

# Trivially and Efficiently Composing Masked Gadgets with Probe Isolating Non-Interference

Gaëtan Cassiers and François-Xavier Standaert

ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium  
e-mails: gaetan.cassiers@uclouvain.be; fstandae@uclouvain.be

**Abstract.** We revisit the analysis and design of masked cryptographic implementations to prevent side-channel attacks. Our starting point is the (known) observation that proving the security of a higher-order masked block cipher exhaustively requires unrealistic computing power. As a result, a natural strategy is to split algorithms in smaller parts (or gadgets), with as main objectives to enable both simple composition (as initiated by Barthe et al. at CCS 2016) and efficient implementations. We argue that existing composition strategies allow either trivial composition with significant overheads or optimized composition with more analysis efforts. As a result, we first introduce a new definition of Probe Isolating Non-Interference (PINI) that allows both trivial composition and efficient implementations. We next prove general composition theorems for PINI gadgets that considerably simplify the analysis of complex masked implementations. We finally design efficient multiplication gadgets that satisfy this definition. As additional results, we exhibit a limitation of existing compositional strategies for the analysis of Multiple-Inputs / Multiple-Outputs (MIMO) gadgets, extend Barthe et al.’s definition of Strong Non-Interference (SNI) to deal with this context, and describe an optimization method to design efficient MIMO-SNI (sub)circuits. Our results allow proving the security of a recent masked AES implementation by Goudarzi and Rivain (EUROCRYPT 2017). From the implementation viewpoint, PINI implementations reach the level of performance of the best composable masking schemes for the AES Rijndael, and outperform them by significant factors for lightweight ciphers.

## 1 Introduction

Side-channel attacks such as the differential power analysis [27] are a significant threat to the security devices implementing cryptographic functionalities. Masking is among the most popular countermeasures to prevent such attacks. Its working principle is to split each sensitive data  $x$  manipulated by an implementation into a randomized sharing  $(x_0, \dots, x_{d-1})$  such that  $x = x_0 \oplus \dots \oplus x_{d-1}$ , and to perform the computations on those shares only [12] by replacing each operation (e.g. boolean gate) by a gadget that performs the operation over randomized sharings. Under now well understood (noise and independence) leakage assumptions, masking guarantees that the security of a masked implementation against any side-channel attack grows exponentially in the number of shares [19,20].

**State-of-the-art.** In the current state-of-the-art, masking schemes usually come with a security proof in the so-called probing model [25,33]. In its simplest definition,  $t$ -probing security requires that the observation of up to  $t$  intermediate computations in the implementation does not reveal anything about the sensitive variables.<sup>1</sup> It has later been shown that while locally sufficient, this definition does not ensure secure composition [17].

Many solutions have been introduced in order to mitigate this issue, but none of them is both generic and efficient. Genericity (that we also name trivial composition) means that a solution can be easily applied to any circuit for any  $t$  using a simple circuit transformation that maps each logic gate to a masked gadget implementing that kind of gate (independently of the structure of the circuit). Efficiency is harder to characterize in a binary way, hence we rely on two criterion's: the number of shares  $d$  should be minimal (i.e.  $d = t + 1$ ), and linear operations should be implemented trivially, by implementing these operations share by share.

Based on this observation, the problem we tackle in this paper is: *Can we define generic composition rules for  $d = t + 1$  masking such that the trivial implementation of linear functions is directly composable (i.e., does not need to be refreshed) without causing significant overheads for the non-linear operations?*

**Contribution.** We answer this question positively by introducing a new security notion that we denote as Probe Isolating Non-Interference (PINI), which is satisfied by linear operations and enjoys the useful property that any composition of PINI gadgets is PINI. In contrast with the (Strong) Non-Interference (NI/SNI) definitions of Barthe et al. [4] that rely on the the number of probes in a target implementation, PINI rather relies on their position (i.e., the shares' indexes). We then show that this approach is applicable by designing multiplication gadgets that are PINI. As a result, we can trivially analyze complex masked circuits, where all linear operations are trivially implemented and all non-linear operations are PINI. We apply our results to analyze the composition strategy of Goudarzi and Rivain [22], which can be viewed as a trivial composition that uses as non-linear element a special "double-SNI" multiplication gadget. We prove that this gadget is PINI, leading to a direct formal proof that the composition strategy of [22] is secure.

In order to confirm that the trivial PINI composition also leads to efficient implementations, we next compare the performances of masked block cipher implementations based on PINI with other published solutions. Since it is currently the most efficient approach for masking, we use bit-level implementations (such as the software bitslice AES by Goudarzi and Rivain [22]) for this purpose, which leads us to a couple of additional observations of independent interest.

---

<sup>1</sup> Admittedly, a security proof in the (abstract) probing model is only a first step in the analysis of a masked implementation. Various physical defaults can contradict probing security, e.g., by re-combining the shares because of glitches [28,30] or transition-based leakages [14,1]. Yet, it is a necessary first step since an insecurity in the probing model usually leads to powerful concrete attacks [16,29].

First, we analyze the limitations of the SNI definition given in [4] for composition in such bit-level implementations (such limitations apply for any gadget with multiple inputs and multiple outputs). We define Multiple-Input Multiple-Output Strong Non-Interference (MIMO-SNI) as the natural extension of SNI that allows composition in this case. Interestingly, it turns out that such cases are not problematic in the PINI framework, and we show that any MIMO-SNI gadget is actually PINI.

Second, the optimized composition of complex circuits such as bit-level AES S-boxes requires significantly more efforts than its counterpart in  $\mathbb{F}_{256}$  studied by Belaïd et al. [6]. We propose a solution to this problem, by representing the circuit to mask as a “computation graph”, and describe how to express the definitions of NI, SNI and MIMO-SNI as graph properties, leading to an algorithm (implemented as an open-source tool) to minimize the number of SNI gadgets.

We finally illustrate that PINI gadgets lead to excellent randomness complexities and operation counts that often improve overall performance over state-of-the-art solutions, by comparing these metrics for implementations of various masked block ciphers according to the use of only SNI gadgets, the double-SNI strategy, our MIMO-SNI optimization, the recent Tight Private Circuit (TPC) approach [7] and the PINI framework. For this purpose, we use the standard AES block cipher as well as some lightweight block ciphers (Noekeon [18], Present [8] and Fantomas [24]). These results allow us to conclude that our tools enable both trivial composition and efficient masked implementations. For the AES, the TPC approach and the PINI framework lead to the best performances. For lightweight ciphers, the PINI approach outperforms all existing solutions by significant factors (2 to 4), making it a particularly relevant solution for lightweight ciphers submitted to the ongoing NIST competition.<sup>2</sup>

This paper is organized as follows. Section 2 recalls the notion of masked circuit and gadget, along with the relevant security notions. We also explain the so-called “probe propagation framework” used to build intuition about those notions. In Section 3, we introduce the PINI definition and its properties. We instantiate it with two gadgets in Section 4, which completes the main contribution of this paper. Section 5 details the limitations of SNI, introduces the notion of MIMO-SNI and covers the design of the optimized MIMO-SNI AES S-box. Related work is reviewed in Section 6. Finally, Section 7 briefly compares the performance of various implementation strategies and concludes.

## 2 Preliminaries

### 2.1 Masked gadgets

We work with circuits and use the definition of [25]. A deterministic circuit  $C$  is a Directed Acyclic Graph (DAG) whose vertices are gates, inputs or outputs,

<sup>2</sup> <https://csrc.nist.gov/projects/lightweight-cryptography/round-1-candidates>.

and whose edges are wires carrying elements of  $\mathbb{F}_q$ . A randomized circuit is a circuit augmented with random gates. A random gate is a gate with fan-in 0 that produces a random output, uniformly and independently of everything else afresh for each invocation of the circuit.

Let  $x_* = (x_i)_{i=0,\dots,d-1}$  be a  $d$ -sharing of a sensitive variable  $x$  if  $x = \sum_{i=0}^{d-1} x_i$ . The index  $i$  is named the *share index*. A set  $A$  is a set of share indexes if  $A \subset \{0, \dots, d-1\}$ , and we denote  $x_A := \{x_i : i \in A\}$ . We say that the sharing  $x_*$  is *fresh* if any tuple of at most  $d-1$  of its shares is distributed uniformly and independently of any other element under consideration.

A gadget  $G$  with  $m$  inputs and  $n$  outputs working with  $d$  shares implementing a function  $f : \mathbb{F}_q^m \rightarrow \mathbb{F}_q^n : (x_0, \dots, x_{m-1}) \mapsto (y_0, \dots, y_{n-1})$  is a circuit with  $md$  inputs grouped into  $m$   $d$ -sharings denoted  $(x_{*,0}, \dots, x_{*,m-1})$  and  $nd$  outputs grouped into  $n$  sharings denoted  $(y_{*,0}, \dots, y_{*,n-1})$ . A gadget must be correct. That is, if  $x_j = \sum_{i=0}^{d-1} x_{i,j}$  for all  $j$ , then  $y_j = \sum_{i=0}^{d-1} y_{i,j}$  for all  $j$  and for any value of the outputs of the random gates.

We additionally use the following notations:  $x_{i,*} = \{x_{i,j} : 0 \leq j \leq m-1\}$ ,  $x_{A,*} = \{x_{i,j} : i \in A, 0 \leq j \leq m-1\}$  where  $A$  is a set of share indexes, and  $x_{*,*} = \{x_{i,j} : 0 \leq i \leq d-1, 0 \leq j \leq m-1\}$ . When it is not clear from the context, we explicitly denote the gadget  $G$  to which the inputs or the outputs are related with a superscript as  $x_{i,j}^G, y_{i,j}^G$ .

For any linear function  $f$ , there is a *trivial implementation* gadget which requires no random gates and consists in applying the function independently to each share:  $y_{i,*} = f(x_{i,*})$  for  $i = 0, \dots, d-1$ .<sup>3</sup>

In this article, we are primarily interested in composing gadgets, that is, connecting gadgets together to build more complex gadgets.

