Amortizing Randomness Complexity in Private Circuits

Sebastian Faust[1,2], Clara Paglialonga[1,2], Tobias Schneider[1,3]

[1] Ruhr-Universität Bochum, Germany
[2] Technische Universität Darmstadt, Germany
[3]Université Catholique de Louvain, Belgium
{sebastian.faust,clara.paglialonga}@crisp-da.de
tobias.schneider@uclouvain.be

Abstract. Cryptographic implementations are vulnerable to Side Channel Analysis (SCA), where an adversary exploits physical phenomena such as the power consumption to reveal sensitive information. One of the most widely studied countermeasures against SCA are masking schemes. A masking scheme randomizes intermediate values thereby making physical leakage from the device harder to exploit. Central to any masking scheme is the use of randomness, on which the security of any masked algorithm heavily relies. But since randomness is very costly to produce in practice, it is an important question whether we can reduce the amount of randomness needed while still guaranteeing standard security properties such as t-probing security introduced by Ishai, Sahai and Wagner (CRYPTO 2003). In this work we study the question whether internal randomness can be re-used by several gadgets, thereby reducing the total amount of randomness needed. We provide new techniques for masking algorithms that significantly reduce the amount of randomness and achieve better overall efficiency than known constructions for values of t that are most relevant for practical settings.

1 Introduction

Masking schemes are one of the most common countermeasures against physical side-channel attacks, and have been studied intensively in the last years by the cryptographic community (see, e.g., [15, 17, 10, 9, 18, 7, 12] and many more). Masking schemes prevent harmful physical side-channel leakage by concealing all sensitive information by encoding the computation carried out on the device. The most widely studied masking scheme is the Boolean masking [7, 15], which encodes each intermediate value produced by the computation using an n-out-of-n secret sharing. That is, a bit b is mapped to a bit string $(b_1, \ldots, b_n)$ such that b_i is random subject to the constraint that $\sum_i b_i = b$ (where the sum is taken in the binary field). To mask computation, the designer of a masking scheme then has to develop masked operations (so-called gadgets) that enable to compute with encodings in a secure way. The security of masking schemes is typically analyzed by carrying out a security proof in the t-probing model [15],

where an adversary that learns up to t intermediate values gains no information about the underlying encoded secret values.

While due to the linearity of the encoding function protecting linear operations is easy, the main challenge is to develop secure masked non-linear operations, and in particular a masked version of the multiplication operation. To this end, the masked multiplication algorithm internally requires additional randomness to securely carry out the non-linear operation in the masked domain. Indeed, it was shown by Belaid et al. [4] that any t-probing secure masked multiplication requires internally $O(t)$ fresh randomness. Notice that complex cryptographic algorithms typically consists of many non-linear operations that need to be masked, and hence the amount of randomness needed to protect the entire computation grows not only with the probing parameter t, but also with the number of operations that are used by the algorithm. Concretely, the most common schemes for masking the non-linear operation require $O(t^2)$ randoms, and algorithms such as a masked AES typically require hundreds of masked multiplication operations.

Unfortunately, the generation and usage of randomness is very costly in practice, and typically requires to run a TRNG or PRNG. In fact, generating the randomness and shipping it to the place where it is needed is one of the main challenge when masking schemes are implemented in practice. There are two possibilities in which we can save randomness when masking algorithms. The first method is in spirit of the work of Belaid et al. [4] who design masked non-linear operations that require less randomness. However, as discussed above there are natural lower bounds on the amount of randomness needed to securely mask the non-linear operation (in fact, the best known efficient masked multiplication still requires $O(t^2)$ randomness). Moreover, such an approach does not scale well, when the number of non-linear operations increases. Indeed, in most practical cases the security parameter t is relatively small (typically less than 10), while most relevant cryptographic algorithms require many non-linear operations. An alternative approach is to amortize randomness by re-using it over several masked operations. This is the approach that we explore in this work, which despite being a promising approach has gained only very little attention in the literature so far.

On amortizing randomness. At first sight, it may seem simple to let masked operations share the same randomness. However, there are two technical challenges that need to be addressed to make this idea work. First, we need to ensure that when randomness is re-used between multiple operations it does not cancel out accidentally during the masked computation. As an illustrative example suppose two secret bits a and b are masked using the same randomness r. That is, a is encoded as $(a+r, r)$ and b is encoded as $(b+r, r)$ (these may, for instance, be outputs of a masked multiplication). Now, if at some point during the computation the algorithm computes the sum of these two encodings, then the randomness cancels out, and the sensitive information $a + b$ can be attacked (i.e., it is not protected by any random mask). While this issue already occurs when $t = 1$, i.e., the adversary only learns one intermediate value, the situation gets much

more complex when t grows and we want to reduce randomness between multiple masked operations. In this case, we must guarantee that the computation happening in the algorithm does not cancel out the randomness, but also we need to ensure that any set of t intermediate values produced by the masked algorithm does not allow the adversary to cancel out the (potentially shared) randomness. Our main contribution is to initiate the study of masking schemes where multiple gadgets share randomness, and show that despite the above challenges amortizing randomness over multiple operations is possible and can lead in certain cases to significantly more efficient masked schemes. We provide a more detailed description of our main contributions in the next section.

1.1 Our contributions

Re-using randomness for $t > 1$. We start by considering the more challenging case when $t > 1$, i.e., when the adversary is allowed to learn multiple intermediate values. As a first contribution we propose a new security notion of gadgets that we call t–SCR which allows multiple gadgets (or blocks of gadgets) to securely re-use randomness. We provide a composition result for our new notion and show sufficient requirements for constructing gadgets that satisfy our new notion. To this end, we rely on ideas that have been introduced in the context of threshold implementations [6].

Blocks of gadgets which re-use randomness. Our technique for sharing randomness between multiple gadgets requires to structure a potentially complex algorithm into so-called blocks, where the individual gadgets in these blocks share their random bits. The randomness of a circuit is not totally reused. Indeed, the last gadget in each of these blocks is usually refreshed, in order to guarantee independence among the blocks. Moreover, an extra refreshing scheme is introduced, when the inputs of gadgets sharing randomness have dependent masks.

Re-using randomness for $t = 1$. We design a new scheme that achieves security against one adversarial probe and requires only 2 randoms for arbitrary complex masked algorithms. Notice that since randomness can cancel out when it is re-used such a scheme needs to be designed with care, and the security analysis cannot rely on a compositional approach such as the 1-SNI property [2].[1] Additionally, we provide a counterexample that securing arbitrary computation with only one random is not possible if one aims for a general countermeasure.

Implementation results. We finally complete our analysis with a case study by applying our new countermeasures to masking the AES algorithm. Our analysis shows that for orders up to $t = 5$ (resp. $t = 7$ for a less efficient TRNG) we can not only significantly reduce the amount of randomness needed, but also improve on efficiency. We also argue that if we could not use a dedicated TRNG

[1] The compositional approach of Barthe et al. [2] requires that all gadgets use independent randomness

(which would be the case for most inexpensive embedded devices), then our new countermeasure would outperform state-of-the-art solutions even up to $t = 9$. We leave it as an important question for future research to design efficient masking schemes with shared randomness when $t > 9$.

1.2 Related work

Despite being a major practical bottleneck, there has been surprisingly little work on minimizing the amount of randomness in masking schemes. We already mentioned the work of Belaid et al. [4], which aim on reducing the amount of randoms needed in a masked multiplication. Besides giving lower bounds on the minimal amount needed to protect a masked multiplication, the authors also give new constructions that reduce the concrete amount of randomness needed for a masked multiplication. However, the best known construction still requires randomness that is quadratic in the security parameter. Another approach for reducing the randomness complexity of first-order threshold implementations of Keccak was also investigated in [5].

From a practical point of view, the concept of "recycled" randomness was briefly explored in [1]. The authors practically evaluated the influence of reusing some of the masks on their case studies and concluded that in some cases the security was reduced. However, these results do not negatively reflect on our methodology as their reuse of randomness lacked a formal proof of security.

From a theoretical point of view it is known that any circuit can be masked using polynomial in t randoms (and hence the amount of randoms needed is independent from the size of the algorithm that we want to protect). This question was studied by Ishai et al. [14]. The constructions proposed in these works rely on bipartite expander graphs and are mainly of interests as feasibility results (i.e., they become meaningful when t is very large), while in our work we focus on the practically more relevant case when t takes small values.

Finally, we want to conclude by mentioning that while re-using randoms is not a problem for showing security in the t-probing model, and hence for security with respect to standard side-channel attacks, it may result in schemes that are easier to attack by so-called horizontal attacks [3]. Our work opens up new research directions for exploring such new attack vectors.

2 Preliminaries

In this section we recall basic security notions and models that we consider in this work. In the following we will use bold and lower case to indicate vectors and bold and upper case for matrices.

2.1 Private Circuits

The concept of *private circuits* was introduced in the seminal work of Ishai et al. [15]. We start by giving the definition of *deterministic* and *randomized circuit*, as

provided by Ishai et al. A *deterministic circuit* C is a direct acyclic graph whose vertices are Boolean gates and whose edges are wires. A *randomized circuit* is a circuit augmented with random-bit gates. A random-bit gate is a gate with fan-in 0 that produces a random bit and sends it along its output wire; the bit is selected uniformly and independently. As pointed out in [14], a *t-private circuit* is a randomized circuit which transforms a randomly encoded input into a randomly encoded output while providing the guarantee that the joint values of any t wires reveal nothing about the input. More formally a *private circuit* is defined as follows.

Definition 1 (Private circuit [14]). *A* private circuit *for* $f : \mathbb{F}_2^{m_i} \to \mathbb{F}_2^{m_o}$ *is defined by a triple* (I, C, O)*, where*

- $I : \mathbb{F}_2^{m_i} \to \mathbb{F}_2^{\hat{m}_i}$ *is a randomized input encoder;*
- C *is a randomized Boolean circuit with input in* $\mathbb{F}_2^{\hat{m}_i}$*, output in* $\mathbb{F}_2^{\hat{m}_o}$ *and uniform randomness* $r \in \mathbb{F}_2^n$
- $O : \mathbb{F}_2^{\hat{m}_o} \to \mathbb{F}_2^{m_o}$ *is an output decoder*

C is said to be a t-private implementation of f with encoder I and decoder O if the following requirements hold:

- *Correctness*: For any input $w \in \mathbb{F}_2^{m_i}$ we have $\Pr[O(C(I(w), \rho)) = f(w)] = 1$, where the probability is over the randomness of I and ρ;
- *Privacy*: For any $w, w' \in \mathbb{F}_{m_i}$ and any set $\mathcal{P}$ of t wires (also called *probes*) in C, the distributions $C_{\mathcal{P}}(I(w), \rho)$ and $C_{\mathcal{P}}(I(w'), \rho)$ are identical, where $C_{\mathcal{P}}$ denotes the set of t values on the wires from $\mathcal{P}$ (also called *intermediate values*).

The goal of a *t-limited attacker*, i.e. an attacker who can probe at most t wires, is then to find a set of probes $\mathcal{P}$ and two values $w, w' \in \mathbb{F}_2^{m_i}$ such that the distributions $C_{\mathcal{P}}(I(w), \rho)$ and $C_{\mathcal{P}}(I(w'), \rho)$ are not the same.
Privacy of a circuit is defined by showing the existence of a *simulator*, which can simulate the adversary's observations without having access to any internal values of the circuit.

According to the description in [15], the input encoder I maps every input value x into n binary values $(r_1, \ldots, r_n)$ called *shares* or *mask*, where the first $n-1$ values are chosen at random and $r_n = x \oplus r_1 \oplus \cdots \oplus r_{n-1}$. On the other hand, the output decoder O takes the n bits $y_1, \ldots, y_n$ produced by the circuit and decodes the values in $y = y_1 \oplus \cdots \oplus y_n$. In its internal working a private circuit is composed by *gadgets*, namely transformed gates which perform functions which take as input a set of masked inputs and output a set of masked outputs. In particular, we distinguish between linear operations (e.g. XOR), which can be performed by applying the operation to each share separately, and non-linear functions (e.g. AND), which process all the shares together and make use of additional random bits. A particular case of randomized gadget is the *refreshing* gadget, which takes as input the sharing of a value x and outputs randomized sharing of the same x. Another interesting gadget is the multiplicative one, which takes as input two values, say a and b shared in $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$, and

outputs a value c shared in $(c_1, \ldots, c_n)$ such that $\bigoplus_{i=1}^{n} c_i = a \cdot b$. We indicate in particular with $\mathsf{g}(x, \boldsymbol{r})$ a gadget which takes as input a value x and internally uses a vector $\boldsymbol{r}$ of random bits, where $\boldsymbol{r}$ is of the form $(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n)$ and each $\boldsymbol{r}_i$ is the vector of the random bits involved in the computation of the i-th output share. For example, referring to Algorithm 6, $\boldsymbol{r}_1$ is the vector (r_1, r_7, r_{13}, r_8). In the rest of the paper, if not needed otherwise, we will mainly specify a gadget with only its random component $\boldsymbol{r}$, so it will be indicated as $\mathsf{g}(\boldsymbol{r})$. Moreover, we suppose that all the gadgets $\mathsf{g}(\boldsymbol{r})$ are such that every intermediate value used in the computation of the i-th output share contains only random bits in $\boldsymbol{r}_i$.

The following definitions and lemma from [2] formalize t-probing security with the notion of *t-Non Interference* and show that this is also equivalent to the concept of *simulatability*.

