Consolidating Security Notions in Hardware Masking

Lauren De Meyer[1], Begül Bilgin[1,2], and Oscar Reparaz[1,3]

[1] KU Leuven, imec - COSIC, Leuven, Belgium
firstname.lastname@esat.kuleuven.be
[2] Rambus, Cryptography Research, Rotterdam, the Netherlands
[3] Square Inc., San Francisco, USA

Abstract. This paper revisits the security conditions of masked hardware implementations. We describe a new, succinct, information-theoretic condition to ensure security in the presence of glitches. This single condition includes, but is not limited to, previous security notions such as those used in threshold implementations. As a consequence, we can prove the security of masked functions that work with non-uniform input sharings. Our notion naturally generalizes to higher orders. Furthermore, we can apply our condition in a tool that efficiently tests and validates the resistance of masked hardware circuits against DPA. Finally, we also treat the notion of (strong) non-interference from an information-theoretic point-of-view in order to unify the different security concepts and pave the way to the verification of composability in the presence of glitches.

Keywords: Glitches, DPA, SCA, Verification, TI, SNI, Non-Completeness, Mutual Information, Information-theory, d-probing, Immunity

1 Introduction

Cryptographic algorithms are designed such that they are mathematically secure. An adversary with access to, for instance, the ciphertext and plaintext, should not be able to derive the secret key with reasonable computing power. However, this *black box* model often does not suffice in practice, as the existence of side-channels can significantly aide the adversary in his quest for secret information. Since the seminal work of Kocher [Koc96], we have learned of many cheap and scalable side-channel attacks (SCA) that succesfully exploit information such as instantaneous power consumption or electromagnetic radiation to recover secret keys, effectively turning the adversary's *black box* into a *grey box*.

In the realm of SCA, the most well known technique is Differential Power Analysis (DPA) [KJJ99], a simple and efficient attack that exploits the fact that the power consumption of an embedded device depends on the intermediate values that the device computes on. In response, many countermeasures have been proposed, among which *masking* is one of the most established.

Constructing masked circuits is non-trivial. Many of the proposed countermeasures of the last years (whether they be Boolean masking or multiplicative

or additive) have been shown to be vulnerable relatively quickly after being published [AG01, PGA06, SP06, RP10, BFGV12, BGN+14, HT16].

This history of trial and error has given rise to a new wave of works on the verification of masking schemes, accompagnied by a variety of verification tools, ranging from formal to statistical [BBD+16, Rep16, ANR17, Cor17a, BGI+18]. Each of these uses its own distinct condition for claiming correctness or security of the masking schemes. This line of work can be subdivided into those that verify software implementations and those that verify hardware implementations.

1.1 Masked Software Implementations

In the traditional setting (e.g. ISW [ISW03]), a *d-probing adversary* gets the noiseless information about the d intermediates that have been probed. Note that the probing in this model is instantaneous. That is, the adversary does not see any transition or change in the intermediate. He sees only the *stabilized* value. Such a naive power model does not take into account physical defaults that impact hardware implementations or memory transitions.

Necessary and Sufficient conditions. A necessary and sufficient condition for the d-probing security of masked software implementations is that any set of d intermediates is statistically independent of the secret. However, it is infeasible to test this condition on a realistic cryptographic circuit in full scale. A notable condition to remedy this problem is the concept of (strong) non-interference [Bel15, BBD+16]. d-Non-Interference (d-NI) is a sufficient condition for d-probing security but it is not necessary. On the other hand, this notion can be used as a stepping stone for d-Strong Non-Interference (d-SNI) which implies composability. More specifically, d-SNI gadgets can be composed to provide d-probing security which allows for a more efficient verification of large circuits, though at the cost of higher randomness. A good example to show the sufficiency but not necessity of d-(strong) non-interference is a first-order threshold implementation (TI) [FGP+17, §3.1].

Security validation tools using these concepts have mainly been limited to software implementations, and glitches are not taken into account [BBD+15, Cor17a].

1.2 Masked Hardware Implementations

Glitches are unintentional and undesirable hardware artifacts causing unnecessary power consumption and hardware designers go to great lengths to minimize them for reasons beyond security. However, reducing glitches requires a careful process of path equalization which is a great challenge given factors such as the product and architecture variation, working environment and device age. Moreover, glitches can momentarily unmask values, invalidating the security guarantees theoretically provided by masking and making it a security hazard [MPG05].

Necessary but not Sufficient. Threshold implementations (TI) [NRR06] were the first provably secure masking scheme in the presence of glitches. The authors identified two key properties: non-completeness and uniformity, which together ensure the provable security of TI against first-order DPA in the presence of glitches. Together, non-completeness and (global) uniformity are sufficient to provide first-order security on any hardware as long as the independent leakage assumption of masking holds.

The concept was extended to higher-order security by Bilgin *et al.* [BGN+14], but it was later shown by Reparaz *et al.* [RBN+15] that non-completeness and uniformity no longer suffice to achieve higher-order security due to the possibility of multivariate attacks. The authors also refined the definition of non-completeness to the level of individual variables and hence opened up the possibility for securely using only $d + 1$ shares in the presence of glitches. The property of non-completeness is a *necessary* requirement for the security of masked hardware implementations, but up to today, it remains unclear how to define a *sufficient* condition.

Validation tools for hardware is a young topic due to aforementioned challenges. A tool that verifies higher-order non-completeness and uniformity properties given a HDL has been developed by Arribas *et al.* [ANR17]. In an alternative line of work, Bloem *et al.* [BGI+18] describe a formal method to verify the security of masked hardware implementations in the presence of glitches, by tracking the correlation between intermediates and secret variables.The verification can be extended to higher order using a SAT solver.

1.3 Our Contribution

In this work, we introduce an information-theoretic metric, which is conceptually extremely simple as well as easy to verify and above all, *sufficient* to achieve d-probing security in the presence of glitches. We revisit the definition of probing security as introduced in [GM10, Def. 4], *i.e.* a circuit is d-probing secure if the mutual information of any set of d probes with the secret is zero. We redefine this notion for hardware implementations using the adversary model of [RBN+15], where each probed wire gives the adversary information about all the inputs to that wire up to the last synchronization point. By replacing each probe with its glitch-extended version (*i.e.* the set of inputs up to the last synchronization point), we obtain an information-theoretic condition for probing security in the presence of glitches.

We compare this metric with existing notions such as non-completeness and (strong) non-interference and we unify these security concepts by introducing new information-theoretic definitions. This allows us to move towards the efficient verification of composability for hardware implementations. In addition, we demonstrate that uniformity is not a necessary condition for secure masked hardware implementations.

Finally, we detail how the new condition for glitch security can be used in a tool to detect flaws and verify schemes. The tool can be used both for exact proofs and statistical validation of practical security. It works directly on the

HDL description of a circuit and needs no other user provided information. The simplicity of the security condition also makes the tool compatible with any type of masking (Boolean, multiplicative, polynomial, ...). Also masked software implementations can be verified. We demonstrate its ability to find vulnerabilities by applying it to known flawed designs and even detect a new problem.

2 SCA Security in the Presence of Glitches

2.1 Preliminaries

Notation. We use lowercase letters, *e.g.* x to denote random variables and capital letters, *e.g.* F for functions. Bold font is used for a sharing of these, *i.e.* $\boldsymbol{x} = (x_0, x_1, \ldots)$ is a sharing of x and $\boldsymbol{F} = (F_0, F_1, \ldots)$ is a masked F. We denote specific realizations of x by superscript x^*. Further, we let $\boldsymbol{x}_{\bar{i}} = (x_0, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots)$ be the vector obtained by removing x_i from $\boldsymbol{x}$.

The probability distribution on x is denoted by $p(x)$ and $H(x)$ is the Shannon entropy of x: $H(x) = -\sum_{x^*} p(x^*) \log_2(p(x^*))$. For any x, y, we use $p(x|y)$ to represent the conditional probability of x on y and $I(x;y)$ the mutual information between x and y. The two notions are connected by the relation $I(x;y) = H(x) - H(x|y)$. Note that mutual information is symmetric, *i.e.* $I(x;y) = I(y;x)$.

