IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 2, pp. 136–158. DOI:10.46586/tches.v2021.i2.136-158

Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry

Gaëtan Cassiers, François-Xavier Standaert

ICTEAM/ELEN/Crypto Group, UCLouvain, Belgium
{gaetan.cassiers,fstandae}@uclouvain.be

Abstract. There exists many masking schemes to protect implementations of cryptographic operations against side-channel attacks. It is common practice to analyze the security of these schemes in the probing model, or its variant which takes into account physical effects such as glitches and transitions. Although both effects exist in practice and cause leakage, masking schemes implemented in hardware are often only analyzed for security against glitches. In this work, we fill this gap by proving sufficient conditions for the security of hardware masking schemes against transitions, leading to the design of new masking schemes and a proof of security for an existing masking scheme in presence of transitions. Furthermore, we give similar results in the stronger model where the effects of glitches and transitions are combined.

Keywords: Side-Channel Analysis · Masking Countermeasure · Physical Defaults · Transition-Based leakages · Glitches · Robust Probing model · Composability

1 Introduction

Masking is a well-known countermeasure against side-channel attacks. A common form of masking is Boolean masking: during computations, a sensitive value $x \in \mathbb{F}_q$ is replaced with a sharing $(x_0, \dots, x_{d-1}) \in \mathbb{F}_q^d$ such that $\sum_i x_i = x$. Moreover, each operation in the computation (viewed as an arithmetic circuit gate) is implemented by a so-called *gadget* that operates on sharings. The probing model [ISW03] is often used to evaluate the security of masking schemes: a computation is t-probing secure if any set of t intermediate variables (leaked "probes") is independent of any sensitive value. Being high-level, it is relatively easy to study complex schemes with this model, and furthermore probing security is a solid starting point to prove security in more realistic, low-level leakage models [DDF19]. In this work, we use $d = t + 1$ shares, since it helps minimizing the cost of the implementation.

Faust et al. [FGP+18] formalized an extension of the probing model: the robust probing model, where probes are extended. Whereas probes in the probing model give access to a single value, extended probes leak multiple intermediate variables. This model, while still being high-level, takes into account physical phenomenons such as glitches, transitions or couplings that may break the practical security of masking schemes [MPG05, CGP+12, CBG+17] that are secure in the probing model.[1]

Glitches are transient "incorrect" computations occurring in combinatorial logic that can be modeled with glitch-extended probes, which leak all the inputs of the probed combinatorial circuit. Glitches have drawn a lot of attention over the last years and

[1]We note that there are other physical defaults (e.g. couplings) or practical concerns (e.g. not analyzing the full circuit or the EDA transformations) that can affect the security of masking [CBG+17, CGLS20]. Whether they can be captured with abstract models is an interesting open problem.

Received: 2020-10-15 Accepted: 2020-12-15 Published: 2021-02-23

many hardware masking schemes are designed to mitigate them (see [NRS11, MPL+11, RBN+15, GMK16]). In particular, the “non-completeness” requirement of “Threshold Implementations” (TIs) [NRS11] (no combinatorial logic should manipulate all the shares), can be seen as a design principle counterpart to the glitch-robust probing model.

Transitions (changes in value of a wire, whose leakage depends on both the current and previous value of the wire) have been shown to be very damaging to software implementations [CGP+12], leading to countermeasures of various efficiency and formalization levels [BGG+14]. They can also be modeled using extended probes: a transition-extended probe leaks the values carried by a wire at two consecutive execution cycles. Despite evidence that transitions are the source of significant leakage in hardware implementations [HSS12], security against transition-based leakages has been devoted little attention in the hardware masking design literature.

We now give a simple example of an implementation strategy that would lead to security-damaging transition leakage: let $L : \mathbb{F}_q^m \to \mathbb{F}_q^n : (x_0, \ldots, x_{m-1}) \mapsto (y_0, \ldots, y_{n-1})$ be a linear function that we want to mask. Since the Boolean masking is linearly homomorphic, this can be implemented share-by-share: let $(x_{i,0}, \ldots, x_{i,d-1})$ be a sharing of x_i, the masked L can be implemented by evaluating L for each share: $(y_{0,j}, \ldots, y_{n-1,j}) = L(x_{0,j}, \ldots, x_{n-1,j})$. In hardware, it is common to implement this gadget using d independent instances of the circuit computing L, which increases significantly the area over the non-masked implementation. Another simple strategy is to trade area for execution time by serializing the implementation: only one instance of L is implemented, and it is used sequentially for each of the shares (as shown in Figure 2a). If no care is taken, and a new set of shares is input in the circuit at each cycle, then transitions can occur, and leak two shares at once. This effectively halves the security order of the implementation.

In this paper, we study the problem of transitions in hardware masked implementations, aiming mainly to prove that some schemes are secure in the transition-robust probing model. Our motivations for this purpose are threefold.

First, we want to provide formal security guarantees that scale with any number of shares and can prevent possibly undiscovered attacks that would exploit transitions. We believe this threat should not be neglected, since popular masking schemes have already been broken in the past (see [MMSS19] for a recent example).

Second, we want to give insights explaining why many published designs do not seem to suffer from transitions, even if there is no proof of their security in that context. From that point of view, the main takeaway message is that some of the conditions for security in our proofs are either satisfied by informal “good practices” used in hardware masking design, or by designs that best fulfill other (e.g. performance) objectives.

Third, we want to enable automation of the security analyses of hardware implementations. Resting on solid theoretical background, automated security verification tools may enable designers to experiment with new designs without putting the security at risk, or having to perform extensive security analysis by hand, which requires both specialized skills and a significant amount of time.

We also analyze the security in a stronger model with glitches and transitions combined. Security in both glitch- and transition-robust probing models considers leakage coming from transition between stable values, or from glitches depending on the combination of the inputs of a combinational circuit. The combined glitch- and transition-robust probing model (denoted glitch+transition-robust probing model) can be interpreted as having glitches that depend on the transitions. That is, on both the current inputs of a combinational circuit and on the previous inputs. This combined model seems to be more realistic than the one considering glitches and transition in isolation [MS06].

Overall, our results extend the applicability of the probing model to a wider range of realistic threats and their combination. They can be used to confirm existing intuitions and to guide new designs. It turns out analyzing transitions is non-trivial and requires additional refinements of the standard circuit models used in probing security proofs. But proofs show that efficient design strategies can be used to rule out these physical defaults and that some state-of-the-art gadgets already lead to implementations that are glitch+transition-probing secure [CGLS20].

Contributions The two main contributions of this work are:

- A formal model for hardware circuits suited to robust probing model analyses that includes the notion of execution cycles and the relationship between operations that are executed by the same gate, and
- Provably secure hardware masking schemes in the transition-robust and glitch+transition-robust probing models, both simple (based on a trivial composition strategy [CS20]) and more optimized/specialized.

Structure The paper is structured as follows: we first introduce necessary background in Section 2. Next, in Section 3, we explain informally the main conditions required for a hardware masked circuit to be transition-robust. Our formal hardware circuit model and the various probing models are described in Section 4. Section 5 covers the security proofs, recalling previous results in the probing model and its glitch-robust variant, then introduces our result for the transition-robust and glitch+transition-robust models. We provide preliminary performance evaluations in Section 6 and conclude in Section 7.

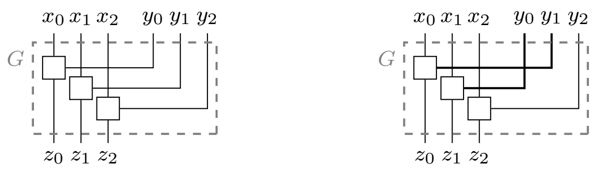
2 Background

In this section, we introduce the main theoretical concepts on which our work is based. We give an intuitive description of those concepts, and give their formalization within our structural circuit model (see Section 4.1) in Section 5. We first recall the framework of simulatability-based definitions for gadgets and the accompanying proof technique (the probe propagation technique) that enables secure composition of gadgets. Next, we cover the Probe-Isolating Non-Interference (PINI) definition and its compositional properties.

Composition of masked gadgets refers to the idea of building complex circuits by connecting together gadgets that implement elementary field operations. The observation that the composition of t-probing secure gadgets is not always t-probing secure [CPRR13] led to the introduction of stronger security definitions that enables proving the probing security of composite circuits using the properties of the gadgets. Simulatability is a building block for this purpose: if the probes on a gadget can be perfectly simulated (i.e. produced with the same distribution as the actual circuit) using only some of its input shares, the security analysis of a composition of gadgets can analyze the sets of input shares only.

The probe propagation technique [BBP+16] proves the security of composite circuits by iteratively using the simulation of gadgets: a gadget made of the composition of two gadgets can be simulated if the probes in a gadget can be simulated using some shares that are the outputs of another gadget and if, for this gadget, the required output shares (so-called propagated probes) and the (internal) probes can be simulated using some of its inputs. This process can be generalized to composition of a larger number of gadgets, and results in a set of shares that suffices to simulate all the probes in the circuit. If those input shares are independent of any sensitive value, so are the probes.