**Definition 1 (Gadget composition).** *A gadget composition  $G$  over  $d$  shares is a directed acyclic graph (DAG) whose vertices are composing gadgets (which are gadgets over  $d$  shares) or inputs/outputs, and edges are connections between those gadgets. For each composing gadget, there is a one-to-one mapping between its  $m$  inputs and the incoming edges of the associated vertex. Furthermore, each outgoing edge is associated to an output of the gadget (there can be multiple edges associated to the same output). Output vertices (resp., input vertices) have one (resp., zero) incoming edge and zero (resp., any number of) outgoing edge(s).*

A gadget composition can be instantiated by mapping each vertex to the corresponding gadget or  $d$  inputs/outputs, and each edge to  $d$  wires (which connect the composing gadgets). The inputs and outputs of the composing gadgets are erased in the instantiation process. We use the term *composite gadget* to refer to the instantiation of a gadget composition.

<sup>3</sup> This can easily be adapted to affine functions:  $f$  is applied to the first share, and the linear function  $f - f(0)$  is applied to the other shares.

## 2.2 Probing model and security definitions

In the  $t$ -probing model, an adversary can choose a set  $P$  of  $t$  probes, which are wires in the target circuit. It has then access to the values carried by each of the chosen wires. A gadget  $G$  is  $t$ -probing secure if the output of any  $t$ -probing adversary is independent of the sensitive variables  $(x_0, \dots, x_{m-1})$  when all the input sharings are fresh. The parameter  $t$  is known as the security order. In the best situation, which is often studied in the literature and which we target,  $t = d - 1$  (e.g., [33,6] in software, [13,23] in hardware), but this is not always the case:  $t$  can also be smaller than  $d - 1$  (e.g., for composability reasons [25], or in order to mitigate physical defaults such as glitches [30]). The trivial implementation of any linear function with  $d$  shares is always  $d - 1$ -probing secure.

On important limitation of  $t$ -probing security is that it is not sufficient to ensure composability: the connection of  $t$ -probing secure gadgets is not necessarily  $t$ -probing secure [17]. This has lead Barthe et al. to introduce stronger notions of security that enable composability in [4]. In order to define them, we use the simulability framework put forward by Belaïd et al. in [6], illustrated in Figure 1. Intuitively, a set of probes is simulatable knowing some of the input shares if there exists a simulator that (knowing the said input shares) can generate simulated probes that have the same statistical distribution as the true probes. Note that the notion of  $(\mathcal{I}, \mathcal{O})$ -Non-Interference introduced in [4] is equivalent.

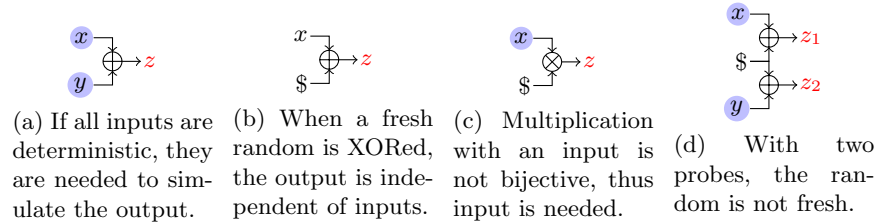


Fig. 1: Simulatability examples. All outputs (in red) are probed. Inputs needed for simulation are circled in blue.

**Definition 2 (Simulatability).** Let  $P = \{p_1, \dots, p_l\}$  be a set of  $l$  probes of a gadget  $C$  and  $C_P$  the tuple of values of the probes for an execution of  $C$ . Let  $I = \{(i_1, j_1), \dots, (i_k, j_k)\} \subset \{0, \dots, d-1\} \times \{0, \dots, m-1\}$  be a set of input wires of  $C$ . A simulator is a randomized function  $\mathcal{S} : \mathbb{F}_q^k \rightarrow \mathbb{F}_q^l$ . The set of probes  $P$  can be simulated with the set of input wires  $I$  if and only if there exists a simulator  $\mathcal{S}$  such that for any inputs  $x_{*,*}$ , the distributions  $C_P(x_{*,*})$  and  $\mathcal{S}(x_{i_1, j_1}, \dots, x_{i_k, j_k})$  are equal, where the probability is over the random coins in  $C$  and  $\mathcal{S}$ .

We can now define Non-Interfering (NI) gadgets and Strong Non-Interfering (SNI) gadgets. We note that for now, those notions are limited to gadgets with

only one output. We take the definitions from [6] and denote probes on shares of a gadget’s outputs as output probes, and probes on any wire of the gadget including inputs and outputs as internal probes.

**Definition 3.** *A gadget with one output sharing is  $t$ -NI if and only if every set of  $t' \leq t$  internal probes can be simulated with at most  $t'$  shares of each input.<sup>4</sup>*

For a gadget with  $d > t$  shares, satisfying  $t$ -NI is strictly stronger than  $t$ -probing security. Indeed, the inputs required by the simulator are independent of the sensitive variables, which implies that the simulated probes are likewise independent of the sensitive variables and the adversarial probes have the same distribution as the simulated probes thanks to indistinguishability.  $t$ -NI is however not a necessary condition for probing security, because it requires indistinguishable simulation for any value of the input shares, not only for any value of the sensitive variables, which sometimes makes the simulation of probing secure gadgets impossible (e.g., in first-order threshold implementations, where non-linear gadgets leverage the input shares in order to reduce the randomness requirements [30]). The trivial implementation of any linear function is NI: the simulator can use the  $x_{i,*}$  values for all the  $i$ ’s for which there is a probe in the evaluation of  $f(x_{i,*})$ .

Next,  $t$ -SNI gadgets guarantee independence between the input and output shares, even in presence of a  $t$ -probing adversary.

**Definition 4.** *A gadget with one output sharing is  $t$ -SNI if and only for if every set  $\mathcal{I}$  of  $t_1$  internal probes and every set  $\mathcal{O}$  of  $t_2$  output probes such that  $t_1+t_2 \leq t$ , the set  $\mathcal{I} \cup \mathcal{O}$  of probes can be simulated with  $t_1$  shares of each input.*

There are many designs of gadgets that implement elementary field operations and are NI or SNI. The most studied ones are NI and SNI field multiplication and SNI refresh gadgets (which implement the identity function in a SNI fashion) [25,6]. Barthe et al. showed that it is possible to build secure composite gadgets based on NI and SNI gadgets [4]:

**Proposition 1.** *A composite gadget  $G$  is  $t$ -NI if all its composing gadgets are  $t$ -NI, and all outputs of composing gadgets (and input vertices) are connected to at most one edge not connected the the input of a SNI refresh gadget.*

This result gives a simple way to securely compose gadgets. However, it usually requires the use of many (expensive) refresh gadgets [22]. One important reason of these overheads, which is also the seed of our following investigations, is that *the trivial implementation of a linear function is not SNI*. It naturally suggests the quest for a security definition that allows simple composition, as Proposition 1, while also leveraging the efficiency and probing security of trivial implementations for linear functions, as an interesting research challenge.

### 2.3 Probe propagation framework

In this section, we describe an intuitive interpretation of the simulatability definition first used in [6], that we next call the probe propagation framework, and discuss its application to the NI and SNI notions.

<sup>4</sup> This definition of NI is equivalent to the definition of tight non-interference in [6].

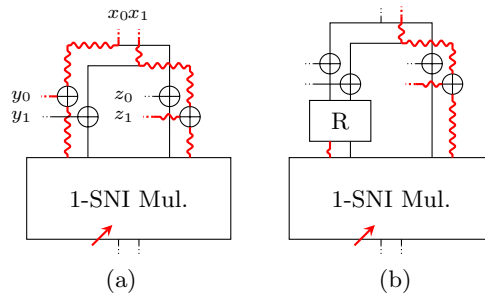


Fig. 2: Implementation of  $(x + y)(x + z)$ . The circuit is made of a SNI multiplication with linear circuits at its inputs (two trivial implementations of the addition), masked with  $d = 2$  shares. The circuits illustrate (a) the limitation of SNI input composability and (b) a simple fix. The arrows indicate the adversarial probes (there is thus one internal probe in the multiplication) and the red snake wires are the propagated probes. The R box is a SNI refresh.

We start with the simple circuit example of Figure 2 which performs a multiplication of dependent values (masked with  $d = 2$  shares). There is one adversarial (internal) probe in the SNI multiplication gadget and we show how to prove (or to fail to prove) that the probe is not an attack in the 1-probing model (i.e., that it is independent of any of the sensitive inputs) by demonstrating that it is possible to simulate it using at most one share of each of the inputs.<sup>5</sup>

According to the SNI definition, it is possible to perfectly simulate the adversarial probe by knowing one share of each of the inputs of the SNI multiplication. Let those required shares be the red snake wires in the circuit (the set of wires shown is an arbitrary example, the shares required by the simulator depend on the position of the adversarial probe). Those wires are called *propagated probes*. The proof then works by observing that if it is possible to simulate the propagated probes, then the adversarial probe can be simulated.

In our example of Figure 2a, we can propagate the probes one step further: a probe at the output of an addition can be simulated with probes on the two inputs of the addition. But then, we obtain four propagated probes at the input of the circuit, which means all the shares of one of the inputs. As a result, we cannot prove that the circuit is probing secure.

In order to circumvent this impossibility, the circuit of Figure 2b (which has the same functionality as the circuit of Figure 2a), implements a simple fix: there is a SNI refresh gadget on one of the inputs of the multiplication gadget. The propagated probe at the output of the refresh gadget can then be simulated using no input of the gadget (thanks to the SNI property), which makes the

<sup>5</sup> Note that this does not prove that the circuit is 1-probing-secure. Proving the probing security would require to analyze all the possible sets of probes. More efficient ways of making such proofs are discussed in the following sections.

circuit secure against this probe. As a result, this composite gadget is 1-NI (and thus 1-probing secure) thanks to Proposition 1.

Summarizing, the main idea of the probe propagation framework is to prove the security of an implementation by replacing the adversarial probes with propagated probes that can be used to simulate the adversarial probes, and by iterating the process until the propagated probes are all at the inputs of the circuit. The conclusion is then easy. More precisely, the propagation of probes always happens backwards in the circuit (probes on the outputs of a gadget propagate to probes on the inputs of the gadget), and the definitions of NI and SNI can be expressed with the following set of simple rules.

*Probe propagation rules:*

- For a NI gadget with  $n_o$  probes on its output shares and  $n_i$  probes inside the gadget, there are propagated probes on  $n_o + n_i$  shares of each input.
- For a SNI gadget with  $n_o$  probes on its output shares and  $n_i$  probes inside the gadget, there are propagated probes on  $n_i$  shares of each input. Hence, SNI gadgets (and SNI refreshes) stop the propagation of probes.