Finally, we denote a set of multiple random variables, *e.g.* (x, y) by caligraphy letters $\mathcal{X}$. The union of two sets $\mathcal{X} \cup \mathcal{Y}$ is the set consisting of all variables that belong to either $\mathcal{X}$ or $\mathcal{Y}$ or both:

$$\mathcal{X} \cup \mathcal{Y} = \{x : x \in \mathcal{X} \text{ or } x \in \mathcal{Y}\}$$

Statistical independence. Recall that there are a multitude of equivalent ways to express the statistical independence of two discrete random variables x and y. For instance, x and y are statistically independent when their mutual information is zero ($I(x;y) = 0$). It then follows directly from the relation between mutual information and entropy that $H(x) = H(x|y)$, *i.e.* the entropy of x does not decrease when conditioned on y (and vice versa). Alternatively, the statistical independence of x and y is also evidenced by $p(x, y) = p(x)p(y)$, which is in turn equivalent to $p(x) = p(x|y)$. In this work, we investigate the statistical independence by verifying that the probability distribution of x, conditioned on a specific instantiation of y^*, is independent of that y^*. In other words, we verify that

$$\forall y^* : p(x|y^*) = p^* \tag{1}$$

with p^* some constant probability distribution independent of y^*. It is easy to show that statistical independence follows from this:

$$p(x) = \sum_{y^*} p(x|y^*)p(y^*) = p(x|y^*) \sum_{y^*} p(y^*) = p(x|y^*) \tag{2}$$

Information-theoretic view. In [GM10, Def. 4], Gammel and Mangard provided an information-theoretic definition for d-probing security. We repeat their definition below for completeness as throughout this paper, we also use information-theoretic notions. Gammel and Mangard trace back the origin of this metric to Siegenthaler's correlation immunity of a Boolean function [Sie84].

***d*-probing security.** *A circuit is d-probing secure, if and only if for any observation set of d wires $\mathcal{Q} = (q_1, q_2, \ldots, q_d)$ the following condition holds:*

$$I((q_1, q_2, \ldots, q_d); x) = 0 \tag{3}$$

with x the secret.

This condition is sufficient and necessary for d-probing security. However, it does not account for glitches. In the rest of this section, we fill in this gap.

2.2 Glitchy Circuits

Before providing the condition, we first discuss our circuit and leakage model.

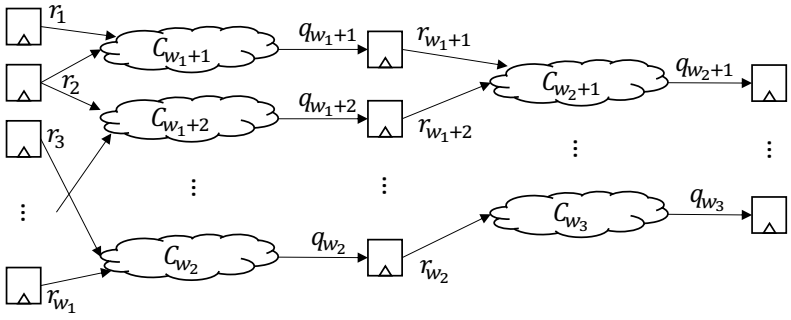

Fig. 1. Masked circuit model

Circuit model. Assume we deal with a masked circuit as shown in Figure 1. The circuit handles a sensitive value x which depends on the key. Every combinational block C_i computes a single output wire q_i from a set of input wires $\mathcal{R}_i = \{r_{i_1}, r_{i_2}, \ldots\}$. For example, in Figure 1, wire q_{w_1+1} is computed by combinational block C_{w_1+1} from inputs $\mathcal{R}_{w_1+1} = \{r_1, r_2\}$. Note that the wires (r_i, q_i) can be constants, shares of multiple sensitive variables, or public values. Their specific roles do not matter here. The output wires of combinational blocks (q_i) are input wires to synchronization points and carry unstabilized values. For brevity, we assume that these synchronization points are simultaneously clocked registers. Data from one register stage (r_i) forms the input to a block of combinational logic that computes the input to the next register stage. These inputs

to combinational blocks are considered *stable*. Since glitch-extended probes of intermediate wires inside a block C_i are obviously included in the glitch-extended probe of wire q_i, we only consider probes on q_i from now on.

The glitch function. It is inefficient or even infeasible to predict the effect of glitches and enumerate all the possible temporary values that can occur on a wire in C_i before the signal stabilizes at its intended function value. We provide an abstraction for this glitch-based leakage with a *glitch function*. The glitch function is *any* function leaked by a combinational block before it stabilizes on the intended function value. This can be an unexpected partial result because of propagation delays in some inputs. Alternatively, upon arrival of a late intermediate, one partial result transitions to another and the glitch function computes the difference between both.

It is difficult to model the glitch function, since it depends on many variables, such as the logic library and synthesis process; hence, in this paper we consider it to be unknown. However, we can state some of its properties. An obvious, yet important property of the glitch function is that, under the independent leakage assumption, it depends exclusively on the inputs of the intended function. That is, when a wire in a combinational block C_i glitches, it momentarily computes a glitch function that depends o n $\mathcal{R}_i$ exclusively.

Leakage model. A model that takes into account glitches considers additional leakage compared to the traditional ISW d-probing model [ISW03]. We use the leakage model introduced in [RBN+15] which covers all possible glitch functions as an abstraction: We assume that, whenever the adversary probes any single value within a combinational function, he also automatically obtains all inputs to the function (up to the last synchronization point), *for free*. That is, an adversary probing q_i obtains the knowledge of the set $\mathcal{R}_i$. We provide the adversary with all inputs, so that he himself can compute the *worst possible glitch function*. This model may be somewhat over-conservative, since this information might not actually be leaked by the circuit. This is the price we pay to work at a high level, where no information about the synthesis process or technology is required.

We note that this adversary model, here-on called *d-glitch-extended probing adversary*, can be extended [FGP+17] to cover other physical defaults, such as cross-talk. In this work, we continue with the above described independent leakage model, including glitches but no coupling effects or register transitions. Moreover, we assume that the clock period is large enough so that the critical path constraint is satisfied in order to avoid attacks such as [MM17].

2.3 A Sufficient Condition to Achieve d-Glitch-Extended Probing Security

In what follows, we introduce a new condition for security against side-channel attacks in the presence of glitches. This security notion is elegant and conceptually simple, as well as easy to verify for circuits in practice.

Property 1 (d-glitch immunity). *A circuit such as that in Figure 1 is d-glitch immune if and only if for any observation set of d wires $(q_{i_1}, q_{i_1}, \ldots, q_{i_d})$ with respective glitch-extended probes $(\mathcal{R}_{i_1}, \mathcal{R}_{i_2}, \ldots, \mathcal{R}_{i_d})$, the condition*

$$I(\mathcal{R}_{i_1} \cup \mathcal{R}_{i_2} \cup \ldots \cup \mathcal{R}_{i_d}; x) = 0 \tag{4}$$

holds, with x the sensitive data.

Lemma 1. *A d-glitch immune circuit is d-glitch-extended probing secure.*

The reasoning is quite elementary: when the mutual information between $\mathcal{R} = \mathcal{R}_{i_1} \cup \mathcal{R}_{i_2} \cup \ldots \cup \mathcal{R}_{i_d}$ and the secret is zero, no combination of input wires $r_i \in \mathcal{R}$ can provide any information on the secret. Hence, no glitch function on $C_{i_1}, .., C_{i_d}$ or their combination, no matter the exact shape, can reveal any secret information. In particular, DPA would not be able to exploit leakage from it.

Multi-variate security. Note that d-glitch immunity puts no limitation on the register stage of probed wires q_i. Therefore, verification of higher-order security, let it be uni-variate or multi-variate, is conceptually as simple as verification of first-order security. To test security against d probes, we have to consider the input wires of d blocks jointly which can be done as mere concatenation. Hence, d-glitch immunity easily covers multi-variate security.

Sufficient and necessary. Note that we only take into account the inputs to the combinational blocks C_i and not the specific functions computed. On the one hand, this makes d-glitch immunity conceptually simple and quite easy to verify. On the other hand, it is a worst-case indication of leaks. A non-zero mutual information

$$I(\mathcal{R}; x) \neq 0 \tag{5}$$

points to the existence of some function on the inputs $\mathcal{R}$ that leaks sensitive information, but there is no guarantee that this function can occur as a glitch function on the circuit that implements C_i. Even if it does, the presence of such a leak does not imply that it can be exploited. Glitch immunity might therefore be over-conservative when one considers the implementation details (including the floorplan, temperature, etc.) and realistic noisy attack scenarios, but in our adversary model, it is both necessary and sufficient.

Comparison to previous work. While no explicit condition such as Property 1 is given in the description of the Formal Verification tool of Bloem *et al.* [BGI+18], its objective is essentially the same, *i.e.* they are both testing independence of worst-case glitch functions of the secret, but with radically different approaches. In [BGI+18], the dependency of all inputs (not just secrets but also masks, public variables, ...) is tracked through the circuit. The masking is assumed to be Boolean. For higher-order security, Bloem *et al.* require a SAT solver to investigate combinations of up to d gates. In contrast, using Property 1 implies *detecting* dependencies of the secret rather than *tracking* all dependencies. Glitch immunity does not assume any particular type of masking. And its extension from first-order to higher-order security is minimal.

3 Threshold Implementations

Traditionally, the provable security of threshold implementations (TI) [NRR06] against first-order attacks relies on three conditions:

T1 Correctness. The masked function should compute a masked representation of the correct unmasked output.
T2 Non-completeness. Any share of the masked function must be independent of at least one input share. This property is central to security in the presence of glitches.
T3 Uniformity. The masked function uses a uniform sharing of the input and transforms this input into a uniform sharing of the output. This property is especially important when composing blocks.