Simulatability-based definitions are properties of gadgets that further simplify the analysis of composite circuits. They describe how probes propagate from the outputs and from the inside to the inputs of a gadget, that is, which input shares are needed to


x_0 x_1 x_2 y_0 y_1 y_2
G
z_0 z_1 z_2
x_0 x_1 x_2 y_0 y_1 y_2
G
z_0 z_1 z_2

(a) Share-isolating **(b)** Non share-isolating

Figure 1: Example of (a) share-isolating and (b) non share-isolating gadgets. Gadget (b) is not share-isolating since its output z_0 depends on y_1.

simulate which output and internal probes. A *(simulatability-based) compositional strategy* is a policy that assigns a property (in the form of a simulatability-based definition) to each of the gadgets in a composite circuits [CGLS20]. It guarantees that if the circuit is instantiated with gadgets that satisfy those definitions, then the circuit is probing secure. Out of the many compositional strategies that exist (see [BBD$^+$16, CS20, BBD$^+$]), we focus on the ones based on the PINI simulatability-based definition.

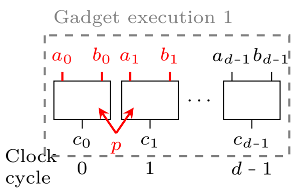
The PINI composition strategy [CS20] was originally introduced for the standard probing model (no glitches nor transitions): a circuit composed of PINI gadgets can be analyzed as if it was split into d shares, and an adversary putting a probe in a circuit share gets only information about the inputs to that circuit share, while learning nothing about the other shares. This property is trivially satisfied by gadgets that can be implemented in a share-wise manner such as linear gadgets (we call them share-isolating, see example in Figure 1). Other gadgets (e.g. multiplications) cannot be implemented in this way, but if they are PINI, they somehow simulate this isolation from an input-output point of view: a probe on an output share depends only on the input shares that are in the same circuit share, and a probe inside the gadget depends on the inputs contained in at most one circuit share. The PINI composition strategy is a *trivial composition strategy*: if a gadget is composed of only PINI gadgets, then it is itself PINI (hence probing secure).

The security based on isolated circuit shares is well suited to withstand glitches: since glitches propagate through wires, a probe that observes glitches in a circuit share gets information only about that circuit share. This observation is the basis for the glitch-robust composition strategy of [CGLS20], which states that composing glitch-robust PINI gadgets guarantees probing security in the presence of glitches. The glitch-robust PINI definition cares for glitches that happen inside a gadget, imposing that, when looking “from outside”, they seem to touch only one circuit share. The propagation of glitches outside of gadgets causes no security issue thanks to the circuit shares principle.

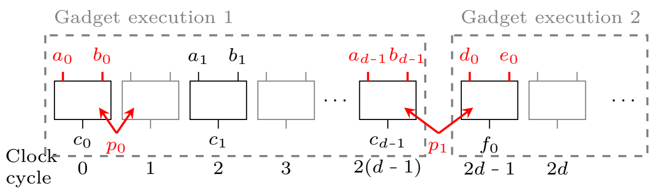
3 Transitions: problems and solutions

In this section, we first give an overview of the issues that can arise when there are transition leakages in a composite masked hardware circuit. We then provide the intuition for the solutions proposed in this paper.

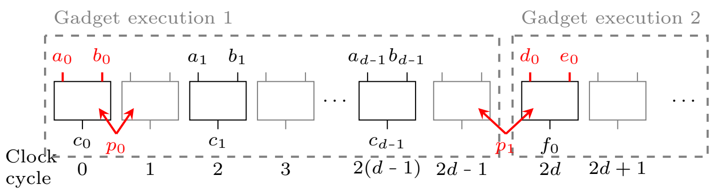
At first, it may seem that the transitions can be handled in a similar way as glitches in the PINI composition strategy. Indeed, since we model a transition-leaking probe as giving knowledge of the value carried on a wire at two consecutive cycles, a transition-leaking probe in a share-isolating gadget gives only knowledge about one circuit share. Similarly to glitch-robust PINI, we can define transition-robust PINI gadgets as gadgets that can be simulated like PINI gadgets even when there are transition probes in the gadget, leaving the transitions that span multiple gadgets to be handled by the composition theorem. Transition-robust PINI gadgets are however not trivially composable for two reasons that


Gadget execution 1
a_0 b_0 a_1 b_1 a_{d-1} b_{d-1}
c_0 p c_1 c_{d-1}
Clock cycle
0 1 d - 1

(a) Transition vulnerable


Gadget execution 1
Gadget execution 2
a_0 b_0 a_1 b_1 a_{d-1} b_{d-1} d_0 e_0
c_0 p_0 c_1 c_{d-1} p_1 f_0
Clock cycle
0 1 2 3 2(d - 1) 2d - 1 2d

(b) Transition-robust, not iterated transition-robust


Gadget execution 1
Gadget execution 2
a_0 b_0 a_1 b_1 a_{d-1} b_{d-1} d_0 e_0
c_0 p_0 c_1 c_{d-1} p_1 f_0
Clock cycle
0 1 2 3 2(d - 1) 2d - 1 2d 2d + 1

(c) Iterated transition-robust

Figure 2: Serial implementation of a 2-input linear gadget: the same gates are used sequentially for all the shares. Each double arrow indicates a transition-extended probe and red inputs are required for simulation. Gadget (2a) is not transition-robust PINI, hence broken in one execution due to transition leakage across shares. Gadget (2b) has interspersed non-sensitive cycles, hence it is transition-robust PINI (p_0 does not break the security), but it is not iterated transition-robust: a transition-extended probe p_1 covers two distinct circuit shares in the two executions. Gadget (2c) is iterated transition-robust PINI thanks to the non-sensitive cycle interspersed between executions.

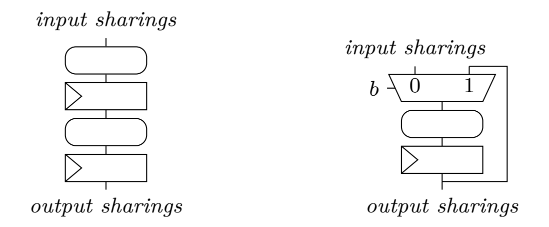
we explain next.

The first reason comes from the difference between the structural (i.e. physical) gates/wires in a gadget, and their algorithmic (i.e. logical) position in the computations performed by the gadget. Indeed, the same physical gadget (a set of physical gates and wires) can be *executed* multiple times with different inputs. Let us assume that a single physical gate appears in different logical positions for two executions of a gadget and there is a transition probe that touches this gate over the two executions. If this can happen without restriction, the two probes are at somewhat independent positions, leading to an effective doubling of the number of probes. An example for this (illustrated in Figure 2) would be an implementation of a linear gadget that intends to minimize area, by applying the operation iteratively for each share. If the processing of each share takes one cycle (Figure 2a), the gadget is not transition-robust: a transition on an input wire leaks two shares from the same input sharing, reducing the security order.[2] However, if a cycle with a non-sensitive (e.g. constant) input is interspersed between the two cycles processing those shares (Figure 2b), then the gadget is transition-robust PINI. Under this property, it is however still allowed to start a second execution immediately after the first one, even when this leads to transition leakage (Figure 2c). This transition leakage (across two different shares) may break the security if the input sharings for the two executions of the gadget are not independent.[3]

A solution to this issue is the *iterated transition-robust* property, which cares about multiple executions of a gadget and requires that when a transition across two executions exists, it appears to probe twice the same logical gate (from an external simulation point of view). This constraint is satisfied by many gadgets, such as the ones built as a pipeline,

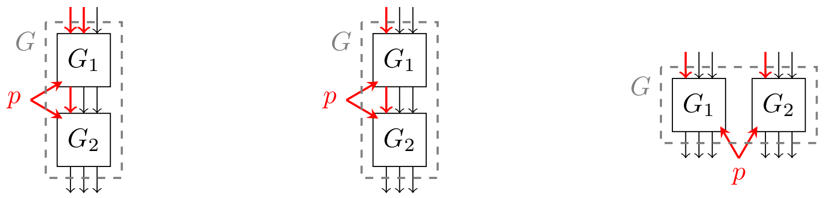
[2]We focus our constructions on masking with $d = t + 1$ shares (where t is the security order) to reach good performance at higher orders. The models and proofs are however valid for other kinds of masking.

[3]In order to achieve generic composition, we cannot assume that the input sharings are independent.



(a) Pipeline gadget **(b)** Non-pipeline gadget

Figure 3: Gadgets implementing the same function in two clock cycles. Rounded boxes represent combinational logic, separated by registers. Gadget (a) is pipeline gadget while gadget (b) is not pipeline ($b = 0$ at the first execution cycle and $b = 1$ at the second cycle).