There are then three probe propagation conditions to verify, in order to guarantee security against the considered adversarial probes.

*Probe propagation security conditions:*

1. For some parameter  $t < d$ , all the gadgets must satisfy  $t$ -NI (or multiple-output variants discussed later such as PINI or MIMO-SNI).
2. For any connection between gadgets or input sharing (i.e., for any edge in the gadget composition graph), there cannot be propagated probes on more than  $t$  shares (out of the maximum  $d$ ).
3. For all SNI gadgets, the following must hold:  $n_i + n_o \leq t$  ( $n_i$  and  $n_o$  are the number of internal and output probes, respectively).<sup>6</sup>

### 3 Trivial composition & PINI

In this section, we introduce a new definition of Probe Isolating Non-Interference (PINI) which is directly satisfied by the trivial implementation of any linear function, and enjoys a simple and practical composition property: any composite gadget whose composing gadgets are all PINI is itself PINI.

We then show that this new definition allows us to build a trivial masking compiler that only requires a PINI implementation for each of the non-linear gates of interest. This compiler instantiates the given PINI non-linear gadgets, trivial implementations of the linear functions, and then makes the appropriate

---

<sup>6</sup> No such restriction is needed for NI gadget since it is redundant: when this condition is violated, then the second probe propagation condition is also violated for the inputs of the gadget (thanks to the probe propagation rule).



connections. In addition to simplicity, this technique is cost-efficient, since it uses no refresh gadgets. In particular, for bit-level implementations, this compiler only needs a PINI multiplication gadget (i.e., an AND gate) for which we provide efficient instances in the next section. Another interest of the PINI definition is that it applies (and its properties apply) easily to gadgets with multiple outputs, which is not the case for NI/SNI, as will be discussed in Section 5.1.

### 3.1 Intuition and probe propagation framework

The main idea behind the PINI definition is to take into account not the number of probes (or of required inputs for simulation) as in the NI/SNI definitions, but instead their position (i.e., the shares' indexes). The whole circuit can then be cut into  $d$  circuit shares that are not interconnected, except for non-linear gadgets. If we neglect those gadgets, the circuit is  $t$ -probing secure (for  $t < d$ ): the adversary can only probe  $t$  of the circuit shares, hence it has no information about at least  $d - t \geq 1$  circuit shares, which contains at least one share of each input. PINI gadgets behave in the probing model as if they had no connection between circuit shares (i.e., they can be simulated as such), which allows to implement non-linear functions while keeping the previous circuit sharing intuition.<sup>7</sup>

In the probe propagation framework, probes propagate through PINI gadgets in a way that respects the isolation of the circuit shares. Output probes propagate to all input shares with the same share index (i.e., inside the same circuit share). Internal probes in PINI gadgets are more subtle since they cannot be trivially associated to a circuit share, because there is no circuit share isolation inside (non-linear) PINI gadgets (the isolation is only simulated). However, we can let those probes carry the same intuition as the output probes: each internal (adversarial) probe gives knowledge of at most one circuit share to the adversary. This preserves the feature that the adversary has knowledge of at most  $t$  of the  $d$  circuit shares, which we formalize with a third *probe propagation rule*.

- Each output probe on a PINI gadget propagates to all the input shares that are in the same circuit share as the output probe. Each internal probe propagates to all the input shares that are in one additional circuit share (this circuit share may depend on the position of the probes).

No new probe propagation security condition is needed: if there are too many probes (internal and output), they propagate to the inputs (thanks to the previous rule), and violate the second security condition.

The way PINI works is illustrated in Figure 3, which takes the case discussed in the previous section (where a refresh was needed to prove security), and a new case not handled by (S)NI definitions: a gadget with multiple outputs.

---

<sup>7</sup> To some extent the probe isolation idea can be connected to the Domain-Oriented Masking described in [23]. However, that approach focuses on the local security of gadgets without discussing composability. The probe isolation principle is also implicitly used in the seminal work of Ishai et al. [25], but they use  $2d + 1$  shares.

In Figure 3a, there is one internal probe which propagates to one share of each input of the multiplication as it is the case for (S)NI multiplications. However, the propagated probes have the same share index (they are in the same circuit share), hence probe propagation through the linear operation does not violate the second probe propagation security condition.

In Figure 3b, the two propagated probes at the output of the S-box have the same share index, hence they propagate to only one circuit share.

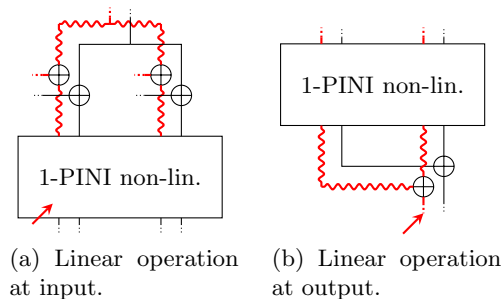


Fig. 3: Examples of PINI circuits masked with  $d = 2$  shares. The rectangle gadget implements a non-linear function. The arrows indicate the adversarial probes and the red snake wires are the propagated probes.

### 3.2 Formalization: definition and properties

We now give the formal definition of PINI, and prove security and composability properties. For this purpose, we first show the link between the notion of circuit share and the notations of Section 2. Namely, for a gadget with inputs  $x_{i,j}$  and outputs  $y_{i,j}$ , all the inputs and outputs in the circuit share  $i$  are  $x_{i,*}$  and  $y_{i,*}$ .

In the following definition, the set  $A$  is the set of share indexes (i.e., the circuit shares) that are probed through output probes, and  $B$  is the set of circuit shares requested to simulate the internal probes. The set of shares required to simulate all the probes is thus  $A \cup B$ .

**Definition 5 (Probe-Isolating Non-Interference).** *Let  $G$  be a gadget over  $d$  shares and  $P$  a set of  $t_1$  probes on wires of  $G$  (called internal probes). Let  $A$  be a set of  $t_2$  share indexes.  $G$  is  $t$ -Probe-Isolating Non-Interfering ( $t$ -PINI) iff for all  $P$  and  $A$  such that  $t_1 + t_2 \leq t$ , there exists a set  $B$  of at most  $t_1$  share indexes such that probes on the set of wires  $P \cup y_{A,*}^G$  can be simulated with the wires  $x_{A \cup B,*}^G$ .*

The following proposition shows that any PINI gadget is probing secure.

**Proposition 2.** *Any  $t$ -PINI gadget (with a number of shares  $d > t$ ) is  $t$ -probing secure.*

*Proof.* Any set of at most  $t$  probes can be simulated with at most  $t$  shares of each input. Thanks to the independent input encodings, this set of input shares is independent from all the sensitive input values.  $\square$

We now look at composability properties for PINI gadgets.

**Proposition 3 (PINI composability).** *Any composite gadget made of  $t$ -PINI composing gadgets is  $t$ -PINI.*

*Proof.* We build a PINI simulator that takes as input a set of probes  $P$  and a set of share indexes of probed output shares  $A$ . Without loss of generality, we assume that there is not probe on wires connecting composing gadgets (hence there are only output probes and probes inside composing gadgets), since such probes can be considered to be inside one of the gadgets connected to the wire.

Let us order gadgets from output to input (in reverse topological sort order for the gadget composition DAG) and denote them  $G_1, \dots, G_l$ . We build iteratively the sets  $A_i$  which are sets of circuit shares needed to simulate probes inside gadgets  $G_1, \dots, G_{i-1}$  and output probes.

Let  $P_i$  be the set of probes inside the gadget  $G_i$ . Let  $A_1 = A$  be the set of share indexes of the output probes. By induction, let  $A_{i+1} = A_i \cup B_i$ , where  $B_i$  is obtained by running the PINI simulator for gadget  $G_i$ , with set of probes  $P_i$  and output share indexes set  $A_i$  (this is possible since  $|P_i| + |A_i| \leq t$ ). Let  $B = A_{l+1} \setminus A$  be the set of input share indexes required to the oracle for the simulation (in addition to the shares with indexes in  $A$ , always given). Simulation is performed from  $G_l$  to  $G_1$ : the PINI simulator for  $G_i$  simulates  $y_{A_i, *}^{G_i}$  and has access to  $x_{A_{i+1}, *}^{G_i}$  (obtained from oracle and/or other PINI simulators). We now have to show that the simulator respects the cardinality constraints of PINI, that is:  $|B| \leq t_1 = \sum_{i=1}^l |P_i|$ . This is obtained by induction on the inequality  $|A_{i+1}| \leq |A_i| + |B_i| \leq |A_i| + |P_i|$  and by the observation that  $A \subset A_{l+1}$ .  $\square$

We finally prove that the trivial implementations of linear gadgets are PINI.

**Proposition 4.** *The trivial implementation of a linear function is  $t$ -PINI for any  $t$ .*

*Proof.* The simulator requires access to inputs from all the circuit shares in which there is a probe. Simulation is then trivial.  $\square$

## 4 PINI multiplication gadgets

Given the results in the previous section, the main remaining challenge to leverage the trivial composition of PINI gadgets is to instantiate PINI field multiplications. We next propose two solutions for this purpose. The first one is based on the composition of existing (SNI) gadgets while the second one is a new design.

## 4.1 Goudarzi and Rivain’s double-SNI gadget

This gadget is based on an implementation strategy used in [22], where only SNI multiplications are used, and one input of every multiplication gadget is refreshed in a SNI manner. Goudarzi and Rivain claim (without proof) that linear operations can then be implemented in the trivial way, leading to a trivial composition using double-SNI multiplications (Algorithm 1) for every non-linear gadget. We next show that the AES implementation using this strategy is secure by showing that the double-SNI multiplication gadget is PINI.

---

**Algorithm 1** Double-SNI multiplication gadget over  $d$  shares.  $\mathsf{R}_d$  and  $\mathsf{Mul}_d$  are SNI  $t$ -refresh and  $t$ -SNI multiplication gadgets over  $d$  shares, respectively.