Recall the definition of a uniform sharing where $Sh(x)$ is the set of all valid sharings of x:

Uniform sharing [Bil15]. *A sharing $\boldsymbol{x}$ is uniform if and only if there exists a constant p such that for all x we have:*

$$p(\boldsymbol{x}|x) = \begin{cases} p & \text{if } \boldsymbol{x} \in Sh(x) \\ 0 & \text{else} \end{cases} \tag{6}$$

and

$$\bigoplus_{\boldsymbol{x} \in Sh(x)} p(\boldsymbol{x}) = p(x) \tag{7}$$

In the rest of this section, we first discuss the relation between d-glitch immunity and the TI conditions. In particular, we elaborate on the necessity and satisfactoriness of these conditions for first order and higher orders. From these relations, we can interpret d-glitch immunity as being a generalization of the TI properties.

3.1 Non-Completeness and Uniformity Imply 1-Glitch Immunity

It is a well-known result that T2 and T3 imply first-order security in the presence of glitches. Hence, T2 and T3 are sufficient conditions for 1-glitch-extended probing security. In what follows, we show that a circuit satisfying T2 and T3 also fulfills 1-glitch immunity.

Lemma 2. *A circuit satisfies 1-glitch immunity if it satisfies non-completeness (T2) and uniformity (T3).*

Proof. The assumptions for each stage of TI are that the input sharing $\boldsymbol{x}$ is uniform and that each wire q_i at the end of that stage must be independent of

at least one input share. Without loss of generality (wlog) we assume that q_i is independent of x_i and thus

$$\mathcal{R}_i = \boldsymbol{x}_{\bar{i}} \tag{8}$$

Lemma 5 of [Bil15] states that $\boldsymbol{x}_{\bar{i}}$ and the secret x are independent for any choice of i if the masking $\boldsymbol{x}$ is uniform. Hence,

$$I(\boldsymbol{x}_{\bar{i}}; x) = 0 \tag{9}$$

and (8) together with (9) forms (4):

$$I(\mathcal{R}_i; x) = 0 \tag{10}$$

□

3.2 Glitch Immunity Implies Non-Completeness

The concept of non-completeness was extended to higher orders by Bilgin *et al.* [BGN+14].

d^{th}-order non-completeness [Bil15]. *Any combination of up to d component functions F_i of a shared function $\boldsymbol{F}$ must be independent of at least one input share.*

Clearly the non-completeness definition aligns with the glitching adversary of this work as the component functions F_i correspond to the combinational block functions C_i calculated from register to register.

Lemma 3. *d-glitch immunity implies d^{th}-order non-completeness.*

Proof. We use a simple proof by contraposition. Suppose that non-completeness is not fulfilled, *i.e.* there exists a set of d blocks $(C_{i_1}, C_{i_2}, \ldots, C_{i_d})$ that depends on all input shares $\boldsymbol{x}$. Since $x = \bigoplus_i x_i$, we clearly have then that $I(\mathcal{R}_{i_1} \cup \mathcal{R}_{i_2} \cup \ldots \cup \mathcal{R}_{i_d}; x) \neq 0$ and hence glitch immunity is not satisfied. □

3.3 A Sufficient Condition for Higher-Order Security

We have demonstrated the well-known fact that T2 and T3 together form a sufficient condition for first-order security from an information-theoretic point-of-view. Moreover, we have shown that non-completeness is a necessary condition for higher-order security as expected. These relations are summarized in Figure 2. However, as presented by Reparaz *et al.* [RBN+15], d^{th}-order non-completeness even together with uniformity is not sufficient for circuits consisting of multiple register stages. This is mainly due to the fact that TI conditions focus on the circuit behavior of a single stage and when higher-order security is considered, this becomes a disadvantage. On the other hand, d-glitch immunity does not have

such a limitation as it considers multiple stages. This observation immediately brings up the idea to extend the TI uniformity condition to cover not only a single stage but multiple stages, for example by requiring the combination of any d non-linear function inputs to be jointly uniform. We do not investigate this idea further in this paper. Instead, we show that uniformity is not a necessary condition to achieve security.

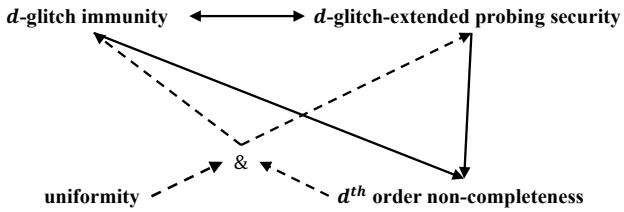

Fig. 2. Illustration of the relations between different security notions for hardware security. The arrows indicate an implication relation. When the line is dotted, it is only valid for security order $d = 1$. The full lines are for any order d.

3.4 (Un)Necessity of Uniformity

Glitch immunity does not imply non-completeness and uniformity (simultaneously). In fact, uniformity is not a necessary condition. The only requirement for input sharings is included in Eqn. (4); that is, the entropy of d input shares does not decrease when conditioned on the secret. In other words, the mutual information of a set of any d input shares with the shared secret x must be zero:

$$I((x_{i_1}, x_{i_2}, \ldots, x_{i_d}); x) = 0 \tag{11}$$

In case of a $d+1$-sharing, we thus only require that $I(\boldsymbol{x}_{\bar{i}}; x) = 0$, $\forall i$.

It is therefore possible to have non-uniform mappings that satisfy Property 1. We show this below with a toy example.

Toy Example: First-order security of non-uniform AND gates We focus on the composition of two AND gates as shown in Figure 3. We will use two types of refreshing after the first gate to illustrate that uniformity is not necessary to achieve theoretical security. Note that these AND gates are designed to demonstrate our cause and therefore are neither optimal nor considered for use in another setting.

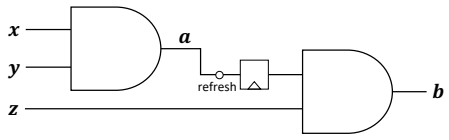

Fig. 3. Circuit of two AND gates

We consider x, y and z as our sensitive variables. The first AND gate receives 3-share input values $\mathbf{x}$ and $\mathbf{y}$ and outputs 6-share value $\mathbf{a}$ as follows.

$$\begin{aligned} a_0 &= x_0y_0 & a_3 &= x_0y_1 \oplus x_1y_0 \\ a_1 &= x_1y_1 & a_4 &= x_1y_2 \oplus x_2y_1 \\ a_2 &= x_2y_2 & a_5 &= x_2y_0 \oplus x_0y_2 \end{aligned} \tag{12}$$

The second AND gate uses the 6-share output $\mathbf{a}$ with 3-share $\mathbf{z}$ as follows.

$$\begin{aligned} b_0 &= a_0z_0 \oplus a_0z_1 \oplus a_1z_0 \oplus a_4z_1 \oplus a_5z_0 \oplus a_5z_1 \\ b_1 &= a_0z_2 \oplus a_1z_1 \oplus a_1z_2 \oplus a_2z_1 \oplus a_3z_1 \oplus a_3z_2 \\ b_2 &= a_2z_0 \oplus a_2z_2 \oplus a_3z_0 \oplus a_4z_0 \oplus a_4z_2 \oplus a_5z_2 \end{aligned} \tag{13}$$

The distribution of output shares a_i which is shown in Table 1 is biased as expected.

Table 1. Scaled probability distributions in the first AND gate with 3 input and 6 output shares for specific inputs (y^*, x^*)

	$p(a_i)$ $i<3$		$p(a_i)$ $i>2$		$p(a_0, a_1, \ldots, a_5)$
(y^*, x^*)	0	1	0	1	00..0 to 11..1
(0,0)	12	4	10	6	70010110000000000000000000000000000000000000000000000000000002202000
(0,1)	12	4	10	6	40000000011000000010100010000100010010001000001010010000000000000
(1,0)	12	4	10	6	40000000011000000010100010000100010010001000001010010000000000000
(1,1)	12	4	10	6	01101001200000002000000000200000200000000200000000002000000000000

Clearly, if there is no remasking after the first AND gate, the second AND gate leaks sensitive information. On the other hand, if we refresh every output share a_i, then the second AND gate receives a uniform sharing and is secure. We look at an intermediate approach where the output of the first gate is re-masked such that it is not completely uniform but the second AND gate does not leak sensitive information.

The Good. Let's consider the following remasking using 2 bits of randomness:

$$\begin{aligned} a_0 &\leftarrow a_0 & a_3 &\leftarrow a_3 \oplus r_1 \oplus r_2 \\ a_1 &\leftarrow a_1 \oplus r_1 & a_4 &\leftarrow a_4 \oplus r_2 \\ a_2 &\leftarrow a_2 & a_5 &\leftarrow a_5 \end{aligned} \tag{14}$$

When applying the second AND gate to the refreshed $\mathbf{a}$, we need to keep track of the input sets of each output share b_i. Since the sharing of $\mathbf{z}$ is uniform and used in a non-complete way, we can ignore its effect for now for brevity. Note that there is synchronization after the randomization, by for example a register stage. The output shares (b_0, b_1, b_2) thus depend respectively on the sets $\mathcal{R}_0 = (a_0, a_1, a_4, a_5), \mathcal{R}_1 = (a_0, a_1, a_2, a_3)$ and $\mathcal{R}_2 = (a_2, a_3, a_4, a_5)$.

