Improved Bitslice Masking: from Optimized Non-Interference to Probe Isolation

Gaëtan Cassiers and François-Xavier Standaert

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

Abstract. We revisit the security analysis of bitslice masking which is currently the most efficient way to protect block ciphers against higher-order side-channel analysis. First, we put forward that the existing definition of Strong Non-Interference (SNI) used to reason about composability in masked implementations requires minor adaptations to capture the multiple-input multiple-output functions that bitslice implementations contain. We argue that the latter adaptations are instrumental in the analysis of a compositional strategy used in the masked AES implementations of Goudarzi and Rivain from EUROCRYPT 2017, where all multiplications are SNI with one input "refreshed" in a SNI manner. Second, we show that this strategy can be improved thanks to integer programming and report on an optimized masked AES S-box with significantly less SNI gadgets than previously required. Eventually we propose a new definition of Probe-Isolating Non-Interference (PINI) which captures a weaker yet sufficient requirement for composability in masked implementations. The latter definition allows major simplifications of the probing security analyzes of complex circuits. We show that it leads to improved performances for masked AES implementations (of order up to 20) by proposing and proving a first PINI multiplication.

1 Introduction

Masking is among the most popular countermeasures to prevent side-channel attacks. Its working principle is to split all the sensitive data manipulated by an implementation in $d+1$ shares, and to perform the computations on those shares only [11]. Under now well understood (noise and independence) leakage assumptions, the latter guarantees that the security of any side-channel attack against a masked implementation grows exponentially in the number of shares [28,15,16]. Concretely, various types of masking schemes have been considered in the literature including additive/boolean masking (see, e.g., [23,29,14] and many follow-ups), multiplicative masking [20,19], affine masking [31,18], Inner Product (IP) masking [1,2], … All these proposals come with significant overheads in execution time and randomness. We next focus on the case of additive/boolean masking, which has been shown recently to provide the best concrete performances thanks to bitslice implementations [21,24].

In the current state-of-the-art, masking schemes usually come with a security proof in the so-called probing model [23,29]. In its simplest definition, d-probing

security requires that the observation of up to d intermediate computations in the implementation does not reveal anything about the sensitive variables.[1] It has later been shown that while locally sufficient, this definition does not ensure secure composition [14]. Two solutions have then been introduced to mitigate this issue: first, formal tools enabling the automated analysis of implementations (up to certain number of shares) [4]; second, more demanding definitions of Non-Interference (NI) and Strong Non-Interference (SNI) that can be proven generically and guarantee composability for any number of shares [5].

In this paper, we start from the observation that the definitions of NI and SNI were originally introduced for operations with single inputs and single outputs, and therefore are not directly applicable to bitslice implementations. Our contributions in this respect are threefold:

First, we introduce a definition of Multiple-Input Multiple-Output Strong Non-Interference (MIMO-SNI), which extends the existing definition of SNI to the bitslice case. The latter is instrumental to formally analyze a compositional strategy proposed in [21] and re-used in [24], which is to use only SNI multiplications and to systematically refresh one of their inputs with a SNI gadget. We next call it the "greedy strategy" due to its high randomness requirements.

Second, we show how the greedy strategy can be optimized by representing the circuit to mask as a "computation graph" and describing how to express the definitions of NI, SNI and MIMO-SNI as graph properties. For simple circuits (such as the AES S-box operating in $\mathbb{F}_{256}$), the number of solutions is sufficiently small for exhaustive search (and we confirm the recent results of [8]). For more complex circuits, exhaustive analysis is impossible and we rely on integer programming. For illustration, we launch our optimization on the bitslice S-box of Boyar, Matthews and and Peralta [10] that is used in [21,24] and can reduce the number of SNI gadgets to 41 (with a lower bound of 34) compared to the 64 of the greedy strategy. In view of the significant (asymptotically dominating) impact of these SNI gadgets in the overall performances of a masked AES implementation, it directly leads to comparable performance gains.

Third, we observe that the definition of MIMO-SNI is still theoretically (over)demanding since it requires to cut the computation graph with refresh operations in order to avoid any distinct path between two nodes. We suggest an alternative approach based on "probe isolation" which rather ensures that such paths cannot be exploited by an adversary (by verifying that the propagated probes on these paths can be simulated with the same input shares). The latter Probe-Isolating Non-Interference (PINI) enables a major simplification of the security analyzes of complex masked circuits, since it allows the composition of any linear gadget with PINI multiplications. We also use this definition to exhibit concrete performance improvements, by proposing and proving a first

[1] 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 or transition-based leakages [25,26,12,3]. Yet, it is a necessary first step since an insecurity in the probing model usually leads to powerful concrete attacks.

PINI multiplication, which reduces the randomness requirements of the previous optimized bitslice S-box mixing SNI multiplications and refreshes for orders up to 20. It is an interesting open problem to investigate more efficient PINI multiplications that would generalize this gain to arbitrary orders.

2 Background

We work with (boolean) circuits and use the definition of [23]. A deterministic circuit C is a Directed Acyclic Graph (DAG) whose vertices are (boolean) gates and whose edges are wires. 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.

From these definitions, it is possible to define the notion of private circuit (which is equivalent to security in the d-probing model [29]).

Definition 1 (Private circuit [23]). *A private circuit for $f : \mathbb{F}_q^n \to \mathbb{F}_q^m$ is defined by a triple (I, C, O), where*

- *$I : \mathbb{F}_q^n \to \mathbb{F}_q^{n'}$ is a randomized circuit with uniform randomness ρ and called input encoder;*
- *C is a randomized circuit with input in $\mathbb{F}_q^{n'}$, output in $\mathbb{F}_q^{m'}$, and uniform randomness $r \in \mathbb{F}_q^{\alpha}$;*
- *$O : \mathbb{F}_q^{m'} \to \mathbb{F}_q^m$ is a circuit, called output decoder.*

We say that C is a d-private implementation of f with encoder I and decoder O if the following requirements hold:

- *Correctness: for any input $w \in \mathbb{F}_q^n$, $\Pr[O(C(I(w;\rho);r)) = f(w)] = 1$, where the probability is over the randomness ρ and r;*
- *Privacy: for any $w, w' \in \mathbb{F}_q^n$ and any set P of d wires in C, the distributions $\{C_P(I(w;\rho);r))\}_{\rho,r}$ and $\{C_P(I(w';\rho);r)\}_{\rho,r}$ are identical, with $C_P(I(w;\rho);r)$ the list of the d values on the wires from P.*

From now on, we assume that I and O are the canonical encoder and decoder: I encodes each input b by a block $(b_j)_{0\le j\le d}$ of $d+1$ random values with sum b, and O takes the sum of each block of $d+1$ values. Each block $(b_j)_{0\le j\le d}$ is called a sharing of b and each b_j is called a share of b.

A gadget C implementing a function $f : \mathbb{F}_q^n \to \mathbb{F}_q^m$ is a private implementation of f working with the canonical encoder and decoder. The gadget has n inputs and m outputs, each of which is a vector of $d+1$ shares. We usually denote the inputs of a gadget as $x_{i,j}$ where $i \in \{1,\dots,n\}$ is the input index and $j \in \{0,\dots,d\}$ is the share index. Outputs are usually written as $y_{i,j}$, $i \in \{1,\dots,m\}$. A set A is a set of share indices if $A \subset \{0,\dots,d\}$. We use the notations $x_{i,A} = \{x_{i,j} : j \in A\}$, $x_{*,A} = \{x_{i,j} : 1 \le i \le n, j \in A\}$ and $x_{*,*} = \{x_{i,j} : 1 \le i \le n, 0 \le j \le d\}$. When it is not clear from the context, we explicitly denote the gadget G to which the inputs or the outputs are related: $x^G_{i,j}$, $y^G_{i,j}$.

The wires in the set P used by the attacker are called the probes and the corresponding values $C_P(I(w;\rho);r)$ the values of the probes. Abusing notation, a probe p is sometimes used to denote the corresponding value.

In order to introduce other security definitions, we use the simulability framework put forward in [8]. Note that the notion of $(\mathcal{I},\mathcal{O})$-Non-Interference introduced in [5] is equivalent.