(a) Serial composition of two iterated transition-robust PINI gadget executions.

(b) Serial composition of two iterated transition-robust O-PINI gadget executions.

(c) Parallel composition of iterated transition-robust PINI gadget executions.

Figure 4: Composition of iterated transition-robust PINI and O-PINI gadget executions with $d = 3$ shares and one transition-extended probe (inputs required for simulations in red). Composition 4a is insecure, while 4b and 4c are secure.

that is, when each structural gate is used once per execution, as shown in Figure 3.

The second reason is an issue that appears when there is a transition between two executions of an iterated transition-robust PINI gadget and additionally an output of the first execution is an input of the second one (Figure 4a). Assuming that such a transition-extended probe is the only probe, simulating the second execution requires one share of each of its input sharings. For the first execution, an output share has thus to be simulated in addition to the internal probe. Hence, the simulator may require two shares of each input sharing. This shows that such a composition is not transition-robust PINI (which could then be exploited to break its probing security).

The intuition to fix this problem comes again from share-isolating gadgets, for which the previous example does not have a problem: we have the guarantee that for the first execution, the internal probe and the output shares to simulate are in the same circuit share. Only one share of each input is therefore required for simulation. The PINI property mimics the share isolation property at the input and output levels, which is why it composes smoothly. However, transitions do not only touch the inputs and outputs, but also the internals of the gadget. In the internals, PINI is counting probes, and not circuit shares, which is insufficient to capture all transitions. We therefore need a property that is even closer to share isolation than PINI, while still allowing to mix circuit shares internally (otherwise some gadgets are impossible to implement). A key observation is that for share isolating gadgets, if the simulator knows the inputs in a given circuit share, then it is able to simulate all outputs in that circuit share in addition to the probes (see Figure 5 for illustration). This property is not necessarily satisfied by PINI gadgets, and we name O-PINI (output-probe isolating non-interferent) the PINI gadgets that, given a set of probes, and knowing a set of input shares that satisfies the PINI constraints, can simulate

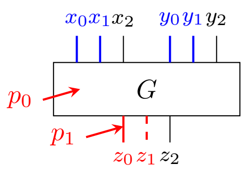


Figure 5: O-PINI gadget. There is one output probe on z_0 and one internal probe p (in red). The simulator has thus access to input shares with two distinct indexes, one of them being index 0 (in blue). Since the simulator has access to input shares with index 1, it must simulate the output share with index 1 (in dashed red).

jointly both the probes and their output shares for which they know the input shares with the same index. An (iterated transition-robust) O-PINI gadget solves our example (Figure 4b): the simulator is asked to simulate the internal probes in both executions, hence requires the inputs with the same share index for both executions. Since it also simulates the outputs with that share index for the first execution, the second gadget can be simulated, and all of this requires only knowledge of inputs in one circuit share.

Another solution, that allows using more efficient gadgets, is to prevent cases similar to the example from happening by constraining how gadgets are composed. The constraint is that adjacent gadgets (i.e. gadgets that a transition-extended probe may cover) must be parallel: none of their inputs may depend of one of their outputs. (Gadget in Figure 4c is a parallel composition while gadget in Figure 4a is not.) This strategy is more efficient as it allows using PINI gadgets instead of O-PINI ones, at the cost of not being a trivial composition strategy. The parallelism requirements results in a more complex security analysis (it must be verified), and restricts the structure of the circuit. However, we will show that a common way of implementing substitution-permutation networks (SPNs), which are arguably the most commonly masked algorithms, satisfies the parallelism assumption.

4 Circuit & security models

4.1 Circuit model

We use a circuit model based on the one of [ISW03]: a circuit is a directed acyclic graph whose vertices are gates and edges are wires carrying elements from a finite field. In the robust probing model, transition-extended probes leak pairs of wires [FGP$^+$18]. The exact specification of such pairs is not determined in the ISW circuit model as it lacks the notion of cycles and does not encode the fact that two logical gates may be implemented by the same physical gates.Therefore, we introduce a more specific circuit model which takes these into account. We first define physical circuits as sets of gates and wires.

Definition 1 (Structural gate)**.** A structural gate is a tuple (I, P, f, lat), where

- I is the set of inputs of the gate,
- P is the set of parameters, it is the disjoint union of P^P (public parameters) and P^S (secret parameters),
- $f : (I \to \mathbb{F}_q) \times (P \to S) \to \mathbb{F}_q$ (where S is any set: the parameters may belong to any set) is the evaluation function,
- $lat : I \to \mathbb{N}$ is the latency of the inputs (a latency of one means the input is required one cycle before the output is produced).

Any structural gate has one output whose value is given by the evaluation function.

We note that the latency is defined "backwards" by taking the output as reference (while it is usually defined forwards as the number of cycles needed to produce the output after the input is available) . The reason for this choice is that our gates always have a single output while they may have any number of inputs. This convention makes our following technical treatments lighter.

We next give a few examples of gates:

- 2-input XOR: $XOR = (\{0,1\}, \emptyset, (f_i, f_p) \mapsto f_i(0) \oplus f_i(1), x \mapsto 0)$ (and similarly for any arithmetic gate).
- Register: $Reg = (\{0\}, \emptyset, (f_i, f_p) \mapsto f_i(0), x \mapsto 1)$.
- 2-input MUX with select bit being a public value:

  $$MUX2 = (\{0,1\}\{0\}, (f_i, f_p) \mapsto f_i(0) \cdot f_p(0) \oplus f_i(1) \cdot (1 - f_p(0)), x \mapsto 0),$$

  where the public parameter 0 is either 0 or 1.
- Randomness generating gate (random gate): $Rnd = (\emptyset, \{0\}, (f_i, f_p) \mapsto f_p(0), \emptyset)$. Randomness is encoded as a random tape in the f_p function, for the secret parameter 0.
- An input gate is encoded similarly to the random gate.

We also define wires that connect the output of a gate to the input(s) of a gate:

Definition 2 (Structural wire)**.** A structural wire is a pair $(g, (g', i))$ where g and g' are gates and i is an input of g'. We say that the wire connects its source g to the input i of g' (its destination).

Based on these definitions, we define structural circuits as follows.

Definition 3 (Structural circuit)**.** A structural circuit is a directed graph whose nodes are structural gates and whose edges are structural wires. A wire connects its source to its destination. In a structural circuit, there must be no combinational loop, that is, no cycle for which all the wires have a destination with latency 0.

Remark The notion of outputs of the circuit is not needed in this work. However for completeness, they may be defined as a map from an output set to the gates of the circuit.

Next, we consider the execution of a circuit over time (cycles).

Definition 4 (Circuit execution)**.** An execution of a structural circuit $C = (G, W)$ for the set of cycles $T = \{t_0, \ldots, t^*\}$ is a directed graph whose set of nodes is $G \times T$ (the gates) and whose set of edges is $W \times T$ (the wires). Wires connect gates according to their latency: let l be the latency of the destination of the structural wire $w = (g, (g', i))$, then the wire (w, t) connects its source $(g, t - l)$ to its destination (g', t). If the source does not exist, then the wire is connected to a fresh "initial state" source gate (no-input gate having as output a public parameter). Each gate may be annotated with a parametrization function, mapping the set of all (resp. public) parameters of the underlying structural gate to values, in which case the execution is full (resp. partial).

Definition 5 (Circuit evaluation)**.** The evaluation of a full circuit execution is a function that maps every gate and wire to an element $\mathbb{F}_q$ or $\perp$. The values are computed recursively by applying the evaluation function for the gates, lazily evaluating the parametrization function for the parameters and the evaluation function of the circuit for the inputs. The result is $\perp$ if the recursion does not terminate.

An evaluation of a partial circuit execution is the evaluation of the full execution obtained using random functions as the evaluation functions of private parameters.

In the following, we work with partial circuit executions. This concept matches the model of circuit from [ISW03, CGLS20] since time ordering and absence of combinational loop guarantee that a circuit execution is a directed acyclic graph (DAG). The addition we bring to these models correspond to the structural information needed to identify the possible transition-extended probes.

We now clarify the notion of gadget in our new framework. First, we define the gadget execution, which is similar to the gadget in the ISW circuit model.

Definition 6 (Gadget execution)**.** A gadget execution with d shares in a circuit execution C is a subset of gates G and wires W of C, such that W is the set of wires in C whose destination is in G. The inputs (also named input shares) of the gadget are the set of its wires whose source is not in G, and its outputs (or output shares) are a subset of the gates of the gadget. The inputs (resp. outputs) are partitioned in n_i (resp. n_o) tuples of d elements designated as input (resp. output) sharings.

The composition of gadgets forms a new gadget, with the constraint that no input a of gadget depends on one of its outputs.