---

**Require:** Shared factors  $(a_i)_i, (b_i)_i \in \mathbb{F}_q^d$  such that  $\sum_i a_i = a$  and  $\sum_i b_i = b$ .

**Ensure:** Output  $(c_i)_i \in \mathbb{F}_q^d$  such that  $\sum_i c_i = a \cdot b$ .

$(x_i)_i \leftarrow \mathsf{R}_d((a_i)_i);$   
 $(c_i)_i \leftarrow \mathsf{Mul}_d((x_i)_i, (b_i)_i);$

---

Intuitively, this result comes from the fact that each probe inside the gadget of Algorithm 1 propagates to at most one share of one of the inputs, and output probes do not propagate, which implies PINI.<sup>8</sup>

**Proposition 5.** *Any double- $t$ -SNI multiplication gadget is  $t$ -PINI.*

*Proof.* Let us assume that there is a set  $P_1$  of probes on the output of the gadget (i.e., probes on  $(c_i)_i$ ), a set  $P_2$  of probes inside the SNI multiplication gadget and a set  $P_3$  of probes inside the SNI refresh gadget, such that  $|P_1| + |P_2| + |P_3| \leq t$ . Any probe on  $(x_i)_i$  is counted as a probe inside the SNI multiplication gadget.

Using the simulator for the SNI multiplication gadget, we can simulate the sets of probes  $P_1$  and  $P_2$  using a set  $P_4$  of shares of  $(x_i)_i$  and a set  $P_5$  of shares of  $(b_i)_i$ , such that  $|P_4| \leq |P_2|$  and  $|P_5| \leq |P_2|$ . Then, using the simulator for the SNI refresh gadget, we can simulate the sets of probes  $P_3$  and  $P_4$  using a set  $P_6$  of shares of  $(a_i)_i$  such that  $|P_6| \leq |P_3|$  (since  $|P_3| + |P_4| \leq t$ ). Overall, we can simulate all the adversarial probes using the sets of input shares  $P_5$  and  $P_6$ .

Since  $|P_2| + |P_3|$  is the number of internal probes and  $|P_5| + |P_6| \leq |P_2| + |P_3|$ , the PINI simulator can request knowledge of all the input shares whose index is the one of a share in  $P_5$  or  $P_6$ . Hence, the PINI simulator knows the values of the shares in  $P_5$  and  $P_6$  and can use the two SNI simulators to complete the simulation.  $\square$

## 4.2 PINI<sub>1</sub>: a more efficient field multiplication

We next introduce in Algorithm 2 a new PINI multiplication gadget for  $d$  shares. It is a variation of the ISW multiplication [25] and has the same randomness

---

<sup>8</sup> Actually, the double-SNI multiplication enjoys the stronger MIMO-SNI property defined in Section 5.1 (or, equivalently for single-output gadgets, MI-SNI).

requirement (i.e.,  $d(d-1)/2$  field elements). Compared to the double-SNI multiplication gadget, it is thus more efficient.

We defer the proof that this algorithm is  $d-1$ -PINI to Appendix A, but we give here the intuition behind it. The only probes that violate the PINI definition (i.e., probes that require knowledge of inputs from more than one circuit share to be simulated) in the ISW multiplication are partial products  $a_i b_j$ , which are intermediate values of the computation of  $z_{ij} = r_{ij} + a_i b_j$ . This issue can be solved by using a fresh random element  $r'_{ij}$  and computing the same result as  $z_{ij} = (r_{ij} + a_i r'_{ij}) + a_i (r'_{ij} + b_j)$ . None of the intermediate values in this computation require the knowledge of both  $a_i$  and  $b_j$  to be simulated. Such a technique requires more random field elements, but it can be optimized by using only the  $r_{ij}$ 's and not fresh  $r'_{ij}$ 's, since the computation  $z_{ij} = (1 + a_i)r_{ij} + a_i(r_{ij} + b_j)$  enjoys the same security and correctness as the previous expression.

---

**Algorithm 2** PINI<sub>1</sub> multiplication gadget over  $d$  shares

---

**Require:** Factors  $a, b \in \mathbb{F}_q^d$  such that  $\sum_i a_i = a$  and  $\sum_i b_i = b$

**Ensure:** Output  $(c_i)_i \in \mathbb{F}_q^d$  such that  $\sum_i c_i = a \cdot b$ .

```

for  $i = 0$  to  $d - 1$  do
  for  $j = i + 1$  to  $d - 1$  do
     $r_{ij} \xleftarrow{\$} \mathbb{F}_q$ ;
     $r_{ji} \leftarrow r_{ij}$ ;
  for  $i = 0$  to  $d - 1$  do
    for  $j = 0$  to  $d - 1$  do
      if  $i \neq j$  then
         $s_{ij} \leftarrow b_j + r_{ij}$ ;
         $p_{ij}^0 \leftarrow (a_i + 1) \cdot r_{ij}$ ;
         $p_{ij}^1 \leftarrow a_i \cdot s_{ij}$ ;
         $z_{ij} \leftarrow p_{ij}^0 + p_{ij}^1$ ;
  Ensure:  $z_{ij} = r_{ij} + a_i \cdot b_j$ 
  for  $i = 0$  to  $d - 1$  do
     $c_i \leftarrow a_i \cdot b_i + \sum_{j=0, j \neq i}^{d-1} z_{ij}$ 

```

---

This algorithm can be adapted to require only linear memory by re-ordering the operations: for each generated  $r_{ij}$ , compute directly  $z_{ij}$  and  $z_{ji}$ , and update  $c_i$  and  $c_j$ . The intermediate values depending on  $r_{ij}$  can then be dropped. This does not change the set of possible probes, hence the security proof is still valid.

## 5 Composition of larger gadgets

In this section, we aim to study a representative example of larger gadget to estimate the cost of using our trivial composition for small (PINI) gadgets compared to other state-of-the-art solutions. We take the bit-level (i.e., manipulated elements are in  $\mathbb{F}_2$ , not  $\mathbb{F}_{256}$ ) S-box of Boyar, Matthews and Peralta [9] that is used in [22] for this purpose. In order to obtain fair comparisons, we then follow

the optimized approach to composition introduced by Belaïd et al. [6], which has been shown to provide better performances than the straightforward application of Proposition 1. Doing so, we face two additional challenges.

First, in order to exploit optimized S-boxes, we ideally need that such S-boxes lead to a secure circuit when composed with the trivial implementation of the AES linear layer. We observe that the SNI definition is not sufficient to reach this goal, and we introduce a natural extension of SNI that allows such a composition. It essentially extends SNI to a multiple-input and multiple-output setting (hence the name MIMO-SNI for the proposed extension).

Next, we observe that the optimization of a complex circuit such as the bit-level S-box in [9] is computationally intensive, and cannot be exhaustively analyzed like the  $\mathbb{F}_{256}$  S-box investigated in [6]. So we propose an automated heuristic based on linear programming to mitigate this limitation.

### 5.1 Multiple-Input-Multiple-Output SNI

*Multiple-Output SNI (MO-SNI).* A first issue with the SNI definition is its specialization to single-output gadgets (whereas a bit-level AES S-box has eight output bits). Therefore, we need to extend this definition to multiple-output gadgets. Two natural extensions can be considered for this purpose: (i) the gadget tolerates at most a total of  $t$  probes for all the outputs, or (ii) it tolerates up to  $t$  probes for each of the outputs.

The next example shows that the first option is not sufficient to ensure composability with linear layers. In order to simplify the discussion, we take a simple case of 2-bit non-linear functions (i.e., each gadget has two inputs and two outputs) masked at order  $t = 1$  (i.e.,  $d = 2$ ), but our reasoning applies to any gadget (e.g., the 8-bit S-boxes of the AES) and any order.

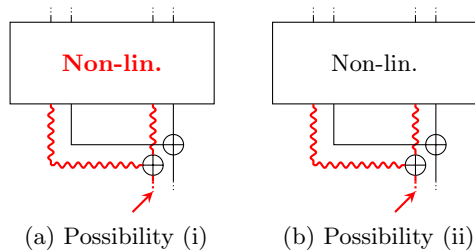


Fig. 4: Two definitions for SNI with multiple output gadgets: only one composes well with a linear layer. Example for  $d = 2$  and 2 bit gadget.

We consider a linear operation between two outputs of a non-linear gadget (depicted in Figure 4). The adversary has  $t$  probes on one output of the linear operation. The probes propagate to  $2t$  probes on the output of the non-linear

gadget. Hence, the extension (i) of the SNI definition is thus not sufficient, and we prefer extension (ii), which we next call MO-SNI.

*Multiple-Input SNI (MI-SNI).* A already discussed in Section 2.3, in general a non-linear SNI gadget cannot have a linear layer at its input without refreshing some of these inputs. Therefore, we introduce a stronger definition of MI-SNI: the simulator must be able to simulate the probes using one input share per probe, while it is one input share per input and per probe for (S)NI.

*Multiple-Input-Multiple-Output SNI (MIMO-SNI).* The combination of those two definitions gives the MIMO-SNI notion, that we first introduce in the probe propagation framework, and then define formally. There is a new probe propagation rule for MIMO-SNI gadgets (to cover the MI-SNI part of the definition):

- For a MIMO-SNI gadget with at most  $n_o$  probes on each output and  $n_i$  internal probes, there is a total of  $n_i$  propagated probes on the input shares.

There is then a fourth probe propagation security condition, which is the condition for SNI gadgets adapted to MO-SNI:

4. For any MIMO-SNI gadget with at most  $n_o$  probes on each output and  $n_i$  internal probes, the following must hold:  $n_i + n_o \leq t$ .

**Definition 6 (MIMO-SNI).** *Let  $O_i$  be a set of share indices for  $i = 0, \dots, n-1$ . A gadget is  $t$ -MIMO-SNI if and only if for any set  $\mathcal{I}$  of  $t_1$  internal probes and any sets  $O_i$  such that there exists a  $t_2$  that satisfies  $t_1 + t_2 \leq t$  and  $|O_i| \leq t_2$  for  $i = 0, \dots, n-1$ , the set of probes  $\mathcal{I} \cup y_{O_0,0} \cup \dots \cup y_{O_{n-1},n-1}$  can be simulated with at most  $t_1$  input shares.*

This definition is very strong, in fact it is strictly stronger than PINI.

