Towards Globally Optimized Masking: From Low Randomness to Low Noise Rate

or Probe Isolating Multiplications with Reduced Randomness and Security against Horizontal Attacks

Gaëtan Cassiers and François-Xavier Standaert

ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium
{gaetan.cassiers,fstandae}@uclouvain.be

Abstract. We improve the state-of-the-art masking schemes in two important directions. First, we propose a new masked multiplication algorithm that satisfies a recently introduced notion called Probe-Isolating Non-Interference (PINI). It captures a sufficient requirement for designing masked implementations in a trivial way, by combining PINI multiplications and linear operations performed share by share. Our improved algorithm has the best reported randomness complexity for large security orders (while the previous PINI multiplication was best for small orders). Second, we analyze the security of most existing multiplication algorithms in the literature against so-called horizontal attacks, which aim to reduce the noise of the actual leakages measured by an adversary, by combining the information of multiple target intermediate values. For this purpose, we leave the (abstract) probing model and consider a specialization of the (more realistic) noisy leakage / random probing models. Our (still partially heuristic but quantitative) analysis allows confirming the improved security of an algorithm by Battistello et al. from CHES 2016 in this setting. We then use it to propose new improved algorithms, leading to better trade-offs between randomness complexity and noise rate, and suggesting the possibility to design efficient masked multiplication algorithms with constant noise rate in $\mathbb{F}_2$.

Keywords: Masking, composability, horizontal attacks, random probing model.

1 Introduction

Masking has been established as a well-founded solution to improve security against side-channel attacks. Intuitively, it works by splitting all the sensitive data manipulated by an implementation in $d = t + 1$ shares, and to perform the computations on those shares only [CJRR99]. Under now well understood (noise and independence) leakage assumptions, the security of a masked implementation grows exponentially in the number of shares.

In the current state-of-the-art, ensuring this security guarantee is generally achieved in three main steps. First, the circuit is analyzed in the abstract probing model of Ishai et al. [ISW03]. The latter is instrumental in detecting composition flaws due to a lack of refreshing [CPRR13]. Concretely, such flaws can be avoided by checking the implementations in order to determine where refreshing gadgets have to be introduced [BBD+15], or by using composable gadgets [BBD+16]. Next, the circuit is analyzed in the qualitative bounded moment model of Barthe et al. [BDF+17], in order to determine the extent to which physical defaults such as (e.g.,) glitches re-combine the shares and reduce the security guarantees [MPG05]. Concretely, such flaws can be avoided both by constraining the algorithmic design of the masking schemes (e.g., by using the non-completeness property of threshold implementations [NRS11], which can be analyzed by extending the probing

IACR Transactions on Cryptographic Hardware and Embedded Systems ISSN 2569-2925, Vol. 2019, No. 2, pp. 162–198
DOI:10.13154/tches.v2019.i2.162-198

model to capture physical defaults [FGP+18]), or by mitigating the problem directly at the hardware level [MM12]. Third and finally, the circuit is analyzed in the quantitative noisy leakage model of Prouff and Rivain [PR13], based on information theoretic metrics [SMY09, DFS15]. In the latter one, masking security proofs typically require that the leakage of each share is sufficiently noisy, which is a purely hardware assumption that has to be verified/falsified empirically [BCPZ16, GS18]. This "*multi-model approach*" (systematized in [JS17]) is theoretically validated by the work of Duc et al., who showed that under the aforementioned noise and independence conditions, security in the noisy leakage model is implied by security in the probing model [DDF14].

Besides, and from the performance viewpoint, masking implies significant overheads in terms of time and randomness complexity (roughly quadratic in the number of shares [GSF14]). Concretely, it has been observed that the randomness complexity dominates as the number of shares in the masking schemes increases [GR17, JS17]. As a result, significant efforts have been devoted to reduce/optimize it for masked multiplications [BBP+16, BBP+17, FPS17].

Based on this state-of-the-art, our contributions are twofold.

First, we push the analysis of masked multiplications with reduced randomness complexity one step further. In this context, we observe that current solutions to ensure probing security are either based on a mix of Non-Interferent (NI) and Strongly Non-Interferent (SNI) multiplications [BBD+16, BBP+16], or can take advantage of a recently introduced notion of Probe-Isolating Non-Interference (PINI) [CS18]. The latter is convenient in the sense that it directly leads to fully composable implementations where all multiplications are PINI and all linear operations are performed share by share (without any refreshing). The PINI multiplication in this reference (next denoted as PINI_1) allows randomness reductions for low to moderate security orders, but was shown to have higher randomness complexity than an optimized combination of NI and SNI gadgets for large security orders (in the 20th range). We propose a new multiplication algorithm (next denoted as PINI_2) which leads to the best known randomness complexity for such large orders – ignoring [BBP+17] and restricting our analysis to boolean masking in $\mathbb{F}_2$ which usually leads to the best concrete performances [GR17, JS17].

Second, we observe that by pushing the reduction of randomness complexity towards optimal, one ignores the increased risk that the noise of each share can be reduced by an adversary (by combining multiple leakage samples), and therefore that this optimization can eventually be detrimental to the concrete security level of an implementation. From the practical viewpoint, this has been demonstrated by the so-called horizontal attacks in [BCPZ16, GS18]. From a theoretical viewpoint, it is captured by the concept of "*noise rate*", which essentially corresponds to the level of noise increase (or shares' information reduction) that is required so that masking still provides an exponential security improvement. The noise rate of Ishai et al.'s original multiplication algorithm is known to be in $\mathcal{O}(1/t)$ [ISW03, PR13, DDF14, DFS15]. Recent works by Andrychowicz et al. and Goudarzi et al. have shown that it is possible to reach noise rates in $\mathcal{O}(1/\log(t))$ or even $\mathcal{O}(1)$ [ADF16, GJR18]. Yet, the latter results require working in larger fields (typically such that $|\mathbb{F}|$ is in $\mathcal{O}(t)$).

Again motivated by the excellent performances of additive masking in small fields, we therefore investigate the noisy leakage security of masked multiplications that can work in $\mathbb{F}_2$. Our starting point for this purpose is an algorithm by Battistello et al. [BCPZ16] (next denoted as $\mathsf{SNI\text{-}H}$) which was shown to have improved resistance against horizontal attacks based on qualitative arguments (i.e., the authors showed that the number of times each share is manipulated is reduced compared to the algorithm by Ishai et al.). We push the understanding of this algorithm one step further by analyzing it in the recently proposed Local Random Probing Model (LRPM) [GGF18], which is a variation of the Random Probing Model (RPM) by Duc et al. [DDF14]. The LRPM restricts the way the information leakage of an implementation is exploited to local propagation rules inspired by the Belief Propagation (BP) algorithm used in Soft Analytical Side-Channel Attacks (SASCA) [VGS14], which allows simple concrete evaluations of actual multiplication algorithms (and more complex circuits). Since SASCA are among the (if not the) most efficient way to perform horizontal attacks in the current state-of-the-art [GS18], the bounds on the information that can be extracted thanks to SASCA (as provided by the LRPM) can be viewed as a good approximation of the worst-case security level in the noisy leakage model. Based on this new tool, we are able to confirm the relevance of the qualitative analysis of Battistello et al. in a quantitative manner, and a noise rate in $\mathcal{O}(1/\log(t))$. We then describe new algorithms (derived from the one of Battistello et al., but also from PINI multiplications and the original multiplication by Ishai et al.) with improved resistance against horizontal attacks, suggesting a noise rate in $\mathcal{O}(1)$. We additionally discuss the types of refreshing to use for improving security against horizontal attacks (i.e., the fact that some internal refreshings used for this purpose within the multiplication algorithms do not need to be SNI, and the randomness complexity gains that can be obtained from this observation). Eventually, we discuss the "*randomness complexity vs. noise rate*" tradeoff based on the different algorithms analyzed, suggesting gradual performance overheads as the noise rate evolves from $\mathcal{O}(1/t)$ to $\mathcal{O}(1/\log(t))$ and $\mathcal{O}(1)$.

Summarizing, our results bring two main conclusions and open problems. First, from a theoretical viewpoint, security against horizontal attacks becomes increasingly important as the number of shares (and claimed security order) in a masking scheme increases, and optimizations based only on reducing the randomness complexity are not sufficient in this context. This conclusion is based on quantitative but heuristic evaluations in the LRPM. Obtaining tight proofs in the RPM is an interesting open problem.

Second, from a practical viewpoint, we show that horizontal attacks are relevant as soon as the implementation is secure at high order (typically, $t \geq 6$), and can cause an exponential security reduction if the noise rate is not adapted. Applying such horizontal attacks in an open source setting (i.e., when all implementation details are given to the adversary) is direct with SASCA. Applying them in a close-source setting (i.e., when these implementation details are not public) may be more challenging. In this respect, another interesting open problem is to find out how strict is the separation between these contexts (if there is one). For example, to what extent black box attacks (e.g., exploiting machine learning / deep learning [MPP16]) can approach the efficiency of SASCA?

2 Background

2.1 Probing security and composability

We work with additive/boolean masking: each sensitive variable x is split into $d = t + 1$ shares $x_0, \ldots, x_t$ that are chosen uniformly at random and independently (except x_t) such that $x = x_0 + \cdots + x_t$ [CJRR99]. Most of our results can be applied for x in any finite field of characteristic 2, but we are mainly interested in the binary ($\mathbb{F}_2$) case since it is known to provide the best implementation performance thanks to bitslice implementations [GR17, JS17]. Concretely, it means that we typically represent a circuit manipulating sensitive variables as a sequence of elementary operations (i.e., additions and multiplications) that we replace by their masked counterparts (that we denote as masked gadgets).

A usual first step in the assessment of a masked circuit is to analyze its security in the (abstract) t-probing model of Ishai et al. [ISW03]. A circuit is secure in this model if and only if all the sets of at most t intermediate values it computes are independent of all the sensitive variables. In the context of block cipher implementations that we consider in this paper, sensitive variables are variables that depend on the plaintext and/or the key [CPR07].

While security in the t-probing model is relevant to concrete security in the noisy leakage model (thanks to the reduction by Duc et al. [DDF14]), it is however not sufficient for composability: a circuit made of the interconnection of multiple t-probing secure circuits is not necessarily t-probing secure [CPRR13].

The stronger definition of Strong Non-Interference (SNI) [BBD$^+$16] has been proposed as a solution to this problem. The composition of SNI gadgets is SNI, and SNI implies probing security [BBD$^+$16] (if the gadgets have only one output [CS18]). However, the simple and inexpensive way of implementing linear operations (i.e., the so-called trivial implementation where each linear operation is applied independently for each share) is not SNI. As a result, full block cipher implementations may require the addition of so-called "refreshing gadgets" (e.g., instantiated by SNI multiplications by one), which cause additional overheads in terms of time and randomness complexity [BBP$^+$16, CS18, BGR18]. The multiplication gadget of Ishai et al. [ISW03] (next denoted as the SNI-ISW multiplication gadget) is SNI.

Another solution for composability has recently been introduced: Probe-Isolating Non-Interference (PINI) [CS18]. It guarantees composability, implies probing security and at the same time, the trivial implementation of linear operations is PINI. We next introduce the formal definition of PINI in the simulatability framework of [BBP$^+$16] (generalized to gadgets with n inputs instead of only 2). We also discuss an intuitive way to reason in this framework using the idea of probe propagation.

We denote the input shares of a gadget as $x_{i,j}$ where $i \in \{1, \dots, n\}$ is the input index and $j \in \{0, \dots, t\}$ is the share index. A set A is a set of share indices if $A \subset \{0, \dots, t\}$. We use the notations $x_{i,A} = \{x_{i,j} : j \in A\}$, $x_{*,A} = \{x_{i,j} : 1 \leq i \leq n, j \in A\}$ and $x_{*,*} = \{x_{i,j} : 1 \leq i \leq n, 0 \leq j \leq t\}$.

Definition 1 (Simulatability (generalization of [BBP$^+$16], Definition 7.1))**.** Let $P = \{p_1, \dots, p_l\}$ be a set of l probes of a gadget C with n inputs. Let $I = \{(i_1, j_1), \dots, (i_k, j_k)\} \subset \{1, \dots, n\} \times \{0, \dots, t\}$ be a set of input wires of C.

A simulator is a randomized function $\mathcal{S} : \mathbb{F}_q^k \to \mathbb{F}_q^l$. A distinguisher is a randomized function $\mathcal{D} : \mathbb{F}_q^l \times \mathbb{F}_q^{nd} \to \{0, 1\}$.

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:

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

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

Simulatability is a sufficient condition for probing security: a circuit C is t-probing secure if any set of probes P of size t can be simulated with t shares of each input. Under the simulatability definition we can reason transitively about sets of probes. A set of probes can be simulated if the simulator has access to another set of probes. Hence, since the simulation is perfect, analyzes can then "forget" the first set of probes and only consider the second one. It allows proving probing security thanks to "probe propagation", by iterating the previous reasoning backwards, from a circuit's outputs to its inputs.

The PINI definition is based on the idea of splitting the whole circuit into shares, and forcing probes to propagate only in their own share. Since there are $d = t + 1$ shares and only t adversarial probes, the latter ensures that the circuit is probing secure. The isolation between circuit shares is naturally respected by the trivial implementation of linear operations, but this is not the case for non-linear operations. Hence, the PINI definition forces a simulated isolation.[1]

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

[1] Since probes inside a non-linear gadget are not naturally associated to a circuit share, the simulator can associate them to a circuit share of its choice.

simulate the internal probes. For a gadget 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 shares that have the same share index j: $x_{*,j}$. The same holds for outputs.

Definition 2 (Probe-Isolating Non-Interference [CS18])**.** 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 indices. G is t-Probe-Isolating Non-Interfering (t-PINI) if and only if for all P and A such that $t_1 + t_2 \leq t$, there exists a set B of at most t_1 indices such that probes on the set of wires $P \cup y^G_{*,A}$ can be simulated with the wires $x^G_{*,A\cup B}$.

2.1.1 Multiplication gadgets

In this section, we introduce a framework for analyzing multiplication gadgets (from the litterature and our new constructions) by splitting them into three stages: the `MatGen` stage produces a $d \times d$ matrix of (possibly refreshed) pairs of shares of x and y, the `Product` stage computes the partial products of those shares, which gives a d^2 sharing of the product, and the `Compression` stage compresses this sharing into a d-share output. This process is illustrated by the next equation (for $d = 4$):

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \xrightarrow{\texttt{MatGen}} \begin{bmatrix} (x_{0,0}, y_{0,0}) & (x_{0,1}, y_{1,0}) & (x_{0,2}, y_{2,0}) & (x_{0,3}, y_{3,0}) \\ (x_{1,0}, y_{0,1}) & (x_{1,1}, y_{1,1}) & (x_{1,2}, y_{2,1}) & (x_{1,3}, y_{3,1}) \\ (x_{2,0}, y_{0,2}) & (x_{2,1}, y_{1,2}) & (x_{2,2}, y_{2,2}) & (x_{2,3}, y_{3,2}) \\ (x_{3,0}, y_{0,3}) & (x_{3,1}, y_{1,3}) & (x_{3,2}, y_{2,3}) & (x_{3,3}, y_{3,3}) \end{bmatrix} \cdots$$

$$\cdots \xrightarrow{\texttt{Product}} \begin{bmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \alpha_{0,3} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \\ \alpha_{3,0} & \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} \end{bmatrix} \xrightarrow{\texttt{Compression}} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}. \quad (1)$$

We next show how this general construction applies in a few particular cases of interest.