Definition 2 ($(\mathcal{S}, \Omega)$-Simulatability, $(\mathcal{S}, \Omega)$-Non Interference). *Let g be a gadget with m inputs $(a^{(1)}, \ldots, a^{(m)})$ each composed by n shares and Ω be a set of t adversary's observations. Let $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_m)$ be such that $\mathcal{S}_i \subseteq \{1, \ldots, n\}$ and $|\mathcal{S}_i| \leq t$ for all i.*

1. *The gadget g is called* $(\mathcal{S}, \Omega)$-simulatable *(or $(\mathcal{S}, \Omega)$–SIM) if there exists a simulator which, by using only $(a^{(1)}, \ldots, a^{(m)})_{|\mathcal{S}} = (a^{(1)}_{|\mathcal{S}_1}, \ldots, a^{(m)}_{|\mathcal{S}_m})$ can simulate the adversary's view, where $a^{(k)}_{|\mathcal{S}_j} := (a^{(k)}_i)_{i \in \mathcal{S}_j}$.*
2. *The gadget g is called* $(\mathcal{S}, \Omega)$-Non Interfering *(or $(\mathcal{S}, \Omega)$–NI) if for any $\boldsymbol{s_0}, \boldsymbol{s_1} \in (\mathbb{F}_2^m)^n$ such that $\boldsymbol{s_0}_{|\mathcal{S}} = \boldsymbol{s_1}_{|\mathcal{S}}$ the adversary's views of g respectively on input $\boldsymbol{s}_0$ and $\boldsymbol{s}_1$ are identical, i.e. $\mathsf{g}(\boldsymbol{s}_0)_{|\Omega} = \mathsf{g}(\boldsymbol{s}_1)_{|\Omega}$.*

In the rest of the paper, when we will talk about *simulatability* of a gadget we will implicitly mean that for every observation set Ω with $|\Omega| \leq t$, where t is the security order, there exists a set $\mathcal{S}$ as in Definition 2 such that the gadget is $(\mathcal{S}, \Omega)$–SIM.

Lemma 1. *For every gadget g with m inputs, set $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_m)$, with $\mathcal{S}_i \subseteq \{1, \ldots, n\}$ and $|\mathcal{S}_i| \leq t$, and observation set Ω, with $|\Omega| \leq t$, g is $(\mathcal{S}, \Omega)$–SIM if and only if g is $(\mathcal{S}, \Omega)$–NI, with respect to the same sets $(\mathcal{S}, \Omega)$.*

Definition 3 (t–NI). *A gadget g is* t-non-interfering *(t–NI) if and only if for every observation set Ω, with $|\Omega| \leq t$, there exists a set $\mathcal{S}$, with $|\mathcal{S}| \leq t$, such that g is $(\mathcal{S}, \Omega)$–NI.*

When applied to composed circuits, the definition of t–NI is not enough to guarantee the privacy of the entire circuit. Indeed, the notion of t–NI is not sufficient to argue about secure composition of gadgets. In [2], Barthe et al. introduced the notion of *$t-$Strong Non-Interference* (t–SNI), which allows for guaranteeing a secure composition of gadgets.

Definition 4 ($t-$Strong Non-Interference). *An algorithm $\mathcal{A}$ is* $t-$Strong Non-Interferent *(t–SNI) if and only if for any set of t_1 probes on intermediate values and every set of t_2 probes on output shares with $t_1 + t_2 \leq t$, the totality of the probes can be simulated by only t_1 shares of each input.*

Informally, it means that the simulator can simulate the adversary's view, using a number of shares of the inputs that is independent from the number of probed output wires. An example of t–SNI multiplication algorithm is the famous ISW scheme in Algorithm 1, introduced in [15] and proven to be t–SNI in [2], and a t–SNI refreshing scheme is Algorithm 2, introduced in [10] by Duc et al. and proven to be t–SNI by Barthe et al. in [2].

Algorithm 1 ISW multiplication algorithm with $n \geq 2$ shares.

Input: shares $(a_i)_{1 \leq i \leq n}$ and $(b_i)_{1 \leq i \leq n}$, such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$.
Output: shares $(c_i)_{1 \leq i \leq n}$, such that $\bigoplus_i c_i = a \cdot b$.

for $i = 1$ to n **do**
  for $j = i + 1$ to n **do**
    $r_{i,j} \xleftarrow{\$} \mathbb{F}_2$;
    $r_{j,i} \leftarrow (r_{i,j} + a_i \cdot b_j) + a_j \cdot b_i$;
  end for
end for
for $i = 1$ to n **do**
  $c_i \leftarrow a_i \cdot b_i + \sum_{j=1, j \neq i}^{n} r_{i,j}$;
end for

Algorithm 2 Refreshing $\mathcal{R}$

Input: shares $(a_i)_{1 \leq i \leq n}$, such that $\bigoplus_i a_i = a$; random shares $(r_{ij})_{1 \leq i \leq n, i+1 \leq j \leq n}$.
Output: shares $(c_i)_{1 \leq i \leq n}$, such that $\bigoplus_i c_i = a$.

for $i = 1$ to n **do**
  $c_i = a_i$;
end for
for $i = 1$ to n **do**
  for $j = i + 1$ to n **do**
    $c_i = c_i + r_{i,j}$
    $c_j = c_j - r_{i,j}$
  end for
end for

As pointed out in [18] and [9], secure multiplication schemes, like ISW, require that the two masks in input are mutually *independent*. This condition is satisfied in two cases: when at least one of the two inputs is taken uniformly at random or when at least one of the two inputs is refreshed by means of a secure refreshing using completely fresh and independent randomness, as shown in Algorithm 2. In this paper, whenever we talk about independence of two inputs, we refer to the mutual independence of the masks, as specified above.

2.2 Threshold Implementation

As shown in [11] and [18], the probing model presented in the last section covers attacks such as the High Order Differential Power Analysis (HO-DPA) attack. The latter, introduced by Kocher et al. in [16], uses power consumption measurements of a device to extract sensitive information of processed operations. The following result from [6] specifies the relation between the order of a DPA attack and the one of a probing attack.

Lemma 2 ([6]). *The attack order in a Higher-order DPA corresponds to the number of wires that are probed in the circuit (per unmasked bit).*

Threshold Implementation (TI) schemes are a $t-$order countermeasure against DPA attacks. It is based on secret sharing and multi party computation, and in addition takes into account physical effects such as glitches.

In order to implement a Boolean function $f : \mathbb{F}_2^{m_i} \to \mathbb{F}_2^{m_o}$, every input value x has to be split into n shares $(x_1, \ldots, x_n)$ such that $x = x_1 \oplus \cdots \oplus x_n$, using the same procedure seen in private circuits. We denote with $\boldsymbol{C}$ is the output distribution $f(\boldsymbol{X})$, where $\boldsymbol{X}$ is the distribution of the encoding of an input x. The function f is then shared in a vector of functions $f_1, \ldots, f_n$, called component functions, which must satisfy the following properties:

1. **Correctness**: $f(x) = \bigoplus_{i=1}^{n} f_i(x_1, \ldots, x_n)$
2. $t-$ **Non-Completeness**: any combination of up to t component functions f_i of f must be independent of at least one input share x_i.
3. **Uniformity**: the probability $\Pr(\boldsymbol{C} = \boldsymbol{c} | c = \bigoplus_{i=1}^{n} c_i)$ is a fixed constant for every $\boldsymbol{c}$, where $\boldsymbol{c}$ denotes the vector of the output shares.

The last property requires that the distribution of the output is always a random sharing of the output, and can be easily satisfied by refreshing the output shares.

TI schemes are strongly related to private circuits. First, they solve a similar problem of formalizing privacy against a t-limited attacker and moreover, as shown in [17], the TI algorithm for multiplication is equivalent to the scheme proposed by ISW.

We additionally point out that the TI aforementioned properties imply simulatability of the circuit. Indeed, if a function f satisfies properties 1 and 2, then an adversary who probes t or fewer wires will get information from all but at least one input share. Therefore, the gadget g implementing such a function is t–NI and due to Lemma 1 is simulatable.

3 Probing security with common randomness

In this section we analyze privacy of a particular set of gadgets $\mathsf{g}_1, \ldots, \mathsf{g}_d$ in which the random component is substituted by a set of bits $\boldsymbol{r} = (r_1, \ldots, r_l)$ taken at random, but reused by each of the gadgets $\mathsf{g}_1, \ldots, \mathsf{g}_d$. In particular, we introduce a new security definition, which formalizes the conditions needed in order to guarantee t-probing security in a situation where randomness is shared among the gadgets.

Definition 5 (*t*–SCR). *Let* $\boldsymbol{r}$ *be a set of random bits. We say that the gadgets* $\mathsf{g}_1(\boldsymbol{r}), \ldots, \mathsf{g}_d(\boldsymbol{r})$ *receiving each* m *inputs split into* n *shares are* $t-$secure with common randomness *(*t–SCR*) if for each set* $\mathcal{P}_i$ *of* t_i *probes on* g_i *such that* $\sum_i t_i \leq t$*, the probes in* $\mathcal{P}_i$ *can be simulated by at most* $n-1$ *shares of the input of* g_i *and the simulation is consistent with the shared random component.*

Let us introduce some notation that we will use in the rest of the paper. With the term *block of gadgets* we define a sub-circuit composed by gadgets, with input an encoding of a certain x and output an encoding of y. Since our analysis focuses on the randomness, when we refer to such a block we only consider the randomized gadgets. In particular, we indicate a block of gadgets as $\mathcal{G}(\boldsymbol{R}) = \{\mathsf{g}_1(\boldsymbol{r_1}), \ldots, \mathsf{g}_d(\boldsymbol{r_d})\}$, where the g_i represent the randomized gadgets in the block and $\boldsymbol{R} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_d})$ constitutes the total amount of randomness used by $\mathcal{G}$. We assume without loss of generality that the input of such a $\mathcal{G}$ is the input of the first randomized gadget g_1. Indeed, even if actually the first gadget of the block was a non-randomized one (i.e. a linear gadget), then this would change the actual value of the input, but not its properties related to the independence. We call *dimension* of a block $\mathcal{G}$ the number of randomized gadgets g_i composing the block. In Figure 1 are represented N blocks of gadgets of dimension 4 each.

The following lemma gives a simple compositional result for multiple blocks of gadgets, where each such block uses the same random component $\boldsymbol{R}$. Slightly informally speaking, let $\mathcal{G}_j$ be multiple sets of gadgets, where all gadgets in $\mathcal{G}_j$ share the same randomness. Then, the lemma below shows that if the gadgets in $\mathcal{G}_j$ are t–SCR, then also the composition of the gadgets in all sets $\mathcal{G}_j$ are t–SCR. We underline that such a block constitutes itself a gadget. For simplicity, we assume that the blocks of gadgets that we consider in the lemma below all have the same dimension d. But our analysis can easily be generalized to a setting where each block has a different dimension.

Lemma 3 (composition of t–SCR **gadgets).** *Consider* N *blocks of gadgets* $\mathcal{G}_1(\boldsymbol{R}) = \{\mathsf{g}_{1,1}(\boldsymbol{r_1}), \ldots, \mathsf{g}_{1,d}(\boldsymbol{r_d})\}, \ldots, \mathcal{G}_N(\boldsymbol{R}) = \{\mathsf{g}_{N,1}(\boldsymbol{r_1}), \ldots, \mathsf{g}_{N,d}(\boldsymbol{r_d})\}$ *sharing the same random component* $\boldsymbol{R} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_d})$ *. If for all* $j = 1 \ldots, d$ *the gadgets* $\mathsf{g}_{1,j}(\boldsymbol{r_j}), \ldots, g_{N,j}(\boldsymbol{r_j})$ *are* t–SCR *, then the blocks of gadgets* $\{\mathcal{G}_1, \ldots, \mathcal{G}_N\}$ *are* t–SCR*.*

Proof. We can prove the statement with an inductive argument on the dimension of the blocks.

- If $d = 1$, then by hypothesis $\{\mathsf{g}_{1,1}, \ldots, g_{N,1}\}$ are t–SCR and then $\{\mathcal{G}_1, \ldots, \mathcal{G}_N\}$ are t–SCR.
- If $d > 1$ and $\{\{\mathsf{g}_{1,1}, \ldots, \mathsf{g}_{1,d-1}\}, \ldots, \{\mathsf{g}_{N,1}, \ldots, \mathsf{g}_{N,d-1}\}\}$ are t–SCR, then by hypothesis $\{\mathsf{g}_{1,d}, \ldots, g_{N,d}\}$ are t–SCR. Now the following cases hold.
  - The probes are placed on the $\{\{\mathsf{g}_{1,1}, \ldots, \mathsf{g}_{1,d-1}\}, \ldots, \{\mathsf{g}_{N,1}, \ldots, \mathsf{g}_{N,d-1}\}\}$: in this case, by the inductive hypothesis, the adversary's view is simulatable in the sense of Definition 5 of t–SCR.

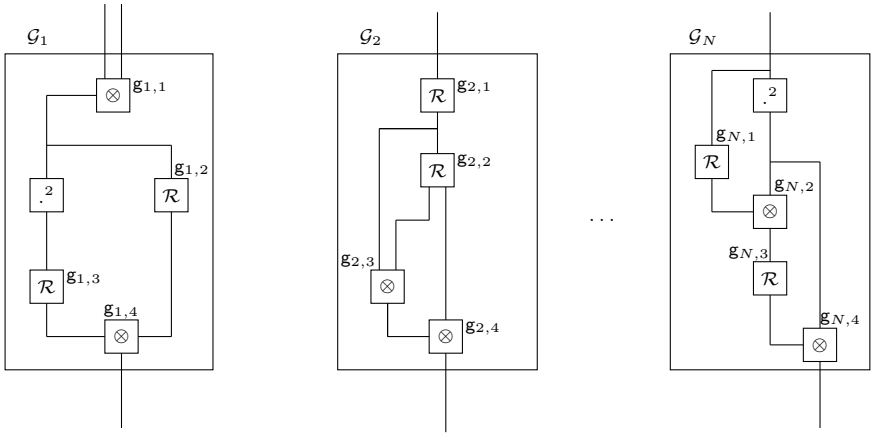

Fig. 1. A set of N blocks of gadgets with dimension $d = 4$ each.