Definition 7 (Gadget execution composition)**.** A gadget execution G is a composition of a set of disjoint gadget executions $(G_i)_{i=0,\dots,n}$ if G is the union of all G_i's. Connections between gadgets must connect the input sharings to the output sharings of the gadgets, respecting the order of shares. Input sharings of the composite gadget G are the input sharings of the composing gadgets G_i which are not connected to any output sharing of a G_i, and output sharings of G are a sub-set of the output sharings of the composing gadgets. The composing gadgets graph (directed graph where the nodes are the gadgets and the edges are the wires connecting them) must be a DAG.

In the "structural" world, defining a gadget as a set of structural gates and wires would lead to a loss of information on the behavior of the gadget (e.g. whether some gadget is a straight pipeline, or it has a serial implementation that uses multiple times the same gate). We therefore define structural gadgets by the fact that all its executions are identical gadget executions, except for a translation in time.

Definition 8 (Translation)**.** Let t be an integer and C a circuit execution. The translation by t of a set of gates $\{(g_i, t_i)\}_i$ and wires $\{(w_i, t_i)\}_i$ of C are the sets of gates $\{(g_i, t_i + t)\}_i$ and wires $\{(w_i, t_i + t)\}_i$. Two sub-sets C_1 and C_2 of C are translation-equivalent if there exists an integer t such that the translation of C_1 by t is equal to C_2.

Definition 9 (Structural gadget)**.** A structural gadget is a non-empty set of gadget executions that are all disjoint and translation-equivalent to each other. For each structural gadget we choose a canonical execution and then each gadget execution is associated to a canonicalization translation t such that the translation by t of the execution is the canonical execution.

We need a final definition to cover the intuitive idea that a structural gadget is an algorithm implemented by a set of wires and gates: when considering a set of structural gadgets, two gadgets should not share any gate or wire.

Definition 10 (Structural gadget composition)**.** The structural gadget S is a composition of structural gadgets $\{S_i\}$ if each of its executions is the union of some S_i executions (possibly multiple executions of each S_i). The composing gadgets S_i cannot intersect (i.e. share any structural gate of wire).

4.2 Probing model

In this section, we formalize the probing model and the robust probing model. Our definitions are adaptations of the definitions of [ISW03, FGP+18] to our circuit model of Section 4.1.

Definition 11 (Probing security). A set of probes P in a gadget execution G is a subset of its gates and input wires. In an evaluation of G with input values x, the values of the probes are denoted as $G_P(x)$. In an evaluation of G, the sensitive values are defined as the sums of the values on the wires of each input sharing. A gadget execution G is *secure* against a set of probes P if $G_P(x)$ is independent of the sensitive values when the inputs x are uniformly distributed. G is *t-probing secure* if it is secure against any P such that $|P| \leq t$.

The robust probing model is a generic model where there is a pre-determined set of possible extended probes for each gadget, which are sets of probes.

Definition 12 (Robust probing security). A probe expansion scheme is a function that maps a gadget execution to a set of extended probes, which are sets of probes in the gadget execution. The probes in those sets are named expanded probes. A gadget execution is t-robust probing secure with respect to a probe expansion scheme if it is secure against all the expanded probes from any set of t extended probes.

We are interested in three particular cases of the robust probing model: glitches, transitions and finally the combination of glitches and transitions. Those are defined by their probe expansion scheme that we describe next.

In the glitch-robust probing model, there is an extended probe for each gate and each wire. For input wires, the extended probe contains only the (non-extended) probe on this wire, while for the other wires, it is equal to the extended probe on the gate whose output is connected to the wire. For gates, the extended probe is made of the union of the extended probes on the gates that produce its input wires that have latency 0, or the extended probe on the wire itself if it is an input wire.

The transition-extended probes in a gadget are all the non-extended probes in it, and additionally the sets $\{(w, t-1), (w, t), \}$ for all w (wires or gate) and t such that both $(w, t-1)$ and (w, t) belong to the gadget execution.

Finally, in the transitions and glitches combined robust (transition+glitch-robust) probing model, the set of extended probes is obtained by first computing the set of transition-extended probes, then, replacing every expanded probe in a transition-extended probe with all the expanded probes contained in the glitch-extended probe that correspond to it. Physically, this models the fact that the propagation of a glitch (e.g. its timing) depends on both the new and the previous values on the wire.

5 Composition results

In this section, we present the formalization of the definitions and prove the security properties. In Sections 5.1 and 5.2, we cover the background presented in Section 2, adapting the notations to our circuit model. Next, in Section 5.3, we formalize the intuitions of Section 3, and we finally handle the transition+glitch case in Section 5.4, using similar techniques as for the transition-only case.

5.1 Composition in the probing model

In this section, we give the formal definitions of simulatability and PINI in the probing model, as well as the definition of a share-isolating gadget. See Section 2 for an intuitive introduction of those concepts.

Definition 13 (Simulatability [BBP+16])**.** A set of probes P in a gadget execution G can be simulated by a set of input shares $I = \{(i_1, j_1), \dots, (i_k, j_k)\}$ if there exists a randomized simulator algorithm S such that the distributions $G_P(x_{*,*})$ (the values of the probes) and $S(x_{i_1,j_1}, \dots, x_{i_k,j_k})$ are equal for any value of the inputs $x_{*,*}$.

Notation. For a gadget execution G, we denote by $x_{i,j}$ the j-th share of its i-th input sharing. The share index of $x_{i,j}$ is j. We denote by $x_{*,*}$ the set of all its inputs sharings.

Definition 14 (Probe-Isolating Non-Interference [CS20])**.** Given a gadget execution G, let I be a set of at most t_1 probes on its internal wires and O a set of probes on its output shares. Let A be the set of the share indexes of the shares in O, and $t_2 = |A|$. Let I and O be chosen such that $t_1 + t_2 \leq t$. The gadget G is t-PINI iff for all I and O there exist a set of at most t_1 share indexes B such that observations corresponding to I and O can be simulated using only the shares with indexes $A \cup B$ of each input sharing.

Definition 15 (Share-isolating gadget)**.** A structural gadget S with d shares is share-isolating if its set of structural gates, inputs and outputs can be partitioned in d components $(S_i)_{i=0,\dots,d-1}$ such that any input or output share with share index i belongs to component S_i, and the source and destination of every structural wire belong to the same component.

5.2 Composition with glitches

The construction of a glitch-robust and composable masking scheme in [CGLS20] is based on the adaptation of simulatability to the glitch-robust probing model.

Definition 16 (Glitch-robust simulatability [CGLS20])**.** A set of extended adversarial probes P in a gadget execution G can be glitch-robustly simulated by a set of input shares $I = \{(i_1, j_1), \dots, (i_k, j_k)\}$ if there exists a randomized simulator algorithm S such that the distributions $G_{rob,P}(x_{*,*})$ and $S(x_{i_1,j_1}, \dots, x_{i_k,j_k})$ are equal for any value of the inputs $x_{*,*}$ when there are no glitches on the inputs.

Extended probes may cover multiple gadgets, therefore it might seem complex to adopt a composable approach where each gadget is analyzed independently. The solution to this problem comes from a separation of responsibilities. Glitches inside a gadget are handled by the simulatability definition, while glitches outside the gadget (i.e. on its inputs) are ignored in the definition and are instead handled at the composition theorem level. We will use a similar approach to handle transitions.

This observation is the basis for a lemma that maps all simulation-based security properties and composition strategies to the glitch-robust probing model if the gadget satisfy the glitch-robust version of the simulatability properties.

Lemma 1 (Glitch-robust composability lemma [CGLS20])**.** *For a gadget execution G made of the composition of gadgets G_i, let S_i^{rob} be a glitch-robust probing simulator for each gadget G_i, and let S_i^{std} be its restriction to a standard probing simulator (by discarding the simulation outputs that are not needed). Let P be a set of standard probes that can be simulated using some inputs I of G using the simulators S_i^{std} according to a simulatability-based compositional strategy (i.e. each simulator is asked to simulate some probes P_i using inputs I_i of G_i where wires in I_i are either in I or in some $P_{i'}$ and $P \subset \bigcap_i P_i$).*

In the glitch-robust probing model, the set of extended probes P can be simulated using inputs I (on which there are no glitches by the definition of simulatability).

5.3 Composition with transitions

We now define simulatability for transitions, following the pattern of simulatability for glitches: transitions that happen inside a gadget must be simulated, but those that happen outside of the gadget (i.e. in other executions) are not considered: they will be taken into account in the composition theorems.

Definition 17 (Transition-robust simulatability.)**.** A set of extended adversarial probes P in a gadget G can be transition-robustly simulated by a set of input shares $I = \{(i_1, j_1), \dots, (i_k, j_k)\}$ if there exists a randomized simulator algorithm S such that the distributions $G_{tr,P}(x_{*,*})$ and $S(x_{i_1,j_1}, \dots, x_{i_k,j_k})$ are equal for any value of the inputs $x_{*,*}$, where $G_{tr,P}$ is the set of transition-expanded probes that are inside the gadget G (i.e. the probes outside G are discarded).