- ISW multplication [ISW03]. This is the most simple case, where the `MatGen` stage is close to the identity: $(x_{i,j}, y_{i,j}) = (x_i, y_j)$, the `Product` stage is a straightforward product: $\alpha_{i,j} = x_{i,j} \cdot y_{i,j}$ and the `Compression` stage sums products and uniform randomness in a way that ensures correctness and security: $c_i = \alpha_{i,i} + \sum_{j=0,j\neq i}^{t} (r_{i,j} + \alpha_{i,j})$, where $r_{i,j} = r_{j,i}$ are random elements.
- Multplication of Belaid et al. [BBP+16]. This multiplication is identical to the ISW gadget for the `MatGen` and `Product` stages. Its `Compression` stage is optimized to use about two times less random elements that the ISW multiplication.[2] Its principle is shown by the following computation:[3]

$$\left(\alpha_{i,i} + \sum_{j} (r_{i,j} + \alpha_{i,j} + \alpha_{j,i} + r_{j-1} + \alpha_{i,j-1} + \alpha_{j-1,i}) \right)_{i=0,\ldots,t},$$

  for fresh randoms $r_{i,j}$ and r_{j-1}.
- PINI$_1$ multiplication [CS18]. This algorithm uses the same `MatGen` stage as the ISW multiplication. Its `Product` stage uses the "masked shares multiplication trick" to achieve PINI which we explain next. A probe on intermediate products $x_i \cdot y_i$ of the ISW multiplication violates the PINI definition: its simulation requires the knowledge

[2] This has the drawback of making it only NI while the ISW multiplication is SNI.
[3] The actual algorithm, reproduced in Algorithm 4, is indeed more complex, as it needs to take into account details such as parity, etc.

of two input shares from distinct circuit shares. The masked shares' multiplication trick solves this problem by computing the tuple $(\alpha_{i,j,0}, \alpha_{i,j,1}) = (x_i \cdot r, x_i \cdot (r + y_j))$ (where r is a fresh random element for each use of the trick), whose sum is indeed $x_i \cdot y_j$, and for which no probe can violates the PINI definition. This tuple is then combined with the randomness of the next `Compression` stage as follows: $c_i = \sum_{j=0}^{t} (r_{i,j} + \alpha_{i,j,0} + \alpha_{i,j,1})$ (addition is performed left-to-right).

In order to reduce the randomness consumption, the PINI_1 multiplication uses the same random bits for the masked shares' multiplication trick and for the `Compression` stage ($r = r_{i,j} = r_{j,i}$ for each application of the trick). For "diagonal" ($i = j$) elements, $r_{i,i} = 0$. This is shown to be secure in [CS18].

- Multiplication of Battistello et al. [BCPZ16]. This algorithm uses the same `Product` and `Compression` stages as the ISW multiplication, but its `MatGen` stage is modified in order to resist horizontal attacks. This is discussed in detail in Section 4.4.

It should be noted that each of the stages may use some random bits (and sometimes, as an optimization, the same random bits a re-used in different stages, e.g. in PINI_1). In the `MatGen` stage, the randomness cost comes from the use of additional "internal" refresh gadgets (which primarily impact security against horizontal attacks), such as for the multiplication of Battistello et al. A good example of randomness use in the `Product` stage is given by the PINI_2 gadget discuss in Section 3. Finally, the randomness in the `Compression` stage is primarily needed for probing security (it is the only stage where randomness is used in the SNI-ISW multiplication).

2.2 Horizontal attacks and the Local Random Probing Model

In actual side-channel attacks, the adversary observes leakage traces which depend on the intermediate values computed by a target implementation. The samples in the traces are typically noisy, which can be quantified by considering that there is a bound MI_o on the mutual information (MI) between each intermediate value computed and its corresponding leakage samples. Such a practical attack scenario can be captured by the noisy leakage model or the Random Probing Model (RPM). The first one was introduced in [PR13] as a realistic model formalizing the information theoretic and security evaluations typically considered in practice [SMY09]. The second one was introduced as a tool in the reduction from the noisy leakage model to the d-probing model [DDF14].

Concretely, the exploitation of noisy leakages can be performed based on different strategies. Standard side-channel attacks typically exploit the information related to the shares of one (or a few) sensitive computations – typically those that can be targeted via a divide-and-conquer approach [MOS11]. Horizontal side-channel attacks rather try to use the information from the leakage of all the variables and all their shares [BCPZ16]. The optimal way to perform such an attack (which would correspond to a worst-case analysis) would be to compute $\Pr[K = k|L]$ for all k's, where K is the sensitive variable and L is the full leakage trace. Yet, computing directly these probabilities through the Bayes rule has a prohibitive (computational) cost.[4] An interesting approach to simplify this computation is the Soft Analytical Side-Channel Attack (SASCA) introduced in [VGS14], which expresses the target implementation as a code (in the meaning of coding theory) and runs the Belief Propagation (BP) algorithm to decode it.

This attack has been recently analyzed in [GGF18], which introduces a technique to bound its data complexity denoted as the Local Random Probing Model (LRPM). It is based on analyzing the BP algorithm from the viewpoint of the mutual information. Guo

[4] More precisely, even if K is restricted to an enumerable value, evaluating this probability requires summing over K's shares, which grows exponentially in d [GS18].

et al. showed that a slightly modified version of the BP algorithm can be used to get an upper bound MI_t on the information that can be extracted from a leaking computation thanks to a SASCA. The idea is that rather than propagating probability densities for performing an attack (This is computationally expensive and gives the success/failure for one attack, which makes it impractical to evaluate high security levels.[5]), the information leakages are propagated. The bound can then be translated into a bound on the success rate of the adversary (and data complexity of the attack) [DFS15].

The LRPM is therefore a specialization of the RPM where the way the leakages are exploited is restricted to some information propagation rules similar to those of the BP algorithm, as we explain next.

Factor graph and BP algorithm. SASCA are based on the construction of a factor graph built from a set of relationships between intermediate values. For the masked implementations of block ciphers, we typically consider three kinds of relationships: sums of two elements: $x = x_1 + x_2$, products of two elements: $x = x_1 \cdot x_2$ and bijections: $y = \mathsf{g}(x)$ where g is a bijection. The factor graph is a bipartite graph made of variable nodes (one for each intermediate result within the leaking implementation) and function nodes (one for each relationship), with an edge if an intermediate result appears in a relationship. Furthermore, to each variable node is associated an intrinsic information – an estimated Probability Density Function (PDF) – which represents the leakage that can be observed on the corresponding intermediate result.

The principle of the BP algorithm is to exploit the relationships between variables in order to constraint the PDF estimates. It works by sending messages (PDF estimates) on the edges of the factor graph. Those messages are called extrinsic information. The BP algorithm alternates two steps until the algorithm converges. The first step sends messages from variable nodes to function nodes, and the second step sends messages from function nodes to variable nodes. The message sent by a variable node to a function node is a combination of the intrinsic information and the extrinsic information coming from function nodes at the previous step, for all the function nodes the variable nodes is connected to, except the function node that is the destination of the message being computed.[6] Likewise, the message sent by a function node to a variable node is a combination of all the incoming messages to the function node, except the message coming from the destination variable node (the actual combination depends on the relationship represented by the function node, and the position – as operand or as result – of the variable). Once the algorithm has converged, the intrinsic and extrinsic information at each node is combined to get the final estimated PDF. We refer to [VGS14] for the details.

Local Random Probing Model. The LRPM models SASCA by using the factor graph and the BP algorithm, with a slight modification: the messages are no longer PDF estimates, but MI values that represent the amount of information contained in the PDF estimates. A first consequence is that the intrinsic information is not a PDF, but the amount of leakage observed on the variable. The second consequence is a modification of the rules to compose messages [GGF18]:

- For messages of the first step (from variables to functions): the message sent is the sum of the intrinsic and the extrinsic information, upper bounded to $\mathrm{MI} = 1$.[7] This follows from the fact that combining estimates for a variable at most sums the information of the estimates.

[5] It also avoids convergence issues when the factor graph has cycles.
[6] All messages are initialized to an a priori (e.g., uniform) PDF.
[7] The limitation comes from the fact that $\mathrm{MI} = 1$ implies no uncertainty on the leaking variable, if the MI unit is the field element.

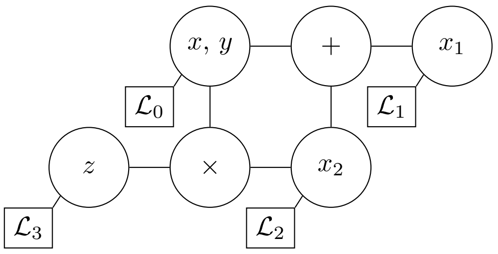


Figure 1: Factor graph with six nodes. $\mathcal{L}$ labels indicate intrinsic information.

- For messages of the second step (from functions to variables): using the random probing model, it is shown in [GGF18] that the information on the result of a sum or on the operand of any operation (sum or product) is bounded by the product of the information sent by the other concerned variables. Another case can happen though, which was not discussed in [GGF18] (since they do not use the result of multiplications in other operations). Namely the information on the result of a multiplication. We discuss it in Appendix A. Simply stated, in our case (where we multiply elements in $\mathbb{F}_2$), the resulting MI is the average of the MI on the operands.
- The case of bijection relationships is special: since the MI on one of the variables related by a bijection translates into the same MI on the other variable, the two nodes can be merged and the bijection function node removed.
- The final combination of information (once the algorithm has converged) is a sum of all the intrinsic and extrinsic MI for each variable node.

For illustration, we next give an example of factor graph in Figure 1. The corresponding equations for the LRPM are the following:

$$x = \mathsf{g}(y), \qquad x = x_1 + x_2, \qquad z = x_2 \cdot y.$$

The messages sent at the first step of the BP algorithm are:

$$\begin{aligned} \mathrm{MI}_{x,y\to+} &\leftarrow \min\left(1, \mathcal{L}_0 + \mathrm{MI}_{\times\to x,y}\right), & \mathrm{MI}_{x,y\to\times} &\leftarrow \min\left(1, \mathcal{L}_0 + \mathrm{MI}_{+\to x,y}\right), \\ \mathrm{MI}_{x_1\to+} &\leftarrow \min\left(1, \mathcal{L}_1\right) = \mathcal{L}_1, & \mathrm{MI}_{x_2\to\times} &\leftarrow \min\left(1, \mathcal{L}_2 + \mathrm{MI}_{+\to x_2}\right), \\ \mathrm{MI}_{x_2\to+} &\leftarrow \min\left(1, \mathcal{L}_2 + \mathrm{MI}_{\times\to x_2}\right), & \mathrm{MI}_{z\to\times} &\leftarrow \min\left(1, \mathcal{L}_3\right) = \mathcal{L}_3, \end{aligned}$$

and those sent at the second step are:

$$\begin{aligned} \mathrm{MI}_{+\to x,y} &\leftarrow \mathrm{MI}_{x_1\to+} \cdot \mathrm{MI}_{x_2\to+}, & \mathrm{MI}_{\times\to x,y} &\leftarrow \mathrm{MI}_{x_2\to\times} \cdot \mathrm{MI}_{z\to\times}, \\ \mathrm{MI}_{+\to x_1} &\leftarrow \mathrm{MI}_{x_2\to+} \cdot \mathrm{MI}_{x,y\to+}, & \mathrm{MI}_{\times\to x_2} &\leftarrow \mathrm{MI}_{x,y\to\times} \cdot \mathrm{MI}_{z\to\times}, \\ \mathrm{MI}_{+\to x_2} &\leftarrow \mathrm{MI}_{x,y\to+} \cdot \mathrm{MI}_{x_1\to+}, & \mathrm{MI}_{\times\to z} &\leftarrow \left(\mathrm{MI}_{x,y\to\times} + \mathrm{MI}_{x_2\to\times}\right)/2. \end{aligned}$$

The two steps are alternated until the algorithm converges.

3 New PINI Multiplication with reduced randomness

As a follow-up of the work in [CS18], we describe a new PINI multiplication gadget denoted as PINI_2 (and described in Algorithm 4) which reduces the randomness cost of PINI_1 by a factor of up to two. This new gadget draws inspiration from the NI multiplication of Belaid et al. [BBP+16] (i.e., its reduced randomness consumption, see Section 2.1.1) and from the PINI_1 multiplication [CS18] (i.e., the masked shares' multiplication trick in order to achieve PINI, see Section 2.1.1). In this respect, we first note that while the masked shares' multiplication trick and the reduced randomness can in principle be combined, the masked shares' multiplication trick needs to be optimized, as its naive use costs d^2 random elements. In the PINI_1 multiplication, this optimization is done by re-using the randomness of the `Compression` stage. Because the reduced randomness consumed in its `Compression` stage, another optimization strategy is needed for PINI_2. We thus do the following: generate d random elements $s_0, \dots, s_t$ and use the random $r = s_{i,j} = s_i + s_j$ for the shares' multiplication trick (for the multiplication of x_i and y_j). The total randomness cost for this new gadget is thus $\lfloor t^2/4 \rfloor + 2t + 1$.

The instantiation of PINI_2 at order $t = 4$ is shown below while the full gadget (Algorithm 4) in Appendix B:

$$\begin{aligned}
c_0 &= x_0 \cdot y_0 + t_{0,4} + t_{0,2},\\
c_1 &= x_1 \cdot y_1 + t_{1,4} + t_{1,2} + r_1,\\
c_2 &= x_2 \cdot y_2 + t_{2,4} + r_{1,2} + r_{0,2},\\
c_3 &= x_3 \cdot y_3 + t_{3,4} + r_3,\\
c_4 &= x_4 \cdot y_4 + r_{3,4} + r_{2,4} + r_{1,4} + r_{0,4},
\end{aligned}$$

where:

$$\begin{aligned}
s_i, r_{i,j}, r_i \xleftarrow{\$} \mathbb{F}_q, &\qquad s_{i,j} = s_i + s_j,\\
p^0_{i,j} = x_i \cdot s_{i,j}, &\quad p^1_{i,j} = x_i \cdot (y_j + s_{i,j}),\\
p^2_{i,j} = y_i \cdot s_{i,j}, &\quad p^3_{i,j} = y_i \cdot (x_j + s_{i,j}),\\
t_{i,i+1} = r_{i,i+1} + p^0_{i,i+1} &+ p^1_{i,i+1} + p^2_{i,i+1} + p^3_{i,i+1},\\
t_{i,j} = r_{i,j} + p^0_{i,j} + p^1_{i,j} + p^2_{i,j} + p^3_{i,j} &+ r_{j-1} + p^0_{i,j-1} + p^1_{i,j-1} + p^2_{i,j-1} + p^3_{i,j-1},
\end{aligned}$$

Proposition 1. *The masked multiplication gadget of Algorithm 4 is t-PINI.*

The full proof is given in Appendix C. We next discuss the main intuitions behind it.

The PINI definition is made of two conditions: each probe on an output share can be simulated using the input shares with the same share index, and each internal probe can be simulated using one share of each of the input shares, all shares having the same index.