We investigate the mutual information between each of the sets $\mathcal{R}_i$ with the secret, by looking at their probability distributions for different secret inputs. Table 2 shows clearly that these distributions are not uniform. However, they are identical for each of the secret inputs (x^*, y^*). The result stays the same for the secret (x^*, y^*, z^*) which is omitted here for readability.

Table 2. Scaled probability distributions of $\mathcal{R}_i$ for any fixed secret (x^*, y^*)

	$00\ldots0$ to $11\ldots1$															
$p(\mathcal{R}_0 \mid x^*, y^*)$	8	2	8	2	8	2	8	2	4	2	4	2	4	2	4	2
$p(\mathcal{R}_1 \mid x^*, y^*)$	9	3	9	3	3	1	3	1	9	3	9	3	3	1	3	1
$p(\mathcal{R}_2 \mid x^*, y^*)$	8	2	8	2	8	2	8	2	4	2	4	2	4	2	4	2

The Bad. Suppose now that instead of the randomization in Eqn. (14), we do the following remasking.

$$\begin{aligned} a_0 &= a_0 & \qquad a_3 &= a_3 \oplus r_1 \oplus r_2 \\ a_1 &= a_1 \oplus r_1 & \qquad a_4 &= a_4 \\ a_2 &= a_2 & \qquad a_5 &= a_5 \oplus r_2 \end{aligned} \tag{15}$$

We see clearly in Table 3 that the joint probability distribution of the input set $\mathcal{R}_0$ depends on the secret inputs even though there is no correlation of the secret with output share b_0 itself. This indicates that the remasking is sufficient in the case of a classic probe on b_0, but not in the presence of glitches. The joint probability distributions are shown in Table 3.

Table 3. Scaled probability distributions of $\mathcal{R}_0$ and b_0 for various secrets (y^*, x^*)

	$p(\mathcal{R}_0 \mid x^*, y^*)$																$p(b_0 \mid x^*, y^*)$	
(y^*, x^*)	$00\ldots0$ to $11\ldots1$																0	1
$(0,0)$	8	2	8	2	4	2	4	2	8	2	8	2	4	2	4	2	160	96
$(0,1)$	8	2	8	2	4	2	4	2	8	2	8	2	4	2	4	2	160	96
$(1,0)$	8	2	8	2	4	2	4	2	8	2	8	2	4	2	4	2	160	96
$(1,1)$	6	4	6	4	6	0	6	0	6	4	6	4	6	0	6	0	160	96

The Ugly. With the good remasking as in Eqn. (14), we can even do certain compositions under the ISW setting with ideal gates. That is, even if the third input $\mathbf{z} = \mathbf{x}$, the output is still secure as shown in Table 4.

Table 4. Scaled probability distributions in the second AND gate with $z = x$

	$p(b_i)$		$p(b_0, b_1, b_2)$							
(y^*, x^*)	0	1	000	011	101	110	001	010	100	111
(0,0)	40	24	28	0	0	12	0	12	12	0
(0,1)	40	24	28	0	0	12	0	12	12	0
(1,0)	40	24	28	0	0	12	0	12	12	0
(1,1)	40	24	0	20	20	0	20	0	0	4

However, in a glitchy environment, we do not necessarily get composability. Neither $\mathcal{R}_0 = (a_0, a_1, a_4, a_5, x_0, x_1)$ nor $\mathcal{R}_1 = (a_0, a_1, a_2, a_3, x_1, x_2)$ are independent from the secret.

Conclusion. In order to avoid leakage of sensitive data, what we need is independence of the secret from the sensitive data. In this case study, we have demonstrated that this does not necessarily require uniform distributions. Using d-glitch immunity as a verifying mechanism, we can create custom gates with minimal randomness consumption.

4 Tools to Validate a Masked Hardware Circuit

In this section we describe how we can use d-glitch immunity to validate a masked hardware circuit. One has to check whether the distributions of sets of input wires corresponding to different secrets are identical or not (cf. Eqn (1)).

4.1 Tool Description

We develop a tool which can take the HDL description of a circuit directly as input and needs no other user-provided information apart from the required security order d. The tool consists of two parts: a preprocessing step prepares the input for the verification step.

Preprocessing. The preprocessing step parses the HDL code and builds a software implementation (in c), which can simulate the entire circuit at bit level. This means that by using this c code during verification, we even consider glitches occurring at the bit level in larger field operations (*e.g.* $GF(2^8)$). Apart from the circuit itself, the tool also extracts from the netlist a list of all register inputs q_i and the corresponding glitch-extended probes $\mathcal{R}_i$, *i.e.* the set of inputs to the combinational block C_i that determines wire q_i. The c implementation is generated such that for a given circuit, it can compute the exact values of

all registers r_i and use these to construct samples for probability distributions. Per simulation (k), one sample $sample_I^k$ is created for each unique d-tuple of intermediate register wires and circuit outputs $\mathcal{Q}_I = \{q_{i_1}, q_{i_2}, \ldots, q_{i_d}\}$ and its corresponding set of glitch extended probes $\mathcal{R}_I = \mathcal{R}_{i_1} \cup \mathcal{R}_{i_2} \ldots \cup \mathcal{R}_{i_d}$. Concatenating the values on the wires in $\mathcal{R}_I$ results in a sample of width $|\mathcal{R}_I|$. Different probe combinations thus result in samples of different widths, but the width of each probe combination is the same for each simulation. The probe $\mathcal{R}_I$ can be a combination of random inputs, public values and shares of various sensitive variables. The specific role of each wire is unimportant and does not need to be tracked.

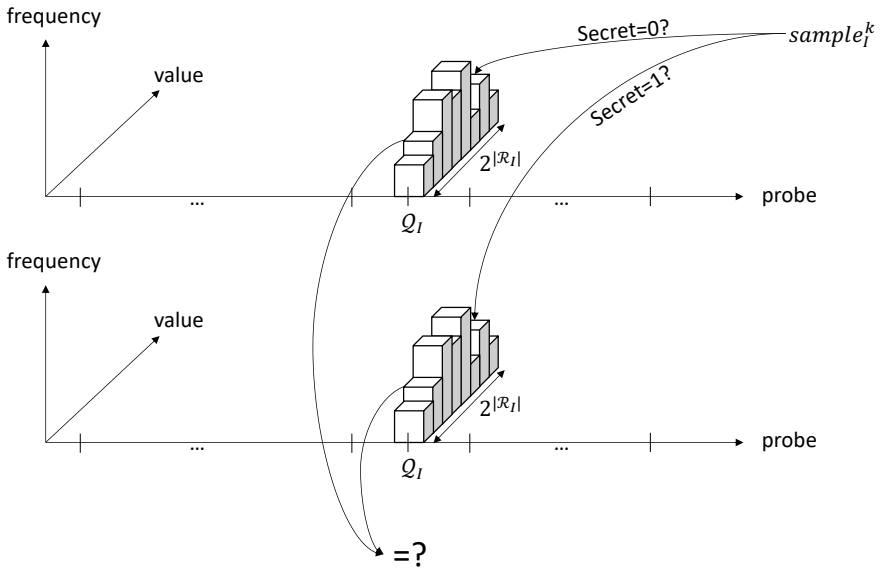

Fig. 4. Illustration of Tool

Verification. The verification tool uses the prepared software implementation to simulate the circuit for different inputs and to collect the samples of glitch extended probes. The tool keeps two histograms for each possible d-probe $\mathcal{Q}_I$. The histograms corresponding to different probes have a different support because of the variable length of the probes $|\mathcal{R}_I|$. In simulation k of the circuit, the tool receives one sample per probe ($sample_I^k$). The sample is added to one of the two histograms, depending on the value of the unshared (secret) input x. This is illustrated in Figure 4. A circuit with K input wires requires 2^K simulations. When all simulations are complete, the two histograms are compared for each probe. If a probe is found for which the histograms do not match, the circuit is

not d-glitch immune. The circuit is d-glitch-extended probing secure if for each d-probe, the histograms are identical for each secret. Note that the functionality of this part is independent of the security order d. The verification tool builds and compares histograms based on variable-width samples it receives from the software implementation. Whether these samples were built from one, two or three probes in the preprocessing step, is of no consequence.

Optimizations. It is possible that different wires in the circuit have the same set of inputs ($\mathcal{R}_i = \mathcal{R}_j$ for $i \neq j$). Also by concatenating different glitch extended probes for higher order security, it is possible to obtain duplicates ($\mathcal{R}_i \cup \mathcal{R}_j = \mathcal{R}_k \cup \mathcal{R}_l$ for $i \neq j \neq k \neq l$). We avoid redundant samples by removing all but one copy of each $\mathcal{R}_I$ from the final list of probes.