This definition extends any NI-based security property such as NI, SNI, PINI, … into a transition-robust variant. The next step towards a composition theorem with transitions is analyzing the behavior of transitions that cross gadgets. However, as discussed in Section 3, the composition of transition-robust PINI gadgets is not necessarily transition-robust itself: to achieve that result, we need iterated transition-robust gadgets and either O-PINI or parallel executions for adjacent gadgets. We next formalize those notions and prove the composition theorems.

Definition 18 (Iterated transition-robust simulatability.)**.** Let G be a structural gadget whose executions are $(G_i)_i$. Let P be a set of transition-extended probes in G. Let P'_j be the set of probes obtained by taking each probe in the expansion of P that is in a gadget G_i and translating it to the execution G_j (through the translation from G_i to G_j). Let I be a set of inputs of the canonical execution of G and I_i be its translation to G_i. If P'_i can be simulated by the set of inputs I_i in G_i for all i, then P is iterated transition-robust simulatable in G by the set I.

In the particular case of gadgets that are a simple pipeline, iterated transition-robust simulatability reduces to simulatability since there are no transitions inside a gadget and probes are always "at the same position" across gadgets.

Definition 19 (Pipeline)**.** A structural gadget is pipeline if its canonical execution G uses each of its structural wires and gates only once: for a gate g (resp. wire w), there exists no $t_1 \neq t_2$ such that $(g, t_1), (g, t_2) \in G$ (resp. $(w, t_1), (w, t_2) \in G$).

Lemma 2. *Let G be a pipeline structural gadget whose executions are $(G_i)_i$ and G_0 be its canonical execution. Let P (resp. I) be a set of probes on (resp. inputs of) G_0, and P_i (resp. I_i) be its translation to G_i. If P_i in G_i can be simulated using the set of inputs I_i for all i, then $P^* = \bigcup_i P_i$ is transition-robust simulatable in G by I.*

Proof. We first observe that the transition-expansion of P^* (restricted to G) is P^* itself: for any (w, t) in a G_i, if $(w, t-1)$ belongs to some G_j, then it is the translation of (w, t) from G_i to G_j, hence it belongs to P^*. Thus, for each gadget G_i, the set of probes to be simulated is P_i, which is simulatable using I_i by hypothesis. □

This lemma implies that if all executions of a pipeline structural gadget satisfy a simulation-based property (such as NI, SNI, PINI) in its transition-robust variant, then the structural gadget satisfies the iterated transition-robust variant. Iterated transition-robustness is however not sufficient since it does not solve the previously discussed problem of the serial composition of executions of the same structural gadgets.

5.3.1 Trivial composition approach

The following O-PINI property captures the requirement explained in Section 3 to compose circuits with transitions trivially: when the simulator has access to an input share index, it should simulate the outputs with the same share index. O-PINI implies PINI, but not the other way around, as shown next (on page 149).

Definition 20 (Output Probe-Isolating Non-Interference)**.** Given a gadget G, let I be a set of at most t_1 probes on its internal wires and O a set of probes on its output shares. Let A be the set of the share indexes of the shares in O, and $t_2 = |A|$. Let I and O be chosen such that $t_1 + t_2 \leq t$. The gadget G is t-O-PINI iff for all I and O there exist a set of at most t_1 share indexes B such that observations corresponding to I, O and output shares with index in B can be simulated using only the shares with indexes $A \cup B$ of each input sharing. We say a gadget is O-PINI if it is t-O-PINI for any t.[4]

First, we observe that share-isolating gadgets are O-PINI.

Proposition 1. *Share-isolating structural gadgets are iterated transition-robust O-PINI.*

Proof. Let G be a share-isolating structural gadget. Let I and O be sets of probes as in the O-PINI definition. Let $(I_i)_{i=0,\dots,n}$ be a partition of I according to the circuit share they are in. At most $|I|$ sets in the partition are non-empty, let B be the set of indexes of these sets. All probes in I and O can then be simulated using all the inputs with share index in $A \cup B$ (where A is the set of share indexes of the probes in O) since there is no path between the other inputs and the probes. ☐

We next prove that any composite gadget made of iterated transition-robust t-O-PINI gadgets is iterated transition-robust t-O-PINI.

Theorem 1 (O-PINI composability)**.** *Let $\{S_i\}$ be a set of iterated transition-robust t-O-PINI structural gadgets, and let $\{G_i\}$ be their executions. If the structural gadget S is a composition of $\{S_i\}$, then it is iterated transition robust t-O-PINI.*

Proof. We build an iterated transition-robust t-O-PINI simulator that takes as input a set of internal probes P, and a set of share indices of probed output shares A. Wlog, we assume that there is no probe on wires connecting gadgets: these can be considered as part of a gadget connected to the wire.

Let $(P_i)_i$ be a partition of P according the structural gadget S_i to which they belong. Let $A_i^0 = A$ and $B_i^0 = \emptyset$ for each gadget S_i. Then, the following algorithm is run for $k = 1, 2, \dots, k^*$, until it converges (i.e. $A_i^{k^*} = A_i^{k^*-1}$ for all i): using the iterated transition-robust O-PINI simulator for each S_i (for output shares A_i^{k-1} and internal probes P_i), the set of input shares B_i^k is computed, and each A_i^{k+1} is computed as follows: $A_i^k \leftarrow A \cup \bigcup_{j \neq i} B_j^k$. Let $A^* = A \cup \bigcup_i B_i^{k^*}$.

Let us observe that at the final iteration, each simulator takes input shares with index in A^* as input, simulates its internal probes that are in P, and simulates all its output shares with index in A^* for every gadget execution. The set of input shares required for simulation is A and $B = \bigcup_i B_i^{k^*} \supset A^* \setminus A$. The simulator proceeds to simulation by feeding each gadget (from input to output, i.e. respecting a topological ordering on the gadgets composition graph) with the inputs with share index in A^* to simulate probes in that gadget and outputs with share index in A^*.

Let us first prove that the algorithm converges: if for all i $A_i^k \subset A_i^{k+1}$, then $B_i^{k+1} \subset B_i^{k+2}$ (if the simulator of S_i requires the minimal input set, which can be assumed wlog), which implies that $A_i^{k+1} \subset A_i^{k+2}$ for all i. Since by construction $A = A_i^0 \subset A_i^1$, it follows by induction that for all i and k, $A_i^k \subset A_i^{k+1}$. Since all A_i^k have cardinality at most d, the algorithm converges.

Next, we prove that the simulation algorithm is correct, that is, all the values it generates have the same distribution as in the true circuit. This is guaranteed by using a correct simulator for each sub-gadget, and feeding it with correct inputs: by induction (starting with the inputs it receives), the simulator knows for each gadget the input shares

[4]As proven in [CGZ20], if a gadget with d shares is $d-1$-PINI, then it is t-PINI for any t, which is denoted as "PINI". The same argument applies to O-PINI.

with index in A^* before simulating the gadget. Therefore, the simulator for the gadget can simulate correctly the probes inside the gadget and the output shares with index in A^*.

Finally, we observe that the simulator simulates all required values: it simulates the translation of P to any execution of S thanks to the iterated transition-robust simulators that can simulate the translation of P_i (and of the outputs with index in $A_i^{k^*}$) to any execution of G_i with the input shares with index A^*. Furthermore, the simulator respects the cardinality constraint $|B| \leq |P|$: $B = \bigcup_i B_i^{k^*}$ and $P = \bigcup_i P_i$, while $\left|B_i^*\right| \leq |P_i|$ for all i (thanks to each gadget O-PINI simulator). □

Multiplication gadget. To instantiate the previous general construction, we build an iterated transition-robust O-PINI multiplication gadget in $\mathbb{F}_q$. We build the O-PINI1 multiplication gadget (Algorithm 2) from the HPC2 multiplication gadget and the observation that the ISW multiplication gadget (on which it is based) is SNI (in the probing model). Intuitively, the SNI property is relevant since it guarantees that outputs are somewhat "easy" to simulate, however it is not sufficient and we have to add a sharing of zero to the output.

As an example of why HPC2 (Algorithm 1) is not O-PINI, we take $d = 3$ and consider probes $\bar{a_1} \otimes r_{01}$ and $\bar{a_2} \otimes r_{21}$, the simulator has to know a_1, a_2, r_{01} and r_{21} (at least sometimes). Hence, it needs the input shares with index 1 and 2. It can however not simulate the output c_1, whose value is $a_1 \otimes (b_0 + b_1 + b_2) + r_{01} + r_{12}$: it does not know b_0, and it outputs r_{01} and r_{12}, thus those are known by the distinguisher.

Remark. We describe the gadget in an algorithmic way for the sake of readability, whereas formally we describe patterns for structural gadgets: a set of gates and wires, and a way to generate gadget executions out of them. The algorithmic description can be mapped to the structural gates and wires by unrolling the loops in the algorithm and adding a gate for each operation (and connecting the wires accordingly). Then, since the gadgets described in this way are all pipeline, there is only one way (up to a translation) of executing them such that the set of inputs matches the algorithmic description.

