Hardware Private Circuits: From Trivial Composition to Full Verification

(aka Repairing Glitch-Resistant Higher-Order Masking)

Gaëtan Cassiers[1], Benjamin Grégoire[2],
Itamar Levi[1,3], François-Xavier Standaert[1]

[1] Crypto Group, ICTEAM Institute, UCLouvain, Belgium
[2] Inria Sophia-Antipolis Méditerranée, France
[3] Department of Electrical Engineering, Bar-Ilan University, Israel

Abstract. The design of glitch-resistant higher-order masking schemes is an important challenge in cryptographic engineering. A recent work by Moos et al. (CHES 2019) showed that most published schemes (and all efficient ones) exhibit local or composability flaws at high security orders, leaving a critical gap in the literature on hardware masking. In this paper, we first extend the simulatability framework of Belaïd et al. (EUROCRYPT 2016) and prove that a compositional strategy that is correct without glitches remains valid with glitches. We then use this extended framework to prove the first masked gadgets that enable trivial composition with glitches at arbitrary orders. We show that the resulting "Hardware Private Circuits" approach the implementation efficiency of previous (flawed) schemes. We finally investigate how trivial composition can serve as a basis for a tool that allows verifying full masked hardware implementations (e.g., of complete block ciphers) at any security order. The tool checks that a synthesized HDL code fulfills the topological requirements of the composability theorems. As side products, we improve the randomness complexity of the best published refreshing gadgets, show that some S-box representations allow latency reductions and confirm practical claims based on implementation results.

1 Introduction

State-of-the-art. Over the last decade, the secure and efficient masking of block cipher implementations against side-channel attacks in hardware has been shown to be a very difficult task. One of the main issues it faces is that transient computations (usually denoted as glitches in the literature) can "re-combine" the shares of a masked hardware implementation, and therefore reduce its security order [26]. While some early attempts tried to mitigate glitches directly at the hardware level, it rapidly appeared that dealing with such low-level physical effects creates hard-to-fulfill engineering constraints [19]. This observation motivated the design of "Threshold Implementations" (TIs) [28], which solved the problem at the algorithmic level and in a principled manner. For this purpose, TIs add a "non-completeness" requirement to all the intermediate computations in a masked implementation (i.e., no combinatorial logic should manipulate all the shares), and store the results of these non-complete operations in registers, in order to block the propagation of the glitches.

The main scientific challenge in the design of TIs (and masked circuits in general) is to best trade the computation, memory, latency and randomness requirements that such countermeasures incur. As a result, and ever since the introduction of the TI concept, a wide literature has tried to minimize these quantities. These optimizations were directly successful for low (typically first) security order(s), as witnessed by [29,10,16]. By contrast, their generalization to higher security orders turned out to be tricky. For example, the first proposal of higher-order TI of Bilgin et al. [9] was rapidly shown insecure against multivariate attacks [30]. Various follow-up papers then proposed efficient and innovative ways to implement higher-order masking in hardware. We mention for example the Consolidated Masking Scheme (CMS) in [13], the Domain-Oriented Masking (DOM) in [23,24], the Unified Masking Approach (UMA) in [22] and the Generic Low Latency Masking (GLM) in [21]. Yet, and in contrast with the literature in software-oriented masking schemes (see [7,8] for example), none of these proposals came with a security proof in a formal model such as Ishai et al.'s probing model [25]. Furthermore, a recent work by Moos et al. showed that this lack of proof is not only a theoretical concern, and that the higher-order generalizations of all these schemes suffer from local or composability flaws [27].[1] These results are therefore revealing a gap in the hardware masking literature (i.e., the lack of efficient solutions for arbitrary-order secure and glitch-resistant implementations), and come as an advocacy for the design of new schemes with provable guarantees, for example in the hardware extension of the probing model (i.e., the robust probing model) put forward by Faust et al. [18].

In this respect, a partial solution to overcome this challenge is to use (what we denote as) direct verification tools that test implementations exhaustively, as initially proposed in the software context by Barthe et al. [4] and generalized to the hardware context with glitches in [11,2]. However, such tools are computationally limited to the analysis of small circuits and low security orders. So for analyzing full circuits at arbitrary security orders, one generally requires stronger properties from the masked gadgets in order to enable composition theorems that can then be exploited in (what we denote as) composition verification tools [5]. Unfortunately, when directly putting such composition requirements together with the glitch-resistance requirements, the cost and latency overheads over efficient schemes such as TIs, DOM, UMA and GLM become significant. To the best of our knowledge, the only solution in this direction is the proposal by Faust et al. to implement a "glitch-robust Strong Non-Interfering (SNI)" multiplication in two cycles [18]. The latter can then be made trivially composable by refreshing one of the two inputs in another two cycles (following the strategy

[1] By local flaws, we mean cases where a single masked gadget (e.g., a multiplication, S-box, ...) does not deliver its security guarantees. An example of local flaw is the attack against the scheme of Schramm and Paar [33] by Coron et al. [14]. Composability flaws happen when the combination of locally secure gadgets leads to additional weaknesses. Such a flaw is at the root of the attack against the scheme of Rivain and Prouff [31] exhibited by Coron et al. [15].

of Goudarzi and Rivain [20] proven by Cassiers and Standaert [12]), leading to a total of four cycles per multiplication, which limits its concrete relevance.[2]

Our contributions. The first main contribution of the paper is the formalization and a proof of the intuition from [18] that any (simulation-based) compositional strategy that is correct in the standard probing model remains valid in the robust probing model. Along the way, we additionally clarify a few subtleties such as the treatment of glitches that span multiple gadgets.

Following this formalization effort, the second contribution of the paper is to close the aforementioned gap in the hardware-oriented masking literature, and to propose efficient (low-latency) and glitch-resistant refreshing and multiplication gadgets that provably compose at arbitrary orders. We denote the resulting circuits as Hardware Private Circuits (HPC) since they can be viewed as a specialization of Ishai et al.'s private circuits to the hardware context [25].

Our first HPC multiplication gadget (HPC1) is a trade-off between the efficient (but non-composable) "DOM-indep" multiplication from [23,24] and the conservative proposal by Faust et al. Our second HPC multiplication gadget (HPC2) is based on the PINI1 gadget of [12]. It has lower randomness cost but is limited to the binary field $\mathbb{F}_2$ (whereas the HPC1 gadget works for any finite field). Both gadgets satisfy the hardware version of the Probe Isolating Non Interference (PINI) concept introduced with the trivial composition framework in [12]. We show that using the PINI framework allows latency gains compared to SNI-based multiplications. For the HPC1 gadget, we additionally design new and randomness-efficient (SNI) hardware refresh gadgets (needed for this multiplication) that achieve minimum latency by performing most of the computations "off-path" and limiting the "on-path" computation to a single XOR.

Our third main contribution tackles the risk that even abstract circuits built only from glitch-robust and trivially composable gadgets may not translate into concrete implementations with the corresponding security guarantees. Typical reasons include programming bugs and synthesis optimizations which may break the topological requirements that circuits must fulfill for our composition theorems to apply. We contribute to this problem by proposing a new tool for (what we denote as) full verification.[3] It takes as input the HDL code describing a masked implementation based on composable gadgets, and can verify its robust probing security after synthesis. This result answers a problem left open by Barthe et al. in [5] who described `maskComp`, an efficient composition verification tool applying to abstract algorithms based on composable gadgets, but not the resulting programs and their physical defaults. As a result, to the best of our knowledge for the first time, we can efficiently verify synthesized masked hardware implementations of complete block ciphers at any security order.

We additionally (1) improve the randomness complexity of the best reported refresh gadgets in the literature and demonstrate their relevance in a hardware

[2] By trivially composable, we mean that implementations mixing such multiplications with linear operations performed independently on each share are secure.

[3] Available under open-source license at `https://github.com/cassiersg/fullverif`.

implementation context; (2) show that some S-box representations allow further latency reductions and describe how to find such a representation for small S-boxes; (3) confirm our practical claims based on implementation results.

Paper structure. We start with some background about the circuit model we consider, (robust) probing security and composability notions (Section 2). We then analyze the solutions that are the basis of our investigations, namely the DOM scheme and Faust et al.'s implementation of the Ishai, Sahai and Wagner (ISW) multiplication (Section 3). We follow with a high-level presentation of our compositional strategy and new constructions (Section 4). Our verification tool is described in Section 5 and implementation results are in Section 6. We conclude with the formalization of our claims and security proofs (Section 7).

Cautionary note. Our new gadgets and full verification tool are based on a trivial compositional strategy. This appears as a natural first step in order to fix the flaws exhibited in [27]. Yet, trivial composability is not a necessary condition for (robust) probing security and we also prove that any (simulation-based) compositional strategy is eligible to design robust masking schemes. So as discussed in Section 2.3, finding out whether more efficient circuits could be obtained (possibly at higher verification cost) thanks to other compositional strategies is an interesting open problem. In this respect, we insist that our implementation results show that HPC gadgets already approach the level of performance of (flawed) DOM implementations which have low latency. So at least for ISW-like masking schemes and with respect to the latency metric, margins for improvements without significant area overheads (e.g., as in [21]) are limited.

2 Background

2.1 Circuit model

We use a circuit model based on the one of [25]: a circuit is a directed acyclic graph whose vertices are gates and edges are wires carrying elements from a finite field. Circuits may also have input and output connections. We focus on the binary field case which is most frequently used in practice. Nevertheless, all the mathematical results and the HPC1 schemes work for larger fields $\mathbb{F}_{2^n}$. We denote field additions by $\oplus$ and field multiplications by $\otimes$. We consider various kinds of gates for the circuit: combinational gates which implement field operations, random gates with fan-in 0 that produce a uniformly distributed random element, and register gates whose functionality is discussed next.

2.2 Standard probing model and security

The t-probing model [25] provides a framework to analyze at an abstract level the security of masked circuits and formalizes the notion of security order. In this model, an adversary can probe up to t wires of a circuit, and each probe gives access to the field element carried by the probed wire. A circuit is secure in the

t-probing model if the values obtained from the probes are (jointly) independent of any sensitive variable. Masking aims to achieve security in the t-probing model by splitting every sensitive variable into at least $t+1$ shares. We focus on $d = t+1$ additive masking: a sensitive variable x is represented by a $(t+1)$-sharing $(x_i)_i$ (where i is named the share index) such that $x = x_0 \oplus \cdots \oplus x_t$ and any set of t shares of x is uniformly distributed. A (d, m_i, m_o)-gadget is a circuit that takes as inputs m_i d-sharings and has m_o output d-sharings. We usually denote by $x_{i,j}$ (respectively $y_{i,j}$) the j-th share of the i-th input (respectively output) sharing of a gadget. The index $*$ represents the whole set of possible values (e.g., $x_{*,*}$ denotes all the shares of all the input sharings).

2.3 Composability

The composition of t-probing secure gadgets is not always t-probing secure [15]. Since testing probing security exhaustively is computationally hard as the security order and size of a circuit increase [4,11,2], various stronger security definitions have been proposed in order to enable the secure compositions of gadgets, paving the way to the security analysis of full circuits.

The definition of simulatability is a key ingredient for such a secure composition of gadgets: if any set of probes on a gadget can be simulated using some of its input shares, the security analysis of a composition of gadgets can be made by analyzing those sets of input probes only. A technique for using simulatability to prove the security of composite circuits (i.e., circuits made of the connection of multiple gadgets) that we call "probe propagation" has been introduced in [7]. Probe propagation works by analyzing gadgets in the composite circuit from the outputs to the inputs, and for each gadget replaces probes on its outputs and internal wires with simulator probes (i.e., the input wires needed for simulation) on its input wires. This technique ends with a set of input wires of the whole composite circuit. Knowing this set allows to simulate all the probes, hence if those inputs are independent of any secret, so are the probes.

Simulatability-based definitions can then be introduced by specifying properties of the input wires required for simulation, given a set of probes. In other words, they describe how probes propagate through a gadget. Let us consider a generic composite circuit structure where the connections between various gadgets are specified, as well as the number of inputs and outputs of each gadget and their functionality (i.e., the logic gate they implement), but not the gadgets themselves. 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. It guarantees that if the circuit is instantiated with gadgets that satisfy those definitions, then the circuit is probing secure. There exists a large variety of compositional strategies based on diverse simulatability-based definitions, such as NI, SNI, MIMO-SNI, PINI and f-NI [5,12,3].

We next introduce the NI definition which can be seen as the most basic simulatability property, the SNI definition and the PINI definition.

Definition 1 (t-Non-Interference [5]). *A gadget with one output sharing is t-Non-Interferent (t-NI) if any set of at most t probes on its wires can be simulated with t shares of each of its m_i input sharings.*

In other words, a probe inside or on the output of a NI gadget propagates into one probe on each of the input sharings of the gadget. Many gadgets are NI, such as the trivial implementation of linear operations (where the operation is implemented share-by-share). Using only NI gadgets is not sufficient to guarantee probing security. In order to propose a sound compositional strategy for any composite circuit structure, Barthe et al. introduced the stronger SNI notion.