The c implementation simulates the circuit at bit level, *i.e.* each multiple-bit variable has been split into single-bit variables and only bitwise operators (AND,OR,NOT,XOR) are used. This allows us to bitslice the simulation: using for example 32-bit integers, we can simulate the circuit for 32 different inputs in parallel on a single core.

Flaw detection vs. validation. We point out the stark contrast between two use cases for the tool. On the one hand, it might be used as a flaw detection mechanism or to "prove" that a circuit is insecure. When a flaw is present, it does not take the tool a lot of effort or time to find it since for large circuits, many optimizations are possible to speed up the process. For example, one can first compare distributions only on a certain subset of the support. Only if the distributions are the same on such a subset of the support, we proceed with a different subset, and so on. This divide-and-conquer method can be distributed among several cores/workers.

On the other hand, one might want to *prove* the security of a circuit. In this case, there are no shortcuts to be made and the tool must exhaust all inputs and compare *exact* and complete histograms. This can be a very slow process for complex circuits.

In what follows, we demonstrate both use cases for small and larger gadgets.

4.2 Small Gadgets: Provable Security

For small gadgets and small security order d, it is feasible to verify d-glitch immunity exhaustively for every d-tuple of register input wires $(q_{i_1}, \ldots, q_{i_d})$. By "exhaustive", we mean we can simulate the circuit for every possible shared input (including fresh random inputs) and build the *exact* joint probability distributions for the glitch-extended probes $\mathcal{R}_I = \mathcal{R}_{i_1} \cup \mathcal{R}_{i_2} \cup \ldots \cup \mathcal{R}_{i_d}$. We then compare these probability distributions for different secrets and if they are not identical, we corroborate that there is a dependency on the secret which *could* result in leakage of sensitive information through some glitch function on those inputs. On the other hand, when the probability distributions of all input sets $\mathcal{R}_I$ are identical for each secret, one can conclude with certainty that the gadget is *provably* secure.

Application to Higher-Order Threshold Implementations. As an example, we consider the higher-order threshold implementation described in [BGN+14]. This higher-order secure KATAN construction has been discussed in depth in other works [SM15, RBN+15, Rep16] and it is well known that it exhibits a multivariate flaw. In particular, the authors of [RBN+15] claim that the secret is leaked when the construction is iterated and one combines probes from cycle 1 and cycle 7. Using the above described tool, we now find that multiple iterations are not needed and that multi-variate leakage of the secret occurs even within one iteration of the round function, i.e. by combining probes from cycle 1 and cycle 2. We recall the mini-cipher described in [RBN+15, Eq. (5)], which targets second-order security. The round function receives three inputs a, b, c, each in five shares. The circuit computes a five-share representation of $d = ab \oplus c$. This is done in two stages. In the first step, d is computed in ten shares:

$$
\begin{aligned}
d_0 &= c_1 \oplus a_1b_1 \oplus a_0b_1 \oplus a_1b_0 & d_1 &= c_2 \oplus a_2b_2 \oplus a_0b_2 \oplus a_2b_0 \\
d_2 &= c_3 \oplus a_3b_3 \oplus a_0b_3 \oplus a_3b_0 & d_3 &= c_0 \oplus a_0b_0 \oplus a_0b_4 \oplus a_4b_0 \\
d_4 &= a_1b_2 \oplus a_2b_1 & d_5 &= a_1b_3 \oplus a_3b_1 \\
d_6 &= c_4 \oplus a_4b_4 \oplus a_1b_4 \oplus a_4b_1 & d_7 &= a_2b_3 \oplus a_3b_2 \\
d_8 &= a_2b_4 \oplus a_4b_2 & d_9 &= a_3b_4 \oplus a_4b_3
\end{aligned}
$$

The ten shares are stored in registers for the sake of non-completeness. Next, they are compressed back to five shares $(\tilde{d}_0, \ldots, \tilde{d}_4)$ by $\tilde{d}_i = d_i$ for $i < 4$ and $\tilde{d}_4 = d_4 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_8 \oplus d_9$. Now, consider two (glitch-extended) probes: the first on $q_1 = \tilde{d}_4$, *i.e.* $\mathcal{R}_1 = \{d_4, d_5, d_6, d_7, d_8, d_9\}$ and a second on $q_2 = b_0$, *i.e.* $\mathcal{R}_2 = \{b_0\}$. The tool detects that the joint probability distribution of $\mathcal{R}_1 \cup \mathcal{R}_2 = \{d_4, d_5, d_6, d_7, d_8, d_9, b_0\}$ is not independent of the secret. Recall that our condition assumes the worst possible glitch function, which does not necessarily occur in the circuit. However, in this case we find that it only takes a glitch on $\tilde{d}_4$ that reveals the intermediate result $d_4 \oplus d_9$. Indeed, the joint distribution of $(d_4 \oplus d_9, b_0)$ depends on the unshared secret b. The single round function circuit is therefore not second-order secure in the presence of glitches. We use a single core of a 3.2 GHz Intel Core i5-6500 CPU. It takes the tool 2,1 seconds to find the flaw. We can correct the compression phase as follows:

$$
\begin{aligned}
\tilde{d}_0 &= d_0 \oplus d_4 \\
\tilde{d}_1 &= d_1 \oplus d_7 \\
\tilde{d}_2 &= d_2 \oplus d_9 \\
\tilde{d}_3 &= d_3 \oplus d_8 \\
\tilde{d}_4 &= d_5 \oplus d_6
\end{aligned}
$$

It takes the tool 12,5 seconds to enumerate all glitch extended probe tuples and compute the exact joint probability distributions for every secret input. With the new compression, the probability distributions are identical for each secret. Note that the gadget is still vulnerable when iterated, as discussed in [RBN+15].

4.3 Larger Circuits: Practical Security

Information theoretic vs computational security. In practice, it typically suffices to have computational security. It was for example noted by Daemen [Dae16a, Dae16b] that it is unclear how to exploit the non-uniformity in his initial Keccak S-box threshold implementation [BDPVA10]. It is then useful to relax d-glitch immunity. We can instead bound the divergence between secret-conditional distributions.

$$\max |p(\mathcal{R}|x^{(0)}) - p(\mathcal{R}|x^{(1)})| \leq \epsilon \tag{16}$$

where ϵ is an arbitrarily small number (*e.g.* 2^{-40}). Because, if we need a lot of samples (measurements) to distinguish two secret-conditioned distributions in a noiseless simulation, an attacker will also need at least as many traces during DPA. Eqn. (4) can be seen as an extreme version of Eqn. (16) when setting $\epsilon = 0$.

Practical Verification. The number of inputs to a masked circuit can easily grow to a point where exhaustive simulation becomes infeasible. In that case, we draw random inputs to feed to the software implementation. We still enumerate every possible d-tuple of register input wires $\mathcal{Q}_I = \{q_{i_1}, \ldots, q_{i_d}\}$ and verify the condition for the corresponding sampled distribution of $\mathcal{R}_I$. For each pair of histograms, we use a Pearson's chi-squared test to verify the null hypothesis that "the two distributions are identical". Our implementation is similar to the one described in [MRSS18]. We compute the p-value of the test, which records the probability of our observations under the null hypothesis. We reject the null hypothesis if the p-value is below a threshold (in this case 10^{-5}). As in [BCD+13, MRSS18], we perform the hypothesis test using two plaintext classes: one fixed and one random, although fixed versus fixed is also possible.

The statistical approach implies that this tool can no longer be used for *provable* security. However, the verification is extremely informative and relevant for *practical* security evaluation. It cannot be compared to tests done in a noisy lab environment [BCD+13] since the tool achieves a lot more accuracy and genericity. Firstly, the tool assumes an ideal noise-free world. It is also independent of the physical aspects implying that we do not need to repeat the test as the layout, library, device etc. change. Moreover, the chi-squared test is much more powerful than the moment-based t-test in its ability to detect *any* dependency of distributions on the secret. Note also the difference with the TVLA methodology used in practical evaluations. There, one constructs a set of traces *once* and evaluates d^{th}-order security by comparing the different order moments of the samples. In contrast, for each security order d, we build different samples and compare entire probability distributions. It is as if we leak entire glitch-extended probes under an identity leakage model to an adversary in a noiseless environment and verify whether the secret can be distinguished from this. We demonstrate the tool's ability to find flaws with some examples below.

Application to higher-order threshold implementations. We repeat the experiment from § 4.2, but this time with the statistical tool. We perform a fixed versus random test with fixed input $(a, b, c) = (0, 0, 0)$. It only takes the tool 0,5 seconds to detect the flaw, when simulating 100 000 times with random inputs. The sample where the dependency is detected corresponds to that found by the exact tool. The chi-square statistic χ and degrees of freedom ν are respectively 30 068.8 and 2047, corresponding to a p-value of 0.0.