Definition 2 (Simulability [8]). *Let $P = \{p_1,\ldots,p_l\}$ be a set of l probes of a gadget C. Let $I = \{(i_1,j_1),\ldots,(i_k,j_k)\} \subset \{1,\ldots,n\} \times \{0,\ldots,d\}$ be a set of input wires of C.*

A simulator is a random function $\mathcal{S} : \mathbb{F}_q^k \to \mathbb{F}_q^l$. A distinguisher is a random function $\mathcal{D} : \mathbb{F}_q^l \times \mathbb{F}_q^{n(d+1)} \to \{0,1\}$.

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 distinguisher $\mathcal{D}$ and any inputs $x_{*,*}$, we have*

$$\Pr[\mathcal{D}(C_P(x_{*,*}),x_{*,*}) = 1] = \Pr[\mathcal{D}(\mathcal{S}(x_{i_1,j_1},\ldots,x_{i_k,j_k}),x_{*,*}) = 1],$$

where the probability is over the random coins in C, $\mathcal{S}$ and $\mathcal{D}$.

Simulability is a sufficient condition for probing security: a circuit C is d-probing secure if any set of probes P of size d can be simulated with d shares of each input. It is not a necessary condition for probing security, because the distinguisher has access to the input shares 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 [26]).

We can now define Non-Interfering (NI) gadgets, Tight Non-Interfering (TNI) gadgets and Strong Non-Interfering (SNI) gadgets. We again take the definitions from [8] and denote output probes on a gadget as probes on shares of outputs of the gadget, and internal probes as probes on any wire of the gadget including inputs and outputs.

Definition 3. *A gadget is d-NI if and only if every set of at most d internal probes can be simulated with at most d shares of each input.*

Definition 4. *A gadget is d-TNI if and only if every set of $t \leq d$ internal probes can be simulated with at most t shares of each input.*

The following property is proved in [8]. Thanks to this property, we can use the TNI definition when we discuss NI gadgets.

Property 1 (d-NI $\Leftrightarrow$ d-TNI). A gadget is d-NI if and only if it is d-TNI.

Definition 5. *A gadget is d-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 $t_1 + t_2 \leq d$, the set $\mathcal{I} \cup \mathcal{O}$ of probes can be simulated with t_1 shares of each input.*

For all linear functions f, there is a *trivial implementation* which requires no random gates and consists in applying the function independently to each share: $y_{*,j} = f(x_{*,j})$ for $j = 0, \ldots, d$. Such an implementation is always NI: the simulator can use the $x_{*,j}$ values for all the j's for which there is a probe in the evaluation of $f(x_{*,j})$. In the following, we always assume that linear operations are implemented in this way.

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 so-called SNI refresh gadgets (which implement the identity function in a SNI fashion). In the following, we will consider the multiplication of Ishai, Sahai and Wagner as an example of SNI multiplication that is proven secure at arbitrary orders and has randomness cost $d(d+1)/2$ [23], the multiplication of Belaid et al. as an example of NI multiplication that is proven secure at arbitrary orders and has randomness cost $\lfloor d^2/4 \rfloor + d$ [8], and the refresh of Batistello et al. as an example of SNI refresh that is proven secure at arbitrary orders and has randomness complexity in $\mathcal{O}(d \log d)$ [7]. We use the randomness as the main cost metric because of its relevance to composability and in view of its impact on the overall performances of masked software implementations such as [21,24], which we aim to improve. In this respect, we note that better randomness complexities can be reached, either by exploiting exhaustive searches (which therefore do not hold for arbitrary orders [8,6]) or by working in larger fields (which is less relevant for our investigations of bitslice masking) [9].

Eventually, in the rest of this article we study how to compose elementary gadgets in order to build more complex functions such as S-boxes or block ciphers that are (at least) probing secure. We use the following definitions of composite function and composite gadget for this purpose.

Definition 6 (Composite function). *A function f from $\mathbb{F}_q^m$ to $\mathbb{F}_q^n$ is a sequential composition of functions f_κ for $\kappa = 1, \ldots, \ell$ if the computation of $(y_1, \ldots, y_n) = f(x_1, \ldots, x_m)$ can be done using the following algorithm:*

$$\begin{aligned} (a_{o_{0,1}}, \ldots, a_{o_{0,m}}) &= (x_1, \ldots, x_m) \\ (a_{o_{\kappa,1}}, \ldots, a_{o_{\kappa,n_\kappa}}) &= f_\kappa(a_{i_{\kappa,1}}, \ldots, a_{i_{\kappa,m_\kappa}}) \qquad \text{for } \kappa = 1, \ldots, \ell \\ (y_1, \ldots, y_n) &= (a_{i_{\ell+1,1}}, \ldots, a_{i_{\ell+1,n}}) \end{aligned}$$

for some set of connection indices $i_{\kappa,p}$ and $o_{\kappa,p}$.

The functions f_κ are called composing functions*.*

This definition is different of the mathematical composition of functions $f_\ell \circ \cdots \circ f_1$ in that all the outputs of f_1 are not necessarily inputs of f_2: an output of f_1 may (for example) be an input of f_3 and f_4 (it is thus also possible to "re-use" values).

Definition 7 (Composite gadget). *Let f be a composite function of functions f_κ for $\kappa = 1, \ldots, \ell$ with connections indices $i_{\kappa,p}$ and $o_{i,p}$. A composite gadget G over $d+1$ shares that implements f is made of gadgets G_j over $d+1$ shares that implement the functions f_κ.*

The connection of the composing gadgets is done as follows, for an evaluation of $(y_1, \ldots, y_n) \leftarrow G(x_1, \ldots, x_m)$ *(where the* x_i*'s and* y_i*'s are elements of* $\mathbb{F}_q^l$*):*

$$\begin{aligned} (a_{o_{0,1}}, \ldots, a_{o_{0,m}}) &\leftarrow (x_1, \ldots, x_m) \\ (a_{o_{\kappa,1}}, \ldots, a_{o_{\kappa,n_\kappa}}) &\leftarrow G_\kappa(a_{i_{\kappa,1}}, \ldots, a_{i_{\kappa,m_\kappa}}) \qquad \text{for } \kappa = 1, \ldots, \ell \\ (y_1, \ldots, y_n) &\leftarrow (a_{i_{\ell+1,1}}, \ldots, a_{i_{\ell+1,n}}) \end{aligned}$$

where $a_\kappa \in \mathbb{F}_q^l$.

The gadgets G_κ *are called* composing gadgets.

3 Composition of bitslice gadgets

The previous definitions of NI and SNI gadgets have been shown to be very effective to avoid compositional issues such as put forward in [14]. For example, the specialization of the $(\mathcal{I}, \mathcal{O})$-Non-Interference they provide is perfectly suited to analyze the amount of refreshing required to implement an AES S-box with operations in $\mathbb{F}_{256}$ [8]. Yet, one possible limitation of these abstractions is that they implicitly assume gadgets with single inputs and single outputs. In this section, we first argue that such definitions are therefore not sufficient to capture the security of certain composite gadgets such as encountered in masked bitslice implementations. We then propose a new definition of Multiple-Input Multiple-Output (MIMO) SNI gadgets as a useful ingredient to analyze the security of any composite gadget. We finally put forward that secure composite gadgets are naturally obtained by the proposal in [21] of using (only) SNI multiplications with one input refreshed in a SNI manner (although we will use an additional ingredient for the full proof of this fact, discussed in Section 5.4).

3.1 Simulation framework and greedy strategy

We start by providing intuition about the simulation framework we use and the greedy strategy with the simple circuit example of Figure 1a which performs a multiplication of dependent values (masked at order $d = 1$). There is one adversarial (internal) probe in the SNI multiplication gadget and we will try to prove that the probe is not an attack (i.e., it is independent of any of the actual inputs) by demonstrating that it is possible to simulate it using at most one share of each of the inputs.[2]

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 of course on the position of the adversarial probe). Those wires are called *propagated*

[2] 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. A more efficient way of making that proof is discussed in Sections 3.3 and 4.