**Proposition 6.** *Any  $t$ -MIMO-SNI gadget is  $t$ -PINI.*

*Proof.* The MIMO-SNI simulator can be used as a PINI simulator, with the set of share indices  $B$  sent to the PINI oracle being made of the indices of the  $t_1$  input shares required by the MIMO-SNI simulator.  $\square$

This Proposition shows that MIMO-SNI benefits from the same composability properties as PINI: a MIMO-SNI S-box can thus be trivially composed with linear layers. Despite it is stronger than PINI (which is already sufficient to compose securely), an interesting feature of MIMO-SNI is that this notion can be obtained by combining NI and SNI elementary gadgets in an optimized composition similar to the one proposed in [6] (see next).

*Additional remark.* The previous definitions are quite connected to the recent work of Belaïd et al. on Tight Private Circuits (TPC) [7], in which the AES S-box is probing secure and its outputs are all outputs of SNI gadgets. This property is similar to MO-SNI: it guarantees independence between outputs,

and between outputs and inputs; but it is weaker than MO-SNI: it does not require simulatability. The authors show that despite the weaker definition, it can be securely composed with linear layers to build a  $t$ -probing secure circuit. This weaker requirement enables more efficient implementations than [22]: S-boxes only requires 32 SNI multiplication gadgets, and no refresh gadgets in this case. However, it does not enable trivial composition. First, this composability result is limited to full linear layers (which results in a need to refresh the key scheduling even if it is linear). Second, the analysis of the S-box itself is not trivial (it requires circuit-specific analyzes to prove its security).

## 5.2 Building large MIMO-SNI gadgets from small (S)NI gadgets

In order to fairly assess the interest of the PINI framework with respect to state-of-the-art solutions, and to allow sound performance comparisons in the next section, we now tackle the problem of building a large MIMO-SNI gadget by composition of smaller (S)NI gadgets. For a given functionality, we try to minimize the amount of SNI gadgets in the implementation in order to reduce its complexity. In other words, we investigate the optimized composition approach mentioned in introduction as a natural competitor to the trivial one.

For this purpose, we first show how to express this optimization based on the properties of a graph describing the computations to perform. We then apply this optimization to the AES S-box in  $\mathbb{F}_{256}$  (confirming the results in [6]) and to the bit-level AES S-box of Boyar, Matthews and Peralta [9], bringing significant improvements over the double-SNI strategy in [22].

**Connecting composability to computation graph properties** We introduce a new computation graph model based on the gadget composition DAG in order to explicitly put into evidence the cases where a sharing is used multiple times. The computation graph restricts the gadget composition DAG by forbidding the connection of more than one edge to an output of a gadget or an input gate. To handle the forbidden cases, we add a new  $\text{Split}_n$  gadget which has one input,  $n$  identical outputs and performs no operations (it only connects input to outputs). For simplicity, we assume that all the composing gadget are NI operation gadgets with one output (in practice mostly additions and multiplications), SNI refresh gadgets (with one input and one output) or  $\text{Split}_n$  gadgets. SNI gadgets are thus modeled as NI ones followed by a SNI refresh. Given a computation graph resulting from our optimization, an implementer can then replace NI multiplications followed by a SNI refresh by (sometimes more efficient) SNI multiplications. This modeling is without loss of generality since it is equivalent from the probing model viewpoint and the respective costs of the different gadgets of a private circuit are parameters of the optimizations.

Using the link between computation graph and the probe propagation framework and capitalizing on the fact that SNI refresh gadgets stop the propagation of probes, we can simply remove them (and their incident edges) from the graph to build a *simplified graph*. The probes inside the refresh gadgets can be reported



to gadgets connected to their input, hence the simplified graph is equivalent to the original graph regarding security in the probing model.

**Definition 7 (Simplified computation graph).** *The simplification of the computation graph  $G$  is the graph that is obtained from  $G$  by removing all SNI refresh vertices and their incident edges.*

Let us now analyze under which condition a simplified computation graph represents a NI composite gadget, leveraging the probe propagation framework. Since we only have NI gadgets in a simplified computation graph, the only case where the probe propagation security conditions are not respected is when there are more than  $t$  propagated probes on one share. Since there are at most  $t$  adversarial probes, violation of the security condition means that a single adversarial probe is duplicated, that is, it propagates to the same share through to different paths. We can thus derive a sufficient NI condition for simplified computation graphs: no probe should propagate backwards from a node to another one through two different paths. In other words, for any pair of vertices there should be at most one (directed) path between them. It can be seen that this condition cannot be made weaker while guaranteeing security for any NI composing gadgets: if probes can propagate backwards through two paths from a node A to a node B and if the adversary has  $t$  probes on the output of A, up to  $2t$  shares of the output of B could be required to simulate.

We now formalize this security condition with the following propositions (which generalize the proof that the AES inversion is  $t$ -SNI in [6]). For this purpose, we first define a property for composite gadgets and their simplified computation graph, which specifies the NI condition of the previous paragraphs.

**Definition 8 (Single-Path-NI-Built gadget).** *A composite gadget  $G$  is Single-Path-NI-Built (SP-NIB) if it is implemented with only NI gadgets and SNI refreshes, and if for any pair of vertices  $u, v$  in the corresponding simplified computation graph there exists at most one path from  $u$  to  $v$ .*

**Proposition 7.** *Let  $G$  be a composite gadget. If  $G$  is SP-NIB (as per Definition 8), then it is  $t$ -NI.*

*Proof.* For each edge  $i$  in the computation graph, there is a number of adversarial probes  $a_i$ , a number of propagated probes  $p_i$  and a total number of probes  $s_i$ . The sum of the  $a_i$ 's is at most  $t$ . For all  $i$ ,  $s_i = a_i + p_i$ . For each edge, the number of propagated probes is:

- 0 if the node at the end of the edge is a refresh or an output gate;
- the sum of the total number of probes of the outgoing edges of the vertex at the end of the edge otherwise (i.e., for split or NI gadgets).

The probes inside a NI gadget are not considered since they can equivalently be replaced with probes on output shares of the gadget. Furthermore, the probes on output gates are modeled as adversarial probes on the edges connected to those gates.

If for each input edge  $i$  (i.e., an edge connected to an input gate), a simulator knows  $s_i$  well-chosen shares, then it can simulate all the probes of the adversary by using the simulator for each gadget in order to get the required intermediate values.

We now prove that the SP-NIB hypothesis implies that for all input edges  $i$ ,  $s_i \leq t$ . This proves that the gadget is NI thanks to the previous observation.

We use a small lemma for this purpose: for all edges  $i$ ,  $p_i = \sum_j \alpha_{ij} a_j$  where  $\alpha_{ij}$  is the number of paths from the output node of  $i$  to the input node of  $j$  in the simplified computation graph. This can be proven by backwards induction on the graph: if all the outgoing edges of a node satisfy this property, it is also satisfied for all the incoming edges to this node if the node is a refresh, split or NI operation. As a base case, this is trivially satisfied for output edges (i.e., edges connected to an output gate).

To conclude the main proof, we observe that the main hypothesis implies that  $\alpha_{ij} \leq 1$  for all pairs of edges  $(i, j)$ , hence  $s_i \leq \sum_j a_j \leq t$ .  $\square$

We now give similar computation graph properties that guarantee SNI and MIMO-SNI. Intuitively, a composite gadget is SNI if no probes can propagate from any output to any input (i.e., there must be no path from an input to an output).

**Proposition 8.** *Let  $G$  be a composite gadget. If the gadget is SP-NIB and if for any input node  $u$  and any output node  $v$ , there is no path from  $u$  to  $v$ , then the gadget is  $t$ -SNI.*

*Proof.* Looking at the proof of Proposition 7, we observe that, under the current stronger hypothesis,  $\alpha_{ij} = 0$  for all input edges  $i$  and output edges  $j$ . Hence for all input edges  $i$ ,  $s_i \leq t_1$  where  $t_1$  is the number of internal probes.  $\square$

For MIMO-SNI, we additionally require that no probe can propagate to two inputs simultaneously (otherwise,  $t$  internal probes could propagate into strictly more than  $t$  input probes). Furthermore, there must be no composing gadget to which probes could propagate from two different outputs: otherwise, up to  $t$  probes could propagate from each output, resulting into up to  $2t$  propagated probes on the output of the composing gadget.

**Proposition 9.** *A composite gadget  $G$  is  $t$ -MIMO-SNI if it satisfies the three following conditions. (i)  $G$  is SP-NIB. (ii) For any pair of output nodes  $u_1, u_2$  there is no node  $v$  such that there is a path from  $v$  to  $u_1$  and a path from  $v$  to  $u_2$ . (iii) For any pair of input nodes  $u_1, u_2$  there is no node  $v$  such that there is a path from  $u_1$  to  $v$  and a path from  $u_2$  to  $v$ .*

*Proof.* We first have to prove that for all edges  $i$ ,  $s_i \leq t$ . Under the current hypothesis, the lemma from the proof of Proposition 7 is strengthened: for any edge  $i$ ,  $\sum_{j \in O_e} \alpha_{ij} \leq 1$  where  $O_e$  is the set of output edges. Furthermore, for any  $i, j$ ,  $\alpha_{ij} \leq 1$ . This implies that  $s_i \leq t_1 + t_2 \leq t$ , taking the definitions of  $t_1$  and  $t_2$  from the MIMO-SNI definition.

Second, we have to prove that  $\sum_{i \in I_e} s_i \leq t_1$  where  $I_e$  is the set of input edges. We know that for all  $j$ ,  $\sum_{i \in I_e} \alpha_{ij} \leq 1$  and for output edges  $j$ ,  $\sum_{i \in I_e} \alpha_{ij} = 0$ . Hence  $\sum_{i \in I_e} s_i \leq \sum_{j \notin O_e} a_j = t_1$ .  $\square$

**Optimizing the AES S-box in  $\mathbb{F}_{256}$**  Using the previous graph formalization, we built a tool<sup>9</sup> that checks if a circuit is (MIMO-)(S)NI. If we want to build a SNI S-box with the multiplication chain from [6], there are 16 wires on which we could insert a refresh. This number is sufficiently small to make an exhaustive search, which confirms the result of [6] and shows that it is the only solution with only three refresh elements (up to the permutation of refresh gadgets with the  $(\cdot)^{2^\alpha}$  power gadgets): two refresh gadgets and one SNI multiplication.<sup>10</sup> It also shows that two refresh gadgets is the minimum possible, even with all multiplications implemented as SNI gadgets.