Algorithm 1 HPC2 multiplication.

```
Require: shares $(a_i)_{0\leq i\leq d-1}$ and $(b_i)_{0\leq i\leq d-1}$, such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$.
Ensure: shares $(c_i)_{0\leq i\leq d-1}$, such that $\bigoplus_i c_i = a \otimes b$.
  for $i = 0$ to $d-1$ do
    for $j = i+1$ to $d-1$ do
      $r_{ij} = r_{ji} \xleftarrow{\$} \mathbb{F}_q$
    end for
  end for
  for $i = 0$ to $d-1$ do
    for $j = 0$ to $d-1$, $j \neq i$ do
      $u_{ij} \leftarrow \bar{a}_i \otimes \mathsf{Reg}\left[r_{ij}\right]$
      $v_{ij} \leftarrow b_j \oplus r_{ij}$
    end for
  end for
  for $i = 0$ to $d-1$ do
    $c_i \leftarrow \mathsf{Reg}\left[a_i \otimes \mathsf{Reg}\left[b_i\right]\right] \oplus \bigoplus_{j=0,j\neq i}^{d-1} \left(\mathsf{Reg}\left[u_{ij}\right] \oplus \mathsf{Reg}\left[a_i \otimes \mathsf{Reg}\left[v_{ij}\right]\right]\right)$
  end for
```

We first prove that the gadget O-PINI1 is still glitch-robust PINI, relying on the fact that HPC2 is glitch-robust PINI.

Proposition 2. *The gadget O-PINI1 (Algorithm 2) instantiated in $\mathbb{F}_2$ is glitch-robust PINI.*

Algorithm 2 O-PINI1 multiplication gadget.

```
Input: shares $(a_i)_{0\leq i\leq d-1}$ and $(b_i)_{0\leq i\leq d-1}$, such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$.
 Output: shares $(c_i)_{0\leq i\leq d-1}$, such that $\bigoplus_i c_i = a \otimes b$.
  $(d_i)_i \leftarrow \mathrm{HPC2}\left((a_i)_i, (b_i)_i\right)$;
  for $i = 0$ to $d-2$ do
    $s_i \overset{\$}{\leftarrow} \mathbb{F}_q$;
  end for
  $s_{d-1} \leftarrow \bigoplus_{i=0}^{d-2} s_i$;
  for $i = 0$ to $d-1$ do
    $c_i \leftarrow d_i \oplus s_i$;
  end for
```

Proof. Any probe on an output c_i can be simulated by knowing d_i and probes on s_i can be simulated by taking fresh randomness. Hence, given a set of probes, a PINI simulator for O-PINI1 can map c_i probes to d_i probes and remove the s_i probes, then require simulation of all the probes by the HPC2 simulator and finally simulate the required c_i and s_i. □

Furthermore, O-PINI1 is iterated transition-robust O-PINI.

Proposition 3. *The gadget O-PINI1 is iterated transition-robust t-O-PINI for $t < d$.*

Proof. For the sake of brevity, we base this proof on the glitch+transition-robust O-PINI property of gadget O-PINI2, proven in Proposition 6.

We observe that the O-PINI2 gadget is equivalent to O-PINI1 regarding probing security and simulatability without glitches, since their only difference is a register layer. Indeed, in a pipeline gadget, a register layer does not change the set of possible transition-extended probes: the inputs of the registers are identical to the wires of the gadget without the registers, and probes on the outputs of the registers have the same leakage as probes their inputs (in the transition-robust probing model). □

5.3.2 Optimized composition approach

In this section, we analyze the particular case of the composition of parallel gadget executions (forming a *parallel gadget*), that is, disjoint executions such that none of them is reachable from another one in the gadget composition graph (i.e. there is no path between two executions). We show that when all the executions of an instance across which there are transitions are parallel executions, O-PINI gadgets are not required and PINI is actually enough. This particular case is of interest in practical implementations of block ciphers, when the same structural gadget is used multiple times, for example to compute sequentially multiple parallel S-boxes. Such an architecture is known as “S-box serial”, where *serial* refers to sequential executions, and should not be confused with the distinct notion of serial composition.

Proposition 4 (Parallel PINI executions)**.** *Let S be a structural gadget whose executions are all parallel. If S is iterated transition-robust t-PINI, then the structural gadget whose only execution is the parallel gadget is iterated transition-robust t-PINI.*

Proof. First of all, since there is only one execution, iterated transition-robustness is equivalent to transition-robustness, that we prove next. Let A be the set of shares indexes for which simulation outputs are required and let P be the set of internal probes. There exists a set of share indexes B of size at most $|P|$ such that, for each execution of S, the iterated transition-robust t-PINI simulator can simulate a set made of those output shares with index in A and the transition-expanded probes of P that are in that execution. □

We are interested in a full block cipher, meaning that there are multiple rounds, and therefore not all the S-boxes are parallel executions. We solve this by observing that if we leave one "empty" cycle between two rounds, no transition-extended probe in the S-boxes can touch more than one. The empty cycle condition is formalized by introducing the notion of adjacent executions: two gadget executions are adjacent to each other if there exists a wire w and a cycle t such that (w, t) belongs to one gadget and $(w, t-1)$ belongs to the other one.

This leads to the next composition theorem which allows PINI gadgets to ensure transition-robust composition, provided their executions are non-adjacent.

Theorem 2. *Let S be a structural gadget composition of iterated transition-robust t-O-PINI gadgets and transition-robust t-PINI gadgets. If every PINI gadget in S has no adjacent executions, then S is transition-robust t-PINI.*

Proof. We build a transition-robust t-PINI simulator for a set of internal probes P and a set of share indices of probed output shares A.

Let $(G_i)_{i=0,\dots,n-1}$ be the gadgets executions is S, in reverse topological order in the gadget composition graph (i.e. from output to input). Let $(S_j)_j$ be the multi-set of structural composing gadgets, where each O-PINI gadget appears once and each PINI gadget appears once for each of its executions. Let $s(i) = j$ if G_i is an execution of S_j and be surjective: two PINI gadgets G_i and $G_{i'}$ are never mapped to the same value. Let us partition the set of probes according to $(S_j)_j$: P_j is the set of probes in S_j for O-PINI gadgets, and it is the set of probes in G_i for PINI gadgets, where i is the only index satisfying $s(i) = j$.

Let $A_0 = A$, and initialize $C_j = C'_j C''_j = A'_j = \emptyset$ for all S_j. For all i, first update $C''_{s(i)} \leftarrow C'_{s(i)}$, $C'_{s(i)} \leftarrow C_{s(i)}$ and $A'_{s(i)} \leftarrow A_i$. Then, let B_i be the set of shares required by the O-PINI (resp. PINI) simulator for G_i when asked to simulate all transition-expanded probes from P_j G_i and all output shares with index in $A_i \setminus C'_{s(i)}$. Finally, set $A_{i+1} = A_i \cup B_i$ and $C_{s(i)} \leftarrow C'_{s(i)} \cup B_i$.

We next prove the following invariants by induction: after each i iteration, $C'_{s(i)} \subset B_i$, and $A_{i+1} = A \cup \bigcup_j C_j$. The base case $i = 0$ is trivial since $C_{s(0)} = \emptyset$. Let us now assume the invariants hold for $i-1$. If $C_{s(i)} = \emptyset$ (before being updated), then both invariants are trivially satisfied. Otherwise, assuming wlog that the simulators require a minimal input set, we know that $(A'_i \setminus C''_{s(i)}) \cap C'_{s(i)} = \emptyset$ and by induction $A'_i \subset A_i$, hence $A'_i \setminus C''_{s(i)} \subset A_i \setminus C'_{s(i)}$. The minimality of the simulator implies then that $C'_{s(i)} \subset B_i$ (between the two calls to the simulator, the set of internal probes is the same, and set of the output share indices grows). Since $C_{s(i)} = B_i$, this implies that $A_{i+1} = A_i \cup C_{s(i)} = (A_i \setminus C'_{s(i)}) \cup C_{s(i)} = A \cup \bigcup_j C_j$.