Definition 2 (t-Strong Non-Interference [5]). *A gadget with one output sharing is t-Strong Non-Interferent (t-SNI) if any set of at most t_1 probes on its wires and t_2 probes on wires from its output sharings such that $t_1 + t_2 \leq t$ can be simulated with t_1 shares of each of its m_i input sharings.*

Informally, SNI guarantees independence between the inputs and outputs even in presence of a t-probing adversary which has access to the internals of the circuit: a probe inside a SNI gadget propagates into one probe on each of the input sharings and a probe on an output share of a SNI gadget does not propagate.

Finally, Probe-Isolating Non-Interference (PINI) has been recently introduced as an alternative basis for composing masked gadgets. Intuitively, PINI formalizes the splitting of a circuit into $t + 1$ shares. If there are no connections between the circuit shares (such as for the trivial implementation of linear operations), then each probe can propagate only into one circuit share, which implies that at least one circuit share remains "untouched" by probes and guarantees probing security. For gadgets which have connections between circuit shares (such as multiplications), the PINI definition imposes that they can be simulated as if these connections did not exist.

Definition 3 (Probe-Isolating Non-Interference [12]). *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-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.*

Based on these definitions, secure composition can follow two main approaches.

Trivial composition is to the strongest possible form of composition. It was initially proposed by Barthe et al. who proved that if all the gadgets in a circuit are SNI, then the circuit is probing secure [5]. The main drawback of this compositional strategy is that it comes with significant performance overheads. In particular, the trivial implementation of linear operations is only NI and must be "refreshed" to satisfy SNI. The Probe Isolating Non-Interference (PINI) framework introduced in [12] provides an alternative and more efficient path to trivial composition: a circuit is probing secure if all its gadgets are PINI and the trivial

implementation of masked linear operations is PINI. Two PINI multiplication gadgets are proposed in [12]: a SNI refresh followed by a SNI multiplication (the strategy of Gourdarzi and Rivain), and an ad-hoc gadget named PINI1.

Optimized composition is a complementary approach which aims at enabling more efficient masked implementations at the cost of more circuit-specific efforts in their design and evaluation. For example, one can combine NI and SNI gadgets so that the amount of refreshing in a protected implementation (hence its randomness complexity) is minimized. The (abstract) `maskComp` tool of Barthe et al. proposed at CCS 2016 is an example of such an approach, where all the multiplications are SNI and the number of SNI refreshes is optimized [5]. Another example can be found in [7], where it is shown that the AES S-box can be implemented securely with a mix of NI/SNI multiplications and SNI refreshes. The Tight Private Circuits (TPC) recently introduced by Belaïd et al. establish yet another set of composition rules with additional optimizations that specifically apply to block ciphers with surjective linear layers [8].

The following sections exploit the trivial composition approach which allows the simplest possible composition rules, and serves as an excellent basis for full verification. We use the PINI framework for this purpose because of the more efficient implementations it enables (compared to using only SNI gadgets) and because it is a natural formalization of the concept of domain (used in DOM) which is quite suitable to design masking schemes that prevent physical defaults such as glitches. Finding out whether an optimized compositional strategy can be exploited in the robust probing model and lead to different performance trade-offs is an interesting scope for further research. For example, generalizing the `maskComp`/TPC tools to this hardware context could lead to reduced randomness. By contrast, it is unlikely to reduce the circuits' latency with such strategies, since the glitch-robust SNI multiplications they rely on (which can be obtained by adding an output register to the DOM-indep multiplication, or by considering the solution of Faust et al. without input refreshing) require two cycles. For this latency budget (and even less thanks to our optimized S-box representations), our gadgets enable trivial composition without the additional refresh gadgets that `maskComp`/TPC aim to minimize (& sometimes cancel).

2.4 Circuits with glitches: the robust probing model

The standard probing model does not handle physical phenomena such as glitches that are encountered when implementing masked circuits in hardware. Since the present work aims at designing gadgets that compose in the presence of glitches, we next describe how to capture them thanks to the robust probing model introduced by Faust et al. [18] as a way to formalize those non-idealities.

In the "glitch-robust" probing model, a probe on a wire does not only give access to the value carried by the wire, but also recursively to all the input wires of the combinational gate that generates the value on that wire. For simplicity, we sometimes name a probe in the glitch-robust probing model an extended probe, and a probe in the standard probing model a standard probe.

In order to limit the power of extended probes, a new kind of gate is introduced: the registers. Registers are sequential gates with one input wire and one output wire that implement the identity function. Formally, for a given execution of a circuit C and an extended probe on a wire p, the adversary has access to the tuple of values $C_{rob,p}$. Let $C_{std,p}$ be the value carried by the wire p (i.e., the value associated to the probe in the standard probing model). If the gate that generates the value on p is a combinational gate, then $C_{rob,p}$ is defined as $\{C_{std,p}\} \cup C_{rob,w_1} \cup \cdots \cup C_{rob,w_n}$ where $w_1, \ldots, w_n$ are the input wires of the combinational gate. If the gate is a register, then $C_{rob,p} = \{C_{std,p}\}$. That is, a register stops the glitches and probes on registers cannot be extended.

A register has however two adverse impacts: it uses additional hardware resources (i.e., silicon area, power) and it acts as a synchronization barrier, increasing the latency of the computation (by one clock cycle).

Following, the definition of probing security in the robust probing model is a direct adaptation of the standard probing security: a circuit is glitch-robust t-probing secure if the values obtained from any set of t extended probes are jointly independent of any sensitive variable.

Remark. In some way, a glitch-extended probe propagates to inputs of its combinational circuit. It should be noted that this propagation is distinct from the simulation-based propagation of probes: in the first case the propagation is due to physical glitches and is stopped by registers while in the second case the propagation is determined by statistical properties of the computed values.

3 Analysis of the state-of-the-art: DOM & Faust et al.

As mentioned in introduction, our following results correspond to a trade-off between the efficient DOM (precisely, DOM-indep) multiplication [23,24] and a conservative proposal by Faust et al. [18]. Both of them can be viewed as solutions to implement the ISW multiplication in hardware. In this section, we discuss these two state-of-the-art solutions and their limitations.

We start with the DOM implementation of the ISW multiplication described in Algorithm 1. If it was implemented using only combinational logic and no registers, then glitch-extended probes would completely break its security: an extended probe on c_i indeed contains all the shares of b. The idea of DOM, which is formalized by the PINI definition, is to isolate domains (that we also name circuit shares) by ensuring that any wire crossing domains (i.e., corresponding to terms $a_i \otimes b_j \oplus r_{ij}$) goes through a register.[4] As a result, extended probes on the outputs contain only terms $a_i \otimes b_j \oplus r_{ij}$, which are independent from the input shares if r_{ij} is not known by the adversary.

Formally, the DOM gadget is glitch-robust NI (i.e., the definition of NI where probes are glitch-extended). Intuitively, this can be seen as follows: the most powerful extended probes are $a_i \otimes b_j \oplus r_{ij}$ and c_i (other extended probes are essentially subsets of those probes). By building adequately the required inputs

[4] Registers for the $a_i \otimes b_i$'s are usually added for synchronization.

Algorithm 1 DOM implementation of the ISW multiplication (in one cycle).

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 $\bigoplus_i c_i = a \otimes b$.

```
for i = 0 to d − 1 do
  for j = i + 1 to d − 1 do
    r_ij ←$ F_{2^n};
    u_ij ← Reg [a_i ⊗ b_j ⊕ r_ij];
    u_ji ← Reg [a_j ⊗ b_i ⊕ r_ij];
  end for
end for
for i = 0 to d − 1 do
  c_i ← Reg [a_i ⊗ b_i] ⊕ ⊕_{j=0,j≠i}^{d−1} u_ij;
end for
```

sets, the simulator is able, for each pair (i, j), to either know a_i and b_j, or to assert that there is no probe on u_{ij}, u_{ji} and at most one probe on c_i or c_j. Therefore, the simulator is able to simulate either by knowing the inputs, or by knowing that the adversary sees u_{ij} from the extension of c_i (or u_{ji} from the extension of c_j) as a fresh random. This (actually a stronger variant) is proven in Proposition 1. The DOM multiplication is not SNI with glitches (since an extended probe on an output cannot be simulated without input shares) and using a NI multiplication is not enough for secure composition. A simple example (although admittedly contrived) is the evaluation of $x \otimes x$: a probe on an internal variable of the multiplication may require two shares of x to be simulated, which breaks the probing security. (More practical examples are discussed in [27].)[5]

In order to solve this issue, Faust et al. start by adding one register layer between the computations of c_i and the outputs. This corresponds to adding a $\mathsf{Reg}[\cdot]$ around the computations of the last line of Algorithm 1. It makes the gadget glitch-robust SNI in two cycles, since the output probes are then stored in (stable) registers and cannot anymore be extended. Hence, they are as powerful as in the standard probing model and the proof that the ISW algorithm is SNI applies (modulo modifications for internal extended probes).

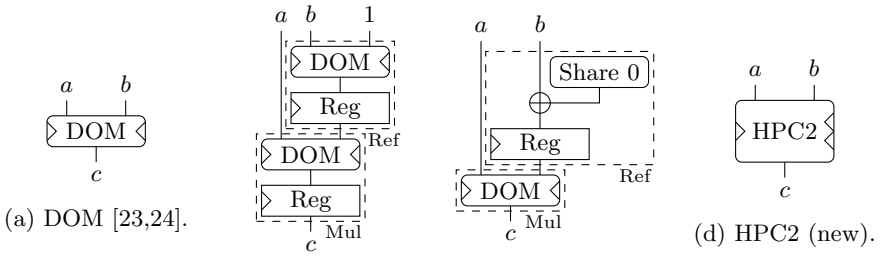
To make their multiplication trivially composable, Faust et al. then additionally exploit the "double-SNI" strategy initially proposed by Goudarzi and Rivain [20]: it consists in the use of a SNI multiplication gadget of which one input is systematically refreshed with one SNI refresh gadget. The refresh gadget is simply the SNI multiplication with the constant 1 as second input.

Yet, although Faust et al. prove that their multiplication (i.e., the DOM multiplication with an additional output register) is glitch-robust SNI, they do not prove that the composition strategy of Goudarzi and Rivain remains secure in the glitch-robust t-probing model when glitch-robust SNI gadgets are used. Furthermore, their solution is quite expensive: the SNI multiplication has a latency

[5] A "DOM-dep" multiplication was proposed in [23], which would be needed to compose securely, but was broken in [27] with no obvious fix.

of two cycles, which means that the double-SNI gadget has a latency of four cycles w.r.t. one of the inputs and two cycles w.r.t. the other one. The refresh gadget is also expensive in randomness compared to the state-of-the-art SNI refreshes in the standard probing model [7,6,3].

The one-cycle DOM glitch-robust NI multiplication and (2+2)-cycles Faust et al. "double-SNI" multiplication are illustrated in Figures 1a and 1b.



(a) DOM [23,24].

(b) Faust et al. [18].

(c) HPC1 (new).

(d) HPC2 (new).

Fig. 1: Hardware-oriented multiplications gadgets. Left and right triangles mark sequential logic; they indicate the output latency with respect to the left and right input, respectively. Rounded corners indicate combinational logic.

4 Efficient, glitch-resistant and composable gadgets

We now introduce our strategy for trivial composition in presence of glitches, together with efficient gadgets that can be used in this strategy. We focus on intuitive explanations, postponing formal definitions and proofs to Section 7. First of all, we propose in Section 4.1 a general technique for applying some standard probing model compositional strategies to the glitch-robust probing model. Those strategies are the ones based on simulatability (e.g., using NI, SNI, PINI). We do this by defining glitch-robust simulatability, by which definitions of glitch-robust NI, SNI, PINI, ... follow immediately. We show that they enjoy the same composition properties as their standard probing model counterparts. Second, we introduce in Sections 4.2 and 4.4 two glitch-robust PINI multiplication gadgets for any masking order. The first one (HPC1) is based on the refresh-then-multiply technique and is generic: it works for any field $\mathbb{F}_q$. The second one (HPC2) is more randomness-efficient, but works only in $\mathbb{F}_2$. We also present in Section 4.3 new constructions for glitch-robust SNI refresh gadgets that are used in HPC1. Finally, we explore how the latency characteristics of HPC multiplication gadgets can be taken into account in logic circuit optimizations, in order to further reduce the overall latency.

4.1 Composability with glitches

We first analyze composition in presence of glitches. For that purpose, we define the concept of glitch-robust simulatability. As mentioned previously, the definitions of glitch-robust NI, SNI and PINI can be adapted directly from their standard probing model counterparts, by replacing the term "probe" (resp., "simulated") by "glitch-extended probe" (resp., "glitch-robustly simulated").

Based on these definitions, it seems natural to assume that simulation-based proofs apply just as for their standard counterparts (i.e., without glitches), which was implicitly assumed by Faust et al. [18]. We next formalize this expectation in Theorem 1 and show that despite essentially correct, some subtleties have to be considered, such as the treatment of glitches that span multiple gadgets.