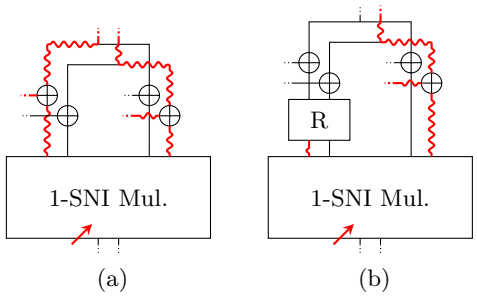


Fig. 1. Implementation of $(x + y)(x + z)$. The circuit is made of a SNI multiplication with linear circuits at the input, masked at order $d = 1$. The circuits illustrate (a) the limitation of SNI input composability and (b) the greedy strategy. The arrows indicate the adversarial probes (there is thus one internal probe in the multiplication) and the red snake wires are the propagated probes. The R box is a SNI refresh.

probes: if it is possible to simulate the propagated probes, then the adversarial probe can be simulated. 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. This gives four propagated probes at the input of the circuit, which probes all the shares of one of the inputs. Because of that, we cannot prove that the circuit is probing secure.

The circuit of Figure 1b has the same functionality as the circuit of Figure 1a, but is implemented using the greedy strategy: 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 be simulated using no input of the gadget (thanks to the SNI property), which makes the circuit 1-NI (and thus 1-probing secure).

The technique of proof used in this section is similar to the one used in [8] to prove the security of the implementation of a masked AES S-box in $\mathbb{F}_{256}$. We call this technique *probe propagation*: it is based on the idea of replacing adversarial probes with propagated probes that can be used to simulate the adversarial probes, and then iterating the process until the propagated probes are all at the inputs of the circuit. The conclusion is then easy.

The propagation of probes happens backwards in the circuit (probes on the outputs of a gadget propagate into probes on the inputs of the gadget). The definitions of NI and SNI can be expressed as the following rules in the probe propagation framework.

Probe propagation rules:

- For a NI gadget with n_o probes on shares of its output[3] and n_i probes inside the gadget, there is a propagated probe on $n_o + n_i$ shares of each input (this comes from Property 1).
- For a SNI gadget with n_o probes on output shares and n_i probes inside the gadget, there is a propagated probe on n_i shares of each input. Hence, the SNI gadgets (and in particular SNI refresh) stop the propagation of probes.

There are then two probe propagation conditions that guarantee security against the considered adversarial probes.

Probe propagation security conditions:

1. For any NI or SNI gadget, the number of output probes must be at most d. This follows from the definitions of NI and SNI.
2. For any input of the circuit, there cannot be propagated probes on all the $d + 1$ shares.

3.2 SNI is not enough (e.g., for bitslice implementations)

Say we are now interested in the situation where a non-linear gadget has multiple inputs and multiple outputs, such as an S-box with a bitslice implementation. To simplify the discussion, we take the case of 2-bit S-boxes (i.e., each S-box has two inputs and two outputs) masked at order $d = 1$, but our reasoning applies to any size of S-boxes (such as the 8 bit S-boxes of the AES) and any order.

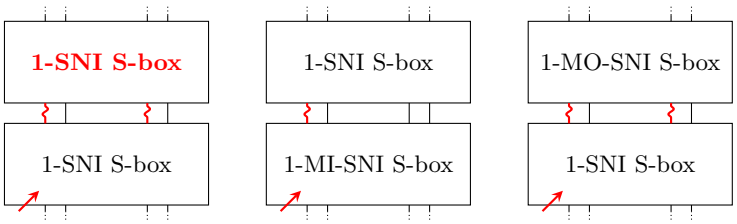
A first example of this context is the serial composition of two S-boxes (in which each of the outputs of the first S-box is connected to one input of the second S-box), depicted in Figure 2a.

This leads to the following problem: if the S-boxes are d-SNI and the adversary has d probes in the second S-box (remember $d = 1$ in our example), the propagated probes can cover up to d shares of each of the inputs of this S-box. There is thus a total of up to $2d$ propagated probes on outputs of the first S-box. Since the number of probes discussed in the first probe propagation security condition is the total number of probes on output shares, the condition is violated in this example.

Next, the situation gets even worse when linear gadgets come into play. A practical example is the AES S-boxes and MixColumns operations: 4 parallel 8-bit S-boxes followed by a 32-bit linear layer, followed again by 4 parallel 8-bit S-boxes. We will however use simpler examples to explain the two additional problems that arise in this case.

For the second problem, we consider a linear operation between two outputs of one S-box (depicted in Figure 3a). The adversary has d probes on one output

[3] We only consider here the gadgets with one output such as a multiplication. Multiple output NI gadgets (such as bitslice S-box) are not investigated since the stronger SNI property is itself is not sufficient for security of a block cipher implementation.

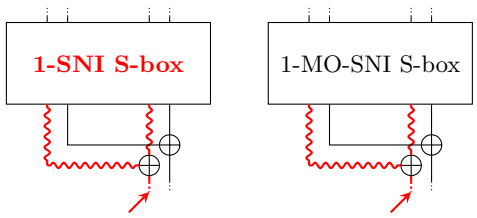


(a) SNI S-boxes: not secure (b) First solution: MI-SNI S-boxes (c) Second solution: MO-SNI S-boxes

Fig. 2. First problem: serial composition of two S-boxes. Example for $d = 1$ and 2-bit S-boxes. The arrows indicate the adversarial probes and the red snake wires are the propagated probes. Red boldface label for the S-box indicates that the first probe propagation security condition is not satisfied.

of the linear operation. The probes propagate to $2d$ probes on the output of the S-box. The first probe propagation security condition is again not respected.

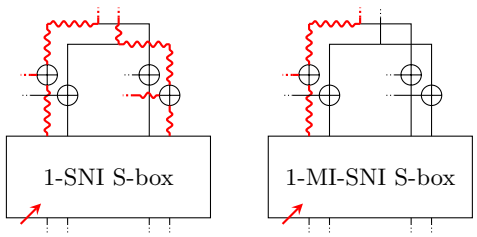
The third problem depicted in Figure 4a is the close to the one discussed previously for SNI multiplications (Figure 1): a linear layer at the input of the S-box is such that the probes propagated by the S-box propagate through the linear layer and reach all the shares of an input. This violates the second probe propagation security condition.



(a) SNI S-box: not secure (b) Solution: MO-SNI

Fig. 3. Second problem: S-box with linear layer at the output. Example for $d = 1$ and 2 bit S-box.

Summarizing, these examples show a limitation in the definition of SNI for the analysis of bitslice S-boxes that does not appear for implementations of the AES S-box in $\mathbb{F}_{256}$ (which have only one input and one output). In the following,



(a) SNI S-box: not secure (b) Solution: MI-SNI

Fig. 4. Third problem: S-box with linear layer at the input. Example for $d = 1$ and 2 bit S-box.

we show how to fix this issue by adapting $(\mathcal{I}, \mathcal{O})$-Non-Interference to this slightly more general context.[4]

3.3 Multiple-Input Multiple-Output SNI

Intuition. Based on the previous examples, natural directions to extend the definition of SNI and ensure composability in a bitslice implementation context are twofold:

- First, require the simulator to work with at most d input shares instead of d input shares on each input. This property is called Multiple-Input Strong Non-Interference (MI-SNI) and adds a third *probe propagation rule*:
  3. For a MI-SNI gadget with n_o probes on output shares and n_i internal probes, there is a total of n_i propagated probes on the shares of the inputs.
- Second, allow the adversary to probe up to d shares of each output, instead of d output shares in total. This property is called Multiple-Output Strong Non-Interference (MO-SNI) and adds a third *probe propagation security condition* (which is the first condition adapted to the MO-SNI gadgets):
  3. For any MO-SNI gadget, the number of probed shares on each output must be at most d.

Either of these extensions (MI-SNI or MO-SNI) solves the first problem (see Figures 2b and 2c), while the second problem is only solved by MO-SNI (see Figure 3b) and the third problem is only solved by MI-SNI (see Figure 4b). As a result, and since we want composability with any composite gadget, we need both

[4] Not being composable does not directly imply the existence of an attack in the probing model, in particular for low security orders that may be tested exhaustively with formal methods [4]. Yet, as the number of shares increases, the sufficient condition of security that composable gadgets provide becomes increasingly useful.

properties for S-boxes: Multiple-Input Multiple-Output Strong Non-Interference (MIMO-SNI).[5]

Formalization. We now formalize the definitions and prove that MIMO-SNI enjoys a useful composability property.