We observe that all transition-expanded probes from P are simulated and each simulator produces a correct simulation (provided it receives correct inputs): this follows directly from the iterated transition-robustness for O-PINI gadgets, while for PINI gadgets G_i, non-adjacency implies that all the expanded probes from $P_{s(i)}$ that belong to S also belong to G_i. We have $|C_j| \leq |P_j|$ (by the PINI/O-PINI definition), therefore, the simulator can ask a set of input share indexes $B = \bigcup_j C_j$ that satisfies $|B| \leq |P_j|$. This set of input share indexes (along with A) enables the generation of all the inputs required for the simulators, using simulators sequentially from G_{n-1} to G_0, since the simulator for each G_i generates its outputs with share index in A_i (either the gadget is O-PINI and it generates outputs $(A_i \setminus C'_{s(i)}) \cup C_{s(i)} \supset A_i$, or $C'_{s(i)} = \emptyset$). □

5.3.3 SPN implementation

We can now prove transition-robustness of the common *round-based and S-box serial* substitution-permutation network (SPN) implementation. This architecture iterates the executions of a round, where one round is made of some hardware implementing the

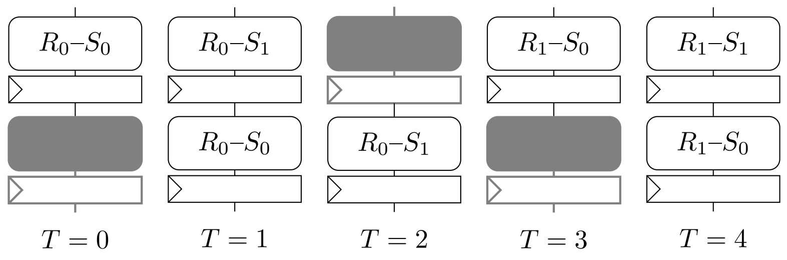


Figure 6: 2-cycle pipeline S-box in a dummy SPN with 2 S-boxes per round (assuming the linear layer takes zero cycles). At the first three cycles, the computations for round 0 (R_0) are executed: the first S-box (R_0–S_0) goes through the pipeline, followed by the second S-box (R_0–S_1). At the beginning and at the end, a pipeline stage is not used (i.e. does not contain sensitive data). The same goes on for the three next clock cycles computing the second round (R_1). The computation of the S-box R_1–S_0 cannot start at cycle $T = 2$ since the output of the first round is not yet computed. No transition can touch S-boxes of two different rounds thanks to this "bubble" (colored in gray) in the pipeline, hence transitions only touch parallel S-boxes.

permutation/linear layer, and some S-boxes (which may be used sequentially multiple times per round). If the S-box is pipeline and PINI, Proposition 4 implies that the gadget made of all its usages for one round is iterated transition-robust PINI. If furthermore all the other gadgets are share-isolating, and there is one cycle where no sensitive data is input to the S-boxes (i.e. they are not part of the cipher gadget for that cycle, see Figure 6), then Theorem 2 implies that the whole SPN implementation is transition-robust PINI (hence probing secure in the transition-robust probing model).

The block cipher implementations of [CGLS20] are based on a trivial glitch-robust PINI composition (using HPC1 or HPC2 multiplications and share-isolating gadgets), have pipeline S-boxes and correspond to the above architecture description. These implementations are thus both transition-robust and glitch-robust probing secure.

5.4 Composition with glitches and transitions

Unlike [FGP+18], we also consider the joint impact of both glitches and transitions. This choice is motivated by the goal of avoiding to rely on physical assumptions as much as possible. At the hardware level, this implies considering a worst-case modeling of glitch propagation, where the logic delay of the gates depends on their current input but also on their state (i.e. the past value of their inputs and output). We model this as the transition+glitch robust probing model, where probes are first transition-extended, then the resulting probes are glitch-expanded to get the set of standard probes.[5]

Security proofs in the transition+glitch robust probing model can be obtained by using Lemma 1. The first step is to have a simulatability-based compositional strategy for the transition-robust probing model, giving a composition of gadget executions that is secure against transitions. Given a set of transition-extended probes P, the circuit is thus secure in the probing model for P_t, the transition-expansion of P. Next, Lemma 1 applies: if all the gadgets are glitch-robust, then the composition is secure against P_t in the glitch-robust model, which means it is secure against P in the glitch+transition probing model.

We first formally define glitch+transition simulatability, then give the composition results, adapting to the glitch+transition setting to both the trivial and optimized composition approaches of the previous section.

[5]Expanding first the glitches, then the transitions gives the same set of standard probes.

Definition 21 (Glitch+transition-robust simulatability.)**.** A set of extended adversarial probes P in a gadget G can be glitch+transition-robustly simulated by a set of input shares $I = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ if there exists a randomized simulator algorithm S such that the distributions $G_{gl+tr,P}(x_{*,*})$ and $S(x_{i_1,j_1}, \ldots, x_{i_k,j_k})$ are equal for any value of the inputs $x_{*,*}$ assuming there are no glitches on the inputs, where $G_{gl+tr,P}$ is the set of glitch+transition-expanded probes inside G (the probes outside G are discarded).

The iterated glitch+transition-robust simulatability follows the same pattern.

Definition 22 (Iterated glitch+transition-robust simulatability.)**.** Let G be a structural gadget whose executions are $(G_i)_i$. Let P be a set of transition-extended probes in G. Let P'_j be the set of probes obtained by taking each probe in the extension of P that is in a gadget G_i and translating it to the execution G_j (through the translation from G_i to G_j). Let I be a set of inputs of the canonical execution of G and I_i be its translation to G_i. If P'_i can be glitch-robustly simulated by the set of inputs I_i in G_i for all i (assuming no glitches on inputs), then P is iterated glitch+transition-robust simulatable in G by the set I.

Corollary 1. *Let $\{S_i\}$ be a set of iterated glitch+transition-robust t-O-PINI structural gadgets, and let $\{G_i\}$ be their executions. If the structural gadget S is a composition of the gadgets $\{S_i\}$, then it is iterated glitch+transition-robust t-O-PINI.*

Proof. Theorem 1 is a simulatability-based composition strategy, hence Lemma 1 applies. □

Corollary 2. *Let S be a structural gadget composition of iterated glitch+transition-robust t-O-PINI gadgets and glitch+transition-robust t-PINI gadgets. If every PINI gadget in S has no adjacent executions, then S is glitch+transition-robust t-PINI.*

Proof. Theorem 2 is a simulatability-based composition strategy, hence Lemma 1 applies. □

Gadgets. We exhibit glitch+transition-robust gadgets, starting with share-isolating gadgets and following with a multiplication gadget.

Proposition 5. *Share-isolating structural gadgets are iterated glitch+transition-robust O-PINI.*

Proof. The proof is identical to the proof of Proposition 1, since the glitches cannot propagate if there is no connection between circuit shares. □

Algorithm 3 O-PINI2 multiplication gadget.

```
Input: shares $(a_i)_{0\le i\le d-1}$ and $(b_i)_{0\le i\le d-1}$, such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$.
 Output: shares $(c_i)_{0\le i\le d-1}$, such that $\bigoplus_i c_i = a \otimes b$.
  $(d_i)_i \leftarrow \mathrm{HPC2}((a_i)_i, (b_i)_i))$;
  for $i = 0$ to $d-2$ do
      $s_i \overset{\$}{\leftarrow} \mathbb{F}_2$;
  end for
  $s_{d-1} \leftarrow \bigoplus_{i=0}^{d-2} s_i$;
  for $i = 0$ to $d-1$ do
      $c_i \leftarrow \mathsf{Reg}[d_i \oplus \mathsf{Reg}[s_i]]$;
  end for
```

Let us now prove that O-PINI2 is iterated glitch+transition-robust O-PINI.

Table 1: Cost of multiplication gadgets in boolean gates, randomness usage and number of cycles of latency of the output with respect to each input.

	$\otimes$	$\oplus$	NOT	Reg $[\cdot]$	Randomness	Lat. a	Lat. b
DOM	d^2	$2d^2-2d$	0	d^2	$\frac{d(d-1)}{2}$	1	1
HPC2	$2d^2-2d$	$3d^2-3d$	d	$4d^2-2d$	$\frac{d(d-1)}{2}$	1	2
O-PINI1	$2d^2-2d$	$3d^2-d-2$	d	$4d^2-2d$	$\frac{d(d-1)}{2}+d-1$	1	2
O-PINI2	$2d^2-2d$	$3d^2-d-2$	d	$4d^2$	$\frac{d(d-1)}{2}+d-1$	2	3

Proposition 6. *The gadget O-PINI2 (Algorithm 3) is iterated glitch+transition-robust t-O-PINI for $t < d$ when instantiated on $\mathbb{F}_2$.*

Proof. First of all, we observe that O-PINI2 is pipeline, and it has no public parameter, hence by Lemma 2 (actually its mapping to the glitch-robust case, which is trivial by Lemma 1), we only have to prove that one execution of it is glitch-robust O-PINI.

We build a simulator based on the glitch-robust PINI simulator of HPC2. The simulation of probes is delegated to the HPC2 simulator, except for c_i and $d_i \oplus \mathsf{Reg}[s_i]$ probes which are given as d_i probes to the HPC2 simulator, and the s_{d-1} probe which is generated as specified by the gadget. Other probes are strictly less powerful and are thus ignored wlog. Simulation of c_i probes is then done by generating the s_i randomness and using the c_i from the HPC2 simulator. Finally, simulation of outputs c_i where i belongs to the set I of input share indexes required is done as follows: if all intermediate variables in c_i have already been simulated by the HPC2 simulator, perform as specified by the gadget. Otherwise, generate a fresh random variable.