Definition 4 (Glitch-robust simulatability). *A set of extended adversarial probes P in a gadget G can be glitch-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 $G_{rob,P}(x_{*,*})$ and $S(x_{i_1,j_1}, \ldots, x_{i_k,j_k})$ are equal for any value of the inputs $x_{*,*}$ when there are no glitches on the inputs.*

Glitch-robust simulatability is illustrated in Figure 2. The first circuit shows that if a probe is at the output of a register, the glitches do not extend the probe. However, the inputs of the circuit are still needed to simulate the circuit and the probes still propagate, as it is the case in the standard probing model. In the second circuit, we see that extended probes are more powerful than non-extended ones: in the standard probing model, no input would be needed to simulate the probe (the probe is $\$ + x + y$, hence independent of both x and y), however due to glitches, the input belongs to the extended probe y. The register prevents glitch propagation, hence the input x is not needed for simulation. For the third circuit, both inputs x and y are needed (both in the standard and glitch-robust probing models): the probes $\$$ and $\$ + x + y$ depend on x and y.

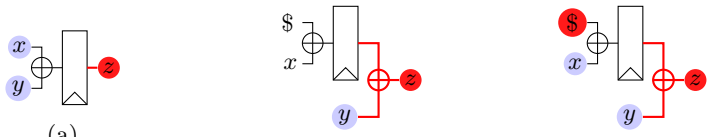


Fig. 2: Glitch-robust simulatability examples. Red letters are probed variables, red wires are glitches (probe extensions), blue variable are inputs needed for simulation and the $ sign denotes a random gate.

This definition of glitch-robust simulatability is stronger than standard simulatability: probes are more powerful and inputs given to the simulator are the same. It however relies on the hypothesis that there are no glitches on the inputs, which simplifies analysis of individual gadgets but is unrealistic when the

gadgets are integrated in a larger circuit. This actually not an issue: if glitches on an input share affect the probes, then the input itself affects the probes and thus the glitch-robust simulator can be extended to deal with glitches on the inputs. This idea is formalized in Lemma 1, Section 7.1. The lemma actually shows that including glitches on inputs would lead to an equivalent definition.

Next, we (informally) introduce a generic composition theorem showing that any simulation-based compositional strategy in the standard probing model also applies to the glitch-robust probing model.

Theorem 1 (Glitch-robust composability (informal)). *Let G be a composition of gadgets where any set of probes in a gadget is glitch-robust simulatable. Restricting a glitch-robust simulator to a standard probing model simulator by discarding some of its inputs, any set of probes in G that can be simulated in the standard probing model using some of the input shares of G can be simulated in the glitch-robust probing model using the same input shares.*

This theorem is a consequence of Lemma 1 and of the way simulation-based composition works. We formalize it (along with the notion of simulation-based compositional strategy) and prove it in Section 7.1.

As a result, the idea of share isolation from the PINI definition is still valid in face of extended probes due to glitches. Therefore, the main properties of PINI are also satisfied by glitch-robust PINI:

- Affine gadgets implemented in the trivial way with $t + 1$ shares are glitch-robust t-PINI since the propagation and extension of probes are limited to one share (thanks to actual share isolation in the gadget).
- The glitch-robust composability theorem (Theorem 1) implies that the composition of glitch-robust t-PINI gadgets is glitch-robust t-PINI.
- Finally, a glitch-robust t-PINI gadget with at least $t+1$ shares is glitch-robust t-probing secure since t extended probes can be simulated with t shares of each input sharing, which are independent of any sensitive value.

These observations lead to the *Hardware Private Circuits (HPC)* trivial composition strategy for glitch-robust masking: using trivial implementations for affine gadgets, along with glitch-robust PINI multiplication gadgets.

A first instance of the HPC strategy was proposed by Faust et al.: the Goudarzi and Rivain refresh-then-multiply gadget is proven to be PINI in [12] using a simulation-based proof. Therefore, Theorem 1 applies and the gadget made of glitch-robust SNI refresh and multiplication is glitch-robust PINI.

4.2 Generic Hardware Private Circuits (HPC1)

We introduce a first efficient glitch-robust PINI multiplication gadget, next denoted as Hardware Private Circuits 1 (HPC1), and prove it secure at all orders and for any field $\mathbb{F}_q$. It is based on the refresh-then-multiply technique and is represented in Figure 1c which highlights the similarities and differences with the DOM multiplication (to which we add a refresh gadget) and the Faust et al.

multiplication (to which we remove an output register and optimize the input refresh gadget). The figure also shows the latency of the gadget: once cycle with respect to one input, and two cycles with respect to the other input.

The main observation here is that while not SNI, the DOM multiplication actually enjoys a property that is stronger than NI, which we use in the HPC1 scheme (we find the same property when removing the register layer on the output of the multiplication of Faust et al.). We next introduce this property and defer its formalization and the full proof of HPC1 to Section 7.2.

We already discussed that each extended probe in the DOM multiplication can be simulated using either a_i and b_i (which satisfies the glitch-robust PINI definition) or a_i and b_j (which does not satisfy PINI), showing that it is glitch-robust NI. The additional property of DOM is that the output probes c_i depend on only the share a_i of a (and some b_j). Therefore, adding a glitch-robust SNI refresh on the input b of the DOM gadget makes it glitch-robust PINI, since it stops the propagation of the probe b_j to the input, and the required inputs a_i satisfy the PINI definition. Interestingly, this result suggests that the probe isolation framework is well suited to enable composition with glitches at limited latency budget. For example, such guarantees would not be possible using the SNI abstraction (since the DOM multiplication is not glitch-robust SNI).

HPC1 also enjoys optimized glitch-robust SNI refreshes with one cycle of latency and reduced randomness compared to Faust et al., as detailed next.

4.3 New refresh gadgets

In this section, we improve glitch-robust SNI refresh gadgets compared to the state-of-the-art (the SNI multiplication by 1 of Faust et al.). We describe several SNI refresh gadgets covering realistic orders d, have best-known randomness complexity (in small fields), are glitch-robust and have a latency of one cycle.

First, we take the order-generic and randomness efficient ($\mathcal{O}(d \log d)$) SNI refresh of Battistello et al. [6] and adapt it to the glitch-robust setting by adding a register after each XOR gate. This construction and its security proof are detailed in Section 7.3. The main drawback of this construction is its high latency of $2 \log_2(d) - 1$ clock cycles. Next, we solve this latency issue by introducing a generic technique to reduce the latency of any glitch-robust SNI refresh gadget to one cycle. It works by using the original gadget while providing it with an all-zero input sharing, then XORing its output with the sharing to be refreshed and adding a register layer after that. Intuitively, the original refresh gadget outputs a "glitch-robust SNI-secure" sharing of zero (we formalize this notion in Section 7.4), which can then be simply XORed with the sharing that must be refreshed. The output register layer prevents output probes from being extended to the inputs of the refresh. Our construction is illustrated in Figure 1c, in the "Ref" frame (where "Share 0" is a SNI refresh provided with an all-zero input). In this way, the latency of the original gadget does not matter: since it is not on the main datapath, it can be computed in advance. The cost of this transformation is at most d registers, since the XOR gates added compensate those that can be removed due to XORing with the all-zero sharing.

Finally, to further reduce randomness utilization, we provide a set of new optimized glitch-robust SNI refresh gadgets for low (and arguably all the practically-relevant) orders (i.e., $d \in \{2, \ldots, 16\}$, see Figure 3), which require less randomness. These gadgets were proven robust-SNI with the maskVerif tool [2]. The randomness complexity of the new refresh gadgets compares favorably to the state-of-the-art for both hardware and software implementations (see Appendix A, Table 2). Randomness gains of more than 30% are obtained for $d \in \{3, 4, 5, 7, 8\}$.

These new refresh gadgets are adaptations of the gadgets in [3], modified by first adding registers where needed, and then optimizing their randomness complexity by relaxing the parallel constraint (at the cost of making some software bitslice implementation strategies impossible). Optimization was performed by hand, iteratively solving flaws found by maskVerif.

$$y_0 \leftarrow \mathsf{Reg}\,[x_0 + r_0]$$
$$y_1 \leftarrow \mathsf{Reg}\,[x_1 + r_0]$$

(a) $d = 2$

$$t_0 \leftarrow \mathsf{Reg}\,[r_0 + r_1]$$
$$y_0 \leftarrow \mathsf{Reg}\,[x_0 + r_0]$$
$$y_1 \leftarrow \mathsf{Reg}\,[x_1 + r_1]$$
$$y_2 \leftarrow \mathsf{Reg}\,[x_2 + t_0]$$

(b) $d = 3$

$$\mathbf{t^0} \leftarrow \mathsf{Reg}\,[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$$
$$\mathbf{y} \leftarrow \mathsf{Reg}\,[\mathbf{x} + \mathbf{t^0}]$$

(c) $d = 4, 5$

$$\mathbf{t_0} \leftarrow \mathsf{Reg}\,[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$$
$$\mathbf{t_1} \leftarrow \mathsf{Reg}\,[\mathbf{s^1} + (\mathbf{s^1} \gg 3)]$$
$$\mathbf{t_2} \leftarrow \mathsf{Reg}\,[\mathbf{t^0} + \mathbf{t^1}]$$
$$\mathbf{y} \leftarrow \mathsf{Reg}\,[\mathbf{x} + \mathbf{t^2}]$$

(d) $d = 13, \ldots, 16$

Fig. 3: Optimized refresh gadgets for some d. (The full set of gadgets $d = 2, \ldots, 16$ is shown in Appendix A and is available at `https://github.com/cassiersg/opt-refresh`.) The input sharing is denoted as $\mathbf{x}$ and the output sharing as $\mathbf{y}$. All r_i variables are independent uniformly random elements, and $\mathbf{s}^i$ are vectors of d independent random elements. The $(\cdot \gg i)$ operator applied to a vector denotes a rotation of its elements: the first element becomes the $i+1$-th, etc.

4.4 Randomness-optimized AND gadget (HPC2)

In this section, we present a multiplication gadget (Algorithm 2) for the practically-relevant field $\mathbb{F}_2$ that has the same randomness cost as the DOM gadget. This gadget is based on the PINI1 multiplication of [12], and adapted to the hardware context by adding registers where needed to prevent glitches.

We explain briefly the main argument of the proof that this gadget is glitch-robust PINI and defer the full proof to Section 7.5.

First, lets us recall the "masked shares multiplication" trick of [12]: whereas ISW-based multiplication schemes such as DOM compute terms $a_i \otimes b_j \oplus r_{ij}$, the HPC2 gadget computes $\bar{a_i} \otimes r_{ij} \oplus a_i(r_{ij} \oplus b_j)$ ($\bar{\cdot}$ denotes the NOT gate). In the standard probing model, this trick ensures that any single probe does not depend jointly on a_i and b_j, enabling the gadget to be PINI. If there is more than one probe, both a_i and b_j are known to the simulator. In presence of glitches, registers are added as follows: $\mathsf{Reg}\,[\bar{a_i} \otimes r_{ij}] \oplus \mathsf{Reg}\,[a_i \otimes \mathsf{Reg}\,[b_j \oplus r_{ij}]]$. None of the glitch-extended probes in this computation depends on both a_i and b_j:

- extended probes on $\bar{a_i} \otimes r_{ij}$ and $b_j \oplus r_{ij}$ do not contain b_j and a_i, respectively;

Algorithm 2 Glitch-robust HPC2 multiplication for $\mathbb{F}_2$.

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 $\bigoplus_i c_i = a \otimes b$.

```
for i = 0 to d − 1 do
  for j = i + 1 to d − 1 do
    r_ij ←$ F_2;
    r_ji ← r_ij;
  end for
end for
for i = 0 to d − 1 do
  for j = 0 to d − 1, j ≠ i do
    u_ij ← ā_i ⊗ Reg[r_ij];
    v_ij ← b_j ⊕ r_ij;
  end for
end for
for i = 0 to d − 1 do
  c_i ← Reg[a_i ⊗ Reg[b_i]] ⊕ ⨁_{j=0,j≠i}^{d−1} (Reg[u_ij] ⊕ Reg[a_i ⊗ Reg[v_ij]]);
end for
```

- for the extended probe on $a_i \otimes \mathsf{Reg}\,[b_j \oplus r_{ij}]$, we observe a_i and $b_j \oplus r_{ij}$ does not depend on b_j, since it is masked with a fresh random r_{ij} (remember that we assume there is a single probe);
- for the extended probe on $\mathsf{Reg}\,[\bar{a_i} \otimes r_{ij}] \oplus \mathsf{Reg}\,[a_i \otimes \mathsf{Reg}\,[b_j \oplus r_{ij}]]$, we observe $\bar{a_i} \otimes r_{ij}$ and $a_i \otimes (b_j \oplus r_{ij})$. If $a_i = 0$, then those observations are r_{ij} and 0, while if $a_i = 1$, the observations are 0 and $b_j \oplus r_{ij}$ (b_j is perfectly masked). In both cases, observations are independent of b_j.

Note that this last probe is the reason why the HPC2 gadget is restricted to $\mathbb{F}_2$: in larger fields, $\bar{a_i}$ would be replaced by $a_i \oplus 1$ for correctness, but it would not be true anymore that one of a_i or $a_i \oplus 1$ is zero.