Application to the Keccak round function. We further apply the tool to the first-order secure round function of a Keccak implementation from [GSM17], which was shown to be insecure in the presence of glitches due to a missing register stage in [ABP+18]. We simulate the whole round function, including all linear operations, since they introduce dangerous dependencies in the inputs to the non-linear blocks. The circuit thus has a 200-bit input in two shares as well as 200 bits of randomness as input. We simulate the round function 1 million times and indeed detect the flaw (p-value 0.0) at the input of the non-linear blocks. This takes approximately 3 minutes.

4.4 Advantages and Extensions

Ease of use and types of masking. A considerable advantage of glitch immunity, is that we do not need to know the "roles" of the wires, *i.e.* which share and which variable a wire belongs to. The tool only needs to know which wires are combined in a combinational block and this is easily derived from the netlist. This is in contrast with tools that must prove for example non-completeness or non-interference. In those cases, the tool needs to know not only which share is carried by each wire but also which sets of wires carry shares of the same secret. The more user-provided input a tool needs, the more room for human error. Another consequence is that the verification is independent of the type of masking (Boolean, multiplicative, polynomial, ...) used in the circuit. Other tools publicly available today assume a Boolean masking (*e.g.* [BGI+18]).

Non-uniform sharings. Recall that uniformity is not a necessary condition for security. By adapting the distribution we draw random inputs from, the tool can also be used to validate the security of low entropy masking schemes. These have only been studied for application in software. Given Lemma 1, investigating low entropy masking for hardware is an interesting direction for future work.

Possibility for leakage functions. A further tradeoff can be made between the accuracy and the efficiency of the verification. While we normally build probability distributions for the exact glitch extended probes, we can also replace this "identity function" by another leakage function, such as for example the Hamming weight. This reduces the histogram sizes considerably and thus allows for the validation of even larger circuits for *practical* security.

Possibility for software implementations. Finally, the tool can naturally also be used to verify the probing security of masked software implementations. Instead of investigating the joint probability distribution of glitch extended probes $\mathcal{R}_I$, we then simply create histograms for the normal probes $\mathcal{Q}_I$. We consider for example the third-order Boolean-to-arithmetic masking conversion from Hutter and Tunstall [HT16][4], which was proven to be flawed by [Cor17b]. We create a netlist of Algorithm 3 for the field $GF(2^2)$ and feed it to the tool. With 1 000 simulations in 10 seconds, a dependency is detected in the probe (V_{26}, V_{34}, V_{37}) (where we use the notation from [HT16, App. B]) with $\chi = 174, \nu = 7$ and thus the p-value is 0.0. We also perform 100 000 and 10 million simulations of the second-order conversion [HT16, Alg. 2]. This takes respectively 28.6 seconds and 4 minutes 51 seconds. No flaw is detected as the minimum p-value is around 0.002418 in both cases.

5 Composability in the Presence of Glitches

When circuits get large, it is infeasible to verify probing security by exhaustive probing. For this reason, the concept of strong non-interference has been introduced for masked software implementations. In this section, we unify the treatment of this security notion with glitch immunity (Property 1) in order to achieve a formal definition for composability in the presence of glitches. The probes in the following definitions are instantaneous.

***d*-non-interference (NI) [BBP+16].** *A gadget is d-non-interferent (or d-NI) if and only if every set of at most d probes can be simulated with at most d shares of each input.*

***d*-strong non-interference (SNI) [BBP+16].** *A gadget is d-SNI if and only if for every set $\mathcal{I}$ of t_1 probes on intermediate variables (i.e., no output wires or shares) and every set $\mathcal{O}$ of t_2 probes on output shares such that $t_1 + t_2 \leq d$, the set $\mathcal{I} \cup \mathcal{O}$ of probes can be simulated using at most t_1 shares of each input.*

When a gadget is d-NI, it is also d-probing secure. However, probing security (and d-NI) is not composable and verifying a circuit by simulating probes on every single wire is not scalable. The authors of [BBD+16] therefore propose to build each circuit from composable gadgets. A gadget is considered composable if it is d-SNI. This essentially means the following two things:

- If a gadget is d-SNI, it is d-NI and thus d-probing secure.
- Any circuit built from d-SNI gadgets, is d-NI and thus d-probing secure.

The use of composable gadgets allows a divide-and-conquer approach for the security verification of large circuits. The ease of verification comes at the cost

[4] Version of 22 Dec 2016

of higher implementation and randomness cost since the demands for strong non-interference are quite high. NI and SNI are both sufficient but not necessary conditions for probing security of masked software implementations. This is illustrated by first-order threshold implementations [NRR06], which are 1-probing secure but neither 1-NI, nor 1-SNI. In addition, SNI is a sufficient but not necessary condition for the composability of gadgets.

The concepts of NI and SNI were defined without regard for hardware defaults such as glitches and the tools available for its verification are only suitable for software implementations [BBD+15, Cor17a]. In [FGP+17], the authors provide a *robust d*-(S)NI definition as follows in order to extend the definition for glitchy circuits.

(g, t, c)-robust d-probing secure *(or d-NI/SNI) circuits are secure in the d-probing model (or d-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).*

In this paper we only consider glitches, *i.e.* $(1, 0, 1)$-robust d-probing. Despite these new definitions, it remains unclear how to automate the verification of composability of hardware gadgets.

In this section, we first redefine NI and SNI for masked software implementation from an information-theoretic point of view. This eventually allows to extend the definitions for masked hardware implementations. For simplicity, we assume from now on that the masked implementation uses $d + 1$ shares even though similar treatment is possible for more shares.

5.1 Redefining Non-Interference

We consider $\mathcal{Q}$ any set of at most d probes in a gadget with input sharing $\boldsymbol{x}$. According to the NI definition, the view of an adversary that probes $\mathcal{Q}$ should be perfectly simulatable by a simulator that has access to only d of the $d+1$ input shares. The view of the adversary is created using knowledge of all input shares $\boldsymbol{x}$, *i.e.* we write it as $p(\mathcal{Q}|\boldsymbol{x})$. Without loss of generalization (wlog), we assume that the simulator uses all input shares except x_i. The view of the simulator can hence be written as $p(\mathcal{Q}|\boldsymbol{x}_{\bar{i}})$.

We therefore consider the following property:

Property 2. *A gadget with $d+1$ input shares $\boldsymbol{x}$ satisfies Property 2 if and only if for any observation set of at most d probes $\mathcal{Q}$, the following condition holds:*

$$\exists i : I(\mathcal{Q}; x_i|\boldsymbol{x}_{\bar{i}}) = 0 \tag{17}$$

In other words, each probe set $\mathcal{Q}$ and at least one share x_i must be conditionally independent given the other shares $\boldsymbol{x}_{\bar{i}}$.

It is well known that d-non-interference is stronger than d-probing security, *i.e.* d-NI implies Eqn. (3) for any set $\mathcal{Q}$. The same can be said for Property 2:

Lemma 4. *Property 2 implies d-probing security, i.e.*

$$I(\mathcal{Q}; x_i | \boldsymbol{x}_{\bar{i}}) = 0 \Rightarrow I(\mathcal{Q}; x) = 0$$

Proof. We use two properties:

1 $I(\boldsymbol{x}_{\bar{i}}, x) = 0$; This is obviously a necessary requirement for any $d+1$-sharing of the input. In particular, it has been shown to be automatically true when the input sharing is uniform in [Bil15, Lemma 5]
2 $p(\mathcal{Q}|\boldsymbol{x}, x) = p(\mathcal{Q}|\boldsymbol{x}) = p(\mathcal{Q}|\boldsymbol{x}_{\bar{i}}, x)$; The equality follows from the redundant information in $(\boldsymbol{x}, x)$.

It follows that:

$$\begin{aligned} p(\mathcal{Q}, x) &= \sum_{\boldsymbol{x}^*_{\bar{i}}} p(\mathcal{Q}, \boldsymbol{x}^*_{\bar{i}}, x) \\ &= \sum_{\boldsymbol{x}^*_{\bar{i}}} p(\mathcal{Q}|\boldsymbol{x}^*_{\bar{i}}, x) p(\boldsymbol{x}^*_{\bar{i}}, x) \end{aligned}$$

Now we use Property 2 and the two properties above:

$$\begin{aligned} &= \sum_{\boldsymbol{x}^*_{\bar{i}}} p(\mathcal{Q}|\boldsymbol{x}^*_{\bar{i}}) p(\boldsymbol{x}^*_{\bar{i}}) p(x) \\ &= \sum_{\boldsymbol{x}^*_{\bar{i}}} p(\mathcal{Q}, \boldsymbol{x}^*_{\bar{i}}) p(x) \\ &= p(\mathcal{Q}) p(x) \end{aligned}$$

□

Moreover, we can carry this one step further and show that d-non-interference and Property 2 are equivalent.