We now discuss correctness of the simulation of c_i's (correctness for s_i's is trivial and for other variables it is a direct consequence of the correctness of the HPC2 simulator).

If there is a probe on s_{d-1}, then there are at most $d-2$ other probes. In that case, in the sum computing d_i, either d_i or $d_i \oplus \mathsf{Reg}[s_i]$ is probed (and simulation of c_i is trivially correct), or at most $d-2$ random r_{ij} are observed, since no probe except d_i and $d_i \oplus \mathsf{Reg}[s_i]$ can observe more than one r_{ij} at once. Therefore, at least one r_{ij} appearing as a term in the c_i expression is fresh (i.e. not observed except through c_i), thus simulation is correct. Otherwise, if d_i nor $d_i \oplus \mathsf{Reg}[s_i]$ is probed, s_i is fresh and therefore simulation is correct. □

Constructions. Similarly to the transitions-only case, we have two approaches. One uses trivial composition (which increases the latency of the multiplication by one cycle and increases the randomness requirements) based on the HPC1 gadget and Corollary 1. The other one is based on an optimized composition (Corollary 2) which proves that the HPC implementations [CGLS20] are glitch+transition-robust probing secure, since, in addition to the discussion of Section 5.3.3, the multiplication gadgets are glitch+transition-robust PINI (they are pipeline and glitch-robust PINI).

6 Performance

As a preliminary performance evaluation, we compare the strategies introduced in this work with a naive implementation based on the DOM-indep [GMK16] multiplication gadget. First, we give in Table 1 the amount of gates, the randomness usage and the latency for each of the multiplication gadgets. We can see the overhead in the number of gates of O-PINI1/O-PINI2 over HPC2 is limited while O-PINI2 significantly increases the latency over HPC2. All the gadgets have a throughput of 1 multiplication per cycle when the pipeline is full.

Table 2: Masked PRESENT implementation in 65 nm technology with various AND gadgets. The same data is presented in Figure 7 (Appendix A).

	Area (kGE)				Randomness usage (bits)	
	DOM	HPC2	O-PINI1	O-PINI2	DOM/HPC2	O-PINI1/O-PINI2
$d = 2$	27.9	31.1	31.6	33.2	2304	4608
$d = 4$	63.4	78.9	80.3	83.5	13 824	20 736
$d = 6$	107	142	145	150	34 560	46 080
$d = 8$	159	224	227	234	64 512	80 640

Next, we consider the example of an implementation of the PRESENT [BKL+07] block cipher, implemented with 18 parallel S-boxes (16 for the datapath and 2 for the key schedule), which maximizes the cost (in area and latency) differences between the implementations. The architecture of the implementation is the one of Cassiers et al. [CGLS20], where the multiplication gadget is changed (along with necessary control signals and randomness buses). The analyzed metrics are the area and the randomness requirements (Table 2), as well as the latency: 64 cycles for DOM, 96 cycles for HPC2/O-PINI1 and 160 cycles for O-PINI2.

We observe first that the HPC2 strategy has significant advantages in randomness and latency over OPINI1/2, hence it is best for cases where the security proof applies (such as an SPN implementation). Second, in cases the HPC2 cannot be used, OPINI1 and OPINI2 have similar performance, except for latency (OPINI2 is 67 % worse than OPINI1). This latency cost can however be amortized if the implementations manage to better fill the gadget's pipeline. Third, as already observed in [CGLS20], the HPC2 strategy has a 10–40 % increase in area and up to 50 % increase in latency over DOM. DOM can be seen as a lower bound for any ISW-based multiplication gadget, even though it has no composability guarantees, and thus the full implementation is not proven to be secure in the robust probing model.

Remark The O-PINI1 and O-PINI2 gadgets only work in $\mathbb{F}_2$ (otherwise they are not glitch-robust), since they are based on HPC2. It is possible to build a variant of these gadgets, replacing HPC2 by HPC1 in the constructions, while keeping the same security properties and proofs. These variants work in larger fields $\mathbb{F}_q$, and they have a lower gate count than O-PINI1/O-PINI2, while increasing slightly the randomness requirement.

7 Conclusion and open problems

We have shown that some state-of-the-art trivial composition strategies for hardware masking that are secure in the glitch-robust probing model (i.e. the ones based on the HPC1 and HPC2 multiplication gadgets [CGLS20]) are also secure against glitches and transitions combined when all adjacent executions of those multiplication gadgets are in parallel. In this common case, security against transitions (and their interactions with glitches) has zero cost in performance for the masked implementation. With such low overheads, a security proof in a practically-relevant model can only be an asset.

More generally, we observe that the implementation guidelines to prevent transition-based leakages that our formal analysis provides are in line with standard strategies to implement (masked) SPNs. In such ciphers, non-share-isolating gadgets (e.g. the S-boxes) are typically found in parallel inside a round. Furthermore, S-boxes are usually implemented as pipelines, meaning that the iterated transition-robustness is not an issue. As a result, if a full cycle is devoted to the permutation layer, or if the S-box has a latency of more than one cycle, S-box executions across rounds are not adjacent, meaning that the construction satisfies some of the requirements for our composition theorems. Formalizing this security guarantee for S-boxes that might not be PINI is an interesting open problem. Analyzing the possible vulnerabilities that may appear when S-box executions across rounds are adjacent would be an interesting direction as well.

A natural extension of our work would be to apply the proposed techniques to software implementations. While recent works on masked software implementations started to take into account the impact of glitches and/or transitions [BDM+20, BGG+20], they use rather high-level abstractions and the lower level modeling of physical defaults we propose could lead to improving both the soundness of these high-level analyses and the efficiency of masked designs with security against glitches and/or transitions. Leveraging an open source (e.g. RISC-V) MCU would be particularly relevant for this purpose. Whether some design ideas used in our hardware gadgets could lead to solutions for software masking is another interesting scope for further research.

Acknowledgements

Gaëtan Cassiers and François-Xavier Standaert are resp. Research Fellow and and Senior Associate Researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in part by the ERC project 724725. This work has been funded in part by the European Union (EU) and the Walloon Region through the FEDER project USERMedia (convention number 501907-379156).

References

[BBD+] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. *Journal of Cryptographic Engineering*, 2019:10.

[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 *ACM Conference on Computer and Communications Security*, 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 *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.

[BDM+20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In *EUROCRYPT (3)*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

[BGG+20] Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Orlt, Clara Paglialonga, and Lars Porth. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR Cryptol. ePrint Arch.*, 2020:603, 2020.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[CBG$^+$17] Thomas De Cnudde, Begül Bilgin, Benedikt Gierlichs, Ventzislav Nikov, Svetla Nikova, and Vincent Rijmen. Does coupling affect the security of masked implementations? In *COSADE*, volume 10348 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2017.

[CGLS20] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. pages 1–1, 2020.

[CGP$^+$12] 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 *COSADE*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

[CGZ20] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. In *EUROCRYPT (3)*, volume 12107 of *Lecture Notes in Computer Science*, pages 342–375. Springer, 2020.

[CPRR13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In *FSE*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2013.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Information Forensics and Security*, 15:2542–2555, 2020.

[DDF19] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. *J. Cryptol.*, 32(1):151–177, 2019.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *TISCCS*, page 3. ACM, 2016.

[HSS12] Annelie Heuser, Werner Schindler, and Marc Stöttinger. Revealing side-channel issues of complex circuits by enhanced leakage models. In *DATE*, pages 1179–1184. IEEE, 2012.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited or why proofs in the robust probing model are needed. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):256–292, 2019.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

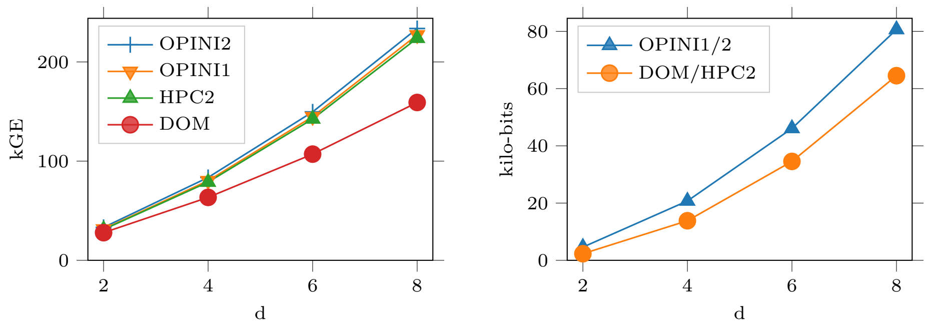
[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[MS06] Stefan Mangard and Kai Schramm. Pinpointing the side-channel leakage of masked AES hardware implementations. In *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2006.

[NRS11] 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.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *CRYPTO (1)*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

A Performance figure



(a) Area (in 65 nm technology) with 18 parallel S-boxes (16 for the datapath and 2 for the key schedule).

(b) Randomness usage.

Figure 7: PRESENT implementation with various AND masked gadgets.