- The probes are placed on $\{\mathrm{g}_{1,d}, \ldots, g_{N,d}\}$: in this case, since by hypothesis $\{\mathrm{g}_{1,d}, \ldots, g_{N,d}\}$ are t–SCR, the adversary's view is simulatable in the sense of Definition 5.
- A set of the probes $\mathcal{P}$ is placed on $\{\mathrm{g}_{1,d}, \ldots, g_{N,d}\}$ and a set of probes $\mathcal{Q}$ is placed on $\{\{\mathrm{g}_{1,1}, \ldots, \mathrm{g}_{1,d-1}\}, \ldots, \{\mathrm{g}_{N,1}, \ldots, \mathrm{g}_{N,d-1}\}\}$: in this case, since the probes in $\mathcal{P}$ and in $\mathcal{Q}$ use different random bits, they can be simulated independently each other. The simulatability of the probes in $\mathcal{P}$ according to Definition 5 is guaranteed by the t–SCR of $\{\mathrm{g}_{1,d}, \ldots, g_{N,d}\}$ and the simulatability of the probes in $\mathcal{Q}$ is guaranteed by the t–SCR of $\{\{\mathrm{g}_{1,1}, \ldots, \mathrm{g}_{1,d-1}\}, \ldots, \{\mathrm{g}_{N,1}, \ldots, \mathrm{g}_{N,d-1}\}\}$.

Therefore, for the inductive step we conclude that for every dimension d of the blocks $\mathcal{G}_i$, with $i = 1, \ldots, N$, the set $\{\mathcal{G}_1, \ldots, \mathcal{G}_N\}$ is t–SCR. □

We point out that the t–SCR property itself is not sufficient to guarantee a sound composition as well. The reason for this is that t–SCR essentially is only t–NI. Therefore, when used in combination with other gadgets, a t–SCR scheme needs additionally to satisfy the t–SNI property. We summarize this observation in the following theorem which gives a global result for circuits designed in blocks of gadgets sharing the same randomness.

Theorem 1. *Let $\mathcal{C}$ be a circuit composed by N blocks of gadgets $\mathcal{G}_1(\boldsymbol{R}), \ldots, \mathcal{G}_N(\boldsymbol{R})$ where $\mathcal{G}_i(\boldsymbol{R}) = \{\mathrm{g}_{i,1}(\boldsymbol{r_1}), \ldots, \mathrm{g}_{i,d}(\boldsymbol{r_d})\}$ for each $i = 1, \ldots, N$ and with inputs encoded in n shares and such that the gadgets outside such blocks are either linear or t–SNI. If*

- $\forall i = 1, \ldots, N$ $\mathcal{G}_i$ *is t–SNI and*
- $\forall j = 1, \ldots, d$ $\mathrm{g}_{1,j}, \ldots, g_{N,j}$ *are t–SCR*

then the circuit $\mathcal{C}$ is $t-$probing secure.

Proof. The proof of the theorem is straightforward. Indeed, Lemma 3 implies that $\mathcal{G}_1, \ldots, \mathcal{G}_N$ are t–SCR. Moreover, we point out that the t–SNI of the $\mathcal{G}_i$, for every $i = 1, \ldots, N$, and the independence of the outputs guarantees a secure composition

- among the blocks $\mathcal{G}_i$
- of the $\mathcal{G}_i$ with other randomized and t–SNI gadgets using fresh randomness
- of the $\mathcal{G}_i$ with linear gadgets.

This is sufficient to prove that the circuit $\mathcal{C}$ is t probing secure. □

To sum up, we showed that, under certain conditions, it is possible to design a circuit which internally reuses the random bits involved and remains probing secure. Therefore, if used appropriately, this result allows us to decrease the amount of randomness necessary in order to have a private circuit (because all the blocks share the same randomness). Nevertheless, we remark that in practice in order to have blocks of t–SCR gadgets it is necessary that the inputs of the first gadget of each block are mutually independent, because otherwise it is not always possible to have the independent simulation required by Definition 5. In the next section we show one possible way to achieve this property.

3.1 The Ind gadget

In the following, we present a gadget called Ind which, given an input x, produces an output $\mathsf{Ind}(x)$ independent from x. Thanks to this property, Ind can be adopted in order to create independence between inputs of gadgets sharing the same randomness. The algorithm, already proposed in [9] as a refreshing scheme, is depicted in Algorithm 3 and it requires the use of n fresh random bits, where n is the number of shares. It was proven to be t–NI and not t–SNI in [2] and therefore it must be used carefully, because not always securely composable with other gadgets.

Algorithm 3 Refreshing gadget Ind

Input: shares $a_1, \ldots, a_n$ such that $\bigoplus a_i = a$
Output: shares $a_1^{in}, \ldots, a_n^{in}$ such that $\bigoplus a_i^{in} = a$

```
(a_1^in, ..., a_n^in) ← (a_1^in, ..., a_n^in);
for i = 2 to n do
    r_i ←$ F_2^n;
    a_1^in ← a_1^in + r_i;
    a_i^in ← a_i^in + r_i;
end for
```

As we already mentioned, the scheme Ind can be applied at the inputs of the multiplication schemes using the same randomness in order to make them independent each other. Whenever the scheme cannot be used because not securely

composable, the multiplication schemes cannot share the same randomness, because they don't have independent inputs.

When designing such circuits, even if on the one hand the randomness involved in the gadgets can be completely reused, on the other hand additional Ind schemes are required to guarantee the independence of the inputs of a multiplication scheme sharing randomness. For this reason, in order to have an actual reduction in the amount of randomness, it is needed to take a couple of precautions when deciding to reuse randomness in a circuit and ensuring that the new randomness injected does not exceed the one that is saved. In Section 5 we will see an example, which shows that in some cases this procedure might actually give a worse result than in the case of not sharing randomness and we will show a practical application on the Sbox of the AES, with promising results.

3.2 A t-SCR Multiplication Scheme

In this subsection, we introduce multiplication schemes, which when sharing the same randomness are t–SCR. In particular, our multiplication schemes are based on two basic properties (i.e., $\lfloor \frac{t}{2} \rfloor$-non-completeness and t–SNI) and we discuss how to construct instantiations of our multiplication according to these properties.

First, we construct a multiplication scheme in accordance with $\lfloor \frac{t}{2} \rfloor$-non-completeness. This process is similar to finding a $\lfloor \frac{t}{2} \rfloor$-order TI of the AND-gate [17] or multiplication [8]. However, for our application we additionally require that the number of output shares is equal to the number of input shares. Most higher-order TI avoid this restriction with additional refreshing- and compression-layers. Since the $\lfloor \frac{t}{2} \rfloor$-non-completeness should be fulfilled without fresh randomness, we have to construct a $\lfloor \frac{t}{2} \rfloor$-non-complete $\mathsf{Mult} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and cannot rely on compression of the output shares. Unfortunately, this is only possible for very specific values of n. Due to this minor difference, we cannot directly use the bounds from the original publications related to higher-order TI. In the following, we derive an equation for n given an arbitrary t for which there exist a $\lfloor \frac{t}{2} \rfloor$-non-complete Mult.

Initially, due to the $\lfloor \frac{t}{2} \rfloor$-non-completeness the number of shares for which we can construct a scheme with the above properties is given by

$$\left\lfloor \frac{t}{2} \right\rfloor \cdot l + 1 = n \tag{1}$$

where l denotes the number of input shares which are leaked by each of the output shares, i.e., even the combination of $\lfloor \frac{t}{2} \rfloor$ output shares is still independent of one input share. To construct a $\lfloor \frac{t}{2} \rfloor$-non-complete multiplication, we need to distribute $\binom{n}{2}$ terms of the form $a_i b_j + a_j b_i$ over n output shares, i.e., each output share is made up of the sum of $\frac{n-1}{2}$ terms. Each of these terms leaks information about the tuples (a_i, a_j) and (b_i, b_j), and we assume the encodings a and b are independent and randomly chosen. For a given l, the maximum number of possible terms, which can be combined without leaking about more

than l shares of a or b, is $\binom{l}{2}$. The remaining $a_i b_i$ are equally distributed over the output shares without increasing l. By combining these two observations, we derive the relation

$$\frac{n-1}{2} = \frac{l^2 - l}{2}. \tag{2}$$

Based on Equation (1), the minimum number of shares for $\lfloor \frac{t}{2} \rfloor$-non-completeness is $n = \lfloor \frac{t}{2} \rfloor \cdot l + 1$. We combine this with Equation (2) and derive

$$n = \left\lfloor \frac{t}{2} \right\rfloor^2 + \left\lfloor \frac{t}{2} \right\rfloor + 1. \tag{3}$$

We use Equation (3) to compute the number of shares for our t–SCR multiplication scheme with $t > 3$. For $t \leq 3$, the number of shares is bounded by the requirement for the multiplication to be t–SNI, i.e., $n > t$.

To achieve t–SCR, it is necessary to include randomness in the multiplications. Initially, $\frac{n^2}{2}$ random components r_i need to be added for the multiplication to be t–SNI. A subset of n random components is added to each output share equally distributed over the sum, and each of these random bits is involved a second time in the computation of a single different output share. This ensures the simulatability of the gadget by using a limited number of input shares as required by the definition of t–SNI. In particular, the clever distribution of the random bits allows to simulate the output probes with a random and independent value.

When dealing with odd order the two properties aforementioned are not sufficient for guaranteeing privacy in our scenarios. Informally speaking, the $\lfloor \frac{t}{2} \rfloor$ non completeness allows probing up to $\lfloor \frac{t}{2} \rfloor$ of the values in two different gadgets sharing the same randomness, without that the cancellation of the random bits let the adversary know more than $n-1$ shares of the secret. But when the order t is odd, the adversary could have, in addition to the $\lfloor \frac{t}{2} \rfloor$ probes per gadget sharing the same randomness, also another probe with same randomness, which might add at least another input share to the knowledge of the adversary. Therefore, it is needed that the scheme satisfies a third property:

Definition 6 (Special Non-Completeness). *For every set of at most t probes* $\{p_1, ..., p_k, q_1, ..., q_h\}$ *such that*

- $p_1, ..., p_k$ *are probes on the output shares depending respectively on the vectors of random bit* $\boldsymbol{r}_{p_1}, ..., \boldsymbol{r}_{p_k}$*, where* $k \leq \frac{t}{2}$*,*
- $q_1, ..., q_h$ *are probes on the internal values depending respectively on the vectors of random bit* $\boldsymbol{r}_{q_1}, ..., \boldsymbol{r}_{q_h}$*, where* $h \leq t - 2 \cdot k$*,*
- $\forall i \in [1, h]\ \exists j \in [1, k]$*such that* $\boldsymbol{r}_{q_i} \subseteq \boldsymbol{r}_{p_j}$*,*

the set $\{p_1, ..., p_k, q_1, ..., q_h\}$ *is non-complete without randomness.*

The construction of a t–SCR multiplication scheme following the aforementioned guidelines is easy for small t. However, finding a distribution of terms

that fulfills $\lfloor \frac{t}{2} \rfloor$-non-completeness or special non-completeness becomes a complex task due to the large number of possible combinations for increasing t. For $t \leq 5$, possible t–SCR multiplication schemes are defined respectively in Algorithms 4, 5, 6 and 7. We underline that the computation in all the algorithms follows from left to right and that the operations in brackets are executed in advance. A complete description of a multiplication algorithm for higher orders fulfilling the aforementioned properties can be found in Appendix A.

Algorithm 4 Mult^2 for order $t = 2$ with $n = 3$ shares.

Input: shares $a_1, \ldots, a_3$ such that $\bigoplus a_i = a$, shares $b_1, \ldots, b_3$ such that $\bigoplus b_i = b$
Output: shares $c_1, \ldots, c_3$ such that $\bigoplus c_i = a \cdot b$

$c_1 = (a_1b_2 + r_1) + (a_2b_1 + r_2) + a_2b_2;$
$c_2 = (a_2b_3 + r_2) + (a_3b_2 + r_3) + a_3b_3;$
$c_3 = (a_3b_1 + r_3) + (a_1b_3 + r_1) + a_1b_1;$

Algorithm 5 Mult^3 for order $t = 3$ with $n = 4$ shares.

Input: shares $a_1, \ldots, a_4$ such that $\bigoplus a_i = a$, shares $b_1, \ldots, b_4$ such that $\bigoplus b_i = b$
Output: shares $c_1, \ldots, c_4$ such that $\bigoplus c_i = a \cdot b$

$c_1 = (a_1b_2 + r_1) + (a_3b_1 + r_5) + (a_3b_2 + r_4) + a_1b_1;$
$c_2 = (a_2b_1 + r_1) + (a_4b_2 + r_6) + (a_4b_1 + r_2) + a_2b_2;$
$c_3 = (a_3b_4 + r_3) + (a_2b_3 + r_5) + (a_2b_4 + r_2) + a_3b_3;$
$c_4 = (a_4b_3 + r_3) + (a_1b_4 + r_6) + (a_1b_3 + r_4) + a_4b_4;$

Algorithm 6 Mult^4 for order $t = 4$ with $n = 7$ shares.