Lemma 5. *A gadget with $d+1$ input shares $\boldsymbol{x}$ is d-NI if and only if it satisfies Property 2, i.e. d-non-interference and Property 2 are equivalent.*

Proof. Clearly, Eqn. (17) is equivalent to the matching of the adversary's view to the simulator's view, *i.e.*

$$\exists i : I(\mathcal{Q}; x_i | \boldsymbol{x}_{\bar{i}}) = 0 \Leftrightarrow \exists i : p(\mathcal{Q}|\boldsymbol{x}) = p(\mathcal{Q}|\boldsymbol{x}_{\bar{i}})$$

$\Rightarrow$ We prove by contraposition that d-NI implies Property 2: if there exists a set $\mathcal{Q}$ for which Eqn. (17) is not true, then we cannot simulate $\mathcal{Q}$ using only d shares.
$\Leftarrow$ In the other direction, Property 2 implies the existence of a simulator for the original NI definition.

□

For uniform input sharings. We can simplify the information-theoretic NI condition when the input sharing is uniform. In that case, we can state that disjunct sets of shares (*e.g.* $\boldsymbol{x}_{\bar{i}}$ and x_i) are independent.

Property 3. *A gadget with a uniform $d+1$ input sharing $\boldsymbol{x}$ satisfies Property 3 if and only if for any observation set of at most d probes $\mathcal{Q}$, the following condition holds:*

$$\exists i : I(\mathcal{Q}; x_i) = 0 \tag{18}$$

In other words, each probe $\mathcal{Q}$ must be marginally independent of at least one input share x_i.

Lemma 6. *A gadget with a uniform $d+1$ input sharing $\boldsymbol{x}$ is d-NI if and only if it satisfies Property 3.*

Proof. We use Lemma 5 and the independence of $\boldsymbol{x}_{\bar{i}}$ of x_i, *i.e.* $p(\boldsymbol{x}_{\bar{i}}|x_i) = p(\boldsymbol{x}_{\bar{i}})$.

$$\begin{aligned}
p(\mathcal{Q}|x_i) &= \sum_{\boldsymbol{x}^*_{\bar{i}}} p(\mathcal{Q}, \boldsymbol{x}^*_{\bar{i}}|x_i) \\
&= \sum_{\boldsymbol{x}^*_{\bar{i}}} p(\mathcal{Q}|\boldsymbol{x}^*_{\bar{i}}, x_i) p(\boldsymbol{x}^*_{\bar{i}}|x_i) \\
&= \sum_{\boldsymbol{x}^*_{\bar{i}}} p(\mathcal{Q}|\boldsymbol{x}^*_{\bar{i}}) p(\boldsymbol{x}^*_{\bar{i}}) \\
&= \sum_{\boldsymbol{x}^*_{\bar{i}}} p(\mathcal{Q}, \boldsymbol{x}^*_{\bar{i}}) \\
&= p(\mathcal{Q})
\end{aligned}$$

□

5.2 Redefining *Strong* Non-Interference

The NI and SNI definitions are very similar apart from the number of input shares that can be used in the simulation. We can thus redefine strong non-inteference in a similar way.

Property 4. *Consider a gadget with $d+1$ input shares $\boldsymbol{x}$. Let $\mathcal{Q}$ be any observation set of at most d probes of which $t_1(\mathcal{Q})$ are intermediates and $t_2(\mathcal{Q})$ are output probes such that $t_1(\mathcal{Q}) + t_2(\mathcal{Q}) \leq d$. The gadget satisfies Property 4 if and only if for any such $\mathcal{Q}$ the following condition holds:*

$$\exists T \subset \{0, \ldots, d\} \text{ with } |T| = t_2(\mathcal{Q}) + 1 \text{ such that } I(\mathcal{Q}; \boldsymbol{x}_T | \boldsymbol{x}_{\bar{T}}) = 0 \tag{19}$$

Lemma 7. *A gadget with $d+1$ input shares $\boldsymbol{x}$ is d-SNI if and only if it satisfies Property 4, i.e. d-strong non-interference and Property 4 are equivalent.*

The proof is very similar to that of Lemma 5. We leave it to the reader.

When no outputs are probed and thus $t_2(\mathcal{Q}) = 0$, then Eqn. (19) aligns with Eqn. (17). For example, when $d = 2$, the tuples of probes can be divided into three groups.

- for each tuple of probes $\mathcal{Q}$ for which $t_2(\mathcal{Q}) = 0$, verify that $\exists i : I(\mathcal{Q}; x_i | \boldsymbol{x}_{\bar{i}}) = 0$.
- for each tuple of probes $\mathcal{Q}$ for which $t_2(\mathcal{Q}) = 1$, verify that $\exists i \neq j : I(\mathcal{Q}; (x_i, x_j) | \boldsymbol{x}_{i,\bar{j}}) = 0$ or equivalently $\exists i : I(\mathcal{Q}; \boldsymbol{x}_{\bar{i}} | x_i) = 0$.
- for each tuple of probes $\mathcal{Q}$ for which $t_2(\mathcal{Q}) = 2$, verify that $I(\mathcal{Q}; \boldsymbol{x}) = 0$

When the input sharing is uniform, we can assume that for any set of indices $T \subset \{0, \ldots, d\}$, $\boldsymbol{x}_T$ is independent from $\boldsymbol{x}_{\bar{T}}$, *i.e.* two disjunct sets of shares are independent of each other. Then, the conditional independence can be replaced by marginal independence.

5.3 Towards Composability in the Presence of Glitches

These information-theoretic definitions are easily extended to include security in the presence of glitches. Following the adversary model of this work, we should only replace each probed wire q_i with its glitch extended probe $\mathcal{R}_i$.

Property 5. *Consider a gadget with d+1 input shares $\boldsymbol{x}$. Let $\mathcal{Q} = (q_{i_1}, q_{i_2}, \ldots, q_{i_d})$ be any observation set of $t \leq d$ wires and let $\mathcal{R} = \mathcal{R}_{i_1} \cup \mathcal{R}_{i_2} \cup \ldots \cup \mathcal{R}_{i_d}$ be the corresponding glitch extended probe. Let t_1 be the number of intermediate wires in $\mathcal{Q}$ and t_2 the number of output wires in $\mathcal{Q}$ such that $t_1 + t_2 = t \leq d$. The gadget satisfies Property 5 if and only if for any such $\mathcal{Q}$ the following condition holds:*

$$\exists T \subset \{0, \ldots, d\} \text{ with } |T| = t_2 + 1 \text{ such that } I(\mathcal{R}; \boldsymbol{x}_T | \boldsymbol{x}_{\bar{T}}) = 0 \qquad (20)$$

Lemma 8. *A gadget with d+1 input shares $\boldsymbol{x}$ is d-SNI in the presence of glitches if it satisfies Property 5.*

From now on, we refer to Property 5 as d-Glitch Strong Non-Interference (d-GSNI). It also corresponds to (1,0,1)-robust d-SNI as defined in [FGP$^+$17].

How to use d-GSNI? We illustrate Lemma 8 by adopting one of the examples of Faust *et al.* [FGP$^+$17]. Consider the following well-known second-order secure multiplier of three shares (x_0, x_1, x_2) and (y_0, y_1, y_2): In the first stage, nine

products $x_i y_j$ are computed, of which only the cross-terms are remasked:

$$\begin{aligned}
t_{0,0} &= x_0 y_0 \\
t_{0,1} &= x_0 y_1 \oplus r_1 \\
t_{0,2} &= x_0 y_2 \oplus r_2 \\
t_{1,0} &= x_1 y_0 \oplus r_1 \\
t_{1,1} &= x_1 y_1 \\
t_{1,2} &= x_1 y_2 \oplus r_3 \\
t_{2,0} &= x_2 y_0 \oplus r_2 \\
t_{2,1} &= x_2 y_1 \oplus r_3 \\
t_{2,2} &= x_2 y_2
\end{aligned} \tag{21}$$

These intermediate nine shares $t_{i,j}$ are stored in a register. In the next stage, they are compressed into three output shares.

$$\begin{aligned}
z_0 &= [t_{0,0}]_{reg} \oplus [t_{0,1}]_{reg} \oplus [t_{0,2}]_{reg} \\
z_1 &= [t_{1,0}]_{reg} \oplus [t_{1,1}]_{reg} \oplus [t_{1,2}]_{reg} \\
z_2 &= [t_{2,0}]_{reg} \oplus [t_{2,1}]_{reg} \oplus [t_{2,2}]_{reg}
\end{aligned} \tag{22}$$