The first condition is satisfied thanks to the structure of the `Compression` stage (this is actually in the proof of Belaid et al. [BBP+16]).

As for the second condition, it would trivially satisfied if each use of the masked shares' multiplication trick used independent randomness.

However, it can be seen that the use of related randomness does not break the security: the security could break if the adversary had multiple probes that depend on the same s_i. Since each use of an s_i is masked with a s_j, there can be security issues only when the adversary also has multiple probes that depend on s_j. In this case, the number of adversarial probes is such that the simulator can have access to the inputs x_i and y_j and can thus run a perfect simulation. An example of such a situation are probes in the computation of $x_{i_1}y_{j_1}$, $x_{i_1}y_{j_2}$, $x_{i_2}y_{j_1}$ and $x_{i_2}y_{j_2}$ for some indices i_1, i_2, j_1 and j_2.

A list of masked multiplication gadgets (secure at any order) together with their randomness complexity is given next:

- SNI multiplication of Ishai et al. [ISW03, BBD+16] (i.e. SNI-ISW): $t(t+1)/2$,
- NI multiplication of Belaid et al. [BBP+16]: $\lfloor t^2/4 \rfloor + t$,
- PINI multiplication of Cassiers and Standaert [CS18] (i.e., PINI_1): $t(t+1)/2$

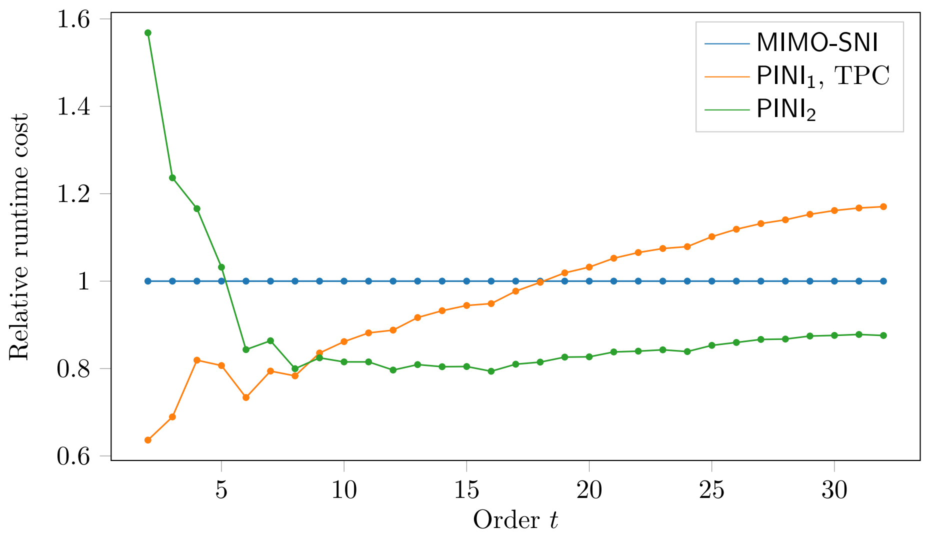


Figure 2: Software runtime cost for a bitslice, masked and composable AES S-box implementation. The cost is measured relatively to the cost of the optimized MIMO-SNI S-box in [CS18]. The curves for PINI_1 and TPC (S-box implementation with only ISW multiplications and no refresh elements, proven secure in [BGR18]) are indistinguishable at the scale of this plot. The source code for this plot is available from [Cas18b].

- New PINI multiplication (i.e., PINI_2): $\lfloor t^2/4 \rfloor + 2t + 1$.

It can be seen that our new construction has state-of-the-art asymptotic complexity of $t^2/4 + \mathcal{O}(t)$, while enjoying the simplicity of PINI composition.

For completeness, we illustrate how this improvement of the randomness complexity translates into improvements of the execution time for concrete circuits. We use the same approach as in [CS18] for this purpose: each kind of elementary operation (bitwise logical operations and randomness generation) has a cost, and the total cost is the sum of the elementary costs; we then estimate the runtime for various masked implementations of the bitslice AES S-box, based on the performance evaluation of [JS17] for software implementations on a ARM Cortex-M4 processor (considering the software PRNG scenario where the cost for generating 32 random bits equals RC = 80 cycles).

As shown in Figure 2 (where the runtime is measured relatively to the one of the optimized S-boxes in [CS18]), the PINI_2 multiplication improves the state-of-the-art for high orders ($t \geq 9$), but performs worse than the PINI_1 multiplication at lower orders ($t \leq 7$). Besides, the PINI approach is the best at any order, even when comparing with the recent Tight Private Circuits (TPC) approach of [BGR18].[8]

4 Modelling SASCA

We next complement the previous analysis by considering a complementary issue. Namely, what is the impact of horizontal attacks on masking schemes, especially when optimized

[8] The TPC S-box has the same randomness cost as the PINI_1 S-box. However, the latter comparison is slightly pessimistic for the PINI approach, since it ignores the fact that applying the TPC composition results may sometimes require additional refresh gadgets for linear operations. For example, in the case of a masked AES implementation, all the output bits of the key rounds must come out of a SNI gadget, which means that 96 additional refreshing gadgets must be added (since only 32 bits go through an S-box layer in the AES key scheduling).

for randomness? For this purpose, we will bound the resistance of an implementation against SASCA using the LRPM introduced in [GGF18]. Our contributions are in two main parts. First, we apply the LRPM to various (SNI and PINI) masked multiplication gadgets (including some with qualitatively improved resistance against horizontal attacks). The latter allows us to obtain good intuition about the impact of their design on their resistance against SASCA, and to obtain a more quantiative view of this resistance. Second, we propose new gadgets which improve the state-of-the-art in terms of resistance against horizontal attacks (leading to significantly better security in low-noise contexts). The variety of gadgets proposed allow us to explore the trade-off between noise requirements and randomness cost in masking, recently put forward in [GGF18].

4.1 Methodology

Our goal is to use the LRPM of [GGF18] (introduced in Section 2.2) in order to measure the resistance of various (recently proposed and new) masking algorithms to horizontal attacks. More precisely, we analyze masked multiplication gadgets at various orders, which are relevant targets of investigation since they usually correspond to the more complex parts of a masked implementation (both in terms of security and efficiency).

Using the LRPM requires the following inputs:

1. The factor graph of the target gadget (or implementation), which is a graph of relationships between all the variables manipulated.
2. The MI between each variable in the graph and the leakages.

For the first point, the relationships on the variables that appear in the gadget first include all the ones that are explicit in the gadget (i.e., each operation of the gadget creates a relationship). There are also variables which do not appear in the gadget such as the (unmasked) sensitive variables: for example, an input x which only appears through it shares $x_0, \dots, x_t$ in the gadget. We include them as well in the factor graph with the sharing relationships $x = x_0 + \cdots + x_t$. Other implicit relationships appear when refresh gadgets are used: for example, the sum of the inputs of a refresh is equal to the sum of its outputs. These relationships are important to consider when a circuit uses refresh gadgets internally (which, as will be seen next, may be relevant to improve security against horizontal attacks). We illustrate it with a simple circuit: $y = \mathsf{R}(x)$, where $\mathsf{R}(\cdot)$ is a refresh gadget, $x = (x_0, \dots, x_t)$ an input sharing and $y = (y_0, \dots, y_t)$ an output sharing. Let us assume that there are many manipulations of the shares y_i. In this case, the BP algorithm can propagate this information to the shares x_i, but (especially if the random elements in the refresh and the x_i shares are manipulated few times and the refresh has a high logic depth – for example by iterating many times a simple refresh algorithm) the information extracted on x_i shares can be much smaller than the information available on y_i. A SASCA targeting the input sensitive variable based on the relationship $x^* = x_0 + \cdots + x_t$ will therefore lead to a much lower information than actually available, by exploiting the relationship $x^* = y_0 + \cdot + y_t$. For this example, the solution is simple: insert in the factor graph not only the equation $x^* = x_0 + \cdots + x_t$, but also $x^* = y_0 + \cdots + y_t$. We generally apply this strategy for all the (possibly more complex) gadgets we analyze. That is, for each refresh gadget, we associate a new variable and insert in the factor graph the facts that (i) the sum of the input variables of the refresh is equal to this variable, and (ii) the sum of the variables associated to the two refresh gadgets is also equal to the associated variable.

Finally, and in order to be reflective of a concrete situation, we next evaluate the security of multiplications $z = x \cdot y$ for different algorithms, where the the fresh inputs x and y are related by a bijection and the output z is also in bijection with the input x.

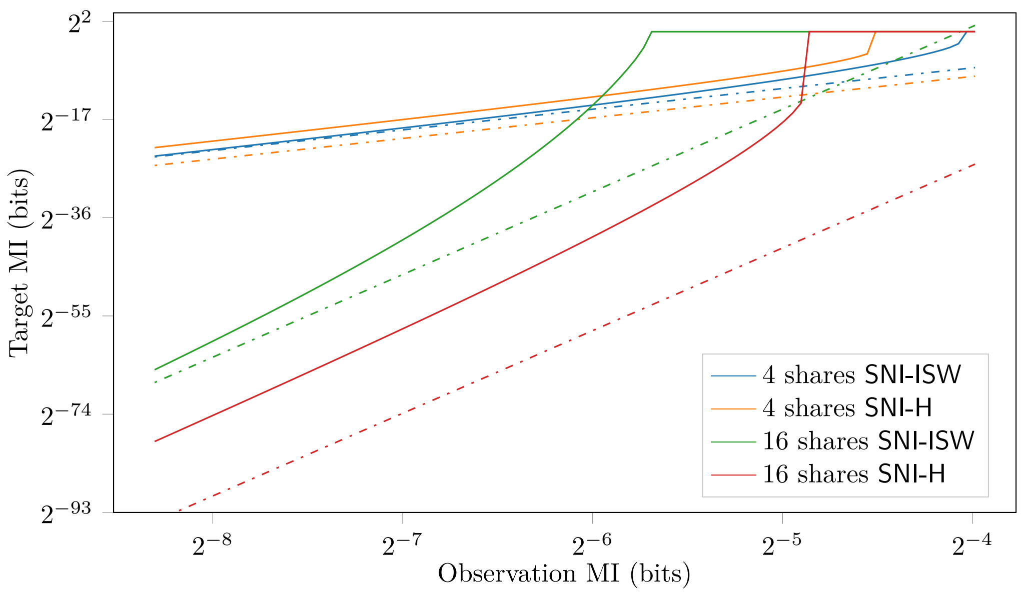


Figure 3: Target MI as a function of observation MI, for the SNI-ISW multiplication gadget [ISW03] and the SNI-H multiplication gadget [BCPZ16]. The continuous line is the bound on the target MI that can be extracted by the BP algorithm and the dashed line is a lower bound for the target MI computed assuming that only the leakages on input shares are exploited. The results for the NI multiplication gadget of Belaid et al. [BBP+16] are identical to the SNI-ISW curves.

The latter typically corresponds to the case of a masked AES S-box implemented as a multiplication chain in $\mathbb{F}_{256}$.[9]

For the second point, we first define the *manipulation* of a variable as its use as an operand of an operation or its apparition as the output of an operation. We assume that each manipulation leaks some amount MI_o of information and that the leakages of all manipulations are independent, hence we approximate the total leakage on any variable as $m\mathrm{MI}_o$ where m is the number of manipulations. The latter corresponds to the Independent Operations Leakages (IOL) assumption validated in [GS18]. For each random variable, we additionally assume that there is one manipulation associated to its generation.

4.2 Analysis of SNI multiplication gadgets

We first apply our methodology to the SNI-ISW gadget, at orders $t = 3$ and $t = 15$. Results are reported in Figure 3, leading to the following conclusions.[10]

In general, for low observation MI (i.e., MI_o) the trend of the target MI (i.e., MI_t) is an asymptote of slope t (as expected for tth order security). For larger MI_o, we see that the curves leave the asymptote until they reach $\mathrm{MI}_t = 1$. There are thus two regions in the graph: the low MI_o and high MI_o regions. The boundary between these regions varies depending on the curves (see for example SNI-ISW for $t = 3$ and $t = 15$ in Figure 3), which essentially reflects the noise rate. The location of the boundary and the slope of the asymptote are the two parameters that determine the security level of a gadget. In

[9] This bijection between the multiplication's inputs implies slightly more leakage than a multiplication between independent values. Yet, its impact is limited if their shares are fresh as we next consider: removing this link generally implies a reduction of MI_t by a factor of approximately 2.

[10] The source code for the LRPM tool and the plots is available from [Cas18a].

the following, since we are primarily interested in horizontal attacks (and the slope of the asymptote is always equal to t since we only consider t-probing secure gadgets anyway), we will focus on the noise rate, captured by the location of the boundary – concretely, we use the point where $\mathrm{MI}_t = 1$ for this purpose.

More specifically, we see that the location of the boundary strongly depends on the number of shares and security order for the SNI-ISW gadget. This comes from the fact that this gadget manipulates each input share d times, which explains why the boundary at order $t = 3$ is better than at order $t = 15$ for large MI_o. We note that these different boundaries correspond to the noise rate of $\mathcal{O}(1/t)$ that is expected for the SNI-ISW multiplication.[11]

By contrast, for the gadget of Battistello et al. [BCPZ16] that we also report on the figure (denoted as SNI-H and for which the details will be given later in the paper), the difference between the boundaries is much smaller. This is due to a different refreshing strategy used in this gadget, that is aimed to prevent horizontal attacks. More precisely, since each input share is manipulated only $\mathcal{O}(\log(t))$ times, the SNI-H gadget gains interest over the SNI-ISW one as t increases, thanks to its better noise rate (e.g., for $t = 15$). At lower orders (e.g., $t = 3$), the SNI-H gadget remains worse than the SNI-ISW gadget due to the additional leakages that its embedded refresh operations imply (which are not compensated by the improved noise rate in this case).

Going deeper into the analysis, we additionally plot lower bounds (represented with dashed lines on the figure) which are computed by applying the methodology under the hypothesis that only the leakages of the input shares are observed and exploited. It gives curves $\mathrm{MI}_t = \min\left(1, (n_o \mathrm{MI}_o)^t\right)$, where n_o is the number of observations of each input share. When the asymptote of the curve is close to the lower bound, which happens for SNI-ISW at low MI_o, the BP algorithm is not able to extract much information from the internal leakages of the gadget. Our interpretation is that the information from internal leakage is too small for precisely estimating the random values in the gadget, and the propagation of this information is "blocked" by the use of random elements in the multiplication. At larger MI_o, the information leaked on the internal variables becomes sufficient to (partially) recover the random elements, hence this information can be propagated to the input shares.

Interestingly, and while the lower bound is tight for the SNI-ISW gadget in the high noise region, it is not for the SNI-H gadget (even though the slope is still t). The latter suggests that there are more useful leakages on intermediate variables in this case, such as the outputs of the embedded refresh gadgets.

These analyzes confirm quantitatively the qualitative conclusion from [BCPZ16] that the repeated manipulation of some shares is the main weakness exploited by horizontal attacks. They cause an horizontal shift of the information theoretic curves in Figure 3, and impose a noise level that increases with the order of the implementation. If the noise level does not increase with the order, this shift directly translates into a security loss on MI_t of a factor $\mathcal{O}(t^t)$ for the SNI-ISW gadget (which is reduced to $\mathcal{O}((\log(t))^t)$ if the countermeasure of Battistello et al. is applied). In other words, this experiment highlights that the use of the SNI-ISW (resp., SNI-H) multiplication gadget at high order requires an amount of noise such that $\mathrm{MI}_o \propto 1/t$ (resp., $\mathrm{MI}_o \propto 1/\log(t)$).