Input: shares $a_1, \ldots, a_7$ such that $\bigoplus a_i = a$, shares $b_1, \ldots, b_7$ such that $\bigoplus b_i = b$
Output: shares $c_1, \ldots, c_7$ such that $\bigoplus c_i = a \cdot b$

$c_1 = r_{15} + a_1b_1 + r_1 + a_1b_2 + a_2b_1 + r_{18} + a_1b_3 + a_3b_1 + r_7 + a_2b_3 + a_3b_2 + r_{13} + r_8;$
$c_2 = r_{16} + a_2b_2 + r_2 + a_2b_4 + a_4b_2 + r_{17} + a_2b_6 + a_6b_2 + r_1 + a_4b_6 + a_6b_4 + r_{14};$
$c_3 = r_{17} + a_4b_4 + r_8 + a_4b_1 + a_1b_4 + r_{10} + a_4b_5 + a_5b_4 + r_2 + a_1b_5 + a_5b_1 + r_3;$
$c_4 = r_{18} + a_5b_5 + r_9 + a_5b_2 + a_2b_5 + r_{16} + a_5b_7 + a_7b_5 + r_4 + a_2b_7 + a_7b_2 + r_{11};$
$c_5 = r_5 + a_7b_7 + r_4 + a_7b_1 + a_1b_7 + r_{15} + a_7b_6 + a_6b_7 + r_{12} + a_1b_6 + a_6b_1 + r_3;$
$c_6 = r_6 + a_6b_6 + r_{13} + a_6b_3 + a_3b_6 + r_5 + a_6b_5 + a_5b_6 + r_9 + a_3b_5 + a_5b_3 + r_{11};$
$c_7 = r_{10} + a_3b_3 + r_{12} + a_3b_4 + a_4b_3 + r_6 + a_3b_7 + a_7b_3 + r_{14} + a_4b_7 + a_7b_4 + r_7;$

According to the multiplication scheme just presented, we now point out that the number of new fresh random bits introduced by the gadget Ind for a certain order t, is $n - 1$, where n is the number of shares required for the corresponding multiplication scheme Mult for that particular order. In the light of this remark,

Algorithm 7 Mult^5 for order $t = 5$ with $n = 7$ shares.

Input: shares $a_1, \ldots, a_7$ such that $\bigoplus a_i = a$, shares $b_1, \ldots, b_7$ such that $\bigoplus b_i = b$
Output: shares $c_1, \ldots, c_7$ such that $\bigoplus c_i = a \cdot b$

$$
\begin{aligned}
c_1 &= (a_2b_3 + r_1) + (a_3b_2 + r_7) + (a_1b_3 + r_8) + (a_3b_1 + r_{13}) + (a_2b_1 + r_{15}) + (a_1b_2 + r_{19}) + a_1b_1;\\
c_2 &= (a_4b_2 + r_2) + (a_2b_4 + r_1) + (a_6b_2 + r_9) + (a_4b_6 + r_{14}) + (a_2b_6 + r_{16}) + (a_6b_4 + r_{20}) + a_2b_2;\\
c_3 &= (a_5b_4 + r_3) + (a_1b_4 + r_2) + (a_4b_5 + r_{10}) + (a_1b_5 + r_8) + (a_5b_1 + r_{17}) + (a_4b_1 + r_{21}) + a_4b_4;\\
c_4 &= (a_2b_7 + r_4) + (a_5b_7 + r_3) + (a_5b_2 + r_{11}) + (a_2b_5 + r_9) + (a_7b_2 + r_{18}) + (a_7b_5 + r_{15}) + a_5b_5;\\
c_5 &= (a_6b_7 + r_5) + (a_7b_6 + r_4) + (a_7b_1 + r_{12}) + (a_1b_7 + r_{10}) + (a_1b_6 + r_{19}) + (a_6b_1 + r_{16}) + a_7b_7;\\
c_6 &= (a_3b_6 + r_6) + (a_6b_3 + r_5) + (a_5b_3 + r_{13}) + (a_5b_6 + r_{11}) + (a_6b_5 + r_{20}) + (a_3b_5 + r_{17}) + a_6b_6;\\
c_7 &= (a_3b_4 + r_7) + (a_7b_4 + r_{14}) + (a_3b_7 + r_6) + (a_4b_7 + r_{12}) + (a_7b_3 + r_{21}) + (a_4b_3 + r_{18}) + a_3b_3;
\end{aligned}
$$

we notice that not always reusing randomness actually reduces the total amount of randomness in the circuit. Let consider for example the case of Figure 2(a), where we aim at reusing the randomness among $\mathsf{Mult}_1 \ldots \mathsf{Mult}_N$ and the inputs of different multiplication schemes are dependent each other. In order to have independent inputs and securely reuse the randomness, we then need to add two Ind gadgets, as in Figure 2(b).

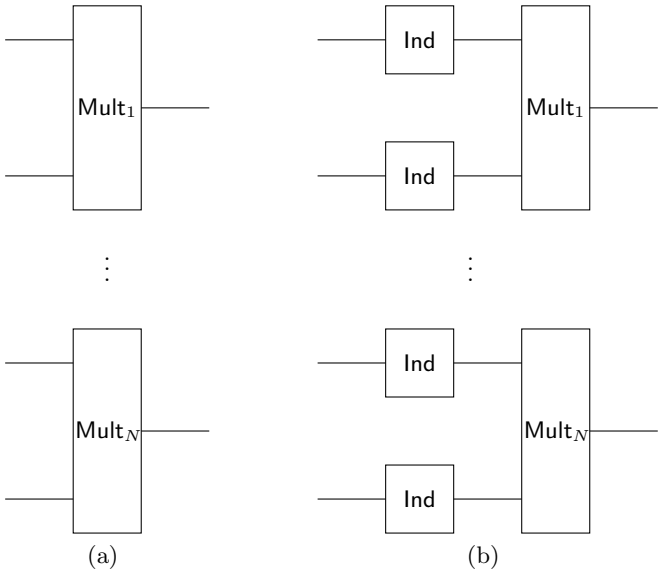

Fig. 2. N parallel multiplication schemes with (a) dependent inputs and (b) independent inputs

It is easy to check that for order 2 and 3, the number of fresh random bits injected in the circuit by the Ind gadgets exceed the one that it would be used if refreshing entirely the Mult_i without reusing randomness. On the contrary, if only one of the two inputs of the multiplication schemes sharing randomness are dependent, only one Ind scheme is needed per multiplication scheme, as in

Figure 3. In this case, reusing randomness is always convenient for every order $t \geq 4$ and for order $t = 3$ only if $N \geq 3$.

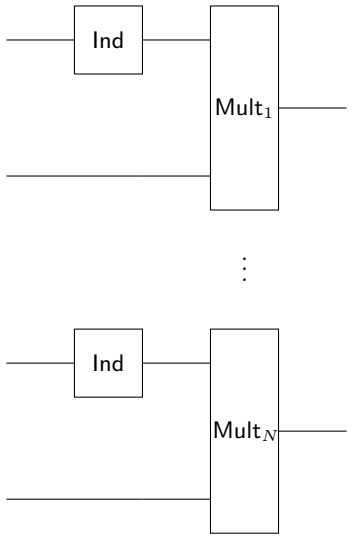

Fig. 3. N multiplication schemes with only one input which needs the Ind gadget in order to become independent

The examples above show that the actual efficiency of the process of reusing randomness is not straightforward, but it is given by the right trade off between the fresh randomness used by the Ind gadgets and the amount of randomness saved by sharing it. It is therefore important to make a careful analysis of the circuit structure before applying our method: first of all in order to understand if the randomness cost is actually amortized and secondly in order to check that the security still holds even with the insertion of the Ind gadgets. As we will see later in Section 5, for circuits with an obvious structure (e.g., layers for symmetric ciphers) which contain easily-exploitable regularities, this is not a hard task and the optimal solutions can be usually found quickly.

3.3 Security proofs

In this section we present the security analysis of the multiplication schemes given in the previous section and we show that they can be securely composed in blocks of gadgets sharing the same random component. We highlight that the schemes in Algorithm 4 and 5 are both already proven to be t–SNI in [2]. We explicitly prove the scheme to be both t–SNI and t–SCR only for the orders greater than 3. Indeed, the general nature of the simulation according to the t–SCR property makes the proof of t–SCR valid for order $t = 2$ and $t = 3$ as well.

Proposition 1 (order $t = 4$). *Let $\mathsf{Mult}_1^4, \ldots, \mathsf{Mult}_N^4$ be a set of N multiplication schemes as in Algorithm 6, with inputs $(\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}), \ldots, (\boldsymbol{a}^{(N)}, \boldsymbol{b}^{(N)})$ and outputs $\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(N)}$. Suppose that the maskings of the inputs are independent and uniformly chosen and that for $k = 1, \ldots, N$ each Mult_k^4 uses the same random bits $\left(r_i\right)_{i=1,\ldots,tn/2}$. Then $\mathsf{Mult}_1^4, \ldots, \mathsf{Mult}_N^4$ are t–SCR and in particular Algorithm 6 is t–SNI.*

Proof. Let $\Omega = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ be a set of t observations respectively on the gadgets $\mathsf{Mult}_1^4, \ldots, \mathsf{Mult}_N^4$. We show that $\mathsf{Mult}_1^4, \ldots, \mathsf{Mult}_N^4$ satisfy definition 5, i.e. that the probes in $\mathcal{P}_i$ can be consistently simulated by at most $n-1$ shares of the inputs of Mult_i^4. Suppose $p_1, \ldots, p_t$ are t adversary's probes. We indicate with $\boldsymbol{r}_{p_i}$ the vector of the respective randomness for every $i = 1, \ldots, t$ and we classify such p_i in the following groups:

(1) $\exists j \in [1,t]$ such that $\boldsymbol{r}_{p_i} = \boldsymbol{r}_{p_j}$
(2) $\exists j \in [1,t]$ such that $\boldsymbol{r}_{p_i} \subset \boldsymbol{r}_{p_j}$
(3) $\forall j \in [1,t]\ \boldsymbol{r}_{p_i} \cap \boldsymbol{r}_{p_j} = \emptyset$
(4) $\boldsymbol{r}_{p_i} = \boldsymbol{0}$
(5) $\exists \mathcal{J} \in [1,t]$such that for all $j \in \mathcal{J}\ \boldsymbol{r}_{p_j} \subset \boldsymbol{r}_{p_i}$ and $\bigcup_j \boldsymbol{r}_{p_j} = \boldsymbol{r}_{p_i}$
(6) $\forall j \in [1,t]$such that $\boldsymbol{r}_{p_j} \subset \boldsymbol{r}_{p_i} : \bigcup_j \boldsymbol{r}_{p_j} \neq \boldsymbol{r}_{p_i}$

We define now the sets $I_1, \ldots, I_N, J_1, \ldots, J_N$ with $|I_i| < n$ $|J_i| < n$ for every $i = 1, \ldots, N$ such that the values of the wires p_h can be perfectly simulated given the values $(a_i^{(1)})_{i \in I_1}$, $(b_i^{(1)})_{i \in J_1}, \ldots, (a_i^{(N)})_{i \in I_N}$, $(b_i^{(N)})_{i \in J_N}$.
The procedure to construct the sets is the following:

- For every wire in the group (1), (2), (4) or (5), add all the indices of the shares of $\boldsymbol{a}^{(j)}$ in I_j and of $\boldsymbol{b}^{(j)}$ in J_j for every $j = 1, \ldots, N$.

We remark that

- by construction of Mult^e, for every probe in group (4) we add at most only one index to each set,
- there exist at most $\left\lfloor \frac{t}{2} \right\rfloor$ probes per each gadget in group (1),
- there exist at most $\left\lfloor \frac{t}{3} \right\rfloor$ probes per each gadget in group (5),
- for every probe p_i with $|\boldsymbol{r}_{p_i}| < t$, the number of indices added is inferior then when $|\boldsymbol{r}_{p_i}| = t$.

Therefore, we have at most $\left\lfloor \frac{t}{2} \right\rfloor$ probes per gadget for which we add all the indices of the shares in the sets I_j and J_j and thanks to the $\left\lfloor \frac{t}{2} \right\rfloor$ non completeness without randomness of Mult^e we can conclude that $|I_i| \leq n$ and $|J_i| \leq n$. In particular, we point out that if the inputs are not independent (or outputs of another t–SCR and t–SNI gadget), we cannot guarantee this bound on the sets of indices, because the shares of inputs of a certain gadget might provide information regarding the shares of another one.

We now simulate, consistently with the randomness involved, the probes p_i by using $(a_i^{(1)})_{i \in I_1}$, $(b_i^{(1)})_{i \in J_1}, \ldots, (a_i^{(N)})_{i \in I_N}$, $(b_i^{(N)})_{i \in J_N}$.

- The simulation starts with a preliminary phase in which for every probe in group (1), (2) and (5) we pick at random the components of $\boldsymbol{r}_{p_i}$.
- For every probe in group (1) and (2), we simulate p_i by using the random bits $\boldsymbol{r}_{p_i}$ selected in the preliminary phase and the indices of the inputs in $I_1, J_1, \ldots, I_N, J_N$.
- For every probe in group (3), since p_i does not share randomness with any other probe, we simulate it as a uniform and random bit.
- For every probe in group (4), since p_i does not contain any randomness, we simulate it only by using the indices of the inputs in $I_1, J_1, \ldots, I_N, J_N$.
- For every probe in group (5) , since all the component of the randomness $\boldsymbol{r}_{p_i}$ have been assigned in the preliminary phase, then we simulate p_i by using the random bits $\boldsymbol{r}_{p_i}$ selected in the preliminary phase and the indices of the inputs in I_1, J_1, I_2, J_2.
- For every probe in group (6), since p_i contains some randomness which does not appear elsewhere in the other probes, we simulate it as a uniform and random bit.

We now show the t–SNI property of Mult^4.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be a set of 4 observations respectively on the internal and on the output wires, where $|\mathcal{I}| = t_1$ and in particular $t_1 + |\mathcal{O}| \leq t$. We classify the internal wires in the following groups:

(1) $a_i, b_j, a_i b_j$
(2.1) r_k
(2.2) $r_k + a_i b_i$
(3.1) $r_k + a_i b_i + r_h$,
(3.2) $r_k + a_i b_i + r_h + a_i b_j$,
(3.3) $r_k + a_i b_i + r_h + a_i b_j + a_j b_i$,
(4.1) $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l$,
(4.2) $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l + a_i b_u$,
(4.3) $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l + a_i b_u + a_u b_i$,
(5.1) $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l + a_i b_u + a_u b_i + r_m$,
(5.2) $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l + a_i b_u + a_u b_i + r_m + a_j b_u$,
(5.3) $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l + a_i b_u + a_u b_i + r_m + a_j b_u + a_u b_j$
(6) $c_i = r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l + a_i b_u + a_u b_i + r_m + a_j b_u + a_u b_j + r_e$
(7) $c_1 = r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l + a_i b_u + a_u b_i + r_m + a_j b_u + a_u b_j + r_e + r_f$

Suppose an adversary corrupts at most 4 wires $w_1, \ldots, w_4$. We define two sets I, J with $|I| < t_1$ and $|J| < t_1$ such that the values of the wires w_h can be perfectly simulated given the values $(a_i)_{i \in I}$, $(b_i)_{i \in J}$.
The procedure to construct the sets is the following:

- We first define a set K such that for all the probes in group (2.1) and (2.2) we add k to K.
- Initially I, J are empty and the w_i unassigned.
- (*) For every wire in the group (1) or a combination of this, add i to I and j to J.