4.5 S-box optimizations

One quite peculiar feature of HPCs is the latency asymmetry that is caused by the fact that only one of their inputs must be refreshed. Take for example a simple (tree-based) implementation of the function $\mathsf{f}(a, b, c, d) = (a \otimes b) \otimes (c \otimes d)$: it will have a latency of four cycles, and three registers will be needed to synchronize the non-refreshed inputs. In this respect, an interesting observation is that if we can find a logic representation of a function to mask (e.g., an S-box) such that one input of each AND gate in the second (or later) stage of the circuit is a linear combination of inputs in an earlier stage, then we can reduce the latency by one cycle. For deep circuits, such an optimization therefore has the potential to reduce the latency by a factor two.

Concretely, we modified a baseline tool from Ko Stoffelen [35] in order to construct circuits that fulfill these conditions. Searching over circuit representations (for a given function) is a complex task. We used SAT (satisfiability)

solvers, which work well for small S-boxes. The advance that we give (on top of the existing tool [35] which generates a circuit representation for a SAT solver) is to jointly encode the SAT solver problem to minimize the number of AND gates for a given AND depth target, while also constraining the logical equations so that one input of each AND gate is always assigned a linear combination of the main inputs and any signal generated in previous AND stages.

We then applied this approach to a set of 4-bit S-boxes to demonstrate its generality and efficiency, using the same SAT solver CryptoMiniSat-5 [34] as used in [35].[6] Precisely, we considered the PRESENT, PRINCE, Rectangle, Class13, Skinny, involutive Class-13 and Prost S-boxes. The resulting circuit with such an asymmetric structure is given in Figure 4 for PRESENT – the S-box circuits are given in Appendix B for completeness. They all cost 4 AND gates (except for PRINCE which costs 6 AND gates) and have AND depth 2, while the XOR gate count varies between 13 and 24. Note that we did not minimize XOR counts and therefore we do not claim any optimality in that respect. This is because for masked implementations, the XOR's associated area/latency penalty becomes negligible compared to the AND's-cost as d increases.

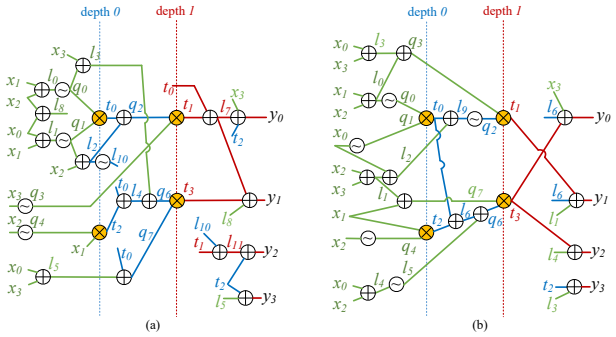


Fig. 4: PRESENT S-box circuit with AND depth 2 and 4 AND gates. SAT solution without optimization (a) and with asymmetry optimization (b).

5 Full verification tool

Despite the previous masking schemes being relatively simple and their composability guarantees being strong, implementing them still requires a skilled hardware designer. Besides implementing the correct functionality, which can be tested through standard techniques such as test vectors, all the assumptions of the underlying security proofs must also be fulfilled to ensure security.

In this respect, while it appears that masking composition proofs are only concerned with high-level assumptions, such as the kind of gadgets and the structure of the circuit, they in fact make other assumptions (implicit or not)

[6] `https://www.msoos.org/cryptominisat5`.

which can be falsified by hardware implementations, have no impact on the functionality of the implementation, and are thus hard to verify by classical testing. Examples of such assumptions include:

- each gadget is used with fresh, independent randomness;
- no more computation on shares is performed than specified by the algorithms (e.g., in parallel with or after useful computations are finished);
- the order of the shares in a sharing should not be shuffled.

We next describe a tool that allows verifying all these assumptions.[7] At a high level, it takes a Verilog implementation as an input and outputs another, pre-synthesized implementation (that is equivalent to the input implementation), along with the result of the verification (error or success). It works by following the steps illustrated in Figure 5 that we detail next.

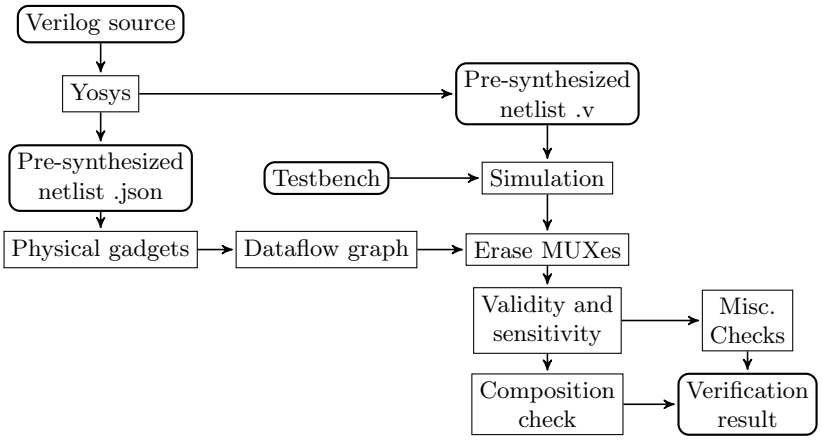


Fig. 5: Process of the verification tool.

First, the open-source synthesizer Yosys [36] is used to produce a netlist of all the gadgets, while preserving the hierarchy of the gadgets. Second, this netlist is simulated using a user-provided testbench (using IcarusVerilog). Third, the netlist is analyzed to build a graph of physical gadgets, which is then unrolled over all execution cycles, leading to a dataflow graph. This graph is close to the gadget composition graphs that we analyze in composition proofs, but with two major differences: it contains so-called "MUX gadgets" (MUXing two sharings according to a non-sensitive control signal), and it contains gadgets for which the inputs are invalid (i.e., do not carry a meaningful value). The next stage is

[7] A library of elementary gadgets (XOR, NOT, AND HPC1 and HPC2, MUX, SNI refresh...) is provided with the tool. Those are annotated with `keep` and `preserve` attributes where needed in order to prevent security-damaging optimizations.

to remove the MUX gadgets from the graph, which is simple: for each cycle, the control signal of the MUX is known (this is the single step where the result of the simulation is used), thus the MUX can be replaced by wires from the inputs to the outputs. Then, we annotate each sharing with validity ("Does it contain a meaningful value ?") and sensitivity ("Does it depend on the input sharings ?") information. Using these annotations, various additional checks are performed. Finally, the composition strategy is verified on the dataflow graph, of which we remove all the gadgets for which no input is sensitive. In the case of trivial composition, it simply checks that all the gadgets satisfy the security property (e.g., PINI). This may involve recursively checking a gadget if it is a composition of sub-gadgets, verifying that it is physically isolating shares (e.g., for linear operation gadgets) or just assuming that it is correct based on annotations. This last case happens for the multiplication gadgets. Automatically verifying them using a tool such as maskVerif is left to future work.

Note that only the choice of which property to check for each sub-gadget (i.e., the "Composition check" box in the figure) is specific to our trivial composition strategy. Therefore, the tool would require only minor modifications to be able to check other composition strategies (such as optimized SNI-based ones).

During the processing stages, many security checks are performed. We list the main ones along with the stage where they are performed.

Physical gadgets:
- Any input sharing of a gadget must be connected to one and only one source gadget of which it is an output sharing;
- Each wire belonging to a sharing is only used as part of that sharing and not elsewhere;
- All the gadgets are connected to the same clock signal, otherwise cycle-based analysis of dataflow over time cannot be done properly. Handling more complex clock circuits (although still all synchronous, such as divided clocks) is left to future work.

Dataflow graph:
- No combinational loop exists in the circuit;

Misc. checks:
- All outputs of the composite gadget (at the specified cycle) should be connected to valid sharings;
- Each random input of a gadget is connected to a wire carrying randomness;
- Each random input of a gadget having a sensitive input should be connected to a fresh random bit. For this check, a dataflow graph of the sub-circuit handling randomness (i.e., wires, registers, MUXes) is built;
- At the cycle after all the outputs have been produced, there should be no sensitive sharing remaining in the gadget (otherwise non-verified computations might happen, since we stop analysis at that cycle).

Related works. Other tools have been proposed to verify different properties of masked circuits. We next provide a brief account of the state-of-the-art to situate our contribution. The main tools to which our proposal compares are listed in

Table 1. Such tools can verify abstract implementations or concrete ones (i.e., actual code, including physical defaults); they can also aim at direct verification (which is limited to small circuits and security orders) or composition-based verification. The fullVerif tool we propose is the first one that can verify the composability of concrete hardware implementations including glitches.

	Abstract	Concrete
Direct	Barthe et al. [4]	REBECCA [11] maskVerif [2]
Composition-based	maskComp [5] Tight Private Circuits [8]	fullVerif (*new*)

Table 1: Masking formal verification tools' overview.

Some other works are less directly comparable, either because they span multiple table cells or because they aim at different goals. For example, the work of Eldib et al. in [17] aims at similar goals as the one of Barthe et al. from Eurocrypt 2015 [4], but it is more concrete (i.e., it applies to concrete C implementations) while still ignoring physical defaults. The work of Arribas et al. rather considers the verification of more specific properties that are required for TIs (namely, non-completeness and uniformity) [1].

We finally mention that the impact of extended probes in the randomness distribution circuit remains excluded from all these tools and, to the best of our knowledge, has never been analyzed. We leave it as an interesting scope for further investigations, together with the general challenge of better understanding the randomness' requirements in masked implementations.

6 Implementation results

In this section, we validate the claimed efficiency of HPCs. This is done for a complete (although non-optimized) encryption architecture test-case which allows realistic estimations of actual area constraints and randomness resources required. We selected the (128-bit version of the) PRESENT block-cipher for this purpose, as it is one of the most popular lightweight ciphers and has been shown to enable efficient masked implementations [29]. Its 4-bit S-box is also well suited to our optimizations of Section 4.5. The obtained results should be representative of other similar (lightweight) ciphers. The section starts by describing our generic architecture, follows with some design considerations, and finally exhibits the good performances that our approach allows.

We evaluate four types of masked implementations, all based on trivial composition.[8] The strategies therefore differ only by their AND gadget:

[8] The HPC1 and HPC2 implementations are available at `https://github.com/cassiersg/present_hpc`.

– The gadget of **Faust et al.**: [18] (where we eliminate some logic in the refresh by propagating the constant sharing $(1, 0, \ldots, 0)$ where it is safe to do so). It uses 4 cycles per AND gate and is based on the standard description of the PRESENT S-box with AND depth 2 (so takes 8 cycles per S-box).
– The **HPC1** gadget with optimized refresh gadget (Figure 1c) and the **HPC2** gadget (Algorithm 2), with the S-box architecture optimized for latency from Section 4.5. These implementations require 3 cycles for the full S-box.
– The **DOM**-indep glitch-robust NI gadget, with the standard description of the PRESENT S-box (our optimization would not reduce the latency in this case), leading to 2 clock cycle for the S-box. While this last design does not come with composability guarantees, we use it as a lower bound for the cost of our masked implementations.

In all cases, the S-boxes are fully pipelined: it requires only one more clock cycle for each additional evaluation (if there is no data dependency across evaluations). This list already highlights one of the concrete achievements of this work. Namely, compared to the Faust et al design, the latency is reduced from 8 to 3 cycles, and compared to the DOM design, only a small overhead is observed.

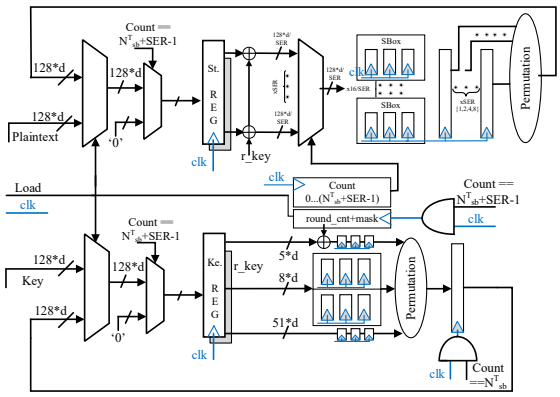
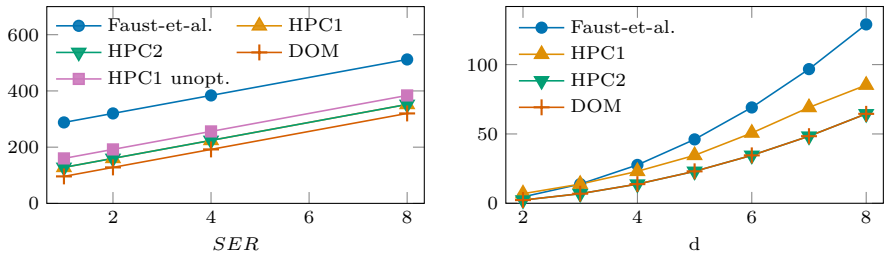


Fig. 6: Architecture of the PRESENT encryption core with serialization (SER) of parallel operations to reduce area cost.