4.3 Analysis of PINI multiplication gadgets

The application of our methodology to the two PINI multiplication gadgets is reported in Figure 4. For the PINI_1 gadget [CS18], we observe that the global trend is very close to the one of the SNI-ISW gadget, except that there is an horizontal shift of a factor of 2 in the asymptotic region. This is due to the multiple manipulations of the input shares during the computation of $\bar{a} \cdot r + a \cdot (r + b)$ (instead of $a \cdot b$ for SNI-ISW). Note that one

[11] Formally, the proofs in [DFS15] require a noise rate of $\mathcal{O}(1/t^2)$, but as discussed in [DFS15] the latter is assumed to be due to the proof that is not completely tight.

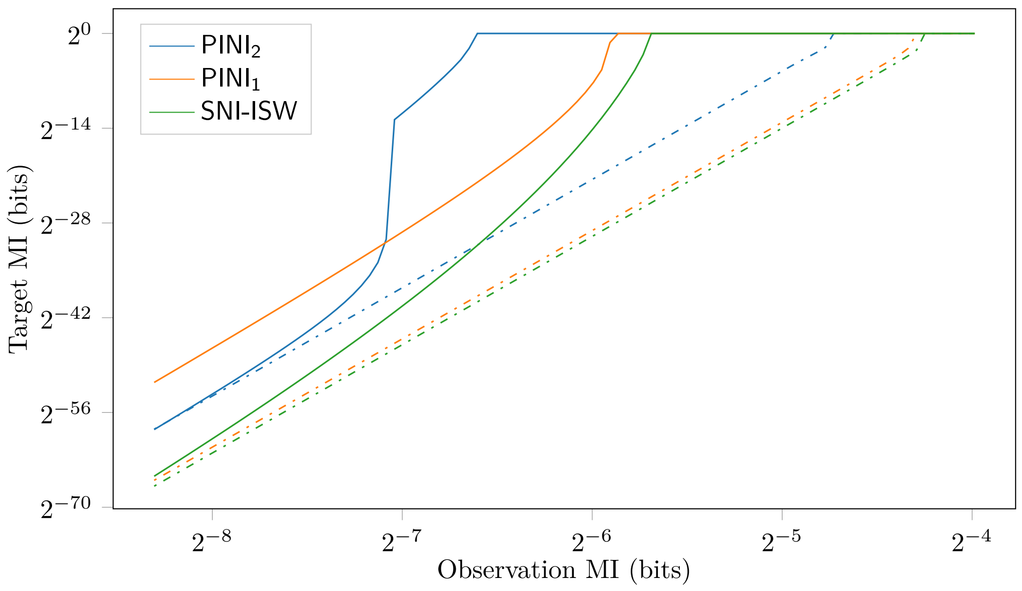


Figure 4: Target MI as a function of observation MI, for the SNI-ISW multiplication gadget [ISW03], the PINI$_1$ multiplication gadget [CS18] and the PINI$_2$ gadget (Section 3). The continuous line is the result of the belief propagation algorithm for the MI and the dashed line is a lower bound for the target MI computed assuming only the leakage on input shares are exploited. Results are shown for order 15.

of those manipulations is a use of $\bar{a}$, which is equivalent to a manipulation of a from the probing security viewpoint, but not from the horizontal attacks viewpoint, which explains why the bound is not tight.

For the PINI$_2$ gadget, we see that the asymptotic performance is slightly better than the one of PINI$_1$, which is due to less manipulations of the input shares. However, there are $\mathcal{O}(t^2)$ operations which involve only input shares and d random bits (i.e., the s_i in Algorithm 4), instead of the $\mathcal{O}(t^2)$ random bits for SNI-ISW. This has only limited impact at low MI$_o$ (where total leakage on the random bits is sufficiently low so they can be considered as unknown to the adversary). But at intermediate MI$_o$ values, the accumulated leakage on these random bits makes them partially known to the adversary, which causes the “staircase of asymptotes” aspect of the curve.

Since both PINI multiplication gadgets have $\mathcal{O}(t)$ manipulations of the input shares, they have a noise rate in $\mathcal{O}(1/t)$. Therefore, their use at high security order faces the same problem as the SNI-ISW gadget: it requires a large amount of noise (i.e., MI$_o \propto 1/t$).

4.4 Design of improved multiplication gadgets

In this section, we first describe the multiplication of Battistello et al. (SNI-H) in the framework of Section 2.1.1, and compare it with the ISW multiplication (SNI-ISW). We then investigate how the SNI-H scheme can be improved against horizontal attacks. Next, we build PINI multiplication gadgets secure against horizontal attacks by applying similar ideas. Finally, we build new gadgets that trade a bit of resistance to horizontal attacks for reduced randomness complexity.

The SNI-ISW multiplication is described in Algorithm 1, with the `MatGen` stage additionally described in Algorithm 2 (using the identity function for `Refresh1` and `Refresh2`), while the two other stages are interleaved in the other operations. In this (simplest) case, `MatGen` is directly obtained by setting $x_{i,j} = x_i$ and $y_{i,j} = y_j$.

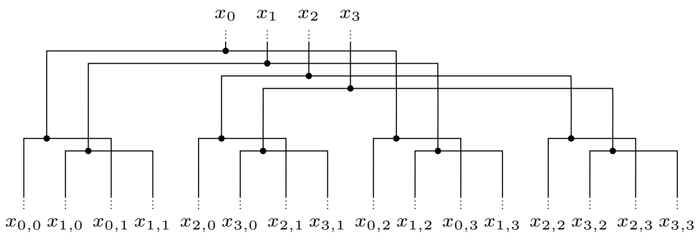
Algorithm 1 Generalized SNI multiplication gadget.

```
Require: shared factors $x, y \in \mathbb{F}_q^d$ such that $\sum_i x_i = x^*$ and $\sum_i y_i = y^*$
Ensure: output $c \in \mathbb{F}_q^d$ such that $\sum_i c_i = x^* \cdot y^*$
  $M \leftarrow \texttt{MatGen}((x_0, \dots, x_t), (y_0, \dots, y_t))$ {See Algorithm 2.}
  for $i = 0$ to $t$ do
    for $j = 0$ to $t$ do
      $(x_{i,j}, y_{i,j}) \leftarrow (M)_{i,j}$
  for $i = 0$ to $t$ do
    for $j = i + 1$ to $t$ do
      $r_{i,j} \overset{\$}{\leftarrow} \mathbb{F}_q$
      $z_{i,j} \leftarrow (r_{i,j} + x_{i,j} \cdot y_{i,j}) + x_{j,i} \cdot y_{j,i}$
      $z_{j,i} \leftarrow r_{i,j};$
  for $i = 0$ to $t$ do
    $c_i \leftarrow x_{i,i} \cdot y_{i,i} + \sum_{j=0, j \neq i}^{t} z_{i,j}$
```

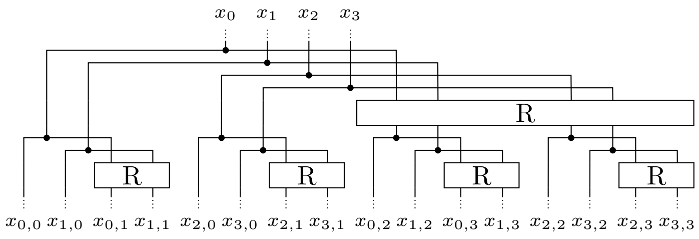
The heuristic countermeasure of Batistello et al. [BCPZ16] makes use of refresh gadgets in the `MatGen` stage to avoid repeated manipulations of the same variables. It proceeds by using a recursive algorithm which is illustrated in Figure 5b (for $d = 4$). This algorithm is applied to the sharing of x and to the sharing of y to get the matrix of pairs of shares. It is formally described by Algorithm 2 where the `Refresh1` and `Refresh2` algorithms are modified as per Table 1. That is, SNI-H multiplication gadgets correspond to the case where `Refresh2` is a SNI refresh (such as the refresh of Battistello et al. [BCPZ16]), and `Refresh1` is an identity gadget (the inputs are simply wired to the outputs). (As already mentioned, the previous SNI-ISW gadget can be instantiated by taking `Refresh1` and `Refresh2` as identity gadgets, as illustrated in Figure 5a).

Algorithm 2 `MatGen` (If d is a power of 2).

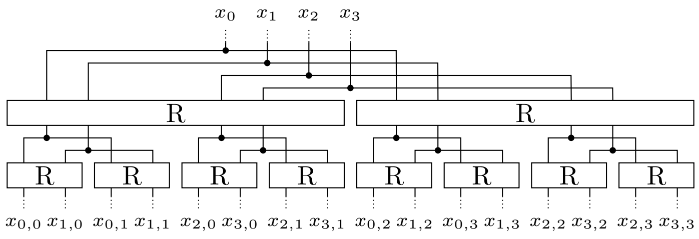
```
Require: Refresh algorithms Refresh1 and Refresh2
Require: Shared factors $x, y \in \mathbb{F}_q^d$ such that $\sum_i x_i = x$ and $\sum_i y_i = y$
Ensure: Output $M \in (\mathbb{F}_q^2)^{d \times d}$ such that $\sum_{i,j} x_{i,j} \cdot y_{i,j} = x \cdot y$, where $(x_{i,j}, y_{i,j}) = M_{i,j}$.
  if d=0 then
    $M \leftarrow [(x_0, y_0)]$
  else
    $X^{(1)} \leftarrow (x_0, \dots, x_{d/2-1})$
    $X^{(2)} \leftarrow (x_{d/2}, \dots, x_t)$
    $Y^{(1)} \leftarrow (y_0, \dots, y_{d/2-1})$
    $Y^{(2)} \leftarrow (y_{d/2}, \dots, y_t)$
    $X^{(1,1)} \leftarrow \text{Refresh1}(X^{(1)})$; $X^{(1,2)} \leftarrow \text{Refresh2}(X^{(1)})$
    $X^{(2,1)} \leftarrow \text{Refresh1}(X^{(2)})$; $X^{(2,2)} \leftarrow \text{Refresh2}(X^{(2)})$
    $Y^{(1,1)} \leftarrow \text{Refresh1}(Y^{(1)})$; $Y^{(1,2)} \leftarrow \text{Refresh2}(Y^{(1)})$
    $Y^{(2,1)} \leftarrow \text{Refresh1}(Y^{(2)})$; $Y^{(2,2)} \leftarrow \text{Refresh2}(Y^{(2)})$
    $M^{(1,1)} = \text{MatRef}(X^{(1,1)}, Y^{(1,1)})$
    $M^{(1,2)} = \text{MatRef}(X^{(1,2)}, Y^{(1,2)})$
    $M^{(2,1)} = \text{MatRef}(X^{(2,1)}, Y^{(2,1)})$
    $M^{(2,2)} = \text{MatRef}(X^{(2,2)}, Y^{(2,2)})$
    $M \leftarrow \begin{bmatrix} M^{(1,1)} & M^{(1,2)} \\ M^{(2,1)} & M^{(2,2)} \end{bmatrix}$
  return $M$
```


x_0 x_1 x_2 x_3
$x_{0,0}$ $x_{1,0}$ $x_{0,1}$ $x_{1,1}$ $x_{2,0}$ $x_{3,0}$ $x_{2,1}$ $x_{3,1}$ $x_{0,2}$ $x_{1,2}$ $x_{0,3}$ $x_{1,3}$ $x_{2,2}$ $x_{3,2}$ $x_{2,3}$ $x_{3,3}$

(a) No refreshing — SNI-ISW.


x_0 x_1 x_2 x_3
R
R
R
R
R
$x_{0,0}$ $x_{1,0}$ $x_{0,1}$ $x_{1,1}$ $x_{2,0}$ $x_{3,0}$ $x_{2,1}$ $x_{3,1}$ $x_{0,2}$ $x_{1,2}$ $x_{0,3}$ $x_{1,3}$ $x_{2,2}$ $x_{3,2}$ $x_{2,3}$ $x_{3,3}$

(b) Battistello et al. refreshing — SNI-H.


x_0 x_1 x_2 x_3
R
R
R
R
R
R
R
R
R
R
$x_{0,0}$ $x_{1,0}$ $x_{0,1}$ $x_{1,1}$ $x_{2,0}$ $x_{3,0}$ $x_{2,1}$ $x_{3,1}$ $x_{0,2}$ $x_{1,2}$ $x_{0,3}$ $x_{1,3}$ $x_{2,2}$ $x_{3,2}$ $x_{2,3}$ $x_{3,3}$

(c) Double refreshing — SNI-H+.

Figure 5: Exemplary instantiations of the generalized `MatGen` algorithm (Algorithm 2) for $d = 4$. The algorithm basically corresponds to two trees of internal refresh gadgets (one for sharings of x, one for those of y). The input shares are $x_0, \dots, x_3$. The $x_{i,j}$ are the outputs of the `MatGen` stage (see Equation (1)).

Table 1: Refreshing gadgets used for various multiplication gadgets. BatRef is the SNI refresh gadget of Battistello et al. ([BCPZ16], Algorithm 6). SimpleRef is a NI refresh using t random bits (Gadget 1 of [BBD+16]). The Identity gadget wires inputs to outputs.

	`Refresh1`	`Refresh2`
SNI-ISW	Identity	Identity
SNI-H	Identity	BatRef
SNI-H+	BatRef	BatRef
SNI-H*	SimpleRef	SimpleRef

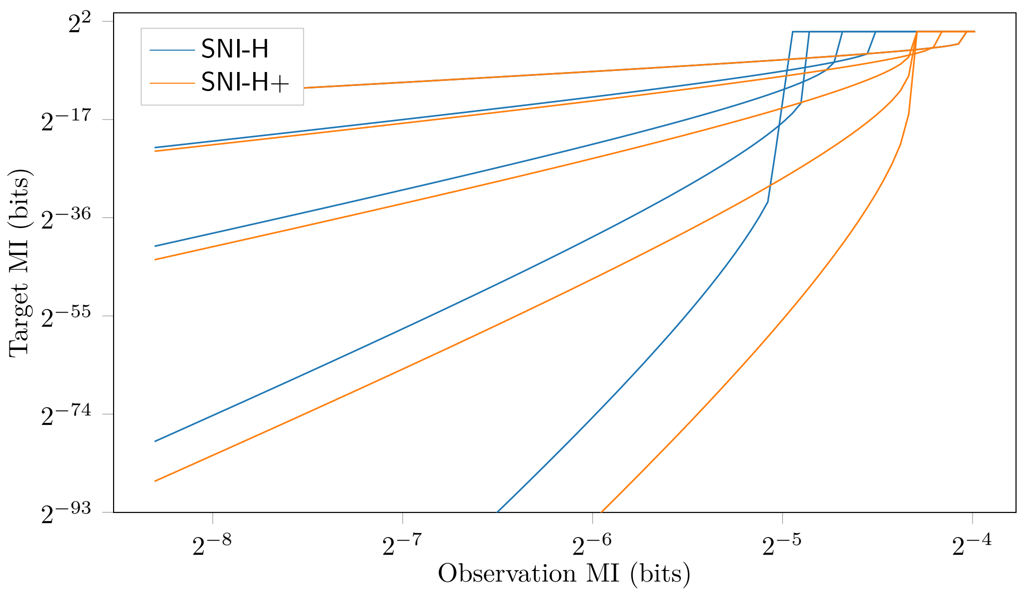


Figure 6: Target MI as a function of observation MI, for the SNI-H and SNI-H+ multiplication gadgets at orders $t = 1, 3, 7, 15$ and 31.