Next, as argued in [FGP+17, §5.2], it is necessary to store also these three shares into a register in order to achieve composability. Indeed, if we consider Eqn. (22) directly as output wires, then the multiplication is not d-GSNI. We verify this by enumerating every single input sharing $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. For each input sharing, we compute the probability distribution of the glitch extended probes of an observation set of two outputs ($t_2 = 2$). For example, for observation set $\{z_0, z_1\}$, we investigate the distribution of $\{t_{0,0}, t_{0,1}, t_{0,2}, t_{1,0}, t_{1,1}, t_{1,2}\}$. For Eqn. (20) to be satisfied, this distribution must be identical for each input sharing. This is not the case. On the other hand, the distribution of $\{z_0, z_1\}$ itself *is* identical for each input sharing $(\boldsymbol{x}^*, \boldsymbol{y}^*)$, *i.e.* $I(\{z_0, z_1\}; (\boldsymbol{x}, \boldsymbol{y})) = 0$. Hence, a register is indeed needed before the output. In that case $\{t_{0,0}, t_{0,1}, t_{0,2}, t_{1,0}, t_{1,1}, t_{1,2}\}$ is the extended probe of an observation set that corresponds to two *intermediate* wires, *i.e.* $t_2 = 0$. We enumerate all possible $(x_0^*, x_1^*, y_0^*, y_1^*)$. For each, we check that the probability distribution of $\{t_{0,0}, t_{0,1}, t_{0,2}, t_{1,0}, t_{1,1}, t_{1,2}\}$ is identical for each (x_2^*, y_2^*). This is the case, hence the conditional mutual information of the glitch extended probe with input shares (x_2, y_2) is zero:

$$I(\{t_{0,0}, t_{0,1}, t_{0,2}, t_{1,0}, t_{1,1}, t_{1,2}\}; (x_2, y_2) | (x_0, x_1, y_0, y_1)) = 0$$

As a final example, consider the observation set $\{[z_0]_{reg}, t_{0,1}\}$ consisting of one output wire and one intermediate wire ($t_2 = 1$). Its glitch extended probe is $\{z_0, x_0, y_1\}$[5]. For each fixed (x_0^*, y_1^*), we find that the probability distribution of $\{z_0, x_0, y_1\}$ is constant for each $(x_1^*, x_2^*, y_0^*, y_2^*)$. The distributions are shown in Table 5. Hence $I(\{z_0, x_0, y_1\}; (x_1, x_2, y_0, y_2) | (x_0, y_1)) = 0$.

[5] The random input has no influence on the probability distributions.

Table 5. Scaled probability distributions of the glitch extended probe of observation set $\{[z_0]_{reg}, t_{0,1}\}$

		$p(z_0, x_0, y_1 \mid \boldsymbol{x}^*, \boldsymbol{y}^*)$							
(y_1^*, x_0^*)	$(x_1^*, x_2^*, y_0^*, y_2^*)$	000	001	010	011	100	101	110	111
(0,0)	*	4	4	0	0	0	0	0	0
(0,1)	*	0	0	4	4	0	0	0	0
(1,0)	*	0	0	0	0	4	4	0	0
(1,1)	*	0	0	0	0	0	0	4	4

Moreover, since we are working with uniform shares, there is no need to fix (x_0, y_1). The probability distribution $p(z_0, x_0, y_1 | x_1^*, x_2^*, y_0^*, y_2^*)$ is $(4, 4, 4, 4, 4, 4, 4, 4)$ for any $(x_1^*, x_2^*, y_0^*, y_2^*)$, hence $I(\{z_0, x_0, y_1\}; (x_1, x_2, y_0, y_2)) = 0$.

This example does not tell us more than the proof already given in [FGP+17] but serves to illustrate that Property 5 could be used to create similar proofs in a more automated way.

6 Conclusion

In this work, we revisited an information-theoretic approach to d-probing security and extended it to include glitches. The result is a new concept of d-glitch immunity, a necessary and *sufficient* condition for d-probing security in the presence of glitches.

We related glitch immunity to the security properties that form the basis of threshold implementations. While non-completeness is necessary and implied in glitch immunity, we showed that uniformity is not a necessary condition.

We demonstrated a tool based on glitch immunity that can be used both for proving the security of small gadgets and for the practical evaluation of larger circuits. The tool does not verify SNI, but has the potential to be extended with this functionality. Instead, in its current state, the tool can be used for customized circuits that reuse randomness and are not SNI, yet are glitch probing secure.

Finally, we redefined the software security notions of (strong) non-interference in the information-theoretic framework. We extended the definitions to include glitches and introduced the concept of d-glitch strong non-inteference, a sufficient condition for composability of hardware gadgets.

Acknowledgements This work was supported in part by the NIST Research Grant 60NANB15D346. Oscar Reparaz and Begül Bilgin are postdoctoral fellows of the Fund for Scientific Research - Flanders (FWO) and Lauren De Meyer is funded by a PhD fellowship of the FWO. The authors would like to thank Thomas De Cnudde, François-Xavier Standaert and Vincent Rijmen for fruitful discussions.

References

ABP+18. Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic Keccak: SCA security and low latency in HW. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):269–290, 2018.

AG01. Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and aes, secure against some attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

ANR17. Victor Arribas, Svetla Nikova, and Vincent Rijmen. VerMI: Verification tool for masked implementations. *IACR Cryptology ePrint Archive*, 2017:1227, 2017.

BBD+15. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.

BBD+16. 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, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

BBP+16. 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, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

BCD+13. G Becker, J Cooper, E DeMulder, G Goodwill, J Jaffe, G Kenworthy, T Kouzminov, A Leiserson, M Marson, P Rohatgi, et al. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, volume 1001, page 13, 2013.

BDPVA10. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Building power analysis resistant implementations of Keccak. In *Second SHA-3 candidate conference*, volume 3, page 2. Citeseer, 2010.

Bel15. Sonia Belaïd. *Security of Cryptosystems Against Power-Analysis Attacks. (Sécurité des cryptosystèmes contre les attaques par analyse de courant)*. PhD thesis, École Normale Supérieure, Paris, France, 2015.

BFGV12. Josep Balasch, Sebastian Faust, Benedikt Gierlichs, and Ingrid Verbauwhede. Theory and practice of a leakage resilient masking scheme. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December*

2-6, 2012. Proceedings, volume 7658 of *Lecture Notes in Computer Science*, pages 758–775. Springer, 2012.

BGI+18. Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal verification of masked hardware implementations in the presence of glitches. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.

BGN+14. Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *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*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

Bil15. Begül Bilgin. *Threshold implementations : as countermeasure against higher-order differential power analysis*. PhD thesis, University of Twente, Enschede, Netherlands, 2015.

Cor17a. Jean-Sébastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. *IACR Cryptology ePrint Archive*, 2017:879, 2017.

Cor17b. Jean-Sébastien Coron. High-order conversion from Boolean to arithmetic masking. In Fischer and Homma [FH17], pages 93–114.

Dae16a. Joan Daemen. On non-uniformity in threshold schemes. `https://www.cosic.esat.kuleuven.be/events/acm-ccs2016/wp-content/uploads/sites/4/2016/11/nonUniformVienna.pdf`, 2016. Talk.

Dae16b. Joan Daemen. On non-uniformity in threshold sharings. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 41. ACM, 2016.

FGP+17. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and Franois-Xavier Standaert. Composable masking schemes in the presence of physical defaults and the robust probing model. Cryptology ePrint Archive, Report 2017/711, 2017. `http://eprint.iacr.org/2017/711`.

FH17. Wieland Fischer and Naofumi Homma, editors. *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*. Springer, 2017.

GM10. Berndt M. Gammel and Stefan Mangard. On the duality of probing and fault attacks. *J. Electronic Testing*, 26(4):483–493, 2010.

GSM17. Hannes Groß, David Schaffenrath, and Stefan Mangard. Higher-order side-channel protected implementations of KECCAK. In Hana Kubátová, Martin Novotný, and Amund Skavhaug, editors, *Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017*, pages 205–212. IEEE Computer Society, 2017.

HT16. Michael Hutter and Michael Tunstall. Constant-time higher-order Boolean-to-arithmetic masking. *IACR Cryptology ePrint Archive*, 2016:1023, 2016.

ISW03. Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

KJJ99. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

Koc96. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

MM17. Thorben Moos and Amir Moradi. On the easiness of turning higher-order leakages into first-order. In Sylvain Guilley, editor, *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*, volume 10348 of *Lecture Notes in Computer Science*, pages 153–170. Springer, 2017.

MPG05. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

MRSS18. Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the x2-test. *IACR Trans. on Cryptographic Hardware and Embedded Systems*, 2018(1), 2018.

NRR06. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

PGA06. Emmanuel Prouff, Christophe Giraud, and Sébastien Aumônier. Provably secure s-box implementation based on fourier transform. In Louis Goubin and Mitsuru Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*, volume 4249 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2006.

RBN+15. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

Rep16. Oscar Reparaz. Detecting flawed masking schemes with leakage detection tests. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 204–222. Springer, 2016.

RP10. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

Sie84. Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. Information Theory*, 30(5):776–780, 1984.

SM15. Tobias Schneider and Amir Moradi. Leakage assessment methodology - a clear roadmap for side-channel evaluations. *IACR Cryptology ePrint Archive*, 2015:207, 2015.

SP06. Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.