Architecture. We refer to Figure 6 for a detailed illustration of our hardware architecture. With the aim of providing a scalable example, we use the number of shares as main parameter. Besides, in order to allow understanding the natural space/speed trade-off in hardware implementations, we use a SER (serialization) factor as a secondary design parameter. Basically, we serialize the computation within an encryption round by this factor and consequently reduce the area by the same factor (e.g., SER=1 is a fully parallel design, SER=2 is a serialization of 2 blocks of 8 S-boxes each, SER=4 is a serialization of 4 blocks, ...).

Our architecture is geared towards simplicity, genericity and reproducibility rather than minimal area/power. This is not an issue since the cost of the architecture is constant for all the implementations we compare. It is built around a $128d$-bit state register, which feeds the serialized S-boxes. The outputs of the S-boxes are stored in a shift register whose output is connected to the state register through the bit-permutation layer. The key schedule unit uses two more S-boxes and is similar to the main datapath (but does not require a shift register).



(a) Total latency (clock cycles). (b) Randomness cost (kilo-bits).

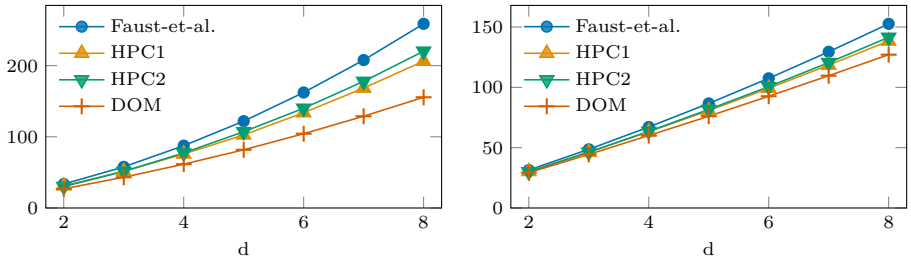
Fig. 7: Randomness and latency cost comparisons for a PRESENT core.

Comparison. Starting the comparison with one main objective of the work, Figure 7a shows the cycle count in function of the serialization factor SER for our architecture (which is independent of the number of shares d). With SER=1 the HPC1 and HPC2 designs have a 60% latency reduction compared to Faust et al. and a 25% latency increase compared to DOM. As SER increase, the S-box (internal) pipeline starts to be filled during encryption rounds, leading to a total latency increase, hence reduced factors of gain. We also show the HPC1 gadget (or equivalently HPC2, since they have the same latency characteristics) without our S-box architecture optimization ("HPC1 unopt."), which confirms that the latency gain over Faust et al. is primarily due the to improved gadgets, but that the optimization helps further approaching the efficiency of DOM.[9]

Moving on to discuss the cost associated with randomness generation, Figure 7b shows the total randomness cost (refresh and multiplication) for the entire PRESENT core per encryption as a function of d. The cost for DOM and HPC2 is half of the cost of Faust et al., while HPC1 is in-between (thanks to the improved refresh gadgets). A designer can immediately deduce the total randomness requirements from his (hers) TRNG/PRNG.

Further investigating the area utilization of the proposed designs, Figure 8 shows the gate equivalent (GE) count as a function of d for two exemplary serialization factors. Overall, the area differences are (relatively) more important for $SER = 1$ (Figure 8a) than for $SER = 8$ (Figure 8b), because all non-S-box related logic is mostly independent of SER, and the S-box cost is proportional

[9] We do not show the "unopt." case in our other analyzes since it leads to almost identical results as the optimized case for the other metrics.



(a) $SER = 1$ (b) $SER = 8$

Fig. 8: Area utilization (in kGE, post-synthesis) of a PRESENT-128 core in a commercial 65 nm ASIC technology.

to $1/SER$. Furthermore, relative area differences are more significant for higher d values, because the S-box cost is quadratic in d while the one of the other parts is linear in d. HPC1 and HPC2 have similar cost, in-between DOM and Faust et al. For very low masking orders ($d = 2, 3$), HPC2 is slightly more area efficient, while HPC1 is getting better at higher orders.

We note that randomness generation is out-of-scope for this work, therefore we did not include any random number generator in the area measurements and assume it comes from some RNG, the cost of which can be estimated from the randomness requirement of Figure 7b) and the RNG technology used.

We conclude that glitch-resistant and composable gadgets can be obtained at affordable cost. On the one hand, our implementations significantly outperform the ones based on the Faust et al. multiplication for all performance metrics. On the other hand, their latency and randomness are comparable with (flawed and potentially problematic at high orders) approaches such as DOM.

In order to confirm that these implementations satisfy minimum concrete security guarantees, we synthesized our architecture on a Xilinx Spartan FPGA and ran leakage detection tests. These preliminary results are provided in Appendix C. We leave the thorough investigation of the worst-case security of our implementations as a scope for further investigations.

7 Security proofs

We finally prove the main results outlined in Section 4.

7.1 Glitch-robustness proofs

First, we show that considering glitches on the input shares of a gadget would lead to an equivalent definition of robust simulatability.

Lemma 1. *Let G be a gadget and P a set of extended probes that can be glitch-robustly simulated using a set of inputs I by a simulator S. In presence of glitches on the inputs of G, there exists a simulator S^g that can simulate P using extended probes on inputs I.*

Proof. For a probe $p \in P$, let $G^g_{rob,p}$ (resp., $G_{rob,p}$) be the wires to which the extended probes p expands to when there are (resp., there are no) input glitches. Then, any wire w belonging to $G^g_{rob,p}$ either belongs to $G^g_{rob,p}$ and can be simulated by S^g (by using S), or is due to a glitch on an input. Let i be that input wire, then w belongs to the extended probe $G^g_{rob,p}$, and i belongs to $G_{rob,p}$ which implies that $i \in I$ since S must have knowledge of i to simulate p. Therefore, the simulator S^g has access to w through the extended input probe i. □

Next, we prove our main composition result in the glitch-robust model.

Theorem 1 (Glitch-robust composability). *For a gadget 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 its outputs corresponding to probe extensions). 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).*

Proof. From simulators S_i^{rob}, Lemma 1 gives simulators $S_i^{rob,g}$ that work with glitches on the inputs of the gadgets. The compositional strategy can then be applied with the simulators $S_i^{rob,g}$ as it would be in the restricted probing model with simulators S_i^{std}, except that the $S_i^{rob,g}$ take extended probes as inputs and produces extended probes. □

7.2 HPC1 is glitch-robust PINI

In this section, we prove that the gadget described in Section 4.2 is glitch-robust PINI. In order to keep the proof simple, we use a composability-based approach. We first give the property (introducing a new technical simulatability-based definition) that is satisfied by the DOM gadget. Then, we prove that composing this gadget with a glitch-robust SNI refresh at one of the inputs (giving the HPC1 multiplication) is glitch-robust PINI.

Definition 5 (t-Limited-PINI). *Let G be a gadget, S a set of its input sharings, P_1 a set of t_1 (extended) internal probes and A a set of t_2 share indexes such that $t_1 + t_2 \leq t$. Let P_2 be the set of all output shares of G whose index is in A. The gadget G is (glitch-robust) t-Limited-PINI (t-LPINI) with respect to I if, for any P_1 and A, there exists a set B of at most t_1 shares indexes such that the set of (extended) probes $P_1 \cup P_2$ can be (glitch-robustly) simulated using the shares with indexes in $A \cup B$ for input sharings not in I, and at most t shares of each input sharing in I.*

Remark. t-LPINI stands between t-PINI and t-NI: if the set S is empty, t-LPINI is the same as t-PINI; if it contains all the input sharings, t-LPINI is t-NI.

Lemma 2. *Let G be a (glitch-robust) t-LPINI gadget with respect to the set of input sharings S. Let G' be the gadget built by adding a (glitch-robust) t-SNI refresh gadget to each of the input sharings of G that are in S. The gadget G' is (glitch-robust) t-PINI.*

Proof. Internal or output probes of G can be simulated by input in one circuit share for sharings not in S, by the t-LPINI definition. For input sharings in S, no input share is required thanks to the SNI gadget. Furthermore, each probe inside a SNI refresh gadget can be simulated using one share of one input sharing, which satisfies the definition. Combining both kinds of probes does not cause any issue, since the total number of probes in G is at most t and the total number of (adversarial of propagated) probes on a SNI refresh gadget is also at most t. The glitch-robust result follows from Theorem 1. □

Corollary 1. *A multiplication gadget built from a (glitch-robust) t-LPINI multiplication (with $S = \{s_0\}$) where the input sharing s_0 is refreshed by a t-SNI refresh is t-PINI.*

We conclude by proving that the DOM multiplication is glitch-robust LPINI.

Proposition 1. *The DOM-indep multiplication gadget (Algorithm 1) with d shares is glitch-robust $(d-1)$-LPINI with respect to input set $\{b\}$.*

Proof. Let us build a simulator. Let P be the set of adversarial extended probes, and set $I = J = \emptyset$. Wlog, let use assume that the only extended probes in P are of the form u_{ij} or c_i, since any other extended probe is less powerful (i.e. the corresponding wires of a probe are all contained in the corresponding wires of either a u_{ij} or c_i probe). The glitch-extended probe on $a_i \otimes b_i$ will be ignored since c_i supersedes it (simulating it requires knowledge of a_i and b_i).

For all c_i probes in P, set $I \leftarrow I \cup \{i\}$ and $J \leftarrow J \cup \{i\}$. Then, for each u_{ij} probe in P, if $i \notin I$, set $I \leftarrow I \cup \{i\}$, otherwise set $I \leftarrow I \cup \{j\}$; and if $j \notin J$, set $J \leftarrow J \cup \{j\}$, otherwise set $J \leftarrow J \cup \{i\}$. We observe that the sets I and J, which are the inputs needed for simulation, satisfy the LPINI definition.

Simulation of the probes proceeds as follows: for each pair (i, j) such that either there is a probe u_{ij} or there is a probe c_i: if $i \in I$ and $j \in J$, compute $a_i \otimes b_j$ and $u_{ij} = a_i \otimes b_j \oplus r_{ij}$ using the provided inputs (and set $r_{ij} = r_{ji}$ to a fresh random if it is not yet set), otherwise set u_{ij} to a fresh random. Simulation is completed by computing c_i as it is done by the true gadget.

We conclude the proof by showing that the simulation is indistinguishable from the gadget. The simulator behaves in the same way as the circuit, except when it needs to simulate u_{ij} where $i \notin I$ or $j \notin J$. In this case, u_{ij} is not probed but appears in a probe, therefore c_i is probed. This implies that $i \in I$, thus $j \notin J$, which implies that neither c_j nor u_{ji} are probed. Since u_{ij} contains the random r_{ij}, which itself does not appear in any probe except c_i (through u_{ij}), u_{ij} behaves as a fresh random from the point of view of the adversary, which is what the simulator generates. □

Remark This proof shows that DOM is glitch-robust LPINI with respect to either $\{a\}$ or $\{b\}$. (This does not imply that it is LPINI w.r.t. $\emptyset$, i.e., PINI).

7.3 Randomness-efficient generic glitch-robust SNI refresh

The refresh gadget of Battistello et al. [6] is the SNI refresh with best known asymptotic complexity in $\mathbb{F}_2$ ($\mathcal{O}(d \log d)$). We briefly recall its working principle (restricting to the cases where d is a power of 2). First, for input sharing $(x_i)_{i=0,\dots,d-1}$, the half refresh gadget R_d^{half} outputs

$$y_i = \begin{cases} x_i \oplus r_i & \text{if } i < n/2, \\ x_i \oplus r_{i-n/2} & \text{if } i \geq n/2. \end{cases}$$

For $d = 2$, the SNI refresh gadget is the half refresh ($R_2^{\text{Bat.}} = R_2^{\text{half}}$), while for $d > 2$, $R_2^{\text{Bat.}}$ is defined as follows (input $(a_i)_i$ and output $(d_i)_i$):

$$\begin{aligned} (b_i)_{i=0,\dots,d-1} &\leftarrow R_d^{\text{half}}\left((a_i)_{i=0,\dots,d-1}\right), \\ (c_i)_{i=0,\dots,d-1} &\leftarrow \left(R_{d/2}^{\text{Bat.}}\left((b_i)_{i=0,\dots,d/2-1}\right), R_{d/2}^{\text{Bat.}}\left((b_i)_{i=d/2,\dots,d-1}\right)\right), \\ (d_i)_{i=0,\dots,d-1} &\leftarrow R_d^{\text{half}}\left((c_i)_{i=0,\dots,d-1}\right). \end{aligned}$$

This gadget can be made glitch-robust by adding a register after each XOR in the half refresh gadgets, which results in a latency of $2\log_2(d) - 1$.

We next sketch how the standard probing model proof from [6] can be adapted to the glitch-robust case. Proceeding by induction, the base case $d = 2$ is glitch-robust SNI. There are two kinds of probes to consider: probes on the inner SNI refresh gadgets and extended probes on the input and output half refresh gadgets. The probes on the internal SNI refresh gadgets are handled inductively as in the original proof. For the output half refresh gadgets probes, we adapt Lemma 4 from [6] by allowing extended probes $\{a_1, r, a_1 \oplus r\}$ and $\{a_2, r, a_2 \oplus r\}$ in $\mathcal{V}$, and the proof of the lemma is trivially extended if the integer restriction on t_1 and t_2 is relaxed to being half of integers.