**Optimizing the bit-level AES S-box of Boyar et al.** We now optimize the implementation of a bit-level AES S-box. We take the logic circuit by Boyar et al. in [9] and search, starting from an implementation with NI gadgets, where it is required to add SNI refresh elements to get a MIMO-SNI implementation.

Due to the large size of the non-linear part of the AES S-Box (more than 124 wires), it is not possible to apply exhaustive search as done for the S-Box in  $\mathbb{F}_{256}$ . We instead re-write this problem as a mixed integer linear optimization problem, for which there exists solvers with efficient heuristics. The formulation of the optimization problem is explained in Appendix B.

We ran the optimization with a uniform cost for all edges, which is sound if we assume that the cost of replacing a NI operation with a SNI one is the same as adding a SNI refresh. This assumption is valid for state-of-the-art gadgets at very high order (as confirmed Section 7). The optimization solver gave a solution with 41 SNI elements and a lower bound of 34 SNI elements (after two hours of running time). In comparison, the implementation of Goudarzi and Rivain in [22] uses two SNI elements per AND gate, totaling 64 SNI elements.

The same technique can be applied to other S-boxes, which gives 7 SNI elements for the Noekeon S-box (4 bit), 8 SNI elements for the Present S-box (4 bit) and 17 SNI elements for the Fantomas S-box (8 bit). Thanks to the lower gate count of those S-boxes, the solver is able to find an optimal solution.

## 6 Related work

In this section, we compare our new composition strategies to other strategies from the literature.

First of all, the seminal masking transformation of Ishai et al. [25] is composable. The composition proof is based on the observation that if a share index

<sup>9</sup> The checking and optimization tools will be open-sourced before publication.

<sup>10</sup> [6] actually mentions two SNI multiplications are needed, but it was observed by Jean-Sébastien Coron that one is enough during Adrian Thillard’s PhD defense.

$i$  belongs to the set of shares  $I$  required for simulation (for both inputs of a multiplication gadget), then the output with share index  $i$  can be simulated. This constraint also appears in the PINI definition. However, Ishai et al. require a number of shares at least  $d = 2t + 1$  (for security at order  $t$ ). The additional requirements in the PINI definition improve this bound to  $d = t + 1$ .

All the composition schemes we discuss next use  $d = t + 1$  masking. The strategy of Goudarzi and Rivain [22], for instance, is based on trivial composition but uses a more complex multiplication gadget, namely the double-SNI construction, where both the multiplication and the refresh are based on the ISW multiplication gadget. However, they do not prove that the composite circuit is probing secure (see Section 4.1).

A more formal and generic composition framework has been put forward by Barthe et al. in [4] through the NI and SNI definitions. Their main composition result (Proposition 1) is simple, but leads to poor performance (see Section 7): it requires refresh gadgets to protect linear operations. The NI and SNI definitions are used by Belaïd et al. [6] as the basis for building a more efficient  $\mathbb{F}_{256}$  AES S-box circuit. This approach is the basis for the techniques developed in Section 5.

Recently, Belaïd et al. [7] introduced the Tight Private Circuit (TPC) composition strategy. It is based on the use of SNI multiplication gadgets and a careful analysis of the structure of non-linear layers, in order to insert SNI refresh gadgets where needed. This approach allows to greatly reduce the number of refresh gadgets needed for masking the AES compared to previous approaches. It exploits the circuit shares isolation property of the linear operations, as well as the bijectivity of the XOR operations with respect to one of its inputs. One (slight) downside of the TPC strategy is that it is specialized to block ciphers with a given (admittedly very usual) structure. More importantly, it requires some specific optimizations leading to better or worse performances depending on the cipher to protect (see the next section). It also requires to add some refresh gadgets in linear key schedulings (as frequently used in lightweight cryptography).

We note that a completely different approach for composing masked gadget is based direct verification of  $t$ -probing security of the composite circuit using automated tools [3]. Circuit verification is a computationally expensive problem, hence those tools are limited in circuit size and masking order.

The main features of these compositional strategies are summarized in Table 1 .

## 7 Performance comparison

We conclude the paper by briefly discussing the performance aspects of various compositional strategies introduced in the literature (including ours).

For this purpose, we consider state-of-the art gadgets at each order (order-specific and generic constructions) for each property needed: SNI refresh, NI multiplication, SNI multiplication and PINI multiplication (whose costs are given in Appendix C). Namely, the refresh gadgets are taken from [2], the NI multiplications and SNI multiplication come from [25,6,5], and the  $\text{PINI}_1$  multiplication comes from this work (Algorithm 2), while the  $\text{PINI}_2$  multiplication was

|                      | Any circ. | Simplicity | $d = t + 1$ | Triv. Lin. Op. |
|----------------------|-----------|------------|-------------|----------------|
| Verification tool    | ✗         | ✗          | ✓           | (✓)            |
| ISW [25]             | ✓         | ✓          | ✗           | ✓              |
| Only SNI [4]         | ✓         | ✓          | ✓           | ✗              |
| MIMO-SNI (this work) | ✓         | ✗          | ✓           | ✓              |
| TPC [7]              | ✓         | ✗          | ✓           | ✓              |
| Double-SNI [22]      | ✓         | ✓          | ✓           | ✓              |
| PINI (this work)     | ✓         | ✓          | ✓           | ✓              |

Table 1: Characteristics of masking strategies:

- Any circuit: the method works for complex circuits (e.g. by enabling composition of smaller gadgets) and any masking order.
- Simplicity: the masking transformation is a straightforward gate-to-gadget translation without more global analysis.
- $d = t + 1$ : the number of shares is minimal.
- Trivial linear operations: no refresh gadget inside linear layers.

introduced in a follow-up work by Cassiers and Standaert (PINI<sub>2</sub>) [11]. For the construction of a SNI multiplication, we observe that for sufficiently high orders ( $d \geq 12$ ), the multiplication of Belaïd et al. followed by a SNI refresh has a lower cost than the multiplication of Ishai, Sahai and Wagner, which justifies the assumptions made for the optimization in Section 5.2.<sup>11</sup>

Next, we analyze the strategies on various block ciphers. We take the AES with the bit-level implementation of Boyar, Matthews and Peralta masked at various orders as a first realistic case study (since it is the basis for the best-reported masked software performances in [22]). We also consider lightweight block ciphers, which are arguably more relevant for embedded devices where masking is needed and when performance is a constraint: Present (with 80 bit key) and Noekeon (with 4 bit S-boxes), and Fantomas (8 bit S-boxes).

A synthetic evaluation is shown for each compositional strategy and each block cipher in Figures 5, 6, 7 and 8. For this purpose, we use a simple cost model assuming that the generation of one random bit has the same cost as 80 Boolean gate evaluations. This model is admittedly abstract and based on the PRNG performances mentioned in [26]. We note that our conclusions are stable for a wide range of PRNG costs, and refer to follow-up [11] for more empirical confirmations of these comparisons.

The implementations considered use the following approaches: using only SNI gadgets (SNI multiplications and SNI refresh after each linear operation, Proposition 1); Tight Private Circuits (TPC) strategy from [7]; optimized MIMO-SNI S-boxes (see Section 5.2); trivial composition using Double-SNI multiplication

<sup>11</sup> Our optimization can be adapted to take into account the actual costs at lower orders, but since the relative costs differ for every order, finding the optimal implementation would require re-running the optimization at each order.

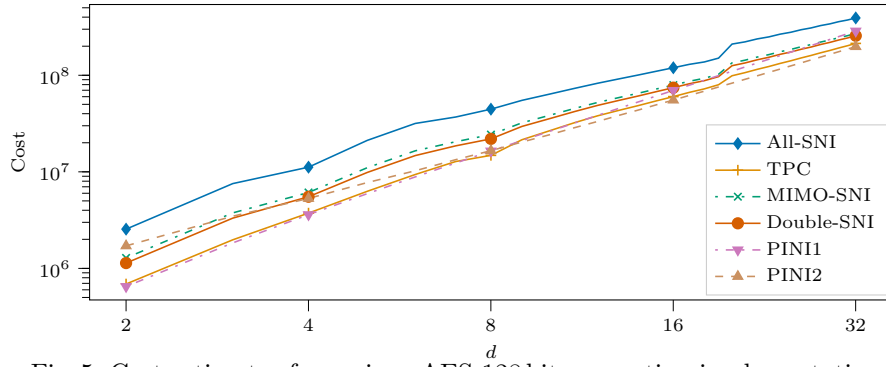


Fig. 5: Cost estimates for various AES 128 bit encryption implementation.

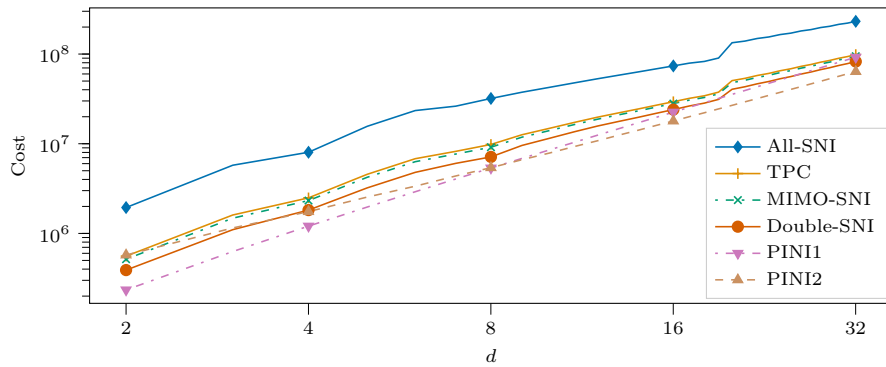


Fig. 6: Cost estimates for various Noekeon encryption implementation.