- (*) For every wire in group (2.2) if $i \notin I$, add i to I, if $i \notin J$, add i to J.
- For every wire in group (3.1), (3.2), (3.3) and such that $k, h \in K$, if $i \notin I$, for every index of the shares of a which is not in I, add the index to I and for every index of the shares of b which is not in J, add the index to J.
- For every wire in group (4.1), (4.2), (4.3) and such that $k \in K$, $h \in K$ and $l \in K$, for every index of the shares of a which is not in I, add the index to I and for every index of the shares of b which is not in J, add the index to J.
- (**)For every wire in group (3.2) or (3.3) such that $r_k + a_i b_i + r_h$ was probed, if $i \notin I$, add i to I, if $j \notin I$, add j to I, if $i \notin J$, add i to J, if $j \notin J$, add j to J.
- (**)For every wire in group (4.2) or (4.3) such that $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l$ was probed, if $i \notin I$, add i to I, if $i \notin J$, add i to J, if $u \notin I$, add u to I, if $u \notin J$, add u to J.
- For every wire in group (5.2) or (5.3) such that $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l + a_i b_u + a_u b_i + r_m$ was probed and $m \notin K$, if $j \notin I$, add j to I, if $j \notin J$, add j to J, if $u \notin I$, add u to I, if $u \notin J$, add u to J.
- (**) For every wire in group (3.2) or (3.3) such that $r_k + a_i b_i$ was probed and $h \in K$, if $i \notin I$, add i to I, if $j \notin I$, add j to I, if $i \notin J$, add i to J, if $j \notin J$, add j to J.
- (**) For every wire in group (3.3) such that $r_k + a_i b_i + r_h + a_i b_j$ was probed if $j \notin I$, add j to I, if $i \notin J$, add i to J.
- For every wire in group (4.2) or (4.3) such that $r_k + a_i b_i + r_h$ was probed and $l \in K$, if $i \notin I$, add i to I, if $i \notin J$, add i to J, if $u \notin I$, add u to I, if $u \notin J$, add u to J.
- For every wire in group (5.2) or (5.3) such that $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l$ was probed and $m \in K$, if $i \notin I$, add i to I, if $i \notin J$, add i to J, if $j \notin I$, add j to I, if $j \notin J$, add j to J, if $u \notin I$, add u to I, if $u \notin J$, add u to J.
- For every wire in group (5.2) or (5.3) such that $r_k + a_i b_i + r_h$ was probed and $l, m \in K$, if $i \notin I$, add i to I, if $i \notin J$, add i to J, if $j \notin I$, add j to I, if $j \notin J$, add j to J, if $u \notin I$, add u to I, if $u \notin J$, add u to J.
- For every wire in group (4.1) such that $r_k + a_i b_i + r_h$ was probed and $l \in K$, or such that $r_k + a_i b_i$ was probed and $h, l \in K$, if $i \notin I$, add i to I, if $i \notin J$, add i to J, if $j \notin I$, add j to I, if $j \notin J$, add j to J.
- For every wire in group (5.1) such that $r_k + a_i b_i + r_h$ was probed and $l, m \in K$, if $i \notin I$, add i to I, if $i \notin J$, add i to J, if $j \notin I$, add j to I, if $j \notin J$, add j to J, if $u \notin I$, add u to I, if $u \notin J$, add u to J.
- For every wire in group (5.1) such that $r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l$ was probed and $m \in K$, if $i \notin I$, add i to I, if $i \notin J$, add i to J, if $j \notin I$, if $u \notin I$, add u to I, if $u \notin J$, add u to J.

Now, if $t_1 = 1$, then according to the distribution of the randomness, the only groups which add indices to I and J are the ones marked with (*). Therefore $|I| \leq 1$ and $|J| \leq 1$. If $t_1 = 2$, then the only groups which add indices to I and J can be the ones marked with (**). Therefore, in all the previous situations $|I| \leq 2$ and $|J| \leq 2$. If $t_1 \geq 3$, then the groups which add indices to I and J

are all the remaining and in any of these cases $|I| \leq 3$ and $|J| \leq 3$ if $t_1 = 3$ or $|I| \leq 4$ and $|J| \leq 4$ if $t_1 = 4$.

We now simulate the wires $w_1, \ldots, w_4$ using only the input shares $(a_i)_{i \in I}$ and $(b_j)_{j \in J}$.

- For every $k \in K$ assign r_k to a random and independent value. This simulates the probes in (2.1)
- For every probe in group (1) , then by construction $i \in I$ and $j \in J$ and the values are perfectly simulated.
- For every probe in group (2.2), r_k has been simulated in the fist step of the simulation. Moreover, by construction $i \in I$ and $j \in J$ and therefore the probe can be perfectly simulated.
- For every probe in (3.1), if r_k and r_h have been observed , then the values have been assigned to a random and independent value at the beginning of the simulation and, since by construction $i \in I, J$, the probe can be simulated by using r_k, r_h and the required shares of a and b. Otherwise, if r_h or r_k have not been observed, then the probe can be simulated as a uniform and random value.
- For every probe in (3.2), if r_k and r_h have been observed , then the values have been assigned to a random and independent value at the beginning of the simulation and, since by construction $i, j \in I, J$, the probe can be simulated by using r_k, r_h and the required shares of a and b. If $p := r_k + a_i b_i + r_h$ was probed, then by construction $i, j \in I, J$ and the probe can be simulated by summing p and $a_i b_j$. Otherwise, if r_h or r_k have not been observed, then the probe can be simulated as a uniform and random value. Otherwise, if $p := r_k + a_i b_i$ and r_h have been probed, then r_h has been assigned to a random and independent value at the beginning of the simulation and by construction $i, j \in I, J$. Therefore, the probe can be simulated by summing p, r_h and the needed shares of a and b. Finally, if $p := r_k + a_i b_i$ but not r_h have been probed, then the observation can be simulated as a random and independent value.
- For every probe in (3.3), the simulation proceeds in a similar way. Moreover, if $p := r_k + a_i b_i + r_h + a_i b_j$ has been probed, then by construction $i \in J$ and $j \in I$, therefore the probe can be simulated by adding p and $a_j b_i$.
- For every probe in group (4.1)
  - if r_k or $r_k + a_i b_i$ and r_h and r_l have been observed , then the values of the random bits have been assigned to a random and independent value at the beginning of the simulation and, since by construction $i, j \in I, J$, the probe can be simulated by using r_k, r_h, r_l and the required shares of a and b;
  - if r_h or r_k or r_l have not been observed, then the probe can be simulated as a uniform and random value;
  - if $p := r_k + a_i b_i + r_h$ and r_l have been probed, then r_l has been assigned to a random and independent value at the beginning of the simulation and by construction $i, j \in I, J$. Therefore the probe can be simulated by using p, r_l and a_i, a_j, b_i, b_j.

  - otherwise we can simulate the probe as a random and independent value.
- For every probe in group (4.2)
  - if r_k or $r_k + a_i b_i$ and r_h and r_l have been observed , then the values of the random bits have been assigned to a random and independent value at the beginning of the simulation and, since by construction $i, j, u \in I, J$, the probe can be simulated by using r_k, r_h, r_l and the required shares of a and b;
  - if $p := r_k + a_i b_i + r_h$ and r_l have been probed, then r_l has been assigned to a random and independent value at the beginning of the simulation and by construction $i, j, u \in I, J$. Therefore the probe can be simulated by using p, r_l and a_i, a_j, b_i, b_j, b_u.
  - if $p := r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l$ has been probed then by construction $i \in I$ and $u \in J$. Therefore the probe can be simulated by using p and a_i, b_u.
  - otherwise we can simulate the probe as a random and independent value.
- For every probe in group (4.3)
  - if r_k or $r_k + a_i b_i$ and r_h and r_l have been observed , then the values of the random bits have been assigned to a random and independent value at the beginning of the simulation and, since by construction $i, j \in I, J$, the probe can be simulated by using r_k, r_h, r_l and the required shares of a and b;
  - if $p := r_k + a_i b_i + r_h$ and r_l have been probed, then r_l has been assigned to a random and independent value at the beginning of the simulation and by construction $i, j, u \in I, J$. Therefore the probe can be simulated by using p, r_l and $a_i, a_j, a_u, b_i, b_j, b_u$.
  - if $p := r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l$ has been probed then by construction $i, u \in I, J$. Therefore the probe can be simulated by using p and a_i, a_u, b_i, b_u.
  - otherwise we can simulate the probe as a random and independent value.
- For every probe in group (5.1), (5.2) or (5.3)
  - if $p := r_k + a_i b_i + r_h$ and r_l, r_m have been probed, then r_l and r_m have been assigned to a random and independent value at the beginning of the simulation and by construction $i, j, u \in I, J$. Therefore the probe can be simulated by using p, r_l, r_m and $a_i, a_j, a_u, b_i, b_j, b_u$.
  - if $p := r_k + a_i b_i + r_h + a_i b_j + a_j b_i + r_l$ and r_m have been probed, then by construction $i, u \in I, J$. Therefore the probe can be simulated by using p and a_i, a_u, b_i, b_u.
  - otherwise we can simulate the probe as a random and independent value.
- For every probe in group (6) or (7), there exists a random bit which was not probed. Therefore, the value can be simulated as a random and independent value.

□

Proposition 2 (order $t = 5$). *Let $\mathsf{Mult}_1^5, \ldots, \mathsf{Mult}_N^5$ be a set of N multiplication schemes as in Algorithm 7, with inputs $(\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}), \ldots, (\boldsymbol{a}^{(N)}, \boldsymbol{b}^{(N)})$ and outputs $\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(N)}$. Suppose that the maskings of the inputs are independent*

and uniformly chosen and that for $k = 1, \ldots, N$ each Mult_k^5 uses the same random bits $(r_i)_{i=1,\ldots,tn/2}$. Then $\mathsf{Mult}_1^5, \ldots, \mathsf{Mult}_N^5$ are t–SCR and in particular Algorithm 7 is t–SNI.

Proof. In the rest of the proof we let $t = 5$ and $n = 7$, but the general nature of the proof allows to generalize it for any order.

Let $\Omega = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ be a set of t observations respectively on the gadget $\mathsf{Mult}_1^5, \ldots, \mathsf{Mult}_N^5$. We show that $\mathsf{Mult}_1^5, \ldots, \mathsf{Mult}_N^5$ satisfy definition 5, i.e. that the probes in $\mathcal{P}_i$ can be consistently simulated by at most $n-1$ shares of the inputs of Mult_i^5, by defining now sets I_i and J_i, where $i = 1, \ldots, N$ with $|I_i| < n$ $|J_i| < n$ such that the values of the wires p_h can be perfectly simulated given the values $(a_i^{(1)})_{i\in I_1}, (b_i^{(1)})_{i\in J_1}, \ldots, (a_i^{(N)})_{i\in I_N}$, $(b_i^{(N)})_{i\in J_N}$. The procedure to construct the sets is exactly the one of the previous proof. Now, the $\lfloor \frac{t}{2} \rfloor$ non completeness without randomness and the special non completeness, which by construction are both satisfied by Mult^5, ensure that $|I_i| \leq n$ and $|J_i| \leq n$. The simulation of the probes follows the same procedure than the one for the case of $t = 4$.

We now show the t–SNI property of Mult^5.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be a set of t observations respectively on the internal and on the output wires, where $|\mathcal{I}| = t_1$ and $|\mathcal{O}| = t_2$ (and in particular $t_1 + t_2 \leq t$). We can classify the internal wires in the following groups:

(1) $a_i, b_j, a_i b_j$
(2) $a_i b_j + r_k$
(3) r_k
(4) $c_{i,j}$, which corresponds to a partial sum of the output share c_i.

For example, $c_{1,0} = a_1 b_1$, $c_{1,2} = a_1 b_1 + (a_1 b_2 + r_{19}) + (a_2 b_1 + r_{15})$. Suppose an adversary corrupts at most t wires $w_1, \ldots, w_t$. We define two sets I, J with $|I| < t_1$ $|J| < t_1$ such that the values of the wires w_h can be perfectly simulated given the values $(a_i)_{i\in I}$, $(b_i)_{i\in J}$.
The procedure to construct the sets is the following:

1. We first define a set K such that for all probes in group (2), (3) or (4) we add the index k of each random bit r_k to K.
2. Initially I, J are empty and the w_i unassigned.
3. For every wire in group (1) or a combination of this, if $i \notin I$ add i to I and if $j \notin J$ add j to J.
4. For every wire in group (2), add i to I and j to J.
5. For every wire in groups (4) such that all the indices of the random bits appear in K at least two times, for every share a_i observed, if $i \notin I$, add i to I and for every share b_j observed, if $j \notin J$, add j to J.
6. For every wire in groups (4) such that a certain $c_{i,j-h}$ was probed (with $h = 1, \ldots, j-1$), if for every r_k in $c_{i,j} - c_{i,j-h}$ $k \in K$, then add all the indices of the shares of a in I and all the shares of b in J, unless they are already in I and J.