For the input half refresh, referring to the proof of Lemma 6 in [6] the only non-trivial case happens when one of R_1 and R_2 is saturated (let us assume wlog it is R_2). In this case, extended probes $\{a_{d/2+i}, r_i, a_{d/2+i} \oplus r_i\}$ are simulated by requiring input $a_{d/2+i}$, and probes $\{a_i, r_i a_i \oplus r_i\}$ are not more powerful than their non-extended $a_i \oplus r_i$ counterparts (since if $a_i \oplus r_i$ is probed, the simulator requires the inputs a_i and $a_{d/2+i}$), hence we can safely ignore them.

7.4 Glitch-robust SNI refresh gadgets with minimal latency

In this section, we prove that our generic latency-reducing transformation is correct. This transformation reduces the latency of any glitch-robust SNI refresh to once cycle (which is the minimum possible) at zero cost (see Section 4.3).

We formalize (and extend to the glitch-robust probing setting) the idea from [3] that if a sharing of zero can be generated without adversarial probes and is then XORed with the input x, one directly obtains a SNI refresh. Of course, adversaries can also probe the generation of this randomness, and we next show how this "zero probe requirement" can be relaxed if the generation of the 0-sharing is done in a proper way. For hardware implementations, registers are needed in the generation of the 0-sharing, and after the XOR operation. We then show that a correct way to instantiate the 0-sharing generation is to use a (glitch-robust) SNI refresh and to feed it with an all-zero input sharing.

First, we formally define what is a 0-sharing generation gadget and give the security property it should satisfy.

Definition 6 (0-sharing generation gadget). *A 0-sharing generation gadget is a gadget with no inputs and one output sharing of $t+1$ shares $r_0, \ldots, r_t$ such that $r_0 \oplus \cdots \oplus r_t = 0$.*

Definition 7. *A gadget with no inputs and one output sharing is (glitch-robust) Strongly Output Independent (SOI) if there exists a simulator S such that for any sets I and O, the distributions of the (extended) probes for the two following games are identical. Let I be a set of (extended) probes in the gadget and O a set of (extended) probes on the output of the gadget such that $|I| + |O| = t = d - 1$.*

Real. *The output of the real game is the values corresponding to the probes (I, O) for an execution of the gadget.*

Simulated. *The simulator S outputs sets of probes (O_1, O_2) such that $O_1 \cup O_2 = O$ and $|O_1| \leq |I|$ and simulates the probes belonging to I and O_1. S takes as input I and O. The probes corresponding to O_2 are generated independently according to the uniform distribution: $P_{O_2} \leftarrow \$$.*

We next prove that a glitch-robust SNI refresh gadget connected to an all-zero input sharing is glitch-robust SOI.

Proposition 2. *Let $z \leftarrow R(x)$ be a glitch-robust t-SNI refresh gadget whose outputs can be written as $z_i = x_i \oplus y_i$, where y_i is independent of x.*[10] *The gadget $G = R(0, \ldots, 0)$ is glitch-robust t-SOI.*

Proof. A t-SOI simulator proceeds as follows given sets I and O: first, run the R t-SNI simulator (with I as internal probes and O as output probes, answering "0" to oracle requests), then set O_1 as the set of probes z_i in O whose corresponding input x_i is asked to the oracle by the SNI simulator. Set $O_2 = O \backslash O_1$. Finally, output the values for the probes in I and O_1 obtained from the SNI simulator. The sets O_1 and O_2 satisfy the SOI definition: their union is O and $|O_1| \leq |I|$ by the SNI definition.

To complete the proof, we show that the statistical distribution of the output of the simulator satisfies the definition. For probes in I and O_1, correct simulation is a consequence of SNI simulation and correctness of the oracle.

[10] As far as we know, this condition is satisfied by all the refresh gadgets in the additive boolean masking literature (such as [25,3,6]).

Finally, let us prove the independence and uniformity of the distribution of the probes in O_2. Without loss of generality, let $O_2 = \{z_1, \ldots, z_m\}$ where $m = |O_2|$ (shares are re-ordered if needed). In the computation $z = R(x)$, the assumption on the structure of R implies that $z_i = x_i \oplus y_i$, which we write in vector form $\mathbf{z} := (z_1, \ldots, z_m) = \mathbf{x} \oplus \mathbf{y}$. Since the SNI simulator can perfectly simulate $T = (I, O_1, \mathbf{x} \oplus \mathbf{y})$ (where we consider I and O_1 as vectors) for any value of $\mathbf{x}$, the distribution of the tuple is independent of $\mathbf{x}$. Therefore, for any $\mathbf{x}$ and any possible value t of the tuple, $\Pr[(I, O_1, \mathbf{y}) = t] = \Pr[(I, O_1, \mathbf{y} \oplus \mathbf{x}) = t]$, and thus for any fixed (I, O_1), the distribution of $\mathbf{y}$ is uniform. □

We finally show that the refresh construction based on a SOI 0-sharing generation gadget XORed with the input sharing with an output register is SNI.

Proposition 3. *Let G be a glitch-robust t-SOI 0-sharing generation gadget. The gadget $u \leftarrow G'(x)$ defined by $r \leftarrow G()$, $u \leftarrow \mathsf{Reg}\,[x \oplus r]$ is a glitch-robust t-SNI refresh.*

Proof. Let us give the following SNI simulator: for an internal probe set P_i and an output probe set P_o (such that $|P_i| + |P_o| \leq t$), run the glitch-robust t-SOI simulator with I being the restriction of P_i to probes in the 0-sharing generation and O being the randoms r_i corresponding to probes u_i in P_o or appearing in the extended probes P_i. The inputs asked to the SNI oracle are the x_i that appear in extended probes P_i and the x_i corresponding to the elements r_i in O_1 (as obtained from the SOI simulator).

The simulator can thus simulate all the probes in P_i: they either are in I (hence are given by the SOI simulator) or are (subsets of) extended probes $(r_i, x_i, r_i \oplus x_i)$, and thus obtained from the oracle and the SOI simulator. For the probes y_i in P_o whose corresponding r_i is in O_1: the x_i is known from the oracle and r_i from the SOI simulator, hence it can be simulated. The output probes whose corresponding r_i is in O_2 can be simulated as fresh uniform independent randoms since r_i are independent of any other input/observation. □

7.5 HPC2 is glitch-robust PINI

As a last result, we prove that the HPC2 gadget is glitch-robust PINI.

Proposition 4. *The HPC2 multiplication gadget (Algorithm 2) with d shares is glitch-robust $(d-1)$-PINI.*

Proof. Let us build a PINI simulator. We assume wlog that only c_i, $a_i \otimes b_i$, u_{ij}, v_{ij} and $a_i \otimes v_{ij}$ are probed (since other extended probes are less powerful). Given a set of probes adversarial extended probes P and probed output shares A, the set of required input shares X is computed as follows: for each probed c_i or $a_i \otimes b_i$, add i to X. Then, for each $i \neq j$ pair, if two out of u_{ij}, v_{ij} and $a_i \otimes v_{ij}$ are probed, or if i of j belongs to X: add i and j to X. Otherwise, if u_{ij} or $a_i \times v_{ij}$ is probed, add i to X, and if v_{ij} is probed, add j to X. The set B is computed as $X \setminus A$.

We observe that the set B satisfies the PINI definition: $|B| \leq |P|$ by construction. All the values to be simulated that depend only on input shares with index in X and on randomness are computed as specified by Algorithm 2 (required randomness is generated). The allows to simulate all $a_i \otimes b_i$, u_{ij} and v_{ij} extended probes by construction of X. Then, for all remaining extended probes (c_i (for which $i \in A$) and $a_i \otimes v_{ij}$), we observe that $i \in X$. They can therefore be computed as it is done by the gadget, except when simulation of $v_{ij} = b_j \oplus r_{ij}$ is needed and $j \notin X$. In this case, the simulator simulates v_{ij} by sampling a fresh random r'_{ij} (we say that the simulator *cheats* for ij).

Let us now show that this algorithm produces the same probe distribution as the true gadget. The behavior of the simulator is identical to the behavior of the gadget, except when it cheats for ij. We next prove that if the simulator cheats for ij, then r_{ij} is not observed in the set of probes, except through v_{ij}, therefore v_{ij} is indistinguishable from r'_{ij} and simulation is correct.

The simulator cheats for ij only if $j \notin X$ and a value depending on v_{ij} is probed. The first condition implies that none of c_j, u_{ji}, $a_j \otimes v_{ij}$ and v_{ij} are probed, and at most of c_i, $a_i \otimes v_{ij}$, u_{ij} and v_{ji} can be probed. The second condition implies that c_i, or $a_i \otimes v_{ij}$ is probed (v_{ij} cannot be probed due to the previous observation). Therefore, the only values depending on r_{ij} that can be probed are c_i or $a_i \otimes v_{ij}$, and one (and only one) of those is probed. If $a_i \otimes v_{ij}$ is probed, then the simulation is correct: the extended probe expands to $\{a_i, v_{ij}, a_i \otimes v_{ij}\}$, which are the only observations depending on r_{ij}. If c_i is probed, then observations depending on r_{ij} are $\bar{a_i} \otimes r_{ij}$ and $a_i \otimes v_{ij}$, and functions of these. If $a_i = 0$, then $a_i \otimes r_{ij} = 0$ does not depend on r_{ij}, which is thus only observed through v_{ij}, hence the simulation is correct. Otherwise, we have $\bar{a_i} = 0$, which implies that $\bar{a_i} \otimes v_{ij} = 0$, thus v_{ij} is not observed and cheating is not observed. □

References

1. Victor Arribas, Svetla Nikova, and Vincent Rijmen. Vermi: Verification tool for masked implementations. In *ICECS*, pages 381–384. IEEE, 2018.
2. Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskverif: Automated verification of higher-order masking in presence of physical defaults. In *ESORICS (1)*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.
3. 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.
4. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.
5. 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.
6. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39. Springer, 2016.
7. 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.
8. Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In *ASIACRYPT (2)*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.
9. Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.
10. Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A more efficient AES threshold implementation. In *AFRICACRYPT*, volume 8469 of *Lecture Notes in Computer Science*, pages 267–284. Springer, 2014.
11. 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 *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 321–353. Springer, 2018.
12. Gaetan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IACR Cryptology ePrint Archive*, 2018:438, 2018.
13. Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.
14. Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, 2007.
15. 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.
16. Joan Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 2017.
17. Hassan Eldib, Chao Wang, and Patrick Schaumont. Formal verification of software countermeasures against side-channel attacks. *ACM Trans. Softw. Eng. Methodol.*, 24(2):11:1–11:24, 2014.
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.
19. Wieland Fischer and Berndt M. Gammel. Masking at gate level in the presence of glitches. In *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 187–200. Springer, 2005.
20. Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, 2017.

21. Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.
22. Hannes Groß and Stefan Mangard. A unified masking approach. *J. Cryptographic Engineering*, 8(2):109–124, 2018.
23. Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *IACR Cryptology ePrint Archive*, 2016:486, 2016.
24. Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In *CT-RSA*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.
25. 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.
26. Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.
27. 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.
28. 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.
29. Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-channel resistant crypto for less than 2, 300 GE. *J. Cryptology*, 24(2):322–345, 2011.
30. Oscar Reparaz. A note on the security of higher-order threshold implementations. *IACR Cryptology ePrint Archive*, 2015:1, 2015.
31. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
32. Tobias Schneider and Amir Moradi. Leakage assessment methodology - extended version. *J. Cryptographic Engineering*, 6(2):85–99, 2016.
33. Kai Schramm and Christof Paar. Higher order masking of the AES. In *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.
34. Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.
35. Ko Stoffelen. Optimizing s-box implementations for several criteria using SAT solvers. In *FSE*, volume 9783 of *Lecture Notes in Computer Science*, pages 140–160. Springer, 2016.
36. Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, 2013.

A Optimized refresh gadgets

The input (resp., output) sharing is denoted as $\mathbf{x}$ (resp., $\mathbf{y}$). All r_i variables are independent uniform random elements, and $\mathbf{s}^i$ are vectors of d independent randoms elements. The $(\cdot \gg i)$ operator applied to a vector denotes a rotation of its elements: the 1st element becomes the $i+1$-th, etc. Registers are denoted as $\mathsf{R}[\cdot]$.

