\Big] \oplus (y_2 \odot z_1 \odot t_2)\Big] \oplus (y_1 \odot z_2 \odot t_2)\Big] \\
y_1 & y_1 \\
y_2 & y_2 \\
z_1 & z_1 \\
z_2 & z_2 \\
t_1 & t_1 \\
t_2 & t_2
\end{array}
$$

### C.7 $C_{13}^4$

$$
\begin{array}{ll}
x_1 & \Big[\Big[\Big[\big[x_1 \oplus (y_1 \odot z_1 \odot t_1)\big] \oplus (y_1 \odot z_1 \odot t_2)\Big] \oplus (y_1 \odot z_2 \odot t_1)\Big] \oplus (y_2 \odot z_1 \odot t_1)\Big] \\
x_2 & \Big[\Big[\Big[\big[x_2 \oplus (y_2 \odot z_2 \odot t_2)\big] \oplus (y_2 \odot z_2 \odot t_1)\Big] \oplus (y_2 \odot z_1 \odot t_2)\Big] \oplus (y_1 \odot z_2 \odot t_2)\Big] \\
y_1 & \Big[\big[y_1 \oplus ((y_1 \oplus z_1) \odot t_1)\big] \oplus ((y_1 \oplus z_1) \odot t_2)\Big] \\
y_2 & \Big[\big[y_2 \oplus ((y_2 \oplus z_2) \odot t_2)\big] \oplus ((y_2 \oplus z_2) \odot t_1)\Big] \\
z_1 & \Big[\big[z_1 \oplus (y_1 \odot t_1)\big] \oplus (y_1 \odot t_2)\Big] \\
z_2 & \Big[\big[z_2 \oplus (y_2 \odot t_2)\big] \oplus (y_2 \odot t_1)\Big] \\
t_1 & t_1 \\
t_2 & t_2
\end{array}
$$

## D Proof of Lemma 4

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be a set of $q$ adversary's observations respectively on the internal and on the output values, where $|\mathcal{I}| = q_1$ and in particular $q_1 + |\mathcal{O}| \leq q$. We construct a perfect simulator of the adversary's probes, which makes use of at most $q_1$ shares of the secrets $x$ and $y$.

Let $w_1, \ldots, w_q$ be the probed values. According to the specific model for glitches presented in Section 4.1, the possible internal extended probes can be classified in the following groups:

(1) $p_{i,j} := (a_i, b_j, r_{i,j})$ with $i, j = 1, \ldots, q+1$
(2) $p_i := (u_{i,1}, \ldots, u_{i,q+1})$ with $i = 1, \ldots, q+1$

On the other hand, since the output shares are stored in registers, glitches do not affect them and so the possible probes on the output shares are, as in the non-robust probing model, the $c_i$ with $i = 1, \ldots, q+1$, as in Algorithm 1.

We define two sets of indices $I$ and $J$ such that $|I| \leq q_1$, $|J| \leq q_1$ and the values of the probes can be perfectly simulated given only the knowledge of $(x_i)_{i \in I}$ and $(y_i)_{i \in J}$. The sets are constructed as follows.

- Initially $I$ and $J$ are empty.
- For every probe as in group (1) add $i$ to $I$ and $j$ to $J$.
- For every probe as in group (2) add $i$ to $I$ and to $J$.

Since the adversary is allowed to make at most $q_1$ internal probes, it holds that $|I| \leq q_1$ and $|J| \leq q_1$.

We now show the simulation phase. First of all, the simulator assigns a random value to every $r_{i,j}$ entering in the computation of any probe. Then we consider an observed value $w_h$ in group (1). In this case, by definition of $I$ and $J$ the simulator has access to $a_i$ and $b_j$ and we distinguish three cases:

- If $i = j$, the simulator assigns $r_{i,i}$ to 0 and then perfectly simulates $w_h$ using $a_i$ and $b_i$.
- If $j \in I$ and $i \in J$, then by definition the adversary has probed also $p_{j,i}$. The simulator then perfectly simulates $w_h$ using $a_i$, $b_j$ and the $r_{i,j}$ assigned in the preliminary phase.
- In all the other cases, $r_{i,j}$ does not enter in the computation of any other probe, and therefore the simulator can assign $w_h$ to a random and independent value.

As for a probe $w_h$ in group (2), by definition $i \in I, J$. So the simulator can perfectly compute the $i$th-component of the probe using $a_i, b_i$. For each of the remaining $j$th-components of $p_i$ we distinguish the following cases.

- If $j \in J$ and $j \notin I$, then the adversary has already probed $p_{i,j}$, which can be simulated as in the first phase and entirely used as $j$th-component of $w_h$.
- If $j \in J$ and $j \in I$, then the adversary has already probed $p_j$ or $p_{j,i}$. In both cases $r_{i,j}$ was assigned in the preliminary phase and can be used with $a_i$ and $b_j$ to simulate the $j$th-component of $w_h$.

– If $j \notin J$, the simulator assigns to the $j$th-component of $p_i$ a random and independent value: indeed, even if $b_j$ has been observed in another probe, by the definition of the algorithm it will appear in addition to a random bit which is different from $r_{i,j}$.

We conclude the proof by showing how to simulate a probe $w_h$ in the output values. We notice that since in this case the probes are as in the traditional probing model, the proof is really similar to the one of Proposition 2 in [2]. We have to take into account the following two cases:

– If the attacker has observed also some of the internal values, then the partial sums previously probed are already simulated. As for the remaining terms, we note that by definition of the scheme there always exists one random bit $r_{k,l}$ in $w_h$, which does not appear in the computation of any other observed element. Therefore the simulator can assign to $w_h$ a random and independent value.
– If the attacker has only observed output shares, then we point out that by definition each of them is composed by $q$ random bits and at most one of them can enter in the computation of each other output variable $c_i$. Since the adversary may has previously probed at most $q-1$ of them, there exist one random bit $r_{k,l}$ in $w_h$, which does not appear in the computation of any other observed value. Thus the simulator can assign to $w_h$ a random and independent element, completing the proof.

□