Definition 8 (MI-SNI). *A gadget is d-MI-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 $t_1 + t_2 \leq d$, the set $\mathcal{I} \cup \mathcal{O}$ of probes can be simulated* ***with at most*** *t_1* ***input shares****.*

Definition 9 (MO-SNI). *Let O_i be a set of share indices for $i = 1, \ldots, m$. A gadget is d-MIMO-SNI if and only if any set $\mathcal{I}$ of t_1* ***internal probes*** *and any sets O_i such that there exists a t_2* ***that satisfies*** *$t_1 + t_2 \leq d$* ***and*** *$|O_i| \leq t_2$* ***for*** *$i = 1, \ldots, m$, the set of probes $\mathcal{I} \cup y_{1,O_1} \cup \cdots \cup y_{m,O_m}$ can be simulated* ***with at most*** *t_1* ***shares of each input****.*

MIMO-SNI can be defined as satisfying both MI-SNI and MO-SNI. For completeness we give the equivalent explicit definition:

Definition 10 (MIMO-SNI). *Let O_i be a set of share indices for $i = 1, \ldots, m$. A gadget is d-MIMO-SNI if and only if any set $\mathcal{I}$ of t_1* ***internal probes*** *and any sets O_i such that there exists a t_2* ***that satisfies*** *$t_1 + t_2 \leq d$* ***and*** *$|O_i| \leq t_2$* ***for*** *$i = 1, \ldots, m$, the set of probes $\mathcal{I} \cup y_{1,O_1} \cup \cdots \cup y_{m,O_m}$ can be simulated* ***with at most*** *t_1* ***input shares****.*

From these definitions, we derive the following composition rules, which are used to prove the main result (Property 6).

Property 2. Serial composition of d-MO-SNI gadgets is d-MO-SNI.

Proof. We make the proof for two gadgets by describing the simulator. The general case follows by induction. The probes to be simulated can be split in

- t_1 internal probes in the first gadget,
- t_2 internal probes in the second gadget,
- output probes y_{i,O_i},

such that $t_1 + t_2 + |O_i| \leq d$ for all i. The probes in the second gadget and the output probes can be simulated with t_2 shares of each of its inputs. These shares and the probes in the first gadget can be simulated with at most t_1 shares of each of its inputs. □

Property 3. Parallel composition of d-MIMO-SNI gadgets is d-MIMO-SNI.

[5] We note that this definition can also be applied to multiplication gadgets, and that the SNI refresh & SNI multiplication composition of the greedy strategy is MIMO-SNI In fact, only the MI-SNI part is relevant since a multiplication gadget has only one output. This is not surprising since the problem solved by MI-SNI is essentially the same as the one solved by the greedy strategy (see Figures 1 and 4).

Proof. The simulator for the composite gadget simply runs the simulator for each internal gadget. The total number of input shares that are requested is at most the number of internal probes. □

Property 4. Serial composition of a linear & a d-MIMO-SNI gadget is d-MO-SNI.

Proof. Let t_1 be the number of internal probes in the linear gadget and t_2 the number of internal probes in the MIMO-SNI gadget. The simulator uses the simulator of the MIMO-SNI gadget to simulate the internal probes of the MIMO-gadget and the output probes. It has then to simulate t_2 output shares of the linear gadget and the t_1 internal probes. Let A the set of share indices corresponding to all the values to be simulated. The simulator can request to know all the shares $x_{*,A}$ and can thus trivially simulate the linear gadget. □

Property 5. Serial composition of a d-MO-SNI & a linear gadget is d-SNI.

Proof. Let B be the set of share indices of the output probes and of the probes in the linear gadget. The simulator of the MO-SNI gadget is used to simulate its internal probes and all the output shares whose index is in B. Simulation of the remaining probes is then trivial. □

We can now prove our main composition theorem about Substitution Permutation Networks (SPN), which shows that our definitions allow to build d-SNI (hence d-probing secure) SPNs. We consider only SPNs whose S-boxes are grouped into layers that operate over the full state.

Property 6. A SPN whose S-boxes are d-MIMO-SNI is d-SNI.

Proof. The SPN can be viewed as a alternating serial composition of linear and S-box layers, whose first and last layers are linear (this is without loss of generality since linear layers can be the identity function). A S-box layer is a parallel composition of S-boxes.

The S-box layers are d-MIMO-SNI thanks to Property 3. A round, the serial composition of one linear layer followed by one S-box layer is d-MO-SNI thanks to Property 4. Serial composition of rounds is d-MO-SNI thanks to Property 2. The combination of the final linear layer with the other rounds is d-SNI thanks to Property 5. □

We insist that this proof is limited to SPNs with full layers of d-MIMO-SNI S-boxes. It is for example not yet a proof of the greedy strategy proposed in [21], which uses MIMO-SNI multiplications within an S-box that does not have such a regular (full-layer) structure. We defer this proof to Section 5.4 where the introduction of "probe isolation" will allow much simplified analyses.

For a gadget with only one input, MI-SNI is equivalent to SNI and for a gadget with only one output, MO-SNI is equivalent to SNI. Hence, MIMO-SNI is equivalent to SNI for gadgets with one input and one output, such as the AES S-box implemented over $\mathbb{F}_{256}$. Property 6 thus applies to an AES implementation using the S-box in [8].

4 Optimized S-box implementations

The previous section gave a general framework for proving the security of refreshing strategies in masked (bitslice) block cipher implementations using MIMO-SNI S-boxes. In this section, we tackle the problem of minimizing the amount of SNI gadgets in the implementation of such an S-box in order to reduce their (e.g., randomness) cost.

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 [8]) and to the bitslice AES S-box of Boyar, Matthews and Peralta [10], bringing significant improvements over the greedy strategy in [21].

4.1 Connecting composability to computation graph properties

Let a masked S-box be given as a mix of elementary operations such as additions, multiplications and refreshes. We can define a high-level computation graph modeling this S-box as a DAG whose vertices represent operations and edges represent intermediate values. The operations can take any number of incoming edges (usually one or two) and produce one outgoing edge. Besides the aforementioned field operations and refreshes, such a computation graph may include three other types of vertices:

- split vertices take one incoming edge and can produce any number of outgoing edge(s). They model multiple uses of an intermediate value;
- input vertices have no incoming edge, one outgoing edge and the edges connected to these vertices are called input edges;
- output vertices have one incoming edge, no outgoing edge and the edges connected to these vertices are called output edges.

For simplicity, we next assume that all the additions and multiplications in our computation graphs are NI, and we model SNI gadgets as NI ones followed by a SNI refresh. Given an optimized computation graph, an implementer can then replace NI multiplications followed by a SNI refresh by (sometimes more efficient) SNI multiplications. We insist that this modeling is without loss of generality since it is equivalent from the probing model point of view and the respective (e.g., randomness) costs of the different gadgets of a private circuit are parameters of the optimizations in the next subsections.

The computation graph model is a formalization of the probe propagation framework discussed in Section 3.1. Capitalizing on the remark that SNI refresh gadgets stop the propagation of probes, we can simple 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 11 (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.*

Since there are at most d adversarial probes in the circuit, a simple security condition is that each probe propagates backwards to a single input through a single path. In this case, there will be at most d probes for each input and the second probe propagation security condition will be satisfied. Furthermore there will be at most d probes a the output of each gadget, since the probe propagation graph is a DAG (it is the reversal of the computation graph) so that the first probe propagation security condition will be satisfied too.

We can relax this constraint and impose that no probe can propagate backwards from a node to another one through two different paths, while still satisfying the probe propagation security conditions. In other words, for any pair of vertices there should be at most one (directed) path between them.

It can be seen that the latter is a necessary condition: if probes can propagate backwards through two paths from a node A to a node B and if the adversary has d probes on the output of A, $d+1$ shares of the output of A could be required for the simulation.

We now formalize this security condition with the following properties (which generalize the proof that the AES inversion is d-SNI in [8]).

Property 7. Let G be a composite gadget. If the gadget 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, then the gadget is 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 d. 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;
- the number of corresponding output probes if it is an output edge;
- 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 operations).

If for each input edge i, 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.