$d = 2$

$y_0 \leftarrow \mathsf{R}[x_0 + r_0]$
$y_1 \leftarrow \mathsf{R}[x_1 + r_0]$

$d = 3$

$t_0 \leftarrow \mathsf{R}[r_0 + r_1]$
$y_0 \leftarrow \mathsf{R}[x_0 + r_0]$
$y_1 \leftarrow \mathsf{R}[x_1 + r_1]$
$y_2 \leftarrow \mathsf{R}[x_2 + t_0]$

$d = 4, 5$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^0}]$

$d = 6$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$t_0^0 \leftarrow \mathsf{R}[t_0^0 + r_0]$
$t_3^0 \leftarrow \mathsf{R}[t_3^0 + r_0]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^0}]$

$d = 7$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$t_0^0 \leftarrow \mathsf{R}[t_0^0 + r_0]$
$t_2^0 \leftarrow \mathsf{R}[t_2^0 + r_1]$
$t_4^0 \leftarrow \mathsf{R}[t_4^0 + r_0]$
$t_6^0 \leftarrow \mathsf{R}[t_6^0 + r_1]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^0}]$

$d = 8$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$t_0^0 \leftarrow \mathsf{R}[t_0^0 + r_0]$
$t_1^0 \leftarrow \mathsf{R}[t_1^0 + r_1]$
$t_2^0 \leftarrow \mathsf{R}[t_2^0 + r_2]$
$t_4^0 \leftarrow \mathsf{R}[t_4^0 + r_0]$
$t_5^0 \leftarrow \mathsf{R}[t_5^0 + r_1]$
$t_6^0 \leftarrow \mathsf{R}[t_6^0 + r_2]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^0}]$

$d = 9$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$t_0^0 \leftarrow \mathsf{R}[t_0^0 + r_0]$
$t_1^0 \leftarrow \mathsf{R}[t_1^0 + r_1]$
$t_3^0 \leftarrow \mathsf{R}[t_3^0 + r_2]$
$t_4^0 \leftarrow \mathsf{R}[t_4^0 + r_0]$
$t_6^0 \leftarrow \mathsf{R}[t_6^0 + r_1]$
$t_7^0 \leftarrow \mathsf{R}[t_7^0 + r_2]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^0}]$

$d = 10$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$t_0^0 \leftarrow \mathsf{R}[t_0^0 + r_0]$
$t_1^0 \leftarrow \mathsf{R}[t_1^0 + r_1]$
$t_2^0 \leftarrow \mathsf{R}[t_2^0 + r_2]$
$t_3^0 \leftarrow \mathsf{R}[t_3^0 + r_3]$
$t_4^0 \leftarrow \mathsf{R}[t_4^0 + r_4]$
$t_5^0 \leftarrow \mathsf{R}[t_5^0 + r_0]$
$t_6^0 \leftarrow \mathsf{R}[t_6^0 + r_1]$
$t_7^0 \leftarrow \mathsf{R}[t_7^0 + r_2]$
$t_8^0 \leftarrow \mathsf{R}[t_8^0 + r_3]$
$t_9^0 \leftarrow \mathsf{R}[t_9^0 + r_4]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^0}]$

$d = 11$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$t_0^0 \leftarrow \mathsf{R}[t_0^0 + r_0]$
$t_1^0 \leftarrow \mathsf{R}[t_1^0 + r_1]$
$t_2^0 \leftarrow \mathsf{R}[t_2^0 + r_2]$
$t_3^0 \leftarrow \mathsf{R}[t_3^0 + r_3]$
$t_4^0 \leftarrow \mathsf{R}[t_4^0 + r_4]$
$t_5^0 \leftarrow \mathsf{R}[t_5^0 + r_0]$
$t_6^0 \leftarrow \mathsf{R}[t_6^0 + r_1]$
$t_7^0 \leftarrow \mathsf{R}[t_7^0 + r_2 + r_5]$
$t_8^0 \leftarrow \mathsf{R}[t_8^0 + r_3]$
$t_9^0 \leftarrow \mathsf{R}[t_9^0 + r_4]$
$t_{10}^0 \leftarrow \mathsf{R}[t_{10}^0 + r_5]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^0}]$

$d = 12$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$t_0^0 \leftarrow \mathsf{R}[t_0^0 + r_0]$
$t_1^0 \leftarrow \mathsf{R}[t_1^0 + r_1]$
$t_2^0 \leftarrow \mathsf{R}[t_2^0 + r_2 + r_6]$
$t_3^0 \leftarrow \mathsf{R}[t_3^0 + r_3]$
$t_4^0 \leftarrow \mathsf{R}[t_4^0 + r_4]$
$t_5^0 \leftarrow \mathsf{R}[t_5^0 + r_5 + r_6]$
$t_6^0 \leftarrow \mathsf{R}[t_6^0 + r_0]$
$t_7^0 \leftarrow \mathsf{R}[t_7^0 + r_1]$
$t_8^0 \leftarrow \mathsf{R}[t_8^0 + r_2 + r_7]$
$t_9^0 \leftarrow \mathsf{R}[t_9^0 + r_3]$
$t_{10}^0 \leftarrow \mathsf{R}[t_{10}^0 + r_4]$
$t_{11}^0 \leftarrow \mathsf{R}[t_{11}^0 + r_5 + r_7]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^0}]$

$d = 13, \ldots, 16$

$\mathbf{t^0} \leftarrow \mathsf{R}[\mathbf{s^0} + (\mathbf{s^0} \gg 1)]$
$\mathbf{t^1} \leftarrow \mathsf{R}[\mathbf{s^1} + (\mathbf{s^1} \gg 3)]$
$\mathbf{t^2} \leftarrow \mathsf{R}[\mathbf{t^0} + \mathbf{t^1}]$
$\mathbf{y} \leftarrow \mathsf{R}[\mathbf{x} + \mathbf{t^2}]$

Table 2: Randomness cost of the best known SNI refresh gadgets at some orders for both HW (glitch-robust) and SW implementations with our two constructions. Green boxes indicate cases where our gadgets improve the randomness complexity compared to both HW and SW state-of-the-art.

d	HW [18]	SW [3,6]	HW-Gen. (App. 7.3)	HW-Opt. (Fig. 3)
2	1	1	1	1
3	3	3	3	2
4	6	4	6	4
5	10	8	8	5
6	15	12	12	7
7	21	13	15	9
8	28	16	20	11
9	36	18	22	13
10	45	20	26	15
11	55	22	30	17
12	66	24	36	20
13	78	26	39	26
14	91	28	44	28
15	105	30	49	30
16	120	32	56	32

B AND depth 2, 4 ANDs, 4-bit (optimized) S-boxes

PRESENT S.

$l_0 = x_1 \oplus x_2$
$q_0 = \neg l_0$
$q_1 = \neg x_0$
$t_0 = q_0 \otimes q_1$
$l_1 = x_2 \oplus x_3$
$q_2 = \neg l_1 \oplus x_0$
$l_3 = x_0 \oplus x_3$
$q_3 = l_3 \oplus l_0$
$t_1 = q_2 \otimes q_3$
$q_4 = \neg x_2$
$t_2 = q_4 \otimes x_1$
$l_4 = x_0 \oplus x_2$
$l_5 = t_0 \oplus t_2$
$q_6 = \neg l_4 \oplus l_5$
$q_7 = l_1 \oplus x_1$
$t_3 = q_6 \otimes q_7$
$l_7 = l_5 \oplus t_3$
$y_0 = x_3 \oplus l_7$
$l_8 = l_5 \oplus t_1$
$y_1 = l_1 \oplus l_8$
$y_2 = l_4 \oplus t_3$
$y_3 = l_3 \oplus t_2$

PRESENT S^{-1}.

$l_0 = x_0 \oplus x_2$
$l_1 = x_1 \oplus x_3$
$q_0 = l_1 \oplus x_2$
$q_1 = \neg l_0$
$t_0 = q_0 \otimes q_1$
$q_2 = \neg x_2 \oplus t_0$
$q_3 = x_1 \oplus x_2$
$t_1 = q_2 \otimes q_3$
$q_4 = \neg l_1$
$q_5 = \neg l_0$
$t_2 = q_4 \otimes q_5$
$q_6 = q_3 \oplus t_0 \oplus t_2$
$q_7 = \neg x_2 \oplus x_3$
$t_3 = q_6 \otimes q_7$
$q_8 = t_1 \oplus t_2$
$y_0 = l_0 \oplus l_1 \oplus q_8 \oplus t_3$
$y_1 = l_1 \oplus x_2 \oplus q_8$
$y_2 = l_0 \oplus x_3 \oplus t_1$
$y_3 = l_0 \oplus l_1 \oplus t_0 \oplus t_2$

PRINCE S.

$q_0 = x_1 \oplus x_3$
$q_1 = \neg q_0 \oplus x_2$
$q_2 = x_2 \oplus x_3$
$q_8 = x_0 \oplus x_1$
$q_5 = \neg q_8 \oplus q_2$
$q_4 = \neg x_0 \oplus x_3$
$t_0 = q_0 \cdot q_1$
$q_3 = q_8 \oplus x_2 \oplus t_0$
$t_1 = q_2 \cdot q_3$
$t_2 = q_4 \cdot q_5$
$q_7 = x_2 \oplus t_2$
$t_3 = (\neg x_3) \cdot q_7$
$q_9 = x_0 \oplus t_2$
$t_4 = q_8 \cdot q_9$
$q_{10} = q_4 \oplus t_0 \oplus t_2$
$q_{11} = q_4 \oplus x_2$
$t_5 = q_{10} \cdot q_{11}$
$l_3 = t_1 \oplus t_2$
$l_4 = t_3 \oplus t_4$
$l_5 = l_3 \oplus l_4$
$y_0 = q_0 \oplus t_0 \oplus t_1 \oplus t_3$
$y_1 = q_0 \oplus l_5 \oplus t_5$
$y_2 = q_0 \oplus l_4$
$y_3 = x_3 \oplus t_0 \oplus l_3$

PRINCE S^{-1}.

$q_0 = x_0 \oplus x_2$
$q_1 = q_0 \oplus x_3$
$q_2 = \neg x_2 \oplus x_3$
$q_8 = \neg x_1 \oplus x_3$
$q_5 = \neg x_1 \oplus x_2$
$q_4 = \neg q_1 \oplus x_1$
$t_0 = q_0 \cdot q_1$
$q_3 = x_1 \oplus t_0$
$t_1 = q_2 \cdot q_3$
$t_2 = q_4 \cdot q_5$
$q_7 = x_2 \oplus t_2$
$t_3 = (\neg x_2) \cdot q_7$
$q_9 = x_0 \oplus q_8 \oplus t_2$
$t_4 = q_8 \cdot q_9$
$q_{10} = q_4 \oplus t_0 \oplus t_2$
$q_{11} = q_0 \oplus x_1$
$t_5 = q_{10} \cdot q_{11}$
$l_3 = t_1 \oplus t_2$
$l_4 = q_0 \oplus t_3$
$l_5 = l_3 \oplus l_4$
$y_0 = \neg l_5 \oplus t_4 \oplus t_5$
$y_1 = l_4 \oplus t_0 \oplus t_5 \oplus t_2$
$y_2 = \neg x_2 \oplus t_0 \oplus l_3$
$y_3 = l_4 \oplus t_4$

Rectangle S.

$q_0 = \neg x_0$
$l_0 = x_0 \oplus x_2$
$l_1 = x_0 \oplus x_1$
$l_3 = l_0 \oplus x_1$
$q_1 = \neg l_0$
$t_0 = q_0 \otimes q_1$
$q_2 = \neg(x_0 \oplus x_3 \oplus t_0)$
$q_3 = \neg l_3$
$t_1 = q_2 \otimes q_3$
$q_4 = \neg l_0 \oplus x_3$
$q_5 = \neg x_2$
$t_2 = q_4 \otimes q_5$
$q_6 = l_0 \oplus x_1 \oplus t_2$
$q_7 = l_1 \oplus x_3$
$t_3 = q_6 \otimes q_7$
$l_2 = t_1 \oplus t_2$
$y_0 = l_0 \oplus t_0 \oplus l_2$
$y_1 = l_3 \oplus l_2 \oplus t_3$
$y_2 = l_1 \oplus x_3 \oplus t_0$
$y_3 = l_1 \oplus t_0 \oplus t_2$

Rectangle S^{-1}.

$l_0 = x_1 \oplus x_2$
$l_1 = l_0 \oplus x_3$
$l_2 = x_0 \oplus l_1$
$t_0 = x_0 \otimes x_3$
$q_2 = \neg l_0 \oplus t_0$
$q_3 = \neg t_0 \oplus x_2$
$t_1 = q_2 \otimes q_3$
$q_4 = \neg x_0 \oplus x_1$
$t_2 = q_4 \otimes x_3$
$q_6 = l_0 \oplus t_2$
$q_7 = \neg x_2$
$t_3 = q_6 \otimes q_7$
$y_0 = l_2 \oplus t_1 \oplus t_2$
$y_1 = l_2 \oplus t_0$
$y_2 = l_1 \oplus t_2$
$y_3 = l_1 \oplus t_1 \oplus t_3$

Class-13 S.

$l_0 = x_0 \oplus x_1$
$l_1 = l_0 \oplus x_2$
$q_0 = x_1 \oplus x_3$
$l_2 = q_0 \oplus x_2$
$q_1 = \neg l_2$
$t_0 = q_0 \otimes q_1$
$q_2 = l_1 \oplus x_3 \oplus t_0$
$q_3 = \neg x_3$
$t_1 = q_2 \otimes q_3$
$q_4 = \neg x_3$
$t_2 = q_4 \otimes x_2$
$l_3 = t_0 \oplus t_2$
$q_6 = l_1 \oplus t_2$
$q_7 = \neg x_0$
$t_3 = q_6 \otimes q_7$
$y_0 = l_2 \oplus t_2 \oplus t_3$
$y_1 = l_0 \oplus l_3$
$y_2 = l_1 \oplus t_1 \oplus l_3$
$y_3 = x_1 \oplus x_2 \oplus t_2$

Class-13 S^{-1}.