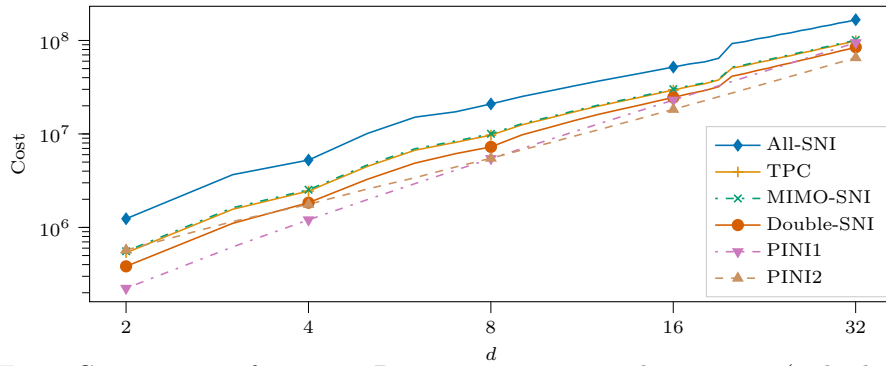


Fig. 7: Cost estimates for various Present encryption implementation (80 bit key).

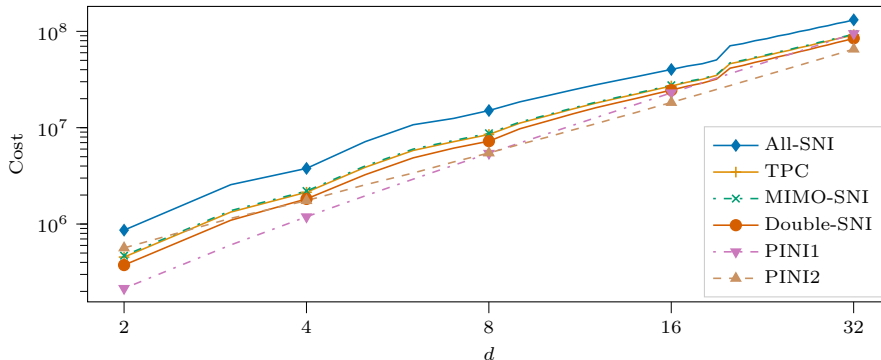


Fig. 8: Cost estimates for various Fantomas encryption implementation.

gadgets [22]; and trivial composition using PINI gadgets. For the PINI strategies, we included an implementation with the  $\text{PINI}_1$  multiplication gadget and another one with the  $\text{PINI}_2$  gadget, which performs better at high orders.

For lightweight ciphers, the PINI strategy performs best (by taking the best amongst  $\text{PINI}_1$  and  $\text{PINI}_2$ ), reducing cost by a factor of roughly 2 compared to the Double-SNI strategy, which comes second in performance.<sup>12</sup>

The AES case is particular: it is a sweet spot for the TPC strategy which then reaches performance similar to PINI. This is due to the large number of AND gates in the AES S-box<sup>13</sup> (32, whereas Fantomas, Noekeon and Present have respectively 11, 4 and 4). This reduces the relative cost of the refreshing of the key schedule (needed for TPC and not for competing strategies). Moreover, contrary to lightweight ciphers, the non-linear part of the AES S-box ends with a full layer of AND gates, avoiding the need of additional refreshing for the TPC strategy (for which the S-boxes must end with a full SNI layer).

These results confirm that PINI is a good strategy to build masked circuits. First, it enjoys the trivial composition property, which leads to simplicity (such as dealing with only one kind of gadget, while e.g., other strategies need rules to decide whether to use NI/SNI multiplications and SNI refresh gadgets). Second, PINI leads to high-performance implementations that reduce by 2 the cost of lightweight block cipher implementations. We believe those results are timely for enabling the comparison of side-channel resistant implementations in the ongoing NIST competition in lightweight cryptography, and are in general relevant to the security and efficiency of any embedded cryptographic implementation, which are important building blocks of many secure systems.

<sup>12</sup> For asymptotic  $d$ , the PINI gadgets are worse than Double-SNI or TPC since they require more arithmetic operations (overhead scaling as  $d^2$ ), which makes the cost of refresh gadgets ( $d \log d$ ) asymptotically negligible. However, this has no significant impact at practical masking orders for bit-level masking. Performance evaluations in larger fields (e.g.  $\mathbb{F}_{256}$ ) where multiplication cost is larger is left to future work.

<sup>13</sup> We use the bit-level implementation from [10], which is the basis for the best-reported masked software performances [22].

Finally, we recall that such performance comparisons (e.g., between the MIMO-SNI, TPC and PINI approaches) are dependent on the cost of the (state-of-the-art) gadgets used. Hence, the search of more optimized such gadgets is an interesting scope for further research, to further refine our understanding of cost-optimized masked implementations.

**Acknowledgments.** Gaëtan Cassiers and François-Xavier Standaert are resp. Research Fellow and and Senior Associate Researcher of the Belgian Fund for Scientific Research (FNRS-F.R.S.). This work has been funded in part by the ERC project 724725.

## References

1. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
2. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, 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*, Jan 2019.
3. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Oswald and Fischlin [31], pages 457–485.
4. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.
5. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Coron and Nielsen [15], pages 535–566.
6. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648. Springer, 2016.
7. Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part*



- II*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.
8. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Paillier and Verbauwhede [32], pages 450–466.
  9. Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *J. Cryptology*, 26(2):280–312, 2013.
  10. Joan Boyar and René Peralta. A new combinational logic minimization technique with applications to cryptology. In Paola Festa, editor, *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20–22, 2010. Proceedings*, volume 6049 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.
  11. Gaetan Cassiers and François-Xavier Standaert. Towards globally optimized masking: From low randomness to low noise rate or probe isolating multiplications with reduced randomness and security against horizontal attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):162–198, 2019.
  12. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Wiener [34], pages 398–412.
  13. Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventsislav Nikov, and Vincent Rijmen. Masking AES with  $d+1$  shares in hardware. In Gierlichs and Poschmann [21], pages 194–212.
  14. Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3–4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.
  15. Jean-Sébastien Coron and Jesper Buus Nielsen, editors. *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, 2017.
  16. Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In Paillier and Verbauwhede [32], pages 28–44.
  17. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11–13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2013.
  18. Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: Noekeon, 2000.
  19. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11–15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.

20. Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Oswald and Fischlin [31], pages 401–429.
21. Benedikt Gierlichs and Axel Y. Poschmann, editors. *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*. Springer, 2016.
22. Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Coron and Nielsen [15], pages 567–597.
23. Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.
24. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. Ls-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 18–37. Springer, 2014.
25. Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.
26. Anthony Journault and François-Xavier Standaert. Very high order masking: Efficient implementation and security evaluation. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 623–643. Springer, 2017.
27. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Wiener [34], pages 388–397.
28. Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.
29. 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 Cryptology ePrint Archive*, 2018:490, 2018.
30. Svetla Nikova, Vincent Rijmen, and Martin Schl affer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.
31. Elisabeth Oswald and Marc Fischlin, editors. *Advances in Cryptology - EURO-CRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*. Springer, 2015.

32. Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*. Springer, 2007.
33. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.
34. Michael J. Wiener, editor. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*. Springer, 1999.

## A PINI<sub>1</sub> security proof

**Proposition 10.** *The multiplication gadget of Algorithm 2 is  $d - 1$ -PINI.*

*Proof.* We prove that the values assigned to the probes by the simulator described in Algorithm 4 are indistinguishable from the multiplication gadget probes.

This behavior of the simulator is identical to the behavior of the gadget, except for values  $z_{ij}$ ,  $s_{ij}$  and  $p_{ij}^1$  for which  $i \notin X$  ( $X$  is generated by Algorithm 4).

In these cases, if  $z_{ij}$  or a sum in which it appears is probed, then there is no probe on  $z_{ji}$  (or their intermediate values, or a sum in which it appears) or on intermediate values of the computation of  $z_{ij}$ , hence  $r_{ij}$  is only observable through the probe  $z_{ij}$ . This means that  $z_{ij}$  behaves like a uniform random variable independent from all other variables, which is what the simulator generates.

For probes on  $s_{ij}$ , the same argument applies: if the behavior of the simulator is not identical to the one of the gadget, then  $r_{ij}$  is only observable through  $s_{ij}$ , hence  $s_{ij}$  behaves as a uniform independent random variable. To simulate  $p_{ij}^1$ , the simulator simulates  $s_{ij}$  as previously (and the same argument applies), then computes  $p_{ij}^1$  in the same manner as Algorithm 2.  $\square$

## B MIMO-SNI S-box optimization problem

The AES S-Box is made of three parts: a top linear transformation, a middle non-linear transformation and a bottom linear transformation. Since our goal is to have a probing secure implementation of the AES, we do not need to have a full MIMO-SNI S-box. Having only the middle non-linear transformation MIMO-SNI is enough since the top and bottom linear transformations can be considered as combined with the other linear operations of the AES (i.e., ShiftRow, MixColumns and AddRoundKey) when applying MIMO-SNI composability.

The non-linear transformation is made of 30 XOR gates and 32 AND gates, hence it contains more than  $2 \cdot (30 + 32) = 124$  wires. This means that it is

---

**Algorithm 3** Input shares chooser for the simulator of  $\text{PINI}_1$  multiplication

---

**Require:** Set of probes  $y_{A,*}^G \cup P$

$X \leftarrow \emptyset;$

**for**  $i = 0$  to  $d - 1$  **do**

**if**  $a_i, a_i + 1, b_i, a_i \cdot b_i$  or  $c_i$  is probed **then**

$X \leftarrow X \cup \{i\};$

**else if** there exists  $k$  such that  $\sum_{j=1}^k z_{ij}$  is probed **then**

$X \leftarrow X \cup \{i\};$

**for**  $j = 0$  to  $d - 1$  **do**

**if** there are at least two probes on intermediate values of computation of  $z_{ij}$  (these values are  $r_{ij}, p_{ij}^k, s_{ij}^k$  and  $z_{ij}$ ) **then**

$X \leftarrow X \cup \{i, j\};$

**else if** there is one probe on an intermediate value of the computation of  $z_{ij}$  **then**

**if**  $i \in X$  or  $j \in X$  **then**

$X \leftarrow X \cup \{i, j\};$

**else**

$X \leftarrow X \cup \{i\};$

$B \leftarrow X \setminus A;$

**Ensure:**  $|B| \leq |P|$

---

impossible to apply the exhaustive search used for the S-Box in  $\mathbb{F}_{256}$ . We therefore reformulate our graph optimization problem into a integer linear programming problem, for which there exists numerous solvers. This does not guarantee that we can find an optimal solution with a reasonable amount of resources, but solvers have efficient heuristics to find good solutions and can prove lower bounds for the solution. Since we take care that our representation as an optimization problem admits as acceptable solutions all the possible implementations of the considered logic circuit, we are able to provide upper and lower bounds on the cost of the optimal implementation.