The probes inside a NI gadget are not considered since they can equivalently be replaced with probes on output shares of the gadget.

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

We use a small lemma for this purpose: for all edges i, $s_i = \sum_j \alpha_{ij} a_j$ where α_{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 children of a node satisfy this property, it is also satisfied for the node itself if the node is a refresh, split or NI operation. Input and output nodes are trivial.

The main hypothesis implies that $\alpha_{ij} \leq 1$ for all edges i and share indices j, hence $s_i \leq \sum_j a_j \leq d$. □

Property 8. Let G be a composite gadget. If the gadget satisfies Property 7 and if for any input node u and any output node v, there is no path from u to v, then the gadget is SNI.

Proof. Looking at the proof of Property 7, we observe that the coefficients $\alpha_{ij} = 0$ 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. □

Property 9. Let G be a composite gadget. If the gadget satisfies Property 8 and if 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 u_2 to v, then the gadget is MO-SNI.

Proof. We have to prove that for all edges i, $s_i \leq d$. Using the lemma from the proof of Property 7, we have that for any edge i, and input edges j, all but one α_{ij} are zero (i.e., $\sum_j \alpha_{ij} \leq 1$). This implies that $s_i \leq t_1 + t_2 \leq d$, taking the definitions of t_1 and t_2 from the definition of MO-SNI. For input edges i, the proof of Property 9 applies. □

Property 10. Let G be a composite gadget. If the gadget satisfies Property 9 and if for any pair of input 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, then the gadget is MIMO-SNI.

Proof. In addition to Property 9, we want 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 $\sum_{i \in I_e} s_i \leq \sum_{j \notin O_e} a_j = t_1$ where O_e is the set of output edges. □

4.2 Optimizing the AES S-box in $\mathbb{F}_{256}$

Using the previous graph formalization, we built a tool that checks if a circuit is (MIMO-)SNI. If we want to build a SNI S-box with the multiplication chain from [8], there are 16 wires on which we could insert a refresh. This number is sufficiently small to make a exhaustive search, which confirms the result of [8] 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.[6] It also shows that two refresh gadgets is the minimum possible, even with all multiplications implemented as SNI gadgets.

[6] [8] 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.

4.3 Optimizing the bitslice AES S-box of Boyar et al.

We now optimize the implementation of a bitslice AES S-box. We take the logic circuit by Boyar et al. in [10] and search where it requires to add SNI refresh elements to get a MIMO-SNI implementation.

The circuit 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 actually need to have a fully 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 the MIMO-SNI composability property.

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 impossible to apply the exhaustive search used in section 4.2. We therefore reformulate our graph optimization problem into a linear programming problem, for which there exists numerous solvers. The latter 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, $\sum_{j \in J} p_j \geq |J| - 1$.

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 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.

We ran this optimization with uniform cost for all edges (which, as discussed in Section 6, is reasonable for very high order making considering the current state-of-the-art for elementary gadget implementations). This gave a solution with 41 SNI elements and a lower bound of 34 SNI elements. The implementation of Goudarzi and Rivain in [21] uses two SNI elements per AND gate, totaling 64 SNI elements. Our optimized S-box is given in Appendix A.

5 Probe-isolating NI multiplications

In this section, we complement the previous optimization by introducing a new kind of Probe-Isolating Non-Interference (PINI) definition, and proving that it enjoys a very simple composition property. Furthermore, we show that any linear gadget satisfies this new definition, and we exhibit a multiplication gadget that also satisfies it. It gives us the ability to directly implement any boolean function with PINI gadgets. This is for example in contrast with the more complex analysis of [8] (or the one in the previous section), which requires a careful combination of NI and SNI gadgets.

5.1 Intuition

The main idea behind this new definition is to take into account not the number of probes (or of required inputs for simulation), but instead their position (i.e., the shares' index). The whole circuit can then be cut into $d + 1$ circuit shares that are not interconnected, except for non-linear gadgets. If we neglect those gadgets, the circuit is d-probing secure: the adversary can only probe d of the circuit shares, hence it has no information about one circuit share, which contains one share of each input. PINI gadgets behave in the probing model as if they

had no connection between circuit shares, which allows to implement non-linear functions while keeping the previous intuition of circuit sharing valid.[7]

Intuitively, the (MIMO-)SNI approach using the computation graph model of Section 4.1 cuts paths in the computation graph to avoid having distinct paths between two nodes. By contrast, the probe-isolating approach allows those paths while making sure that they are redundant from the adversarial viewpoint, by ensuring that the propagated probes on both paths can be simulated with the same shares.

5.2 Probe propagation framework

In the probe propagation framework, probes propagate through PINI gadgets in a way that respects circuit shares isolation.

Internal probes in PINI gadgets cannot be trivially associated to a circuit share, since there is no actual 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 to the adversary of at most one circuit share. This preserves the intuition that the adversary has knowledge of at most d of the $d + 1$ circuit shares.

We thus add a fourth *probe propagation rule*:

4. 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).

The probe isolation principles also impact security conditions: we count the number of circuit shares probed at the output, instead of the number of probes. There is thus a fourth *probe propagation security condition*:

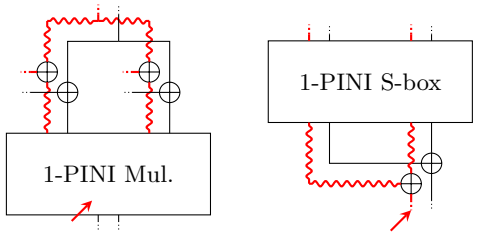
4. For any PINI gadget, the number of circuit shares touched by output probes must be at most d.

The way PINI works is illustrated in Figure 5, which takes two cases where SNI is not sufficient (Figures 1a and 3a) and shows how PINI solves the problem.

In Figure 5a, 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 5b, the two propagated probes at the output of the S-box have the same share index, hence it does not violate the fourth probe propagation security condition.

[7] To some extent the probe isolation idea can be connected to the Domain-Oriented Masking implementation approach described in [22]. However, the latter focuses on the local security of gadgets without discussing composability issues.



(a) Multiplication (b) S-box

Fig. 5. PINI circuits, masked at order $d = 1$. The arrows indicate the adversarial probes and the red snake wires are the propagated probes.

5.3 Probe-Isolating NI gadgets & composability

In this section, we give the formal definition of PINI, and prove security and composability properties. Furthermore, we exhibit PINI addition and multiplication gadgets.

We first show the link between the notion of circuit share and the notations of Section 2. For a gadgets with inputs $x_{i,j}$ and outputs $y_{i,j}$, all the shares of one input are denoted as $x_{i,*}$ and all the input shares that are in the same circuit share are all the input share that have the same share index j: $x_{*,j}$. The same goes for outputs.

In the following definition, the set A is the set of share indices (i.e., circuit shares) that are probed through output probes, and B is the set of circuit shares requested to simulate the internal probes.

Definition 12 (Probe-Isolating Non-Interference). *Let G be a gadget over $d+1$ shares and P a set of d_1 probes on wires of G (called internal probes). Let A be a set of d_2 share indices. G is d-Probe-Isolating Non-Interfering (d-PINI) if and only if for all P and A such that $d_1 + d_2 \leq d$, there exists a set B of at most d_1 indices such that probes on the set of wires $P \cup y^G_{*,A}$ can be simulated with the wires $x^G_{*,A\cup B}$.*

The following property shows that PINI satisfies the probing security requirement up to a slight additional requirement of independent input encodings. The latter is generally not a strong constraint, since we will use this property for large circuits, such as complete block ciphers, but it prevents the use of the optimizations described in [17].

Property 11 (PINI implies probing security). A d-PINI gadget (with $d+1$ sharing) is d-probing secure if the encodings of the inputs are independent of each other.

The probe isolation principle is also implicitly used in the seminal work of Ishai et al. [23], but they use $2d + 1$ masking.

Proof. Any set of at most d probes can be simulated with at most d shares of each input. Thanks to the independent input encodings, this set of simulation input probes is independent from all the input values. □

We now look at composability properties for PINI gadgets.

Property 12 (PINI composability). A composite gadget made of d-PINI composing gadgets is d-PINI.