$l_0 = x_1 \oplus x_3$
$l_1 = l_0 \oplus x_2$
$l_2 = x_0 \oplus x_3$
$q_0 = \neg l_0$
$t_0 = q_0 \otimes x_1$
$q_2 = l_2 \oplus x_2 \oplus t_0$
$t_1 = q_2 \otimes x_2$
$q_4 = \neg l_2$
$q_5 = x_0 \oplus l_0$
$t_2 = q_4 \otimes q_5$
$l_3 = t_0 \oplus t_2$
$l_4 = l_3 \oplus t_1$
$q_6 = x_2 \oplus t_0$
$q_7 = x_0 \oplus x_2$
$t_3 = q_6 \otimes q_7$
$y_0 = x_2 \oplus x_3 \oplus l_4 \oplus t_3$
$y_1 = l_1 \oplus l_4$
$y_2 = x_1 \oplus x_2 \oplus l_3$
$y_3 = l_2 \oplus t_0$

Skinny S.

$q_1 = x_0 \oplus x_2$
$t_0 = x_3 \otimes q_1$
$q_2 = x_0 \oplus x_1 \oplus t_0$
$t_1 = q_2 \otimes x_0$
$q_4 = x_3$
$q_5 = \neg x_0 \oplus x_3$
$t_2 = q_4 \otimes q_5$
$l_0 = t_1 \oplus t_2$
$q_7 = x_1 \oplus x_3$
$q_6 = \neg q_1 \oplus q_7 \oplus t_2$
$t_3 = q_6 \otimes q_7$
$y_0 = x_0 \oplus x_3 \oplus l_0$
$y_1 = l_0 \oplus t_3$
$y_2 = x_1 \oplus t_0 \oplus t_2$
$y_3 = x_2 \oplus t_2$

Skinny S^{-1}.

$q_1 = x_1 \oplus x_3$
$q_0 = q_1 \oplus x_2$
$t_0 = q_0 \otimes q_1$
$l_0 = x_0 \oplus x_1$
$q_2 = l_0 \oplus t_0$
$q_3 = \neg x_3$
$t_1 = q_2 \otimes q_3$
$l_1 = x_2 \oplus x_3$
$q_5 = \neg l_1$
$t_2 = x_2 \otimes q_5$
$q_6 = l_0 \oplus x_2 \oplus t_2$
$q_7 = x_0 \oplus x_2$
$t_3 = q_6 \otimes q_7$
$y_0 = x_1 \oplus t_2$
$y_1 = q_1 \oplus t_0 \oplus t_1 \oplus t_3$
$y_2 = x_0 \oplus l_1 \oplus t_3$
$y_3 = x_0 \oplus q_1 \oplus t_0$

iClass13 S.

$l_0 = x_2 \oplus x_3$
$l_2 = x_0 \oplus x_3$
$q_0 = \neg x_1$
$t_0 = q_0 \otimes x_3$
$q_2 = l_2 \oplus t_0$
$q_3 = \neg x_2$
$t_1 = q_2 \otimes q_3$
$q_4 = l_0$
$q_5 = \neg x_1$
$t_2 = q_4 \otimes q_5$
$l_1 = t_0 \oplus t_2$
$q_6 = \neg x_0 \oplus x_2 \oplus l_1$
$q_7 = x_0 \oplus l_0$
$t_3 = q_6 \otimes q_7$
$y_0 = q_7 \oplus x_1 \oplus t_1 \oplus t_2$
$y_1 = q_7 \oplus t_2$
$y_2 = l_0 \oplus l_1$
$y_3 = l_2 \oplus t_1 \oplus t_3$

Prøst S.

$q_1 = x_0 \oplus x_2$
$q_0 = q_1 \oplus x_1$
$t_0 = q_0 \otimes q_1$
$q_2 = \neg q_1 \oplus x_3 \oplus t_0$
$t_1 = q_2 \otimes x_0$
$q_4 = x_0 \oplus x_1$
$q_5 = \neg x_0$
$t_2 = q_4 \otimes q_5$
$l_1 = t_0 \oplus t_2$
$q_6 = q_0 \oplus t_2$
$t_3 = q_6 \otimes x_3$
$y_0 = x_1 \oplus x_2 \oplus t_2$
$y_1 = q_0 \oplus x_3 \oplus l_1$
$y_2 = q_1 \oplus t_0 \oplus t_1 \oplus t_3$
$y_3 = q_1 \oplus l_1 \oplus t_3$

C Leakage detection tests

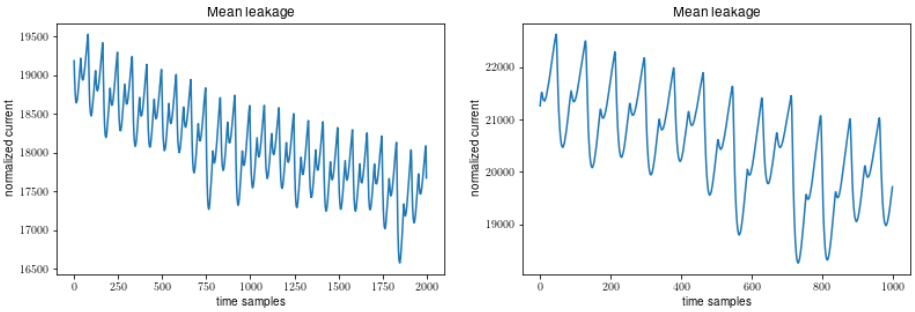
C.1 Low-orders leakage detection checks

The evaluation setup which is used in this manuscript is composed of a PicoScope oscilloscope and a SAKURA-G board. The SAKURA-G board embeds a Xilinx FPGA (Spartan-6 in 45nm technology) was utilized to inhabit the evaluated PRESENT-128 architecture. The PicoScope 5244B oscilloscope was used to capture the power supply current with a passive inductive probe (Tektronix-CT1) connected serially to the measurement points. Sampling rate of 100 MS/s was practiced and the device was clocked at 4 MHz. Inputs were asserted through a UART interface to the FPGA. Fresh randomness was generated on the FPGA with an AES architecture in CTR mode supplied with a random key.

For leakage detection the detection method used is the traditional univariate one, based on Welch's (two-tailed) T-test. It is computed on two input sequences (Set_0 and Set_1). In this work we compare two classes of leakages with so-called specific "*fixed* vs. *fixed*" tests to detect leakages, using the following statistic:

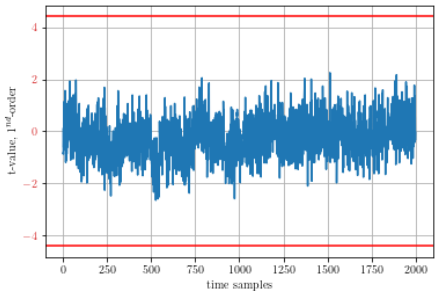
$$T_{value} = (\mu_{Set_0} - \mu_{Set_1}) \Big/ \sqrt{{\sigma_{Set_0}}^2/|Set_0| + {\sigma_{Set_1}}^2/|Set_1|}, \tag{1}$$

where μ and σ are the populations' mean and standard-deviation, respectively. The leakages from the fixed sequences were recorded with fixed input and key. Detection was assumed for estimated statistics beyond a certain threshold.In addition, we use the generalization in [32] to analyze higher-order statistical leakages. The left Subfigures column of Figure 9(a, c and e) shows the mean leakage, the 1^{st} and 2^{nd} order leakage-detection (T-tests) of a 2-shared implementation. The right Subfigures column of Figure 9(b, d, f and g) shows the mean leakage, the 1^{st}, 2^{nd} and 3^{rd} order leakage-detection (T-tests) of a 3-shared implementation. Figure 10 shows the mean leakage, the 1^{st}, 2^{nd}, 3^{rd} and 4^{th} order leakage-detection (T-tests) of a 4-shared implementation. As demonstrated in the figures for an d^{th} order implementation, leakage is only visible at the d^{th} statistical moment as exacted. The figures present the results with $240 \cdot 10^3/6 \cdot 10^6$ and $9 \cdot 10^6$ traces (samples) for the 2/3 and 4-shares designs.

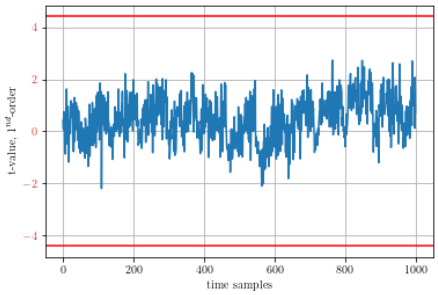


(a) $d = 2$, average trace

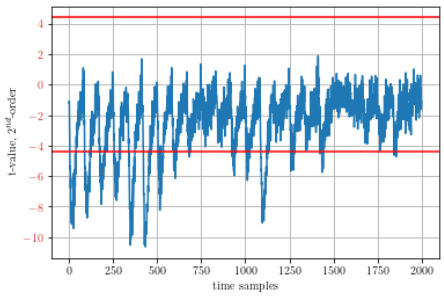
(b) $d = 3$, average trace



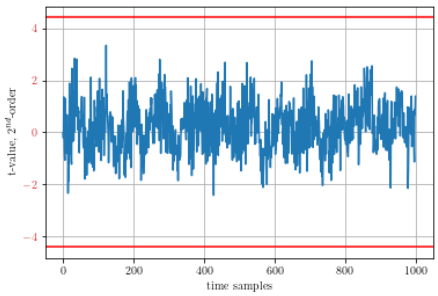
(c) $d = 2$, first order



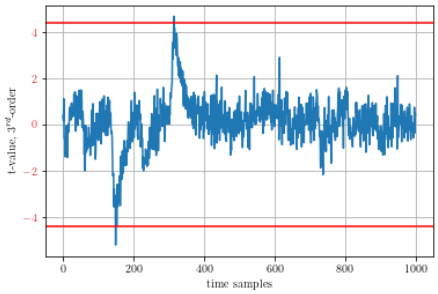
(d) $d = 3$, first order



(e) $d = 2$, second order

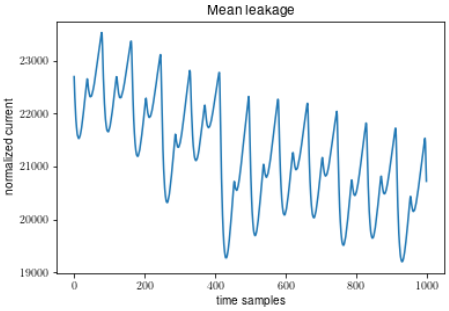


(f) $d = 3$, second order

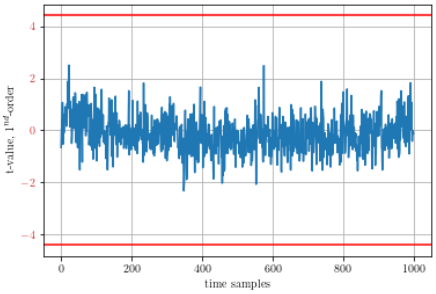


(g) $d = 3$, third order

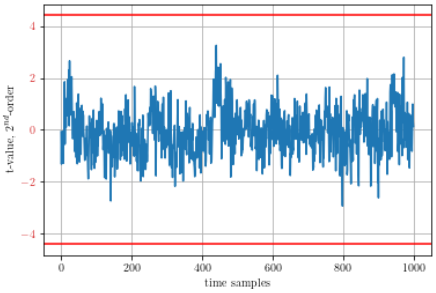
Fig. 9: T-tests of masked PRESENT-128 HPC1 on FPGA for $d = 2, 3$.



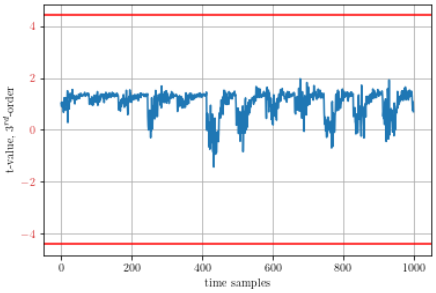
(a) $d = 4$, average trace



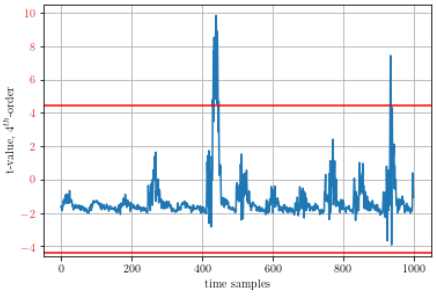
(b) $d = 4$, first order



(c) $d = 4$, second order



(d) $d = 4$, third order



(e) $d = 4$, fourth order

Fig. 10: T-tests of masked PRESENT-128 HPC1 on FPGA for $d = 4$.