Improved SNI gadgets. The representation of the SNI-H tree (Figure 5b) compared to the tree of SNI-ISW (Figure 5a) suggest a natural improvement (from the security against horizontal attacks viewpoint) where more refresh gadgets are added, that we denote as SNI-H+ (Figure 5c). It instantiates Algorithm 2 with `Refresh1` and `Refresh2` as SNI refresh gadgets. (see again Table 1). In this case, the number of manipulations of each variable is independent of the number of shares t, hence we can expect lower (possibly constant) MI_t. The security of this gadget is shown in Figure 6. We observe that the curves for SNI-H+ are shifted to the right compared to the SNI-H curves. Furthermore, whereas the boundary between the two regions of the SNI-H curves shifts to the left as the order increases (in a $\mathcal{O}(\log(t))$ manner), the location of the boundary is almost constant for SNI-H+ (except for a small shift at orders $t \leq 6$).

An interesting note here is that the SNI-H+ gadget is the first one (in $\mathbb{F}_2$) for which increasing the order increases the security level for all values of MI_o and large enough t (i.e., excepted for $t \leq 6$ and $\mathrm{MI}_o \geq 5 \times 10^{-2}$ bit). This confirms the intuition that having a constant number of share manipulations translates into having a constant MI_o noise requirement: the location of the boundary between the two regions of the graph is indeed almost independent of the order in this case. The latter observation can be connected to the concept of noise rate [ADF16]: the SNI-ISW multiplication has a $\mathcal{O}(1/t)$ noise rate, SNI-H exhibits a behavior of $\mathcal{O}(1/\log(t))$ noise rate, and we conjecture that SNI-H+ (and also SNI-H*, p. 180) has an effective $\mathcal{O}(1)$ noise rate.

Improved PINI gadgets. The idea of "internal refreshes" used for SNI-H+ can be combined with the "masked shares' multiplication" trick of the PINI gadgets. Applying this combination to the PINI_2 multiplication gadget leads to a PINI_2-H+ gadget with improved security against horizontal attacks. A similar idea can be applied to the PINI_1 gadget, but requires some slight modifications, leading to a new algorithm that we denote as PINI_3-H+

(given in Algorithm 3).[12] The PINI$_3$ and PINI$_3$-H variants are less interesting and will not be discussed in detail.

Algorithm 3 Generalized PINI$_3$ multiplication gadget.

```
Require: shared factors $x, y \in \mathbb{F}_q^d$ such that $\sum_i x_i = x^*$ and $\sum_i y_i = y^*$
Ensure: output $c \in \mathbb{F}_q^d$ such that $\sum_i c_i = x^* \cdot y^*$
  $M \leftarrow \texttt{MatGen}((x_0, \dots, x_t), (y_0, \dots, y_t))$; {See Algorithm 2.}
  for $i = 0$ to $t$ do
    for $j = i + 1$ to $t$ do
      $r_{i,j} \xleftarrow{\$} \mathbb{F}_q$;
      $r_{j,i} \leftarrow r_{i,j}$;
  for $i = 0$ to $t$ do
    for $j = 0$ to $t$ do
      $(x_{i,j}, y_{i,j}) \leftarrow (M)_{i,j}$;
      if $i \neq j$ then
        $z_{i,j} \leftarrow (x_{i,j} \cdot r_{i,j} + r_{i,j}) + x_{i,j} \cdot (y_{i,j} + r_{i,j})$;
Ensure:         $z_{i,j} = r_{i,j} + x_{i,j} \cdot y_{i,j}$
  for $i = 0$ to $t$ do
    $c_i \leftarrow x_{i,i} \cdot y_{i,i} + \sum_{j=0, j\neq i}^{t} z_{i,j}$;
```

In Figure 7, we observe that the evolutions of PINI$_1$ and PINI$_2$ into PINI$_3$-H+ and PINI$_2$-H+ follow the same trend of shifting information curves to the right (as it happens for the evolution from SNI-ISW into SNI-H+), while preserving their other distinguishing features. In particular, the PINI$_2$-H+ curve still has the "staircase of asymptotes" aspect which reduces the security level in the medium/large MI$_o$ region.

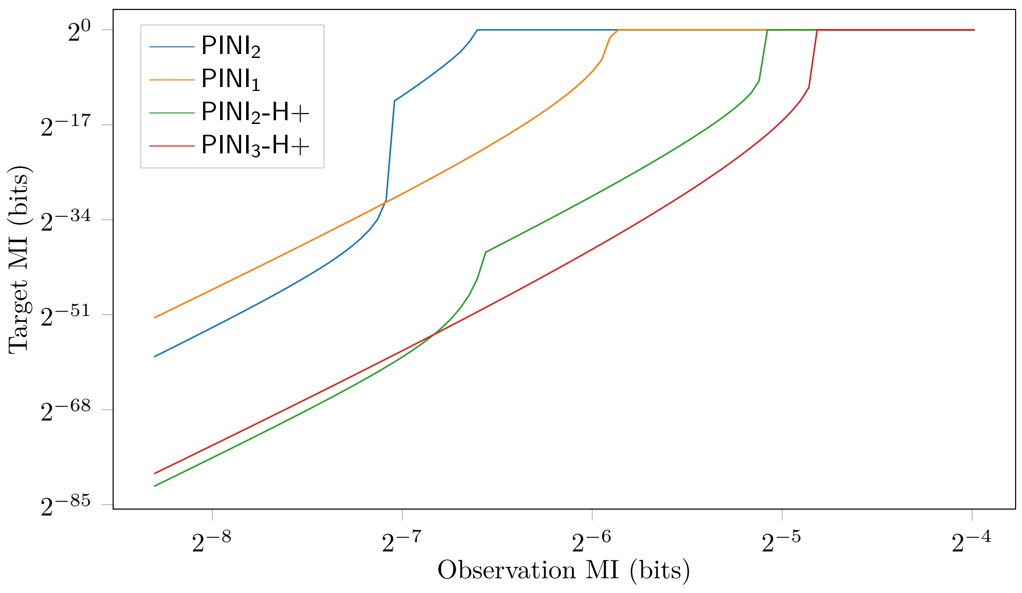


Figure 7: Target MI as a function of observation MI, for various PINI multiplication gadgets at order $t = 15$.

[12] More precisely, the negation of the inputs $\bar{a}$ cannot be computed once for each a_i due to the use of refreshed a_i's. In order to avoid the $2d^2$ leakage on (refreshed) input shares (which decreases significantly the security level), we replace the evaluation of $r \cdot \bar{a}$ by a computation $r \cdot a + r$ (which uses the same number of arithmetic operations but trades leakage on a for leakage on r).

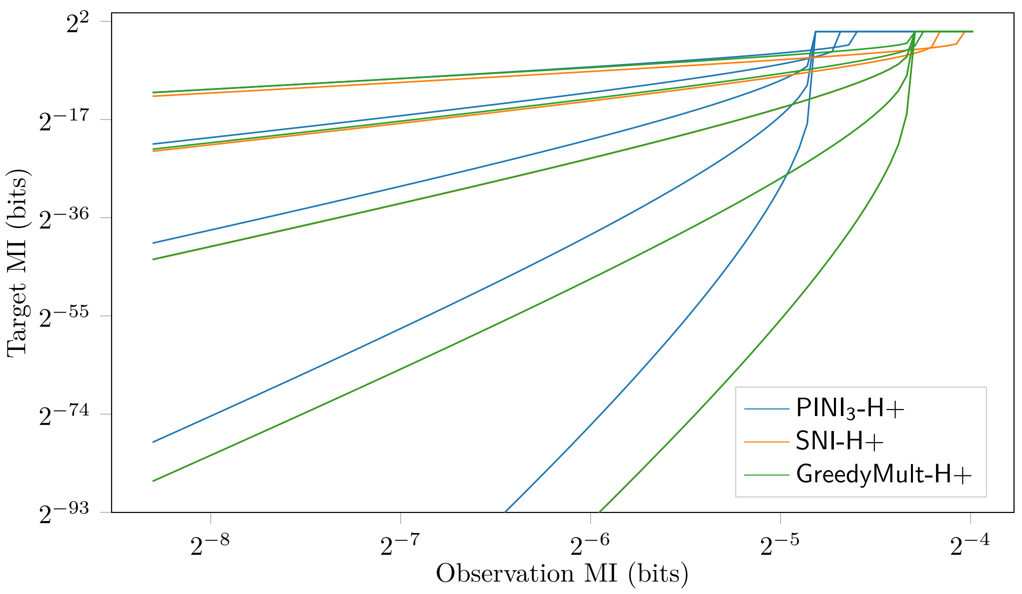


Figure 8: Target MI as a function of observation MI, for the SNI-H+, PINI$_3$-H+ and GreedyMult-H+ multiplication gadgets at orders $t = 1, 3, 7, 15, 31$. The curves for SNI-H+ and GreedyMult-H+ are superimposed for $t \geq 7$.

Figure 8, shows that the asymptotic behavior of the PINI$_3$-H+ gadget is identical to the one of SNI-H+: the location of the boundary between regions is also almost independent of the order. The main difference between the two gadgets is that the PINI$_3$-H+ curve is shifted of a factor of 2 to the left compared to SNI-H+, due to more manipulations of the shares (in the masked shares' multiplication trick), as when comparing PINI$_1$ and SNI-ISW.

We finally note that there exists a possibility to get a PINI multiplication gadget that does not suffer from this shift compared to SNI-ISW: the use of a SNI multiplication gadget for which one of the inputs is refreshed (with a SNI refresh). This technique was introduced in [GR17]. It is called "greedy strategy" and shown to be PINI in [CS18]. We can instantiate it with the SNI-ISW gadget and the refresh of Batistello et al., and denote it as GreedyMult. GreedyMult(x, y) = SNI-ISW(BatRef(x), y). Its variation GreedyMult-H+ uses the SNI-H+ gadget. The security of GreedyMult-H+ is displayed in Figure 8: it is the same as that of SNI-H+ (except for low orders and large MI$_o$).

Security vs randomness cost trade-off. Battistello et al. use a SNI refresh with randomness complexity $\mathcal{O}(t \log(t))$ in the `MatGen` stage. However, since this refresh is not used to prove composability in the t-probing model, the SNI property is not strictly required.[13] We hence analyze the case where this SNI refresh (Algorithm 6 of [BCPZ16]) is replaced with a simple refresh using t random bits (Gadget 1 of [BBD^{+}16]). The latter leads to new gadgets SNI-H* and PINI$_3$-H*, which are adapted versions of SNI-H+ and PINI$_3$-H+, (as per Table 1). We observe in Figure 9 that the security level of the these new gadgets is almost the same as the one of the gadgets based on the refresh of Battistello et al.

Regarding the randomness cost of the gadgets, they all have a complexity in $\mathcal{O}(t^2)$ (this is immediately visible for the `Compression` stage, and proven in Appendix D for the H+ `MatGen`), but the use of the H* `MatGen` versus the H+ one results in a reduced randomness cost of about 50 %. We defer a detailed performance comparison to Section 5.

[13] We note that this property might help to formally prove the countermeasure in the RPM.

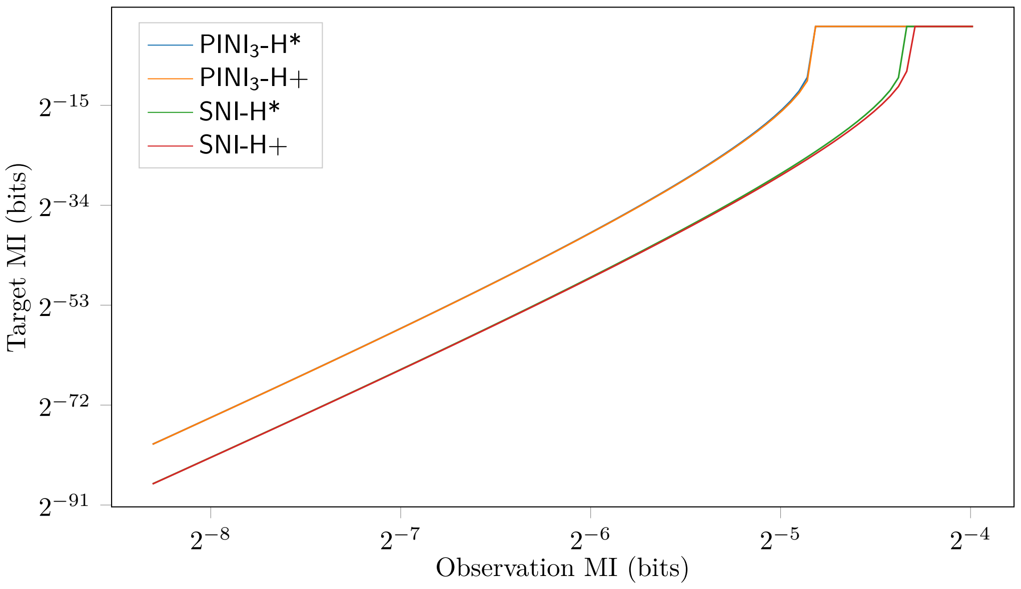


Figure 9: Target MI as a function of observation MI, for the SNI-H+, SNI-H*, PINI$_3$-H+ and PINI$_3$-H* multiplication gadgets at order $t = 15$.

4.5 A note on SASCA and factor graphs

Before discussing performance comparisons in detail, we empirically validate the argument of Section 4.1 that adding equations about all the input/output relationships of refresh gadgets is actually needed to perform worst-case SASCA. As illustrated in Figure 10, removing those relationships from the factor graph gives a less informative "SNI-H+ naive" curve.[14] We observe that the naive attack causes a significant loss to the adversary, exhibited by a shift of the curve to the left, by a factor that depends on the order and the type of protection against horizontal attacks considered.

We conclude that this technique should be used when analyzing SASCA in the LRPM, since it can lead to significant reductions of the worst-case complexity and does not add stronger hypotheses on the capabilities of the adversary. The only potential disadvantage of this technique is that it adds cycles in the factor graph, which might harm the convergence of the BP algorithm in practice (the analysis of which is a scope for further research).

5 Cost comparison and discussion

We finally evaluate the runtime cost of different multiplication gadgets in Figure 11, using the estimation framework of [CS18]. Four groups of curves can be distinguished on the plot:

- The (possibly randomness-optimized) gadgets that are not protected against horizontal attacks: SNI-ISW, PINI$_1$, PINI$_2$, GreedyMult;
- The H+ gadgets that are strongly protected against horizontal attacks with a cost up to six times larger than the SNI-ISW multiplication gadget;
- The H* gadgets, costing almost half the cost of the H+ gadgets (but with almost the same security against horizontal attacks as the H+ gadgets);

[14] The SNI-H+ gadget is the most sensitive to the removal of the input/output refresh relationships (yet, visible impact occurs for PINI$_2$-H+ and PINI$_3$-H+ as well).