Proof. All internal probes are either inside a gadget, or they are on a share of a a_j wire, in which case it can be considered as internal to one gadget for which a_j is an input or an output. Hence, if the set P of internal probes is of size d_1, there exists sets P_κ of probes internal to G_κ and of sizes $d_{1,\kappa}$ for $\kappa = 1, \ldots, \ell$ such that $P_1 \cup ... \cup P_\ell = P$ and $d_{1,1} + ... + d_{1,\ell} = d_1$.

Let $A_\ell = A$ the set of d_2 indices for output probes shares. For $\kappa = \ell$ to 1, we apply the definition of d-PINI: there exists a set of share indices B_κ of size at most $d_{1,\kappa}$ such that the probes P_κ and output probes $y^{G_\kappa}_{*,A_\kappa}$ can be simulated with the input probes $x^{G_\kappa}_{*,A_{\kappa-1}}$, where $A_{\kappa-1} = A_\kappa \cup B_\kappa$.

By induction, we have $A_{\kappa-1} = A \cup B_\kappa \cup \cdots \cup B_\ell$. This implies that $|A_{\kappa-1}| \leq d_2 + \sum_{p=\kappa}^{\ell} d_p \leq d_2 + d_1 \leq d$, hence the PINI conditions are satisfied for each gadget.

Let $B = A_0 \setminus A$, we have $|B| \leq |B_1 \cup \cdots \cup B_\ell| \leq d_1$. Finally, looking at the whole circuit, we observe that the wires $x_{*,A\cup B}$ allow to simulate all the required probes. □

We next prove that linear gadgets are PINI.

Property 13. The trivial implementation of a linear function is d-PINI.

Proof. Take B as the set of all share indices for which there is a probe in P. □

We finally introduce in Algorithm 1 a new multiplication gadget for $d+1$ shares with $d(d+1)/2$ randomness requirement. It is a small variation of the ISW multiplication [23].

Property 14. The multiplication gadget of Algorithm 1 is d-PINI.

Proof. We prove that the values assigned to the probes by the simulator described in Algorithm 3 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^1_{ij} for which $i \notin X$ (X is generated by Algorithm 3).

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 to the distinguisher through z_{ij}. This means that z_{ij} is seen by the distinguisher as a uniform random variable independent from all other variables, which is what the simulator generates.

Algorithm 1 PINI multiplication gadget over $d+1$ shares

Require: shared factors $a, b \in \mathbb{F}_q^{d+1}$ such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$
Ensure: output $c \in \mathbb{F}_q^{d+1}$ such that $\bigoplus_i c_i = a \cdot b$

```
for i = 0 to d do
  for j = i + 1 to d do
    r_ij ←$ F_q;
    r_ji ← r_ij;
  end for
end for
for i = 0 to d do
  for j = 0 to d do
    if i ≠ j then
      s_ij ← b_j ⊕ r_ij;
      p^0_ij ← ā_i · r_ij;
      p^1_ij ← a_i · s_ij;
      z_ij ← p^0_ij ⊕ p^1_ij;
Ensure: z_ij = r_ij ⊕ a_i · b_j
    end if
  end for
end for
for i = 0 to d do
  c_i ← a_i · b_i ⊕ ⨁^d_{j=0,j≠i} z_ij;
end for
```

For probes on s_{ij}, the same argument applies: r_{ij} is only observable to the distinguisher through s_{ij}, hence s_{ij} is seen by the distinguisher as a uniform independent random variable. To simulate p^1_{ij}, the simulator simulates s_{ij} as previously (and the same argument applies), then computes p^1_{ij} in the same manner as Algorithm 1. □

5.4 Relationship between PINI and MIMO-SNI

We first observe that MIMO-SNI implies PINI.

Property 15. Any d-MIMO-SNI gadget is d-PINI.

Proof. In the definition of PINI, take B as the share indices of t_1 input shares required by the simulator. □

This leads immediately to the following composability result.

Corollary 1. *A composite gadget whose composing gadgets are only d-MIMO-SNI gadgets and linear gadgets is d-probing secure.*

Proof. Direct from Property 15, Property 13, Property 12 and Property 11. □

We note that this corollary now applies to the AES implementation proposed in [21]: using only SNI multiplications and systematically refreshing one of their inputs guarantees the input and output independence conditions which are required to move from SNI to MIMO-SNI and therefore PINI gadgets.

Algorithm 2 Input shares chooser for the simulator of PINI multiplication

Require: Set of probes $y^G_{*,A} \cup P$
  $X \leftarrow \emptyset$;
  for $i = 0$ to d **do**
    if a_i, $\overline{a_i}$, 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 $\bigoplus_{j=1}^{k} z_{ij}$ is probed **then**
      $X \leftarrow X \cup \{i\}$;
    end if
    for $j = 0$ to d **do**
      if there are at least two probes on intermediate values of computation of z_{ij} (these values are r_{ij}, p^k_{ij}, s^k_{ij} 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
          $X \leftarrow X \cup \{i\}$;
        end if
      end if
    end for
  end for
  $B \leftarrow X \setminus A$;
Ensure: $|B| \leq |P|$

Algorithm 3 Simulator of probes for the PINI multiplication (Algorithm 1)

Require: Set of probes $y^G_{*,A} \cup P$
  Run Algorithm 2 and get X and B.
Require: Knowledge of input shares $x^G_{*,A\cup B} = x^G_{*,X}$.
  for $0 \leq i \leq d$ **do**
    for $0 \leq j \leq d$ **do**
      if $i \in X$ and $j \in X$ **then**
        Compute w^k_{ij}, s^k_{ij} 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 values of 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^0_{ij} is probed **then**
          $r_{ij} \xleftarrow{\$} \mathbb{F}_q$;
          $p^0_{ij} \leftarrow \overline{a_i} \cdot r_{ij}$;
        else if p^1_{ij} is probed **then**
          $s_{ij} \xleftarrow{\$} \mathbb{F}_q$;
          $p^1_{ij} \leftarrow a_i \cdot s_{ij}$;
        end if
      end if
    end for
  end for
  Compute (partial) sums of assigned z_{ij}, products $a_i b_i$ and associated c_i.
Ensure: All the probed values are now assigned.

6 Performance comparison

We conclude the paper by discussing and comparing the expected performances of our two proposals for the implementation of a (masked, bitslice) composable AES S-box. As mentioned in Section 2, we use the randomness complexity (in bits) as our main cost metric for this purpose (and since the computational complexity is essentially similar for the gadgets we consider that all have to compute the full matrix of shares' products). To verify the validity of this assumption, we compare the execution time of the various algorithms on a microprocessor.

We report the state-of-the-art randomness cost for the gadgets used in Table 1. Using these cost formulas, we first observe that for sufficiently high orders ($d \geq 17$), the multiplication of Belaid et al. followed by the refresh of Batistello et al. has a lower cost than the multiplication of Ishai, Sahai and Wagner, which justifies the assumptions made for the optimization in Section 4.3.[8]

Order d	NI mul.	SNI refresh	SNI mul.	PINI mul.
1	1	1	1	1
2	2 [8]	3	3	3
3	4 [8]	4 [6]	6	6
4	5 [8]	8	10	10
5	11	12	15	15
6	15	14 [6]	21	21
7	19	20	24 [6]	28
d	$\mathcal{C}_{mul,NI}$	$\mathcal{C}_{ref}$	$\mathcal{C}_{mul,SNI}$	$\mathcal{C}_{mul,PINI}$

$$\mathcal{C}_{mul,NI} = \lfloor d^2/4 \rfloor + d \text{ [8]}$$
$$\mathcal{C}_{ref} = (d+1)\left(\log_2(d+1) - 1\right) \text{ [7]}$$
$$\mathcal{C}_{mul,SNI} = \min\left(d(d+1)/2, \mathcal{C}_{mul,NI} + \mathcal{C}_{ref}\right)$$
$$\mathcal{C}_{mul,PINI} = d(d+1)/2 \text{ [Algorithm 1]}$$

Table 1. Randomness cost of best known gadgets at various orders. Numbers in the table with no reference are instantiations of the formula valid at all orders. The formula for $\mathcal{C}_{ref}$ is valid only if $d+1$ is a power of 2. It is more complicated in other cases but it is still $\mathcal{O}(d \log d)$. The implementation of the SNI multiplication is either the ISW multiplication [23] or a NI multiplication followed by a SNI refresh.