We point out that the sets just constructed are such that $|I| \leq t_1$ and $|J| \leq t_1$. Indeed, for each probe in group (1) and (2) at most one index is added. For each probe in (4), we add more than one index to I and J only in the following four possible scenarios:

(a) The attacker's probes are of the form

$$p_1 := (a_i b_j + r_{k_1}) + (a_e b_f + r_{k_2}),\ p_2 := a_g b_h + r_{k_1},\ p_3 := a_l b_m + r_{k_2}$$

(b) The attacker's probes are of the form

$$p_1 := (a_i b_j + r_{k_1}) + (a_e b_f + r_{k_2}) + (a_p b_q + r_{k_3}),$$

$$p_2 := a_g b_h + r_{k_1},\ p_3 := a_l b_m + r_{k_2}, p_4 := a_r b_s + r_{k_3}$$

(c) The attacker's probes are of the form

$$p_1 := (a_i b_j + r_{k_1}) + (a_e b_f + r_{k_2}) + (a_p b_q + r_{k_3}), +(a_u b_v + r_{k_4})$$

$$p_2 := a_g b_h + r_{k_1},\ p_3 := a_l b_m + r_{k_2}, p_4 := a_r b_s + r_{k_3},\ p_5 := a_w b_z + r_{k_4}$$

(d) The attacker's probes are of the form

$$p_1 := (a_i b_j + r_{k_1}) + (a_e b_f + r_{k_2}) + (a_p b_q + r_{k_3}),\ p_3 := a_l b_m + r_{k_2},$$

$$p_2 := (a_g b_h + r_{k_1}) + (a_u b_v + r_{k_4}), p_4 := a_r b_s + r_{k_3},\ p_5 := a_w b_z + r_{k_4}$$

We give below table with all the possible set of probes in the previous four scenarios and the corresponding dimension of I and J, which is indeed always smaller than t_1.

Probes in scenario (a)	$\|I\|$	$\|J\|$
$p_1 := (a_2 b_3 + r_1) + (a_3 b_2 + r_7),\ p_2 := (a_2 b_4 + r_1),\ p_3 := (a_3 b_4 + r_7)$	2	3
$p_1 := (a_4 b_2 + r_2) + (a_2 b_4 + r_1),\ p_2 := (a_2 b_3 + r_1),\ p_3 := (a_1 b_4 + r_2)$	3	3
$p_1 := (a_5 b_4 + r_3) + (a_1 b_4 + r_2),\ p_2 := (a_4 b_2 + r_2),\ p_3 := (a_5 b_7 + r_3)$	3	3
$p_1 := (a_2 b_7 + r_4) + (a_5 b_7 + r_3),\ p_2 := (a_7 b_6 + r_4),\ p_3 := (a_5 b_4 + r_3)$	3	3
$p_1 := (a_6 b_7 + r_5) + (a_7 b_6 + r_4),\ p_2 := (a_6 b_3 + r_5),\ p_3 := (a_2 b_7 + r_4)$	3	3
$p_1 := (a_3 b_6 + r_6) + (a_6 b_3 + r_5),\ p_2 := (a_3 b_7 + r_6),\ p_3 := (a_6 b_7 + r_5)$	2	3
$p_1 := (a_3 b_4 + r_7) + (a_7 b_4 + r_{14}),\ p_2 := (a_3 b_2 + r_7),\ p_3 := (a_4 b_6 + r_{14})$	2	3

Probes in scenario (b)	$\|I\|$	$\|J\|$
$p_1 := (a_2b_3 + r_1) + (a_3b_2 + r_7) + (a_1b_3 + r_8), p_2 := (a_2b_4 + r_1),$ $p_3 := (a_3b_4 + r_7), p_4 := (a_1b_5 + r_8)$	3	4
$p_1 := (a_4b_2 + r_2) + (a_2b_4 + r_1) + (a_6b_2 + r_9),\ p_2 := (a_2b_3 + r_1),$ $p_3 := (a_1b_4 + r_2), p_4 := (a_2b_5 + r_9)$	4	4
$p_1 := (a_5b_4 + r_3) + (a_1b_4 + r_2) + (a_4b_5 + r_{10}),\ p_2 := (a_4b_2 + r_2),$ $p_3 := (a_5b_7 + r_3), p_4 := (a_1b_7 + r_{10})$	3	4
$p_1 := (a_2b_7 + r_4) + (a_5b_7 + r_3) + (a_5b_2 + r_{11}),\ p_2 := (a_7b_6 + r_4),$ $p_3 := (a_5b_4 + r_3), p_4 := (a_5b_6 + r_{11})$	3	4
$p_1 := (a_6b_7 + r_5) + (a_7b_6 + r_4) + (a_7b_1 + r_{12}),\ p_2 := (a_6b_3 + r_5),$ $p_3 := (a_2b_7 + r_4), p_4 := (a_4b_7 + r_{12})$	4	4
$p_1 := (a_3b_6 + r_6) + (a_6b_3 + r_5) + (a_5b_3 + r_{13}),\ p_2 := (a_3b_7 + r_6),$ $p_3 := (a_6b_7 + r_5), p_4 := (a_3b_1 + r_{13})$	3	1
$p_1 := (a_3b_4 + r_7) + (a_7b_4 + r_{14}) + (a_3b_7 + r_6),\ p_2 := (a_3b_2 + r_7),$ $p_3 := (a_4b_6 + r_{14}), p_4 := (a_3b_6 + r_6)$	3	4

Probes in scenario (c)	$\|I\|$	$\|J\|$
$p_1 := (a_2b_3 + r_1) + (a_3b_2 + r_7) + (a_1b_3 + r_8) + (a_3b_1 + r_{13}),$ $p_2 := (a_2b_4 + r_1),\ p_3 := (a_3b_4 + r_7), p_4 := (a_1b_5 + r_8), p_5 := (a_5b_3 + r_{13})$	4	5
$p_1 := (a_4b_2 + r_2) + (a_2b_4 + r_1) + (a_6b_2 + r_9) + (a_4b_6 + r_{14}),$ $p_2 := (a_2b_3 + r_1),\ p_3 := (a_1b_4 + r_2), p_4 := (a_2b_5 + r_9), p_5 := (a_7b_4 + r_{14})$	5	5
$p_1 := (a_5b_4 + r_3) + (a_1b_4 + r_2) + (a_4b_5 + r_{10}) + (a_1b_5 + r_8),$ $p_2 := (a_4b_2 + r_2),\ p_3 := (a_5b_7 + r_3), p_4 := (a_1b_7 + r_{10}), p_5 := (a_1b_3 + r_8)$	3	5
$p_1 := (a_2b_7 + r_4) + (a_5b_7 + r_3) + (a_5b_2 + r_{11}) + (a_2b_5 + r_9),$ $p_2 := (a_7b_6 + r_4),\ p_3 := (a_5b_4 + r_3), p_4 := (a_5b_6 + r_{11}), p_5 :=$	3	5
$p_1 := (a_6b_7 + r_5) + (a_7b_6 + r_4) + (a_7b_1 + r_{12}) + (a_1b_7 + r_{10}),$ $p_2 := (a_6b_3 + r_5),\ p_3 := (a_2b_7 + r_4), p_4 := (a_4b_7 + r_{12}), p_5 := (a_6b_2 + r_9)$	5	5
$p_1 := (a_3b_6 + r_6) + (a_6b_3 + r_5) + (a_5b_3 + r_{13}) + (a_5b_6 + r_{11}),$ $p_2 := (a_3b_7 + r_6),\ p_3 := (a_6b_7 + r_5), p_4 := (a_3b_1 + r_{13}), p_5 := (a_5b_2 + r_{11})$	3	5
$p_1 := (a_3b_4 + r_7) + (a_7b_4 + r_{14}) + (a_3b_7 + r_6) + (a_4b_7 + r_{12}),$ $p_2 := (a_3b_2 + r_7),\ p_3 := (a_4b_6 + r_{14}), p_4 := (a_3b_6 + r_6), p_5 := (a_7b_1 + r_{12})$	4	5

Probes in scenario (d)	$\lvert I\rvert$	$\lvert J\rvert$
$p_1 := (a_2b_3 + r_1) + (a_3b_2 + r_7) + (a_1b_3 + r_8)$, $p_2 := (a_3b_4 + r_7)$, $p_3 := (a_4b_2 + r_2) + (a_2b_4 + r_1), p_4 := (a_1b_5 + r_8), p_5 := (a_1b_4 + r_2)$	4	4
$p_1 := (a_2b_3 + r_1) + (a_3b_2 + r_7) + (a_1b_3 + r_8)$, $p_2 := (a_2b_4 + r_1)$, $p_3 := (a_3b_4 + r_7) + (a_7b_4 + r_{14}), p_4 := (a_1b_5 + r_8), p_5 := (a_7b_4 + r_{14})$	5	5
$p_1 := (a_4b_2 + r_2) + (a_2b_4 + r_1) + (a_6b_2 + r_9)$, $p_2 := (a_2b_5 + r_9)$, $p_3 := (a_2b_3 + r_1) + (a_3b_2 + r_7), p_4 := (a_3b_4 + r_7), p_5 := (a_1b_4 + r_2)$	5	3
$p_1 := (a_4b_2 + r_2) + (a_2b_4 + r_1) + (a_6b_2 + r_9)$, $p_2 := (a_2b_5 + r_9)$, $p_3 := (a_5b_4 + r_3) + (a_1b_4 + r_2), p_4 := (a_2b_3 + r_1), p_5 := (a_5b_7 + r_3)$	5	5
$p_1 := (a_5b_4 + r_3) + (a_1b_4 + r_2) + (a_4b_5 + r_{10})$, $p_2 := (a_1b_7 + r_{10})$, $p_3 := (a_4b_2 + r_2) + (a_2b_4 + r_1), p_4 := (a_5b_7 + r_3), p_5 := (a_2b_3 + r_1)$	4	5
$p_1 := (a_2b_7 + r_4) + (a_5b_7 + r_3) + (a_5b_2 + r_{11})$, $p_2 := (a_7b_6 + r_4)$, $p_3 := (a_5b_4 + r_3) + (a_1b_4 + r_2), p_4 := (a_4b_2 + r_2), p_5 := (a_5b_6 + r_{11})$	5	4
$p_1 := (a_2b_7 + r_4) + (a_5b_7 + r_3) + (a_5b_2 + r_{11})$, $p_2 := (a_5b_4 + r_3)$, $p_3 := (a_6b_7 + r_5) + (a_7b_6 + r_4), p_4 := (a_6b_3 + r_5), p_5 := (a_5b_6 + r_{11})$	4	5
$p_1 := (a_6b_7 + r_5) + (a_7b_6 + r_4) + (a_7b_1 + r_{12})$, $p_2 := (a_5b_4 + r_3)$, $p_3 := (a_2b_7 + r_4) + (a_5b_7 + r_3), p_4 := (a_6b_3 + r_5), p_5 := (a_4b_7 + r_{12})$	5	5
$p_1 := (a_3b_6 + r_6) + (a_6b_3 + r_5) + (a_5b_3 + r_{13})$, $p_2 := (a_2b_7 + r_4)$, $p_3 := (a_6b_7 + r_5) + (a_7b_6 + r_4), p_4 := (a_3b_1 + r_{13}), p_5 := (a_3b_7 + r_6)$	5	5
$p_1 := (a_3b_4 + r_7) + (a_7b_4 + r_{14}) + (a_3b_7 + r_6)$, $p_2 := (a_2b_4 + r_1)$, $p_3 := (a_2b_3 + r_1) + (a_3b_2 + r_7), p_4 := (a_3b_6 + r_6), p_5 := (a_4b_6 + r_{14})$	4	5
$p_1 := (a_3b_4 + r_7) + (a_7b_4 + r_{14}) + (a_3b_7 + r_6)$, $p_2 := (a_6b_7 + r_5)$, $p_3 := (a_3b_6 + r_6) + (a_6b_3 + r_5), p_4 := (a_3b_2 + r_7), p_5 := (a_4b_6 + r_{14})$	4	5

Now we simulate the wires w_h using only the values $(a_i)_{i\in I}$ and $(b_i)_{i\in J}$.

- We start the simulation with a preliminary phase in which for every $k \in K$ assign r_k to a random and independent value.
- For every probe in group (1), then $i \in I$ and $i \in J$ and so the values are perfectly simulated.
- For every probe in group (2), then, since $k \in K$ and r_k has been simulated in the preliminary phase as a random and independent value, we can compute w_i by using r_k and the indices of the inputs which are, by construction, in the sets I and J;
- For every probe in group (4), then:
  - if for all the random bits r_k used in the computation, $k \in K$, then by construction the indices of the shares of a and b are respectively in I and J and the bits r_k have been taken randomly in the initial step of the simulation and the probe can be simulated using these values;
  - if none of the sums $c_{i,j-h}$ have been probed and there exists at lest one random bit r_k used in the computation such that $k \notin K$, then $c_{i,j}$ can be simulated as a uniform and random value;
  - if a sum $c_{i,j-h}$, with $h = j-1,\ldots,1$, has been probed and every r_k in $c_{i,j} - c_{i,j-h}$ is such that $k \in K$, then we can simulate $c_{i,j}$ from r_k assigned at random in the preliminary phase, the probe $c_{i,j-h}$ and the shares of a and b which are in I and J by construction.

- if a sum $c_{i,j-h}$, with $h = j-1, \ldots, 1$, has been probed and there exists a random bit r_k in $c_{i,j} - c_{i,j-h}$ such that $k \notin K$, then we can simulate $c_{i,j}$ as a uniform and random bit.

Finally, the simulation of the output probes c_i follows the same steps of the probes in group (4) and it can be therefore performed by using at most t_1 shares of the inputs, completing the proof. □

From the lemmas above we deduce that we can compose the multiplication scheme Mult in blocks of gadgets sharing the same randomness, but we point out once more that such blocks need to receive independent inputs, otherwise the hypothesis of the lemmas are not satisfied.