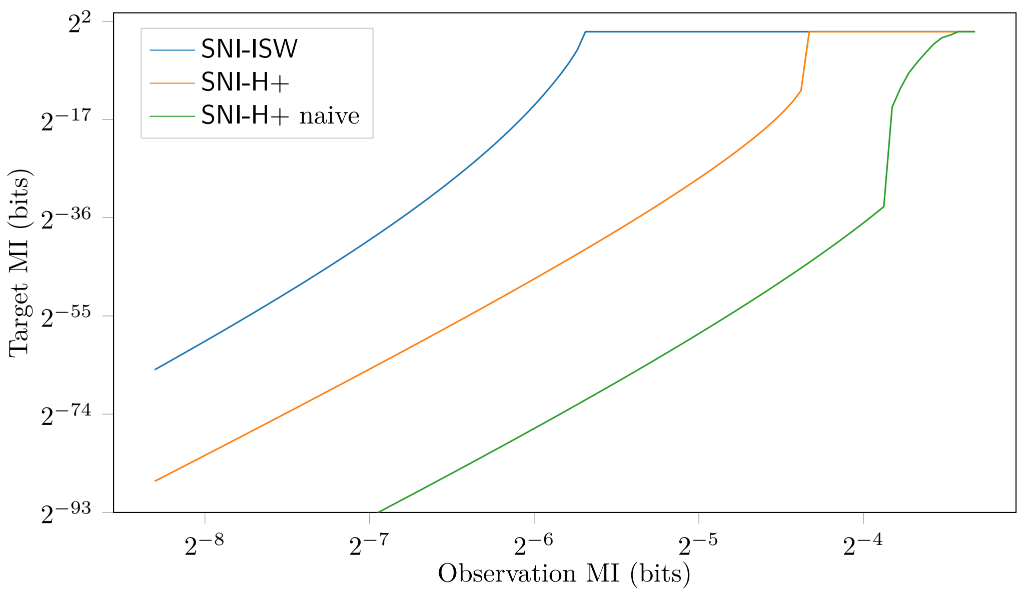


Figure 10: Target MI as a function of observation MI, for the SNI-H+ multiplication gadget at order $t = 15$. For the SNI-H+ naive curve, the target MI is extracted through the sums of the input and output shares only (naive attack). For the SNI-H+ curve, the improved attack is considered (equations about sums of refresh input/outputs are added. The SNI-ISW curve (which is the same for both attacks) is shown as a reference point.

- The H gadgets, which are only a bit less costly compared to the H* gadgets but have significantly worse resistance to horizontal attacks.

Combined with the previous observation that security against horizontal attacks gains relevance as the security order increases, the figure first suggests that the H and H+ gadget families are in general less interesting than the (new) H* gadget family which combines the advantages of both. Namely, it has roughly the same runtime cost as the H family, and ensures almost the same security against horizontal atttacks as the H+ family.

Therefore, it remains two main options for concrete applications. If security against horizontal attacks is ignored (which is not advisable – see the discussion below), the PINI_1 and PINI_2 algorithms provide the best performances and easy composition (depending on the order). If security against horizontal attacks is required (which should be the default situation at high orders), one can choose between the GreedyMult-H* and PINI_3-H* gadgets if easy composition is required (the first one has better constant security against horizontal at the cost of a slightly higher runtime), or SNI-H* if the addition of refreshing gadgets can be optimized in order to guarantee safe composition.

Note that our results are limited to one multiplication gadget and their security is only analyzed in the (still heuristic) LRPM. Extending our analysis to formal proofs in the RPM and finding way to analyze full ciphers without the computational cost of extending the BP algorithm to such full ciphers (which is expensive) is an interesting open problem.

Besides, and to conclude, we recall that the relevance of horizontal attacks in practical contexts depends on the adversarial assumptions, for which we can distinguish two main cases: the open-source setting and the close-source setting. In the first one, a SASCA can be directly conducted, and the bounds we provide are therefore directly applicable. By contrast, if the target implementation is close-source, building the factor graph and identifying the leakage points for all the manipulations of each variable may be more

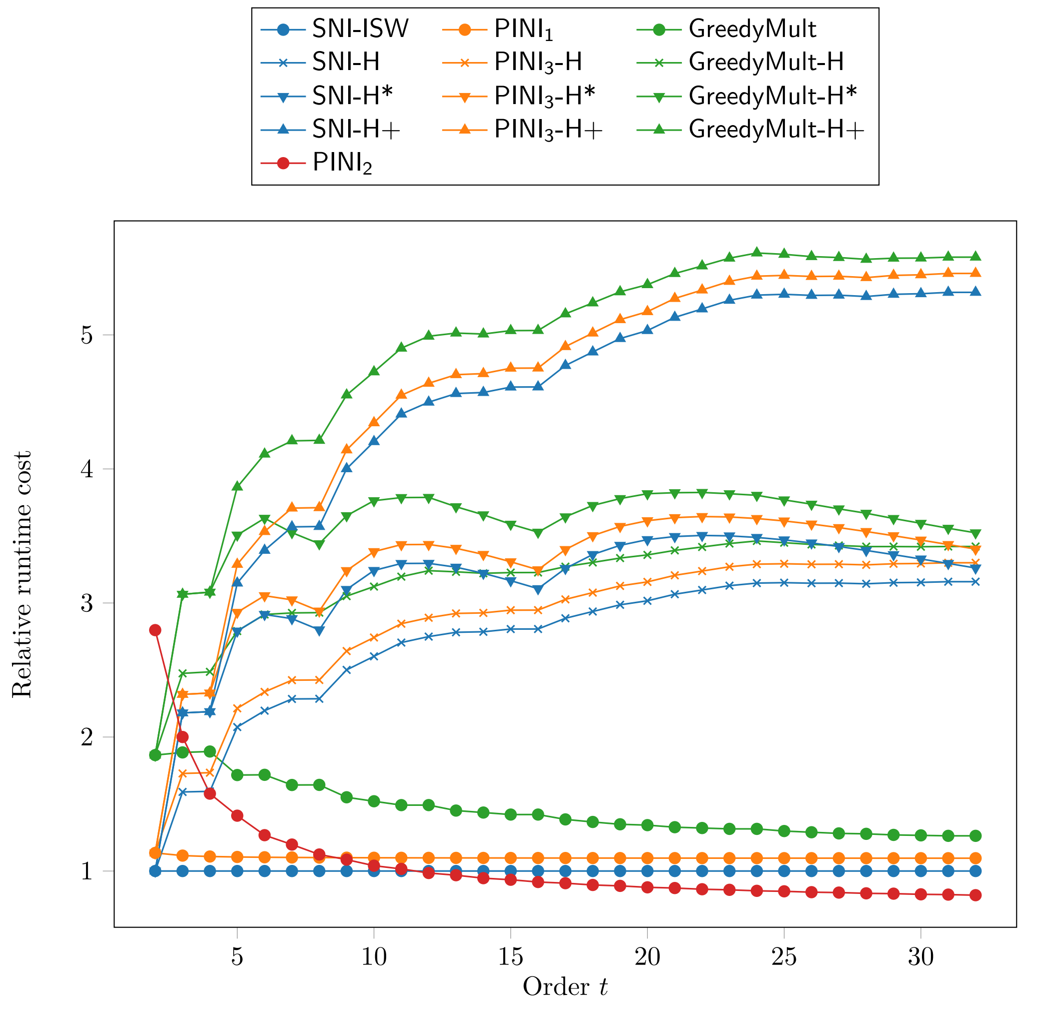


Figure 11: Runtime cost (relative to the SNI-ISW multiplication gadget) of different masked multiplication gadgets (with different noise rates).

challenging. At least, the state-of-the-art attacks in [BCPZ16, GS18] both exploit the knowledge of the source code. In this respect, we note that if an adversary can determine at least one leakage point for each input share, then he can (e.g., by simple correlation) identify the leakage points of all the manipulations of the targeted shares, even in the closed-source setting. Hence, while a worst-case SASCA exploiting all the combinations of shares (e.g., the merge trick for linear operations discussed in [GGF18]) may indeed be hard to mount, this example suggests that certain types of manipulations (e.g., repetitions) can lead to exponential security reductions even in the close-source setting.

In general, we believe the latter is a good reason to consider the security against horizontal attacks heuristically bounded in this paper (or, if possible, formal proofs in the future) in the evaluation of any implementation. Even more assuming the possibility that state-of-the-art black box attacks may continuously improve in the future. Ignoring such a threat is a risky choice since it anyway imply considering the implementation details as a long-term secret (while standard countermeasures against side-channel attacks do not protect against reverse-engineering). Yet, we note that it may be relevant for some applications with high cost constraints, where moderate / mid-term security is acceptable (and the technical difficulty of exploiting the leakages is assumed to be a sufficient barrier for the considered case studies).

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

[ADF16] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with o(1/\log (n)) leakage rate. In Fischlin and Coron [FC16], pages 586–615.

[BBD+15] 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 [OF15], pages 457–485.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BBP+16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Fischlin and Coron [FC16], pages 616–648.

[BBP+17] 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.

[BCPZ16] 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.

[BDF$^+$17] 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 [CN17], pages 535–566.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In Peyrin and Galbraith [PG18], pages 343–372.

[Cas18a] Gaëtan Cassiers. lrpm-bounder: Target MI bound computation in the LRPM. `https://github.com/cassiersg/lrpm-bounder`, June 2018.

[Cas18b] Gaëtan Cassiers. Plot runtimes of execution (on Cortex M4) of various masked AES S-boxes. `https://gist.github.com/cassiersg/5d451e1cab6b250f2e55a9d014040d6b`, November 2018.

[CJRR99] 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.

[CN17] 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.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side channel cryptanalysis of a higher order masking scheme. In 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*, pages 28–44. Springer, 2007.

[CPRR13] 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.

[CS18] Gaetan Cassiers and François-Xavier Standaert. Improved bitslice masking: from optimized non-interference to probe isolation. Cryptology ePrint Archive, Report 2018/438, 2018. `https://eprint.iacr.org/2018/438`.

[DDF14] 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.

[DFS15] 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 [OF15], pages 401–429.

[FC16] 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*. Springer, 2016.

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

[FPS17] Sebastian Faust, Clara Paglialonga, and Tobias Schneider. Amortizing randomness complexity in private circuits. In 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*, pages 781–810. Springer, 2017.

[GGF18] Qian Guo, Vincent Grosso, and François-Xavier. Modeling soft analytical side-channel attacks from a coding theory viewpoint. Cryptology ePrint Archive, Report 2018/498, 2018. `https://eprint.iacr.org/2018/498`.

[GJR18] Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain. How to securely compute with noisy leakage in quasilinear complexity. In Peyrin and Galbraith [PG18], pages 547–574.

[GR17] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Coron and Nielsen [CN17], pages 567–597.

[GS18] Vincent Grosso and François-Xavier Standaert. Masking proofs are tight and how to exploit it in security evaluations. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 385–412. Springer, 2018.

[GSF14] Vincent Grosso, François-Xavier Standaert, and Sebastian Faust. Masking vs. multiparty computation: how large is the gap for aes? *J. Cryptographic Engineering*, 4(1):47–57, 2014.

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

[JS17] 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.

[MM12] Amir Moradi and Oliver Mischke. Glitch-free implementation of masking in modern fpgas. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2012, San Francisco, CA, USA, June 3-4, 2012*, pages 89–95. IEEE, 2012.

[MOS11] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.

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

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.

[OF15] 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.

[PG18] 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*. Springer, 2018.

[PR13] 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.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung,* *Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.

A BP rule for result of multiplication

We first recall the formal context of the random probing model in which the rules of the LRPM are inferred. In the random probing model, the adversary gets for each intermediate variable X a probe P such that:

$$P = \begin{cases} \perp & \text{with probability } \epsilon \\ X & \text{with probability } 1-\epsilon \end{cases},$$

and all probes are independent.

We can thus compute the MI for this probe:

$$\begin{aligned}
\mathrm{MI}(P,X) &= \sum_p \sum_x \Pr[X=x, P=p] \log_{|\mathbb{F}|}\left(\frac{\Pr[X=x,P=p]}{\Pr[X=x]\Pr[P=p]}\right), \\
&= \sum_x \Pr[X=x,P=\perp] \log_{|\mathbb{F}|}\left(\frac{\Pr[X=x,P=\perp]}{\Pr[X=x]\Pr[P=\perp]}\right), \\
&\quad + \sum_x \Pr[X=x,P=x] \log_{|\mathbb{F}|}\left(\frac{\Pr[X=x,P=x]}{\Pr[X=x]\Pr[P=x]}\right), \\
&= (1-\epsilon)\sum_x \Pr[X=x] \log_{|\mathbb{F}|}\left(\frac{\Pr[X=x](1-\epsilon)}{\Pr[X=x]^2(1-\epsilon)}\right), \\
&= (1-\epsilon)H(X), \\
&= 1-\epsilon = \Pr[P=X],
\end{aligned}$$

assuming that X has a uniform distribution.

Let us now assume that $Y = X_1 \cdot X_2$, and that P_1, P_2 are probes on X_1, X_2 (Y is not directly probed). We write $\mathrm{MI}_1 = \mathrm{MI}(P_1, X_1)$ and $\mathrm{MI}_2 = \mathrm{MI}(P_2, X_2)$. The extrinsic information on Y is $\mathrm{MI}_Y = \mathrm{MI}(Y,(P_1,P_2))$. The value of Y is known if either $P_1 = X_1 = 0$, or $P_2 = X_2 = 0$, or $(P_1,P_2) = (X_1,X_2)$, in those cases $\mathrm{MI}_Y \leq 1$. Otherwise, there is not information on Y: $\mathrm{MI}_Y = 0$.

We hence compute

$$\begin{aligned}
\mathrm{MI}_Y \leq\ & \Pr[P_1 = X_1]\cdot(\Pr[X_1=0] + \Pr[X_1 \neq 0]\cdot\Pr[P_2=X_2]) \\
& + \Pr[P_1 \neq X_1]\cdot\Pr[P_2=X_2]\cdot\Pr[X_2=0], \\
=\ & \mathrm{MI}_1\left(\frac{1}{|\mathbb{F}|} + \left(1-\frac{1}{|\mathbb{F}|}\right)\mathrm{MI}_2\right) + (1-\mathrm{MI}_1)\,\mathrm{MI}_2\frac{1}{|\mathbb{F}|}, \\
=\ & \frac{\mathrm{MI}_1+\mathrm{MI}_2}{|\mathbb{F}|} + \mathrm{MI}_1\mathrm{MI}_2\left(1-\frac{2}{|\mathbb{F}|}\right).
\end{aligned}$$

In the bitslice case, $|\mathbb{F}| = 2$, thus $\mathrm{MI}_Y \leq (\mathrm{MI}_1 + \mathrm{MI}_2)/2$, which is the result announced in Section 2.2.

B PINI$_2$ algorithm and Multiplication of Belaid et al.