Using those costs, we can evaluate the cost for a full bitslice, masked and composable AES S-box for three different approaches: the greedy strategy, the MIMO-SNI optimized S-box, and the S-box that uses PINI multiplications. The bitslice S-box has 32 multiplications, hence the greedy strategy requires 32 SNI

[8] Our optimization can be easily 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 for each order.

multiplications and 32 SNI refresh gadgets. The optimized MIMO-SNI S-box requires 41 refresh elements, and 12 of those refresh the output of a multiplication (hence can be implemented with a SNI multiplication). The PINI implementation simply requires 32 PINI multiplication gadgets.

The total randomness cost is shown in Figure 6. First, we can see that the MIMO-SNI optimization reduces costs by approximately 20 % over the greedy strategy. The PINI multiplication is even better at low orders ($d \leq 23$), with cost reduction up to 50 %. However, for large d values, PINI is less interesting, which is logical since PINI multiplications have a cost of roughly $d^2/2$ while NI multiplications have a cost of roughly $d^2/4$ and SNI refreshes $d\log(d)$.

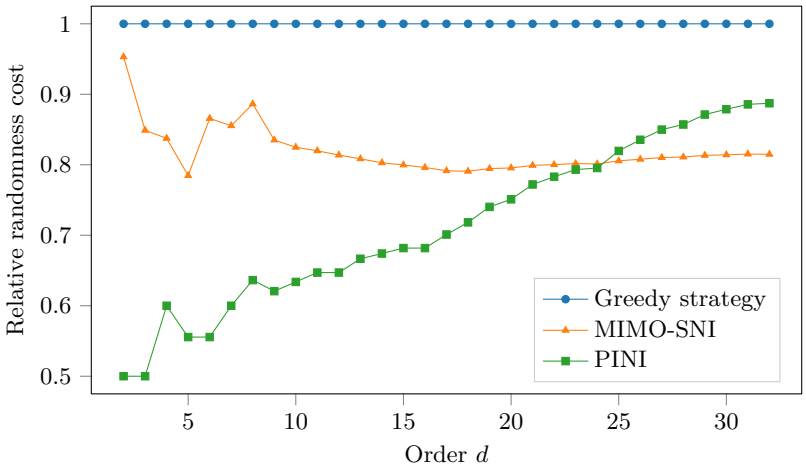


Fig. 6. Randomness cost for a bitslice, masked and composable AES S-box implementation. The cost is measured relatively to the cost of the greedy strategy.

It is interesting to note that the generic PINI construction is more efficient at low orders than the other solutions, even though those solution use automatically optimized gadgets.

We conclude by discussing the performance in a broader sense by looking at the execution time of a software implementation. We take the RC = 80 scenario from [24]: for their algorithm, the linear layers cost 1 % of execution time, the arithmetic operations of the S-box 7 % and the randomness generation 92 %. Their algorithm can be analyzed to find the execution time (expressed as percentage of their total runtime) of one arithmetic operation (including load/store overheads) and the time to generate one random bit. Knowing the number of arithmetic operations and amount randomness required for each of our algorithms, we compute their execution time, which gives Figure 7.[9]

[9] The only assumption of this estimation method is that the arithmetic operation execution time and randomness generation time are constant across algorithms. The

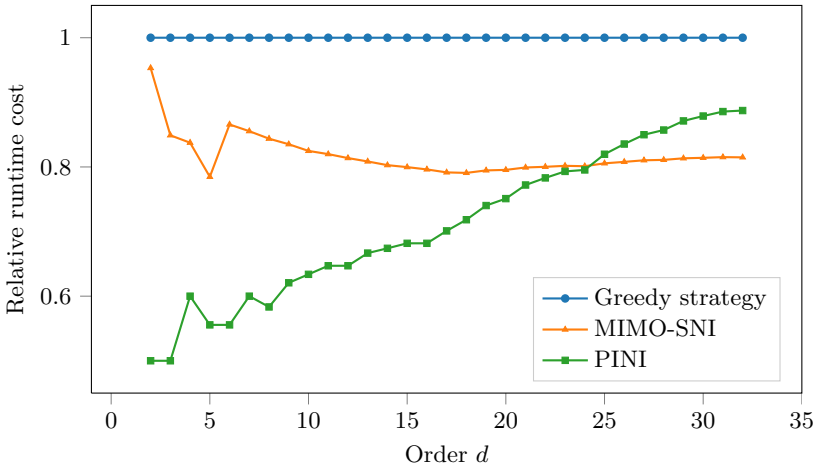


Fig. 7. Software runtime cost for a bitslice, masked and composable AES S-box implementation. The cost is measured relatively to the cost of the greedy strategy.

We first note that since the randomness generation time is dominant, our goal of reducing randomness requirements is justified. For the remaining part of the execution time, our MIMO-SNI optimization approach reduces the number of refresh operations, hence it reduces the amount of time spent in the S-box computation compared to the greedy strategy.

For the PINI implementation, the arithmetic cost of the multiplications is twice the cost of the greedy strategy (as it is the case for the randomness cost), but there is no refresh cost. The conclusion is hence similar: the PINI implementation is very competitive for small orders, and becomes worse than the MIMO-SNI optimization at large orders.

Finally, this comparative result (i.e., the choice between the MIMO-SNI optimization and PINI multiplications) is strongly dependent on the randomness complexity of the (state-of-the-art) gadgets used. In this respect, it is worth emphasizing that one core advantage of the PINI definition is that it allows trivial proofs of complex circuits with one single type of gadget, namely PINI multiplications (while the optimization has to deal with NI multiplications, SNI multiplications and SNI refreshes). In this respect, the investigation of more efficient PINI multiplications (that would make it the solution of choice for any number of shares) is an interesting open problem.

Source codes. We join an archive containing the source codes used in this submission as supplementary material.

estimation error is thus small and independent of many algorithmic parameters such as the masking order.

References

1. Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. Inner product masking revisited. In Oswald and Fischlin [27], pages 486–510.
2. Josep Balasch, Sebastian Faust, Benedikt Gierlichs, Clara Paglialonga, and François-Xavier Standaert. Consolidating inner product masking. In Takagi and Peyrin [30], pages 724–754.
3. 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.
4. 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 [27], pages 457–485.
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 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.
6. 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 [13], pages 535–566.
7. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In 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*, pages 23–39. Springer, 2016.
8. 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.
9. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 397–426. Springer, 2017.
10. Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *J. Cryptology*, 26(2):280–312, 2013.
11. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology*

Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
12. Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In 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.
13. 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.
14. 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.
15. 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.
16. 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 [27], pages 401–429.
17. Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In Takagi and Peyrin [30], pages 781–810.
18. Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010.
19. Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.
20. Jovan Dj. Golic and Christophe Tymen. Multiplicative masking and power analysis of AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.
21. Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Coron and Nielsen [13], pages 567–597.
22. 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.

23. 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.
24. 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.
25. 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.
26. 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.
27. Elisabeth Oswald and Marc Fischlin, editors. *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*. Springer, 2015.
28. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.
29. 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.
30. Tsuyoshi Takagi and Thomas Peyrin, editors. *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*. Springer, 2017.
31. Manfred von Willich. A technique with an information-theoretic basis for protecting secret data from differential power attacks. In Bahram Honary, editor, *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings*, volume 2260 of *Lecture Notes in Computer Science*, pages 44–62. Springer, 2001.

A Optimized AES S-box implementation

$\mathrm{R}(\cdot)$ means a SNI refresh gadget. The inputs are $x_0, \ldots, x_7$ and the outputs are $s_0, \ldots, s_7$.