Moreover, we remark that, due to the use of $n > t+1$ shares in the multiplication algorithm for order $t > 3$, the refreshing scheme in Algorithm 2 makes use of a not optimal amount of randomness. Indeed the amount of randomness required, namely $\frac{n(n-1)}{2}$ random bits, can be easily reduced. We propose a refreshing scheme which, for each security order, adds to each input share fresh random bits such that for each output share there are t random bits involved in the computation and each of them is used a second time in a distinct output share. An example for order $t = 4$ is depicted in Algorithm 8. The new scheme makes use of $\frac{n \cdot t}{2}$ random bits and in Lemma 4 it is presented its security analysis.

Algorithm 8 Refreshing scheme $\mathcal{R}'$ for order $t = 4$

Input: shares $a_1, \ldots, a_7$ such that $\bigoplus a_i = a$
Output: shares $c_1, \ldots, c_7$ such that $\bigoplus c_i = a$

$c_1 = a_1 + r_1 + r_7 + r_{13} + r_8$;
$c_2 = a_2 + r_2 + r_1 + r_{14} + r_9$;
$c_3 = a_3 + r_8 + r_{10} + r_3 + r_2$;
$c_4 = a_4 + r_9 + r_4 + r_3 + r_{11}$;
$c_5 = a_5 + r_5 + r_{10} + r_4 + r_{12}$;
$c_6 = a_6 + r_6 + r_{13} + r_{11} + r_5$;
$c_7 = a_7 + r_{14} + r_6 + r_{12} + r_7$;

Lemma 4 (Efficient refreshing when $n \neq t+1$). *Let $\mathcal{R}'_1, \ldots, \mathcal{R}'_N$ be a set of N multiplication schemes as in Algorithm 8, with inputs $\boldsymbol{a}^{(1)}, \ldots, \boldsymbol{a}^{(N)}$ and outputs $\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(N)}$. Suppose that the maskings of the inputs are independent and uniformly chosen and that for $k = 1, \ldots, N$ and that for $k = 1, \ldots, N$ each $\mathcal{R}'_k$ uses the same random bits $\left(r_i\right)_{i=1,\ldots,tn/2}$. Then $\mathcal{R}'_1, \ldots, \mathcal{R}'_N$ are t–SCR and in particular Algorithm 8 is t–SNI.*

Proof. Let $\Omega = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ be a set of t observations respectively on the gadget $\mathcal{R}'_1, \ldots, \mathcal{R}'_N$. We show that $\mathcal{R}'_1, \ldots, \mathcal{R}'_N$ satisfy definition 5, i.e. that the probes in $\mathcal{P}_i$ can be consistently simulated by at most $n-1$ shares of the inputs of $\mathcal{R}'_i$.

Suppose $p_1, \ldots, p_t$ are t adversary's probes. We indicate with $\boldsymbol{r}_{p_i}$ the vector of the respective randomness for every $i = 1, \ldots, t$ and we classify such p_i in the following groups:

(1) $\exists j \in [1,t]$ such that $\boldsymbol{r}_{p_i} = \boldsymbol{r}_{p_j}$
(2) $\exists j \in [1,t]$ such that $\boldsymbol{r}_{p_i} \subset \boldsymbol{r}_{p_j}$
(3) $\forall j \in [1,t]\ \boldsymbol{r}_{p_i} \cap \boldsymbol{r}_{p_j} = \emptyset$
(4) $\boldsymbol{r}_{p_i} = \mathbf{0}$
(5) $\exists j \in [1,t]$such that $\boldsymbol{r}_{p_j} \subset \boldsymbol{r}_{p_i}$ and $\bigcup_j \boldsymbol{r}_{p_j} = \boldsymbol{r}_{p_i}$
(6) $\forall j \in [1,t]$such that $\boldsymbol{r}_{p_j} \subset \boldsymbol{r}_{p_i} : \bigcup_j \boldsymbol{r}_{p_j} \neq \boldsymbol{r}_{p_i}$

We define now the sets $I_1, \ldots, I_N, J_1, \ldots, J_N$ with $|I_i| < n$ $|J_i| < n$ for every $i = 1, \ldots, N$ such that the values of the wires p_h can be perfectly simulated given the values $(a_i^{(1)})_{i \in I_1}$, $(b_i^{(1)})_{i \in J_1}, \ldots, (a_i^{(N)})_{i \in I_N}$, $(b_i^{(N)})_{i \in J_N}$.
The procedure to construct the sets is the following:

- For every wire in the group (1), (2), (4) or (5), add the index of the shares of $\boldsymbol{a}^{(j)}$ in I_j.

We note that, since for every value probed we add at most one input share, $|I_j| < t$ for all j.

We now proceed with the simulation, consistently with the randomness involved.

- First of all, we pick at random every component of $\boldsymbol{r}_{p_i}$.
- Then we simulate every probe, by using the respective vector of random bits defined in the fist phase and by using the share of the input $(a_i^{(j)})_{i \in I_j}$, which exist by construction.

We now simulate the t–SNI property of $\mathcal{R}$.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be a set of t observations respectively on the internal and on the output wires, where $|\mathcal{I}| = t_1$ and in particular $t_1 + |\mathcal{O}| \leq t$. We can classify the internal wires in the following groups:

(1) a_i
(2) $a_i + r_h$, $a_i r_h + r_k$, $a_i r_h + r_k + r_l$
(3) $c_i = a_i r_h + r_k + r_l + r_m$

Suppose an adversary corrupts at most t wires $w_1, \ldots, w_t$. We define two sets I with $|I| < t_1$ such that the values of the wires w_h can be perfectly simulated given the values $(a_i)_{i \in I}$.
The procedure to construct the sets is the following:

- Initially I is empty and the w_i unassigned.
- For every wire in the group (1), (2), (3), (4) add i to I.

Note that by construction $|I| \leq t_1$.

- For each observation as in the group (1), $i \in I$ and then by definition of I the simulator has access to the value of a_i.

– For each observation as in the group (2), by construction $i \in I$ and if all the random bits have been probes, we can pick r_h (resp. r_h, r_k, or r_h, r_k, r_l) at random and simulate the value by using the share a_i. On the other hand, if there exists at least one random value which has not been observed, the probe can be sample uniformly at random.

As for the output wires, we distinguish two cases. If some partial sum has already been observed, we remark that each output share involves the computation of t random bits and each of them appears a second time in a different output share. Therefore, since we have at most $t-1$ additional probes to the current one, there exists at least one random value which has not been observed, the probe can be sample uniformly at random. □

4 1-probing security with constant amount of randomness

The first order ISW scheme is not particularly expensive in terms of randomness, because it uses only one random bit. Unfortunately, when composed in more complicated circuits, the randomness involved increases with the size of the circuit, because we need fresh randomness for each gadget. Our idea is to avoid injecting new randomness in each multiplication and instead alternatively use the same random bits in all gadgets. In particular, we aim at providing a lower bound to the minimum number of bits needed in total to protect any circuit, and moreover show a matching upper bound, i.e., that it is possible to obtain a 1-probing secure private circuit, which uses only a constant amount of randomness. We emphasize that this means that the construction uses randomness that is *independent* of the circuit size, and in particular uses only 2 random bits in total per execution.

We will present a modified version of the usual gadgets for refreshing, multiplication and the linear ones, which, in place of injecting new randomness, use a value taken from a set of two bits chosen at the beginning of each evaluation of the masked algorithm. In particular, we will design these schemes such that they will produce outputs depending on at most one random bit and such that every value in the circuit will assume a fixed form. The most crucial change will be the one at the multiplication and refreshing schemes, which are the randomized gadgets, and so responsible for the accumulation of randomness. On the other hand, even tough the gadget for the addition does not use random bits, it will be subjected at some modifications as well, in order to avoid malicious situations that the reusing of the same random bits in the circuit can cause. As for the other linear gadgets, such as the powers $.^2$, $.^4$, etc., they will be not affected by any change, but will perform as usual share-wise computation.

We proceed by showing step by step the strategy to construct such circuits. First, we fix a set of bits $R = \{r_0, r_1\}$ where r_0 and r_1 are taken uniformly at random. The first randomized gadget of the circuit does not need to be substantially modified, because there is no accumulation of randomness to be avoided yet. The only difference with the usual multiplication and refreshing gadgets is

that, in place of the random component, we need to use one of the random bits in R, as shown in Algorithm 9 and Algorithm 10. Notice that when parts of the operations are written in parenthesizes, then this means that these operations are executed first.

Algorithm 9 1-SecMult case (i)

Input: shares a_1, a_2 such that $a_1 \oplus a_2 = a$, shares b_1, b_2 such that $b_1 \oplus b_2 = b$
Output: shares c_i depending on a random number $r_k \in R$ such that $c_1 \oplus c_2 = a \cdot b$, the value r_k

$r_k \xleftarrow{\$} R$;
$c_1 \leftarrow a_1 b_1 + (a_1 b_2 + r_k)$;
$c_2 \leftarrow a_2 b_1 + (a_2 b_2 - r_k)$;

Algorithm 10 Refreshing case (i)

Input: shares a_1, a_2 such that $a_1 \oplus a_2 = a$
Output: shares c_i depending on the random number $r_k \in R$ such that $c_1 \oplus c_2 = a$, the value r_k

$r_k \xleftarrow{\$} R$;
$c_1 \leftarrow a_1 + r_k$;
$c_2 \leftarrow a_2 - r_k$;

Secondly, we analyze the different configurations that an element can take when not more than one randomized gadget has been executed, i.e. when only one random bit has been used in the circuit. The categories listed below are then the different forms that such an element takes if it is respectively the first input of the circuit, the output of the first refreshing scheme as in Algorithm 2 and the one of the first ISW multiplication scheme as in Algorithm 1 between two values x and y:

(1) $a = (a_1, a_2)$;
(2) $a = (a_1 + r, a_2 - r)$, where r is a random bit in R;
(3) $a = (x_1 y_1 + x_1 y_2 + r, x_2 y_1 + x_2 y_2 - r)$, where r is a random bit in R.

This categorization is important because according to the different form of the values that the second randomized gadget takes in input, the scheme will accumulate randomness in different ways. Therefore, we need to modify the gadgets by taking into account the various possibilities for the inputs, i.e. distinguish if:

(i) both the inputs are in category (1);
(ii) the first input is as in category (1), i.e. $a = (a_1, a_2)$, and the second one in category (2), i.e. $b = (b_1 + r_1, b_2 - r_1)$;

(iii) the first input is as in category (1), i.e. $a = (a_1, a_2)$, and the second one in category (3), i.e. $b = (c_1d_1 + c_1d_2 + r_1, c_2d_1 + c_2d_2 - r_1)$;
(iv) the first input is in category (3), i.e. $a = (c_1d_1 + c_1d_2 + r_0, c_2d_1 + c_2d_1 - r_0)$, and second one in category (2), i.e. $b = (b_1 + r_1, b_2 - r_1)$;
(v) both inputs are in category (2), i.e. $a = (a_1 + r_1, a_2 - r_1)$ and $b = (b_1 + r_0, b_2 - r_0)$;
(vi) both inputs values are in category (3), i.e. $a = (c_1d_1 + c_1d_2 + r_1, c_2d_1 + c_2d_2 - r_1)$ and $b = (c'_1d'_1 + c'_1d'_2 + r_0, c'_2d'_1 + c'_2d'_2 - r_0)$.

where for the moment we suppose that the two inputs depend on two different random bits each, but a more general scenario will be analyzed later. The goal of the modified gadgets that we will present soon will be not only to reuse the same random bits, avoiding an accumulation at every execution, but also to produce outputs in the groups (1), (2) or (3), in order to keep such a configuration of the wires unchanged throughout the circuit. In this way we guarantee that every wire depends only on one random bit and that we can use the same multiplication schemes in the entire circuit. According to this remark we modify the ISW as depicted in Algorithms 11 and 12.

Algorithm 11 1-SecMult case (ii) and (iii)

Input: shares a_1, a_2 such that $a_1 \oplus a_2 = a$, shares b_1, b_2 depending on a random number $r_i \in R$ such that $b_1 \oplus b_2 = b$, the set $R = \{r_0, r_1\}$, r_i
Output: shares c_i depending on the random number r_{1-i} such that $c_1 \oplus c_2 = a \cdot b$, the value r_{1-i}

$c_1 \leftarrow a_1b_1 + (a_1b_2 + r_{1-i})$;
$c_2 \leftarrow a_2b_1 + (a_2b_2 - r_{1-i})$;

Algorithm 12 1-SecMult case (iv), (v) and (vi)

Input: shares a_1, a_2 depending on the random number r_i such that $a_1 \oplus a_2 = a$, shares b_1, b_2 depending on the random number r_{1-i} satisfying $b_1 \oplus b_2 = b$, the set $R = \{r_0, r_1\}$
Output: shares c_i depending on the random number $r_{1-i} \in R$ satisfying $c_1 \oplus c_2 = a \cdot b$, the value r_{1-i}

$\delta \leftarrow -r_{1-i}$;
$\delta \leftarrow \delta + r_ib_1$;
$\delta \leftarrow \delta + r_ib_2$;
$c_1 \leftarrow a_1b_1 + (a_1b_2 - \delta)$;
$c_2 \leftarrow a_2b_1 + (a_2b_2 + \delta)$;

It is easy to prove that the new multiplication algorithms are such that their outputs always belong to group (3).

Lemma 5. *Let a and b be two input values of Algorithm 11 or of Algorithm 12. Then the output value $e = a \cdot b$ is of the form* (3).

As specified before, in the previous analysis we supposed to have as input of the multiplication schemes values depending on different random bits. Since this is not always the case in practice, we need to introduce a modified refreshing scheme, which replaces the random bit on which the input depends with the other random bit of the set R. The scheme is presented in Algorithm 13 and it has to be applied to one of the input values of a multiplication scheme every time that they depend on the same randomness. Algorithm 13 is also useful before a XOR gadget with inputs depending on the same random bit, because it avoids that the randomness is canceled out. The proof of correctness is quite

Algorithm 13 Modified refreshing $\mathcal{R}'$

Input: shares a_1, a_2 such that $a_1 \oplus a_2 = a$ depending on a random bit r_i, the value r_i
Output: shares c_i depending on the random number r_{1-i} such that $c_1 \oplus c_2 = a$, the value r_{1-i}

$c_1 \leftarrow (a_1 + r_{1-i}) - r_i$;
$c_2 \leftarrow (a_2 - r_{1-i}) + r_i$;

straightforward, therefore we provide only an exemplary proof for a value in category (3).