Algorithm 4 is the NI multiplication of Belaid et al. [BBP+16] where we abstracted away the computation of the products of shares and their addition to random elements. The exact multiplication of Belaid et al. can be obtained by instantiating:

- `init_cprod` doing nothing,
- `cprod` computing $t_{i,j} = r_{i,j} + x_i \cdot y_j + x_j \cdot y_i + r_{j-1} + x_i \cdot y_{j-1} + x_{j-1} \cdot y_i$,

Algorithm 4 Multiplication gadget over d shares with reduced randomness for `Compression` stage. Parameterized with three sub-algorithms `init_cprod`, `cprod` and `cprod'`.

```
Require: shared factors $x, y \in \mathbb{F}_q^d$ such that $\bigoplus_i x_i = x$ and $\bigoplus_i y_i = y$
Ensure: output $c \in \mathbb{F}_q^d$ such that $\bigoplus_i c_i = x \cdot y$
  Run init_cprod;
  for $i = 0$ to $t$ do
    for $j = 0$ to $t - i - 1$ by 2 do
      $r_{i,d-j} \xleftarrow{\$} \mathbb{F}_q$;
  for $j = t - 1$ downto 1 by 2 do
    $r_j \xleftarrow{\$} \mathbb{F}_q$;
  for $i = 0$ to $t$ do
    $c_{i,d} \leftarrow a_i \cdot b_i$;
    for $j = t$ downto $i + 2$ by 2 do
      $t_{i,j} \leftarrow$ cprod$(i, j)$;
Ensure:     $t_{i,j} = r_{i,j} + x_i \cdot y_j + x_j \cdot y_i + r_{j-1} + x_i \cdot y_{j-1} + x_{j-1} \cdot y_i$
      $c_{i,j-2} \leftarrow c_{i,j} + t_{i,j}$;
    if $i \not\equiv t \mod 2$ then
      $t_{i,j} \leftarrow$ cprod'$(i, i + 1)$;
Ensure:     $t_{i,i+1} = r_{i,i+1} + x_i \cdot y_{i+1} + x_{i+1} \cdot y_i$
      $c_{i,i} \leftarrow c_{i,i+1} + t_{i,i+1}$;
      if $i \equiv 1 \mod 2$ then
        $c_{i,0} \leftarrow c_{i,i} + r_i$;
      else
        $c_{i,0} \leftarrow c_{i,i}$;
    else
      for $j = i - 1$ downto 0 do
        $c_{i,j} \leftarrow c_{i,j+1} + r_{j,i}$;
    $c_i \leftarrow c_{i,0}$;
```

- `cprod'` computing $t_{i,j} = r_{i,j} + x_i \cdot y_j + x_j \cdot y_i$.

The PINI_2 multiplication is obtained by instantiating the generic construction as follows:

- `init_cprod` computing the terms and random elements for the masked shares' multiplication trick, as specified in Algorithm 5.
- `cprod` computing $t_{i,j} = r_{i,j}+p^0_{i,j}+p^1_{i,j}+p^2_{i,j}+p^3_{i,j}+r_{j-1}+p^0_{i,j-1}+p^1_{i,j-1}+p^2_{i,j-1}+p^3_{i,j-1}$,
- `cprod'` computing $t_{i,j} = r_{i,j} + p^0_{i,j} + p^1_{i,j} + p^2_{i,j} + p^3_{i,j}$.

Algorithm 5 `init_cprod` for PINI_2.

```
for $i = 0$ to $t$ do
  $s_i \xleftarrow{\$} \mathbb{F}_q$;
for $i = 0$ to $t$ do
  for $j = i + 1$ to $t$ do
    $s_{i,j} \leftarrow s_i + s_j$;
    $p^0_{i,j} \leftarrow a_i \cdot s_{i,j}$;
    $p^1_{i,j} \leftarrow a_i \cdot (b_j + s_{i,j})$;
    $p^2_{i,j} \leftarrow b_i \cdot s_{i,j}$;
    $p^3_{i,j} \leftarrow b_i \cdot (a_j + s_{i,j})$;
```

C PINI proof

In order to prove that the multiplication gadget described in Algorithm 4 is PINI, we present a simulation algorithm and show that it satisfies the PINI definition. The simulator algorithm is split into two parts: the first one has access to the list of adversarial probes and chooses the set of input share indices I that is sent to the oracle, and the second part uses the inputs given by the oracle to simulate the probes.

Proof idea. In order to satisfy the PINI definition, the first algorithm has to satisfy two conditions: for each adversarial probe, at most one element can be added into the output set I, and for each adversarial probe on an output wire, only the share index of the probed output wire can be added to I.[15]

To conclude the proof, we then have to show that the output of the second algorithm is indistinguishable from the actual probes. That is, to prove that the joint distribution of the inputs of the gadget and the values of the probes is identical to the joint distribution of the inputs of the gadgets and the simulated probes. For probes whose expression involves only one input variable, the simulation is simple, as the simulator can request access to the value of the input variable. For other probes, indistinguishable simulation is possible thanks to the random variables generated in the algorithm: if a probe is of the form $x + r$ where x is an expression that depends on inputs and r is a generated random variable, then $x + r$ can be perfectly simulated as an independent random variable as long as r is not involved in any other probe. If r is involved in another probe, we prove that either

- this other probe is of the form $y + r + r'$ where r' is not observed through any other probe (hence both probes can be simulated as independent random variables), or r' only appears in a probe that contains a random variable r'', which itself. . . ; or

[15] As a comparison, if the goal was to prove the NI property, the first algorithm would generate one set of share indices for each (unshared) input of the gadget (whereas for PINI there is only one set that applies to all the inputs) and add at most one element in each of those sets for each adversarial probe. Furthermore, in a NI proof, the restriction related to the probes on output wires does not exist.

- the simulator can request access to all the inputs involved in x, in which case the simulator makes the same computation as the gadget.

Relationship to a previous proof by Belaid et al. Our proof follows the strategy of the NI proof for the multiplication gadget in [BBP+16]. Indeed, our gadget uses the same randomized compression scheme as theirs: the main modification is a more complex partial products generation step. Furthermore, the PINI property is very close to the NI property. In fact, if we neglect the possibility to put probes on partial products $\alpha_{i,j} = a_i \cdot b_j$, then the proof of Belaid et al. guarantees the PINI property. Since those partial products are not present in our modified gadget, we only need to describe how to handle probes on variables that appear in the new partial products generation step to complete the proof.

This is done in Sections C.1.2 and C.2.2. For the sake of completeness, we nevertheless present a full proof below: Section C.1 describes the algorithm to build the set I and Section C.2 describes the simulation algorithm.

Classification of probes. For conciseness, we introduce here a way to name the possible probes in the gadget. The values that can be probed are: a_i, b_i, s_i, $r_{i,j}$, r_j, $s_{i,j}$, $p^k_{i,j}$, $t_{i,j}$, $c_{i,j}$, and intermediate values in the computation of $p^k_{i,j}$ and $t_{i,j}$. When discussing probes over $t_{i,j}$ and their intermediate values, we distinguish two cases: probes over intermediate values which do not involve r_{j-1} (hence sums of at most five terms, including $t_{i,i+1}$) denoted collectively $t^-_{i,j}$, and probes which involve r_{j-1} (sums of strictly more than five terms and $t_{i,j}$ for $j \neq i+1$), denoted collectively $t^+_{i,j}$. Probes over any $s_{i,j}$ or intermediate values of $p^k_{i,j}$ (including $p^k_{i,j}$ itself) are denoted collectively $p^*_{i,j}$.

C.1 Building I

C.1.1 Global probes

We cover here the probes a_i, b_i, s_i, $r_{i,j}$, r_j, $t^-_{i,j}$, $t^+_{i,j}$, $c_{i,j}$. Let I_1 be the empty set. For each adversarial probe, we next add at most one element (a share index) to I_1.

1. For any probe $c_i = c_{i,0}$, add i to I_1.[16] Then, for any other probe $c_{i,j}$, if $i \notin I_1$ then add i to I_1, else if $i > 0$ then add $i-1$ to I_1.
2. For any observed variable a_i or b_i or s_i, add i to I_1.
3. For any observed variable r_j add j to I_1.
4. For any observed variable $r_{i,j}$, if $i \notin I_1$ add i to I_1, else if $j \notin I_1$, add j to I_1. Otherwise add $j-1$ to I_1.
5. For any observed $t^-_{i,j}$, if $i \notin I_1$, add i to I_1, else if $j \notin I_1$, add j to I_1. Otherwise add $j-1$ to I_1.
6. Iterate the following until all observed $t^+_{i,j}$ have been processed.
   - Select an observed and not yet processed $t^+_{i,j}$ such that $i \in I_1$. If there is no such $t^+_{i,j}$, select an observed and not yet processed $t^+_{i,j}$ such that there are two observed $t^+_{i,j}$. If there is no such $t^+_{i,j}$, select any observed and not yet processed $t^+_{i,j}$.
   - Process the selected $t^+_{i,j}$, which means: if $j-1 \notin I_1$, add $j-1$ to I_1. Otherwise, if $i \notin I_1$, add i to I_1, otherwise add j to I_1.

[16] This step ensures that the set I_1 satisfies the PINI condition that the input shares required for a given output probe have the same share index.

C.1.2 Local probes

We cover the remaining probes: $p^*_{i,j}$.

Let I_2 be the empty set. If there are at least two probes in $p^*_{i,j}$ for a fixed pair (i, j), then add i and j to I_2, and these probes are ignored in the next (graph-building) step.

Let G be a graph whose nodes are all the share indices i. There is an edge between i and j if there is a probe on $p^*_{i,j}$ (remember that there can only be zero or one probe thanks to the previous step).

Let I_3 be the empty set. For each connex component G' of G, if G' contains a node which is in $I_1 \cup I_2$ or if G' contains a cycle then add all the nodes of G' to I_3. Otherwise, for each probe $p^*_{i,j}$ which generates and edge $\{i, j\}$ in G', add i to I_3.

The number of elements in $I_2 \cup I_3$ is at most the number of probes $p^*_{i,j}$ thanks to the following proposition.

Proposition 2. *For any connected graph with v vertices and e edges, $v \leq e + 1$ with equality only if the graph does not contain any cycle (i.e. is a tree).*

Finally, set $I = I_1 \cup I_2 \cup I_3$.

C.2 Simulation

We now prove that a simulator knowing the input shares with share index in I can perfectly simulate the probes.

We first show how to simulate the global probes (a_i, b_i, s_i, $r_{i,j}$, r_j, $t^-_{i,j}$, $t^+_{i,j}$, $c_{i,j}$), and then show how to simulate the local probes ($p^*_{i,j}$).

C.2.1 Global probes

Observations. Before simulating, we make the following observations:

(i) all variables whose expression involves $r_{i,j}$ are $r_{i,j}$, $t^-_{i,j}$, $t^+_{i,j}$, $c_{i,k}$, $c_{j,k}$.

(ii) all variables whose expression involves r_{j-1} are r_{j-1}, $t^+_{k,j}$, $c_{j-1,k}$, $c_{k,l}$.

(iii) if two variables of the form $t^+_{i,j}$ are probed, then $i, j - 1 \in I$.

(iv) if $k \in I_1$ before Step 6[17] and $t^+_{i,j}$ and $t^+_{k,j}$ are probed, then $i, k, j - 1 \in I$.

(v) if any two of i, k, j, $j - 1$ are in I_1 before Step 6 and $t^+_{k,j}$ and $t^+_{i,j}$ is probed, then $i, k, j, j - 1 \in I$.

(vi) if $k \in I_1$ or $j \in I_1$ before Step 6 and $t^+_{k,j}$ is probed and $t^+_{i,j}$ is probed twice, then $i, k, j, j - 1 \in I$.

(vii) if $t^+_{k,j}$ is probed twice and $t^+_{i,j}$ is probed twice, then $i, k, j, j - 1 \in I$.

Description of the simulation algorithm. Any probe on a_i or b_i can be assigned to the correct value. For any probe on variables s_i, r_j or $r_{i,j}$, the variable is assigned to a fresh random, as it is the case in the gadget.

For $t^-_{i,j}$ probes, if $i, j \in I$, then the probe can be computed as it is the case in the real algorithm. Otherwise, it is assigned a fresh random.

For $t^+_{i,j}$ probes, if $i, j, j - 1 \in I$, then the probe can be computed as it is the case in the real algorithm. Otherwise, it is assigned a fresh random.

For $c_{i,j}$ probes, the intermediates $t_{i,k}$, $r_{k,i}$ and r_i are simulated and the sum is computed.

[17]Numbered Steps refer to steps in Section C.1.1.

Indistinguishability proof. The only cases where the simulation is different from the real algorithm is the simulation of values $t^-_{i,j}$ for which i or j is not in I and the simulation of $t^+_{i,j}$ for which i or j or $j-1$ is not in I. In those two cases, we show that $t^\pm_{i,j}$ appear to the adversary as randoms indenpendent of any other input and probe.

Furthermore, probes on $c_{i,j}$ may involve $t_{i,k}$ for which i, k or $k-1$ is not in I. We prove that any $t_{i,k}$ can be perfectly simulated (i.e. is independent of any input and any probed except $c_{i,j}$) if i, k or $k-1$ is not in I) if $c_{i,j}$ is probed.

This proves indistinguishability of the whole simulation.

Independence of $t^-_{i,j}$. The variable $t^-_{i,j}$ (hereafter denoted t) involves the random $r_{i,j}$. We show that if $r_{i,j}$ is not independent of all the other probes and inputs, then $i, j \in I$. Using Observation (i), we analyze the possibilities for probes on values which depend on $r_{i,j}$.

- If $r_{i,j}$ is probed: Step 4 (for $r_{i,j}$) of the algorithm building I_1 adds[18] i to I and Step 5 (for t) adds j.
- If $r_{i,j}$ appears in probed $c_{i,k}$: Step 1 (for $c_{i,k}$) adds i to I and Step 5 (for t) adds j.
- If $r_{i,j}$ appears in probed $c_{j,k}$: Step 1 (for $c_{j,k}$) adds j to I and Step 5 (for t) adds i.
- If $r_{i,j}$ appears in another probed $t^-_{i,j}$ (denoted t'): Step 5 for t and t' adds i and j.
- If $r_{i,j}$ appears in a probed $t^+_{i,j}$ (denoted t'): Step 5 for t adds i to I. Since t' involves the random r_{j-1}, we have to prove that either $j \in I$ or r_{j-1} appears as independent of all the probes except t' (this implies that $r_{i,j}$ appears as independent of t' and thus t appears as an independent random variable). Observation (ii) gives the variables which depend on r_{j-1}:
  - If r_{j-1} (respectively $c_{j-1,k}$) is probed, then Step 3 (resp. Step 1) adds $j-1$ to I, hence Step 6 for t' adds j to I.
  - If a $t^+_{k,j}$ (denoted t'') is probed, then Step 6 for t' adds $j-1$ or j and Step 6 for t'' adds $j-1$, k or j.