Top linear layer:

$$
\begin{array}{llll}
t_0 = x_1 + x_2 & y_5 = y_1 + x_6 & y_{11} = y_{20} + y_9 & y_{17} = y_{10} + y_{11} \\
t_1 = x_4 + y_{12} & y_6 = y_{15} + x_7 & y_{12} = y_{13} + y_{14} & y_{18} = x_0 + y_{16} \\
y_1 = t_0 + x_7 & y_7 = x_7 + y_{11} & y_{13} = x_0 + x_6 & y_{19} = y_{10} + y_8 \\
y_2 = y_1 + x_0 & y_8 = x_0 + x_5 & y_{14} = x_3 + x_5 & y_{20} = t_1 + x_1 \\
y_3 = y_5 + y_8 & y_9 = x_0 + x_3 & y_{15} = t_1 + x_5 & y_{21} = y_{13} + y_{16} \\
y_4 = y_1 + x_3 & y_{10} = y_{15} + t_0 & y_{16} = t_0 + y_{11} &
\end{array}
$$

Middle non-linear layer:

$$
\begin{array}{llll}
x_{7,0} = \mathrm{R}(x_7) & t_{12} = y_{9,0} \cdot y_{11,0} & t_{33} = \mathrm{R}(t_{32}) + t_{24,0} & z_{10} = y_{3,0} \cdot t_{37} \\
y_{1,0} = \mathrm{R}(y_1) & t_{13} = y_{17,0} \cdot y_{14,0} & t_{33,0} = \mathrm{R}(t_{33}) & z_{11} = y_{4,0} \cdot t_{33} \\
y_{2,0} = \mathrm{R}(y_2) & t_{14} = t_{13} + t_{12} & t_{34} = t_{23,0} + t_{33} & z_{12} = y_{13,0} \cdot t_{43} \\
y_{3,0} = \mathrm{R}(y_3) & t_{14,0} = t_{14} & t_{35} = t_{27,0} + t_{33,0} & z_{13} = y_{5,0} \cdot t_{40,0} \\
y_{4,0} = \mathrm{R}(y_4) & t_{15} = y_{10,0} \cdot y_{8,0} & t_{36} = t_{35} \cdot t_{24} & z_{14} = t_{29,0} \cdot y_{2,0} \\
y_{5,0} = \mathrm{R}(y_5) & t_{16} = t_{15} + t_{12} & t_{36,0} = \mathrm{R}(t_{36}) & z_{15} = t_{42,0} \cdot y_{9,0} \\
y_{6,0} = \mathrm{R}(y_6) & t_{17} = t_4 + t_{14,0} & t_{37} = t_{34} + t_{36,0} & z_{16} = t_{45,0} \cdot y_{14,0} \\
y_{7,0} = \mathrm{R}(y_7) & t_{18} = t_6 + t_{16} & t_{37,0} = \mathrm{R}(t_{37}) & z_{17} = y_{8,0} \cdot t_{41} \\
y_{8,0} = \mathrm{R}(y_8) & t_{19} = t_9 + t_{14,0} & t_{38} = t_{27,0} + t_{36,0} & o_0 = z_0 \\
y_{9,0} = \mathrm{R}(y_9) & t_{20} = t_{11} + t_{16} & t_{39} = t_{38} \cdot t_{29,0} & o_1 = z_1 \\
y_{10,0} = \mathrm{R}(y_{10}) & t_{21} = t_{17} + y_{20} & t_{40} = t_{39} + t_{25} & o_2 = z_2 \\
y_{11,0} = \mathrm{R}(y_{11}) & t_{21,0} = \mathrm{R}(t_{21}) & t_{40,0} = \mathrm{R}(t_{40}) & o_3 = \mathrm{R}(z_3) \\
y_{12,0} = \mathrm{R}(y_{12}) & t_{22} = t_{18} + y_{19} & t_{41} = t_{37,0} + t_{40,0} & o_4 = \mathrm{R}(z_4) \\
y_{13,0} = \mathrm{R}(y_{13}) & t_{22,0} = \mathrm{R}(t_{22}) & t_{41,0} = t_{41} & o_5 = z_5 \\
y_{14,0} = \mathrm{R}(y_{14}) & t_{23} = t_{19} + y_{21} & t_{42} = t_{29,1} + t_{33,0} & o_6 = \mathrm{R}(z_6) \\
y_{15,0} = \mathrm{R}(y_{15}) & t_{23,0} = t_{23} & t_{42,0} = \mathrm{R}(t_{42}) & o_7 = \mathrm{R}(z_7) \\
y_{16,0} = \mathrm{R}(y_{16}) & t_{24} = t_{20} + y_{18} & t_{43} = t_{29,1} + t_{40,0} & o_8 = \mathrm{R}(z_8) \\
y_{17,0} = \mathrm{R}(y_{17}) & t_{24,0} = \mathrm{R}(t_{24}) & t_{44} = t_{33,0} + t_{37} & o_9 = \mathrm{R}(z_9) \\
t_2 = y_{12,0} \cdot y_{15,0} & t_{25} = t_{21,0} + t_{22,0} & t_{45} = t_{41,0} + t_{42} & o_{10} = \mathrm{R}(z_{10}) \\
t_{2,0} = t_2 & t_{26} = t_{23,0} \cdot t_{21,0} & t_{45,0} = t_{45} & o_{11} = z_{11} \\
t_3 = y_{6,0} \cdot y_{3,0} & t_{26,0} = \mathrm{R}(t_{26}) & z_0 = y_{15,0} \cdot \mathrm{R}(t_{44}) & o_{12} = \mathrm{R}(z_{12}) \\
t_4 = t_3 + t_{2,0} & t_{27} = t_{24,0} + t_{26,0} & z_1 = t_{37,0} \cdot y_{6,0} & o_{13} = z_{13} \\
t_5 = y_{4,0} \cdot x_{7,0} & t_{27,0} = t_{27} & z_2 = x_{7,0} \cdot t_{33,0} & o_{14} = z_{14} \\
t_6 = t_5 + t_{2,0} & t_{28} = t_{27} \cdot t_{25} & z_3 = y_{16,0} \cdot t_{43} & o_{15} = z_{15} \\
t_7 = y_{13,0} \cdot y_{16,0} & t_{29} = t_{28} + t_{22} & z_4 = y_{1,0} \cdot t_{40,0} & o_{16} = \mathrm{R}(z_{16}) \\
t_{7,0} = t_7 & t_{29,0} = \mathrm{R}(t_{29}) & z_5 = t_{29,1} \cdot y_{7,0} & o_{17} = \mathrm{R}(z_{17}) \\
t_8 = y_{1,0} \cdot y_{5,0} & t_{29,1} = \mathrm{R}(t_{29}) & z_6 = t_{42,0} \cdot y_{11,0} & \\
t_9 = t_8 + t_{7,0} & t_{30} = t_{24,0} + t_{23,0} & z_7 = t_{45,0} \cdot y_{17,0} & \\
t_{10} = y_{7,0} \cdot y_{2,0} & t_{31} = t_{26,0} + t_{22,0} & z_8 = t_{41,0} \cdot y_{10,0} & \\
t_{11} = t_{10} + t_{7,0} & t_{32} = t_{31} \cdot t_{30} & z_9 = y_{12,0} \cdot t_{44} &
\end{array}
$$

Bottom linear layer:

$$
\begin{array}{llll}
t_{46} = o_{15} + o_{16} & t_{54} = o_6 + o_7 & t_{62} = t_{52} + t_{58} & s_2 = \mathrm{Not}(t_{55} + t_{67}) \\
t_{47} = o_{10} + o_{11} & t_{55} = o_{16} + o_{17} & t_{63} = t_{49} + t_{58} & s_3 = t_{53} + t_{66} \\
t_{48} = o_5 + o_{13} & t_{56} = o_{12} + t_{48} & t_{64} = o_4 + t_{59} & s_4 = t_{51} + t_{66} \\
t_{49} = o_9 + o_{10} & t_{57} = t_{50} + t_{53} & t_{65} = t_{61} + t_{62} & s_5 = t_{47} + t_{65} \\
t_{50} = o_2 + o_{12} & t_{58} = o_4 + t_{46} & t_{66} = o_1 + t_{63} & s_6 = \mathrm{Not}(t_{56} + t_{62}) \\
t_{51} = o_2 + o_5 & t_{59} = o_3 + t_{54} & t_{67} = t_{64} + t_{65} & s_7 = \mathrm{Not}(t_{48} + t_{60}) \\
t_{52} = o_7 + o_8 & t_{60} = t_{46} + t_{57} & s_0 = t_{59} + t_{63} & \\
t_{53} = o_0 + o_3 & t_{61} = o_{14} + t_{57} & s_1 = \mathrm{Not}(t_{64} + s_3) &
\end{array}
$$