We write the linear optimization problem in the following way. A binary variable  $e_i$  is associated to each edge  $i$  of the graph, indicating if it is cut (i.e., if a refresh is inserted). All the paths in the graph are then computed and a binary variable  $p_j$  is assigned to each path  $j$ , again indicating if it is cut. A path is cut if any edge in the path is cut. It implies a first general constraint  $p_j \leq \sum_i e_i$  (the sum is over the edges in the path).

We can then add the various constraints related to Non-Interference properties. First, to enforce NI, for each pair of vertices  $(u, v)$  all but one paths from  $u$  to  $v$  must be cut. Let  $J$  be the set of paths from  $u$  to  $v$ ,  $\sum_{j \in J} p_j \geq |J| - 1$ . Next, to enforce SNI, when  $u$  is an input node and  $v$  an output node, the constraint becomes  $\sum_{j \in J} p_j \geq |J|$ . For the MI part we need: for any node  $u$ , let  $J$  be the set of paths from any input node to  $u$ ,  $\sum_{j \in J} p_j \geq |J| - 1$ . Finally, for the MO part we need: for any node  $u$ , let  $J$  be the set of paths from  $u$  to any output node,  $\sum_{j \in J} p_j \geq |J| - 1$ .

---

**Algorithm 4** Simulator of probes for the PINI<sub>1</sub> multiplication (Algorithm 2)

---

**Require:** Set of probes  $y_{A,*}^G \cup P$

Run Algorithm 3 and get  $X$  and  $B$ .

**Require:** Knowledge of input shares  $x_{A \cup B,*}^G = x_{X,*}^G$ .

**for**  $0 \leq i \leq d-1$  **do**

**for**  $0 \leq j \leq d-1$  **do**

**if**  $i \in X$  and  $j \in X$  **then**

      Compute  $w_{ij}^k, s_{ij}^k$  and  $z_{ij}$  as specified by the algorithm of the multiplication gadget;

**else if**  $i \notin X$  **then**

      Leave  $z_{ij}$  and its intermediate values unassigned as they are not involved in any probe;

**else**

**Ensure:**  $i \in X$  and  $j \notin X$ .

**Ensure:** Only one intermediate value from the computation of  $z_{ij}$  is probed, or a sum in which  $z_{ij}$  appears.

**Ensure:**  $z_{ji}$  or its intermediate values do not appear in any probe.

**if**  $z_{ij}$  or a sum in which  $z_{ij}$  appears is probed **then**

$z_{ij} \xleftarrow{\$} \mathbb{F}_q$ ;

**else if**  $s_{ij}$  is probed **then**

$s_{ij} \xleftarrow{\$} \mathbb{F}_q$ ;

**else if**  $p_{ij}^0$  is probed **then**

$r_{ij} \xleftarrow{\$} \mathbb{F}_q$ ;

$p_{ij}^0 \leftarrow (a_i + 1) \cdot r_{ij}$ ;

**else if**  $p_{ij}^1$  is probed **then**

$s_{ij} \xleftarrow{\$} \mathbb{F}_q$ ;

$p_{ij}^1 \leftarrow a_i \cdot s_{ij}$ ;

  Compute (partial) sums of assigned  $z_{ij}$ , products  $a_i b_i$  and associated  $c_i$ .

**Ensure:** All the probed values are now assigned.

---

The objective function to be minimized is a weighted sum of the  $e_i$  variables. The weigh associated to each variable is the cost of adding a refresh on the corresponding edge. This cost can be any metric, such as the amount of random bits required, the computation time, etc. Since each edge has a distinct associated cost parameter, this is the point where we can take into account that the cost of adding a SNI refresh gadget may not be the same as replacing a NI multiplication with a SNI multiplication.

This simple way of writing our problem has two limitations. First, there are many paths in the computation graph (in the order of magnitude of  $10^5$  for the AES S-box) which leads to many variables and constraints in the optimization problem. This can be mitigated by grouping paths into clusters that share common parts and associating them to a single variable.

The second issue is related to split nodes: there are multiple trees of binary split nodes that represent the split of a value in more than two parts, and all these representations do not give equivalent possibilities for inserting refresh elements. Furthermore, no tree can provide all the optimization degrees of freedom. Since it would be impractical to run the optimization for all the possible trees, we instead modified the optimization problem. Each split node is replaced by a set of split nodes that form a fully connected DAG and constraints are set to ensure that a constant number of added edges is cut, which ensures that the added edges do not distort the objective function.

The tool used for translating a boolean circuit into the corresponding optimization problem is available as supplementary material and will be published.

## C Cost of masked gadgets

The randomness and field operations cost for SNI refresh, NI & SNI multiplication, and PINI multiplication gadgets are given in Table 2 for some small orders. The cost formula for any order are given next:

- Randomness (the formula for  $\mathcal{R}_{ref}^{SNI}(d)$  is  $\mathcal{O}(d \log d)$ ):

$$\begin{aligned} \mathcal{R}_{ref}^{SNI}(d) &= 2 \lfloor d/2 \rfloor + \mathcal{R}_{ref}^{SNI}(\lfloor d/2 \rfloor) + \mathcal{R}_{ref}^{SNI}(\lceil d/2 \rceil) \\ \mathcal{R}_{mul}^{NI}(d) &= \lfloor (d-1)^2/4 \rfloor + d - 1 \\ \mathcal{R}_{mul}^{SNI}(d) &= \min(d(d-1)/2, \mathcal{R}_{mul}^{NI}(d) + \mathcal{R}_{ref}^{SNI}(d)) \\ \mathcal{R}_{mul1}^{PINI}(d) &= d(d-1)/2 \\ \mathcal{R}_{mul2}^{PINI}(d) &= \lfloor (d-1)^2/4 \rfloor + 2d - 1 \end{aligned}$$

| $d$ | SNI refresh               | NI mul.                  | SNI mul.                  | PINI <sub>1</sub> mul.      | PINI <sub>2</sub> mul.      |
|-----|---------------------------|--------------------------|---------------------------|-----------------------------|-----------------------------|
| 2   | 1                         | 1                        | 1                         | 1                           | 3                           |
| 3   | 3                         | 2                        | 3                         | 3                           | 6                           |
| 4   | 4                         | 4                        | 6                         | 6                           | 9                           |
| 5   | 8                         | 5                        | 10                        | 10                          | 13                          |
| 6   | 12                        | 11                       | 15                        | 15                          | 17                          |
| 7   | 13                        | 15                       | 21                        | 21                          | 22                          |
| 8   | 16                        | 19                       | 24                        | 28                          | 27                          |
| 16  | 32                        | 71                       | 103                       | 120                         | 87                          |
| 32  | 96                        | 271                      | 367                       | 496                         | 303                         |
| $d$ | $\mathcal{R}_{ref}^{SNI}$ | $\mathcal{R}_{mul}^{NI}$ | $\mathcal{R}_{mul}^{SNI}$ | $\mathcal{R}_{mul1}^{PINI}$ | $\mathcal{R}_{mul2}^{PINI}$ |
| 2   | 2                         | 4                        | 4                         | 8                           | 14                          |
| 3   | 6                         | 10                       | 12                        | 21                          | 36                          |
| 4   | 8                         | 20                       | 24                        | 40                          | 66                          |
| 5   | 16                        | 30                       | 40                        | 65                          | 106                         |
| 6   | 24                        | 52                       | 60                        | 96                          | 154                         |
| 7   | 26                        | 72                       | 84                        | 133                         | 212                         |
| 8   | 32                        | 94                       | 104                       | 176                         | 278                         |
| 16  | 64                        | 382                      | 446                       | 736                         | 1134                        |
| 32  | 192                       | 1534                     | 1726                      | 3008                        | 4574                        |
| $d$ | $\mathcal{A}_{ref}^{SNI}$ | $\mathcal{A}_{mul}^{NI}$ | $\mathcal{A}_{mul}^{SNI}$ | $\mathcal{A}_{mul1}^{PINI}$ | $\mathcal{A}_{mul2}^{PINI}$ |
| 2   | 0                         | 4                        | 4                         | 6                           | 6                           |
| 3   | 0                         | 9                        | 9                         | 15                          | 15                          |
| 4   | 0                         | 16                       | 16                        | 28                          | 28                          |
| 5   | 0                         | 25                       | 25                        | 45                          | 45                          |
| 6   | 0                         | 36                       | 36                        | 66                          | 66                          |
| 7   | 0                         | 49                       | 49                        | 91                          | 91                          |
| 8   | 0                         | 64                       | 64                        | 120                         | 120                         |
| 16  | 0                         | 256                      | 256                       | 496                         | 496                         |
| 32  | 0                         | 1024                     | 1024                      | 2016                        | 2016                        |
| $d$ | $\mathcal{M}_{ref}^{SNI}$ | $\mathcal{M}_{mul}^{NI}$ | $\mathcal{M}_{mul}^{SNI}$ | $\mathcal{M}_{mul1}^{PINI}$ | $\mathcal{M}_{mul2}^{PINI}$ |

Table 2: Randomness and field operations cost of known gadgets.

– Additions:

$$\mathcal{A}_{ref}^{SNI}(d) = 2\mathcal{R}_{ref}^{SNI}(d)$$

$$\mathcal{A}_{mul}^{NI}(d) = 2\mathcal{R}_{mul}^{NI}(d) + d(d-1)$$

$$\mathcal{A}_{mul}^{SNI}(d) = 2\mathcal{R}_{mul}^{SNI}(d) + d(d-1)$$

$$\mathcal{A}_{mul1}^{PINI}(d) = 2\mathcal{R}_{mul1}^{PINI}(d) + 2d(d-1) + d$$

$$\mathcal{A}_{mul2}^{PINI}(d) = 2\mathcal{R}_{mul2}^{PINI}(d) + 4d(d-1)$$

– Multiplications:

$$\mathcal{M}_{ref}^{SNI} = 0$$

$$\mathcal{M}_{mul}^{NI} = d^2$$

$$\mathcal{M}_{mul}^{SNI} = d^2$$

$$\mathcal{M}_{mul1}^{PINI} = d(2d-1) + d$$

$$\mathcal{M}_{mul2}^{PINI} = d(2d-1) + d$$