    The variable t'' involves the random $r_{k,j}$. Using again Observation (i), we can analyze the probes which depend on $r_{k,j}$.
    * If $r_{k,j}$, $t^-_{k,j}$ or $c_{k,l}$ is probed, then k is added to I before Step 6, which implies that Step 6 for t' and t'' adds $j-1$ and j to I.
    * If another $t^+_{k,j}$ is probed (denoted t'''), then Step 6 ensures $k, j, j-1 \in I$.
    * If $c_{j,l}$ is probed, then $j \in I$.
    * Otherwise, no such variable is probed, hence t'' is seen as a random independent of r_{j-1}, which in turn implies that t' is seen as a random indepent of $r_{i,j}$ and thus t is also independent.
  - If r_{j-1} appears in a probed $c_{k,l}$ (with $k \neq i$, $k \neq j-1$: those case have already been discussed), $r_{k,j}$ also appears in $c_{k,l}$. Step 1 for $c_{k,l}$ adds k to I. We can now discuss the probes which depend on $r_{k,j}$.
    * If $r_{k,j}$ or $t^-_{k,j}$ is probed, then j is added to I.
    * If $c_{k,l'}$ is probed then either $c_{k,l}$ is a sub-sum of $c_{k,l'}$ or the other way around. In any of these situation, these sums only observe $r_{k,j} + r_{j-1}$, hence r_{j-1} is independent of these sums in the view of the adversary, since $r_{k,j}$ is not observed elsewhere. This implies that t' is seen as independent of $r_{i,j}$ and thus t also looks independent.

[18]All occurences of "adds x to I" actually mean "ensure that $x \in I$".

    * If a $t^+_{k,j}$ (denoted t'') is probed, then Step 6 for t' and t'' ensures $j, j-1 \in I$.
    * If $c_{j,l'}$ is probed, then $j \in I$.
    * Otherwise, no such variable is probed, hence $c_{k,l}$ is seen as a random independent of r_{j-1}, which in turn implies that t' is seen as a random indepent of $r_{i,j}$ and thus t is also independent.
  - Otherwise, no such variable is probed, hence t' is seen as a random independent of $r_{i,j}$, which in turn implies that t is seen as independent of any probe or input.

- Otherwise, $r_{i,j}$ is only observed through t, hence t is independent of all probes and inputs.

Independence of $t^+_{i,j}$. The variable $t^+_{i,j}$ (hereafter denoted t) involves the randoms $r_{i,j}$ and r_{j-1}. We show that if none of $r_{i,j}$ and r_{j-1} is independent of all the other probes and inputs, then $i, j, j-1 \in I$ (which means that the simulator knows all the inputs involved in t). Using Observations (i) and (ii), we analyze the possibilities for probes on values which depend on $r_{i,j}$ and r_{j-1}, assuming that both appear in at least one probe in addition to t (otherwise t is an independent random variable).

We look at probes which involve r_{j-1} (using Observation (ii)):

- If r_{j-1} or $c_{j-1,k}$ is probed, then Step 3 or 1 adds $j-1$ to I. We then analyze to possible probes involving $r_{i,j}$:
  - If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ is probed, then $i \in I$ before Step 6, and Step 6 (for t) adds j.
  - If $c_{j,k'}$ is probed, then $j \in I$ before Step 6, and Step 6 (for t) adds i.
  - If another $t^+_{i,j}$ is probed, then Observation (vi) applies.

- If there is a $t^+_{k,j}$ probe (denoted t'), then this probe involves $r_{k,j}$. Let us analyze the possible probes involving this variable:
  - If $r_{k,j}$, $t^-_{k,j}$ or $c_{k,l}$ is probed, then $k \in I$ after Step 5. We then analyze the possible probes for $r_{i,j}$:
    * If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or $c_{j,k'}$ is probed, then Observation (v) applies: $i, j, j-1 \in I$.
    * If $t^+_{i,j}$ is probed, then Observation (vi) applies: $i, j, j-1 \in I$.
  - If another $t^+_{k,j}$ is probed, then there are two $t^+_{k,j}$ probes. We then analyze the possible probes for $r_{i,j}$:
    * If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or $c_{j,k'}$ is probed, then i or j is in I before 6 and Observation (vi) applies: $i, j, j-1 \in I$.
    * If $t^+_{i,j}$ is probed, then Observation (vii) applies: $i, j, j-1 \in I$.
  - If $c_{j,l}$ is probed, then Step 1 adds j to I. We then analyze the possible probes for $r_{i,j}$:
    * If $r_{i,j}$, $t^-_{i,j}$ or $c_{i,k}$ is probed, then $i, j \in I$ before Step 6, hence Step 6 for t adds $j-1$ to I.
    * If $t^+_{i,j}$ is probed, then Observation (iii) implies $i, j, j-1 \in I$.
    * If $c_{j,k'}$ is probed, then after Step 1, $j, j-1 \in I$. Step 6 adds i to I.
  - Otherwise, $r_{k,j}$ is observed only through t', hence r_{j-1} is independent of t'.

- If $c_{k,l}$ is probed, then $k \in I$ after Step 1 and this probe involves $r_{k,j}$. Let use analyze the possible probes involving $r_{k,j}$:
  - If $r_{k,j}$ or $t^-_{k,j}$ is probed, then $j \in I$ after Step 5. We then analyze the possible probes for $r_{i,j}$:

    * If $r_{i,j}$, $t^-_{i,j}$ or $c_{i,k}$ is probed, then $i, j \in I$ before Step 6, hence Step 6 for t adds $j-1$ to I.
    * If $t^+_{i,j}$ is probed, then Observation (iii) implies $i, j, j-1 \in I$.
    * If $c_{j,k'}$ is probed then after Step 1, $j, j-1 \in I$. Step 6 adds i to I.
  - If $t^+_{k,j}$ is probed, again analyzing possible probes depending on $r_{i,j}$:
    * If $r_{i,j}$, $t^-_{i,j}$, $c_{i,k}$ or $c_{j,k'}$ is probed, then Observation (v) applies: $i, j, j-1 \in I$.
    * If $t^+_{i,j}$ is probed, then Observation (vi) applies.
  - If $c_{j,l}$ is probed, then $j \in I$ before Step 6. We then analyze the possible probes for $r_{i,j}$:
    * If $r_{i,j}$, $t^-_{i,j}$ or $c_{i,k}$ is probed, then $i, j \in I$ before Step 6, hence Step 6 for t adds $j-1$ to I.
    * If $t^+_{i,j}$ is probed, then Observation (iii) implies $i, j, j-1 \in I$.
    * If $c_{j,k'}$ is probed then after Step 1, $j, j-1 \in I$. Step 6 adds i to I.
  - Otherwise, $r_{k,j}$ is observed only through $c_{k,l}$, hence r_{j-1} is independent of $c_{k,l}$.

Independence of $t_{i,j}$ if $c_{i,k}$ is probed. If $c_{i,k}$ is probed, then $i \in I$ (before Step 4). We distinguish two cases: $j \neq i+1$ and $j = i+1$.

First, if $j = i+1$, the variable $t_{i,j} = t_{i,i+1}$ involves the random $r_{i,i+1}$. If $r_{i,i+1}$ is probed through $r_{i,i+1}$, $t^-_{i,i+1}$, $c_{i+1,l}$, then $i, i+1 \in I$ and all the inputs required to compute $t_{i,i+1}$ are known. If $r_{i,i+1}$ is probed through a $c_{i,k'}$ or not probed, then $r_{i,i+1}$ is only observed through $t_{i,i+1}$, which can thus be simulated as an independent random.

Second, if $j \neq i+1$, the variable $t_{i,j}$ involves the randoms $r_{i,j}$ and r_{j-1}. If any of those is not observed through any probe, $t_{i,j}$ is independent from any other input or probe (except possibly some $c_{i,k'}$) and can thus be independently simulated. We analyze the possible probes for r_{j-1}:

- If r_{j-1} is observed through r_{j-1}, $t^+_{k,j}$ or $c_{j-1,k'}$, then $j-1 \in I$. We then analyze the possible probes for $r_{i,j}$:
  - If $r_{i,j}$ is observed through $r_{i,j}$, $t^-_{i,j}$ or $c_{j,l}$, then $i, j, j-1 \in I$.
  - If $r_{i,j}$ is observed through $t^+_{i,j}$, then $j-1 \in I$. In this case, $r_{i,j}$ is only obseved through the sum $r_{i,j} + p^0_{i,j} + p^1_{i,j} + p^2_{i,j} + p^3_{i,j}$, which itself looks like an independent random and the inputs for the remaining part of $t_{i,j}$ ($r_{j-1} + p^0_{i,j-1} + p^1_{i,j-1} + p^2_{i,j-1} + p^3_{i,j-1}$) are known.
  - The other possibility for $r_{i,j}$ to be observed is a probe on $c_{i,l}$. In this case all observations of $r_{i,j}$ are through $t_{i,j}$, which can thus be simulated as an independent random.
- If r_{j-1} is observed through a $c_{k',l}$, then $k' \in I$ and this $c_{k',l}$ probe involves $t_{k',j}$. This variable involves in turn $r_{k',j}$. We then analyze the possible probes for $r_{k',j}$:
  - If $r_{k',j}$, $t^-_{k',j}$ or $c_{j,l'}$ is observed, then $k', i, j \in I$. We now list the possible probes for $r_{i,j}$:
    * If $r_{i,j}$, $t^-_{i,j}$, $t^+_{i,j}$ or $c_{j,l}$ is probed, then $j-1 \in I$.
    * The other possibility for $r_{i,j}$ to be observed is a probe on $c_{i,l}$. In this case all observations of $r_{i,j}$ pass through $t_{i,j}$, which can thus be simulated as an independent random.
  - If $t^+_{k',j}$ is probed, then $j-1 \in I$. Possible probes for $r_{i,j}$:
    * If $r_{i,j}$, $t^-_{i,j}$ or $c_{j,l}$ is probed, then $j \in I$.

  - $*$ If $t_{i,j}^+$ is probed, then Observation (v) applies.
  - $*$ The other possibility for $r_{i,j}$ to be observed is a probe on $c_{i,l}$. In this case all observations of $r_{i,j}$ are through $t_{i,j}$, which can then be simulated as an independent random.

- Finally, if $r_{k',j}$ is only observed through a $c_{k',l'}$ (and $c_{k',l}$), then those observations only observe $r_{k',j}$ and r_{j-1} through the sum $r_{k',j}+r_{j-1}$, which implies that r_{j-1} is indepentent from any other probe or input, which proves independence of $t_{i,j}$.

C.2.2 Local probes

In this section, we discuss how to simulate the local probes $p^*_{i,j}$ and the probes on s_i and s_j.

Description of the simulation algorithm. To simulate a probe on s_i or s_j, we assign to it a fresh random. For probes on $p^*_{i,j}$, we always have $i \in I$ thanks to the algorithm in Section C.1.2. In the case where $j \in I$ or if the probe does not involve a_j or b_j, the simulator imitates the gadget. Otherwise, the term $a_j + s_{i,j}$ or $b_j + s_{i,j}$ or $s_{i,j}$ is taken as a fresh random, and the remaining part of the computation imitates the gadget.

Indistinguishability proof. The only case where the simulation is different from the real algorithm is the simulation of a value $p^*_{i,j}$ (denoted p) which involve $a_j + s_{i,j}$ or $b_j + s_{i,j}$ or $s_{i,j}$ and for which j is not in I. In this case, we show that $a_j + s_{i,j}$, $b_j + s_{i,j}$ or $s_{i,j}$ (denoted collectively $u_{i,j}$) appears to the adversary as a random indenpendent of any input and probe (except p itself), which proves the indistinguishability. Assuming that $j \notin I$ and that some $p^*_{i,j}$ is probed, we firstly show that s_i and s_j are not probed, secondly that $u_{i,j}$ is independent of any global probe simulated in the previous section, and thirdly that all $u_{i,j}$'s that have to be simulated and for which $j \notin I$ are independent of each other and of the inputs.

Let us observe that if $p^*_{i,j}$ is probed and s_i or s_j is probed (which implies $i \in I_1$ or $j \in I_1$), then $i, j \in I$ (thanks to steps in Section C.1.2), which proves the first statement.

For the second statement, looking at the previous proof for the simulation of global probes (Section C.2.1), we observe that if a $p^*_{i,j}$ is involved in a global probe, then either $i, j \in I$, or the global probe observes only the sum $p^*_{i,j} + r$ where r is an independent random. If $j \notin I$, $p^*_{i,j}$ is thus independent of the global probes.

We now consider the third statement. Since two $p^*_{i,j}$ probes that are in distinct connected components of G (G is defined in Section C.1.2) are independent of the global probes and have no observed link between their random $s_.$ variables, they are thus independent of each other. Let us consider probes that are in the same connected component of G. This component is a tree (otherwise $i, j \in I$). Let i be a leave of the tree (which, by definition, is connected to only one edge). This means that the only observation of s_i is a probe $p^*_{i,j}$, or $p^*_{j,i}$ (denoted p) that is represented by the edge connected to i.

Re-writing the equation $s_{i,j} = s_i + s_j$ as $s_i = s_j + s_{i,j}$ and considering that s_i is never observed (except through $s_{i,j}$) puts into evidence that $s_{i,j}$ (which is the only way through which s_i and s_j are involved in p) is a random variable independent of everything except p and that s_j is a random variable independent of p. Thus, the required $u_{i,j}$ or $u_{j,i}$ can be generated as an independent random variable for the purpose of simulating p. Finally, since s_j and all the other nodes in G' are independent of s_i, we can safely remove i form G' and apply iteratively the procedure to the new tree, which concludes the proof.

D SNI-H costs $\Theta(d^2)$ random bits.

In this section, we compute preciesly the randomness complexity of the multiplication gadget of Battistello et al. (SNI-H): $\Theta(d^2)$ (previous bound was $\mathcal{O}(d^2 \log d)$).

Let R_d be the randomness cost of the refresh of Battistello et al. for a d-sharing input (where d is a power of 2). Let C_d be then randomness cost of the refreshing part of the SNI-H multiplication[19].

Inspection of the algorithm gives $C_2 = 0$ and $C_d = 4C_{d/2} + 4R_{d/2}$ (for $d > 1$). Let $C'_i = C_{2^i}$ and $R'_i = R_{2^i}$. We can rewrite the recurrence equation as $C'_i = 4C'_{i-1} + 4R'_{i-1}$ (with $C'_1 = 0$), which gives:

$$C'_i = \sum_{j=1}^{i-1} 4^{i-j} R'_j.$$

Using that $R_d = d\left(\log_2 d - \frac{1}{2}\right)$ (which can be shown by building a recurrence equation thanks to inspection of the algorithm), and hence $R'_i = \left(i - \frac{1}{2}\right) 2^i$, we find:

$$C'_i = 4^i \sum_{j=1}^{i-1} 2^{-j}\left(j - \frac{1}{2}\right) = 4^i \left(\sum_{j=1}^{i-1} j2^{-j} - \frac{1}{2}\sum_{j=1}^{i-1} 2^{-j}\right).$$

Using the identities:

$$\sum_{k=1}^{n} z^k = z\frac{1-z^n}{1-z},$$

and

$$\sum_{k=1}^{n} kz^k = z\frac{1-(n+1)z^n + nz^{n+1}}{(1-z)^2},$$

we get:

$$C'_i = 4^i\left(\frac{3}{2} - \frac{1+2i}{2^i}\right).$$

This implies:

$$\frac{1}{2}4^i \le C'_i \le \frac{3}{2}4^i,$$

which leads to:

$$\frac{1}{2}d^2 \le C_d \le \frac{3}{2}d^2.$$

and to the conclusion $C_d = \Theta(d^2)$.

We note that this computation gives also the cost of the refreshing part of SNI-H+, which is twice the refreshing cost of SNI-H.

[19] The non-refreshing part costs $d(d-1)/2$ random bits.