Lemma 6. *Let a be an input value of the form* (3) *depending on a random bit $r_i \in R$ for Algorithm 13. Then the output value is of the form* (3) *and depends on the random bit r_{1-i}.*

Proof. Suppose without loss of generality that the input a depends on the random bit r_1, so that $a = (c_1d_1 + c_1d_2 + r_0, c_2d_1 + c_2d_1 - r_0)$. Then the output $e = \mathcal{R}'(a)$ is:

$$e_1 = (c_1d_1 + c_1d_2 + r_0 + r_1) - r_0 = c_1d_1 + c_1d_2 + r_1$$
$$e_2 = (c_2d_1 + c_2d_1 - r_0 - r_1) + r_0 = c_2d_1 + c_2d_1 - r_1$$

completing the proof. □

Lastly, in Algorithm 14 we define a new scheme for addition, which allows to have outputs in one of the three categories (1), (2) or (3). Note that thanks to the use of the refreshing $\mathcal{R}'$, we can avoid having a dependence on the same random bit in the input of an addition gadget. The proof of correctness is again quite simple

In conclusion, we notice that by using the schemes above and composing them according to the instructions just given, we obtain a circuit where each wire carries a value of a fixed form (i.e. in one of the categories (1), (2) or (3)) and therefore we can always use one of the multiplication schemes given in the Algorithms 9, 11 and 12 without accumulating randomness and without the risk of canceling the random bits. Moreover, it is easy to see that all the schemes just presented are secure against a 1-probing attack.

Algorithm 14 Modified addition XOR'

Input: shares a_1, a_2 such that $a_1 \oplus a_2 = a$ depending on a random bit r_i, shares b_1, b_2 such that $b_1 \oplus b_2 = b$ depending on a random bit r_{1-i}
Output: shares c_i depending on a random number $r_k \in R$ such that $c_1 \oplus c_2 = a + b$, the value r_k

$r_k \xleftarrow{\$} R$;
$c_1 \leftarrow a_1 + b_1 - r_k$;
$c_2 \leftarrow a_2 + b_2 + r_k$;

4.1 Impossibility of the 1-bit randomness case

In the following we show that is impossible in general to have a 1st-order probing secure circuit, which uses only 1 bit of randomness in total. In particular, we present a counterexample which breaks the security of a circuit using only one random bit.

Let us consider c and c' two outputs of two multiplication schemes between the values a, b and a', b' respectively, and let r be the only random bit which is used in the entire circuit. Then c and c' are of the form

$$\begin{cases} c_1 = a_1b_1 + a_1b_2 + r \\ c_2 = a_2b_1 + a_2b_2 + r \end{cases} \quad \text{and} \quad \begin{cases} c'_1 = a'_1b'_1 + a'_1b'_2 + r \\ c'_2 = a'_2b'_1 + a'_2b'_2 + r \end{cases}.$$

Suppose now that these two values are inputs of an additive gadget, as in Figure 4. Such a gadget could either use no randomness at all and just add the components each other, or involve in the computation the bit r maintaining the correctness. In the first case we obtain

$$\begin{cases} c'_1 + c_1 = a_1b_1 + a_1b_2 + a'_1b'_1 + a'_1b'_2 = a_1b + a'_1b' \\ c'_2 + c_2 = a_2b_1 + a_2b_2 + a'_2b'_1 + a'_2b'_2 = a_2b + a'_2b' \end{cases}$$

and then the randomness r will be completely canceled out, revealing the secret. In the second case, if we inject in the computation another r, then, in whatever point of the computation we put it, it will cancel out again one of the two r revealing one of the secrets during the computation of the output. For example, we can have

$$\begin{cases} c'_1 + c_1 = r + a_1b_1 + a_1b_2 + r + a'_1b'_1 + a'_1b'_2 + r = a_1b + a'_1b'_1 + a'_1b'_2 + r \\ c'_2 + c_2 = r + a_2b_1 + a_2b_2 + r + a'_2b'_1 + a'_2b'_2 + r = a_2b + a'_2b'_1 + a'_2b'_2 + r \end{cases}.$$

In view of this counterexample, we can conclude that the minimum number of random bits needed in order to have a 1st-order private circuit is 2.

5 Case study: AES

To evaluate the impact of our methodology on the performance of protected implementations, we implemented AES-128 without and with common randomness. In particular, we consider the inversion of each Sbox call (cf. Figure 5) as

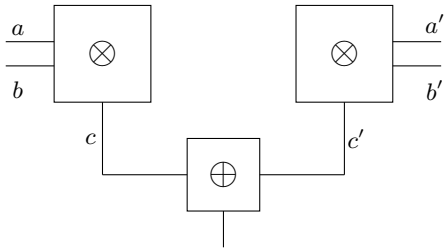

Fig. 4.

a block of gadgets $\mathcal{G}_{i=1,\ldots,200}$ using the same random components and the last multiplication of each of these inversions is using fresh randomness. For the im-

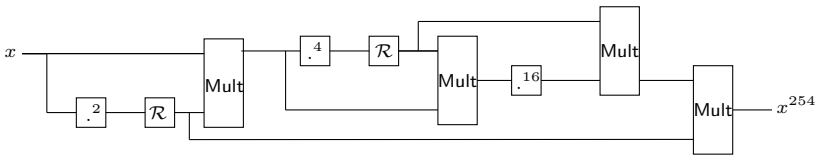

Fig. 5. Inversion gadget of the AES Sbox with the insertion of the Ind

plementation without common randomness, we use the multiplication algorithm from [18] and the refresh from [10] (cf. Algorithm 2). To enable the use of common randomness, we replace the multiplication with our t–SCR multiplication, the refresh with Algorithm 3.3 for $t > 3$, and we place the Ind gadgets where needed. In particular, since the inputs of the first call of the inversion are independent, we do not need to add Ind before the first multiplication scheme with common randomness, while this is needed for the next multiplications in the circuit. On the other hand, the last multiplication scheme of each inversion needs to be completely refreshed, because the insertion of the gadgets Ind for guaranteeing the independence of the inputs would make the circuit insecure. For this reason, the outputs of each inversion are independent again and therefore the same procedure can be used for each round. The algorithm is depicted in Figure 6 and in Lemma 7 it is presented its security analysis after the introduction of the Ind gadgets.

Lemma 7. *Gadget* $.^{254}$*, depicted in Figure 6, is* t–SNI.

Proof. Let $\Omega = (\bigcup_{1\leq i\leq 11}, \mathcal{O})$ be a set of t probes such that $\sum_{1\leq i\leq 11} |\mathcal{I}^i| \leq t_1$ $|\mathcal{O}|+t_1 \leq t$. In the following, we show the existence of a simulator which simulates the probes by using at most t_1 internal values. The simulation is processed from right to left, according to the enumeration in Figure 6.

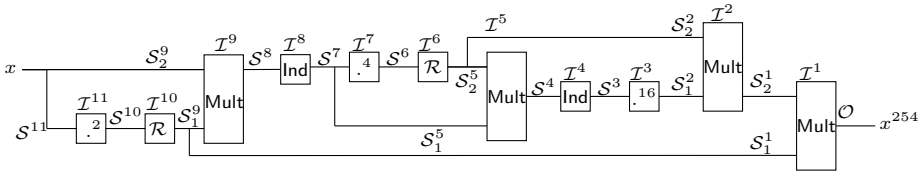

Fig. 6. Inversion gadget of the AES Sbox with the insertion of the Ind

- *Gadget* 1: Since Mult is t–SNI and $|\mathcal{I}^1 \cup \mathcal{O}| \leq t$, then there exist two sets of indices $\mathcal{S}_1^1$ and $\mathcal{S}_2^1$ such that $|\mathcal{S}_1^1| \leq |\mathcal{I}^1|$, $|\mathcal{S}_2^1| \leq |\mathcal{I}^1|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}_1^1$ and $\mathcal{S}_2^1$.
- *Gadget* 2: Since Mult is t–SNI and $|\mathcal{I}^2 \cup \mathcal{S}_2^1| \leq t$, then there exist two sets of indices $\mathcal{S}_1^2$ and $\mathcal{S}_2^2$ such that $|\mathcal{S}_1^2| \leq |\mathcal{I}^2|$, $|\mathcal{S}_2^2| \leq |\mathcal{I}^2|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}_1^2$ and $\mathcal{S}_2^2$.
- *Gadget* 3: Since $.^{16}$ is linear then there exists a set of indices $\mathcal{S}^3$ such that $|\mathcal{S}^3| \leq |\mathcal{I}^3| + |\mathcal{S}_1^2| \leq |\mathcal{I}^3| + |\mathcal{I}^2|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}^3$.
- *Gadget* 4: Since Ind is t–NI then there exists a set of indices $\mathcal{S}^4$ such that $|\mathcal{S}^4| \leq |\mathcal{I}^4| + |\mathcal{S}^3| \leq |\mathcal{I}^4| + |\mathcal{I}^3| + |\mathcal{I}^2|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}^4$.
- *Gadget* 5: Since Mult is t–SNI and $|\mathcal{I}^5 \cup \mathcal{S}^4| \leq t$, then there exist two sets of indices $\mathcal{S}_1^5$ and $\mathcal{S}_2^5$ such that $|\mathcal{S}_1^5| \leq |\mathcal{I}^5|$, $|\mathcal{S}_2^5| \leq |\mathcal{I}^5|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}_1^5$ and $\mathcal{S}_2^5$.
- *Gadget* 6: Since $\mathcal{R}$ is t–SNI and $|\mathcal{I}^6 \cup \mathcal{S}_2^5| \leq t$, then there exist a set of indices $\mathcal{S}^6$ such that $|\mathcal{S}^6| \leq |\mathcal{I}^6|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}^6$.
- *Gadget* 7: Since $.^4$ is linear then there exists a set of indices $\mathcal{S}^7$ such that $|\mathcal{S}^7| \leq |\mathcal{I}^7| + |\mathcal{S}^6| \leq |\mathcal{I}^7| + |\mathcal{I}^6|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}^7$.
- *Gadget* 8: Since Ind is t–NI then there exists a set of indices $\mathcal{S}^8$ such that $|\mathcal{S}^8| \leq |\mathcal{I}^8| + |\mathcal{S}^7| \leq |\mathcal{I}^8| + |\mathcal{I}^7| + |\mathcal{I}^6|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}^8$.
- *Gadget* 9: Since Mult is t–SNI and $|\mathcal{I}^9 \cup \mathcal{S}^8| \leq t$, then there exist two sets of indices $\mathcal{S}_1^9$ and $\mathcal{S}_2^9$ such that $|\mathcal{S}_1^9| \leq |\mathcal{I}^9|$, $|\mathcal{S}_2^9| \leq |\mathcal{I}^9|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}_1^9$ and $\mathcal{S}_2^9$.
- *Gadget* 10: Since $\mathcal{R}$ is t–SNI and $|\mathcal{I}^{10} \cup \mathcal{S}_1^9| \leq t$, then there exist a set of indices $\mathcal{S}^{10}$ such that $|\mathcal{S}^{10}| \leq |\mathcal{I}^{10}|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}^{10}$.

– *Gadget* 11: Since $.^2$ is linear then there exists a set of indices $\mathcal{S}^{11}$ such that $|\mathcal{S}^{11}| \leq |\mathcal{I}^{11}| + |\mathcal{S}^{10}| \leq |\mathcal{I}^{11}| + |\mathcal{I}^{10}|$ and the gadget can be perfectly simulated from its input shares corresponding to the indices in $\mathcal{S}^{11}$.

Each of the previous steps show the existence of a simulator each of the respective gadgets and by composing them we obtain a global simulator of the entire circuit which uses $|\mathcal{S}^{11} \cup \mathcal{S}_2^9|$ shares of the input. Since $|\mathcal{S}^{11} \cup \mathcal{S}_2^9| \leq \mathcal{I}^{11}| + |\mathcal{I}^{10}| + |\mathcal{I}^9| \leq t_1$ we conclude that the gadget $.^{254}$ is t–SNI.

□

Table 1 summarizes the randomness requirements of both types of refresh and multiplication algorithms for increasing orders.

Table 1. Number of random elements required for the multiplication and refresh algorithms with and without common randomness from $t = 1$ to $t = 11$.

	Without Common Randomness			*With Common Randomness*		
t	n	**Multiplication**	**Refresh**	n	**Multiplication**	**Refresh**
1	2	1	1	2	1	1
2	3	3	3	3	3	3
3	4	6	6	4	6	6
4	5	10	10	7	18	14
5	6	15	15	7	21	18
6	7	21	21	13	78	39
7	8	28	28	13	78	46
8	9	36	36	21	210	84
9	10	45	45	21	210	95
10	11	55	55	31	465	155
11	12	66	66	31	465	171

Both types of protected AES were implemented on an ARM Cortex-M4F running at 168 MHz using C. The random components were generated using the TRNG of the evaluation board (STM32F4 DISCOVERY) which generates 32 bits of randomness every 40 clock cycles running in parallel at 48 MHz. To assess the influence of the TRNG performance on the result, we considered two modes of operation for the randomness generation. For $\texttt{TRNG}_{32}$, we use all 32 bits provided by the TRNG by storing them in a buffer and reading them in 8-bit parts when necessary. To simulate a slower TRNG, we also evaluated the performance of our implementations using $\texttt{TRNG}_8$ which only uses the least significant 8 of the 32 bits resulting in more idle states waiting for the TRNG to generate a fresh value. We applied the same degree of optimization on both

implementations to allow a fair comparison. While it is possible to achieve better performances using Assembly (as recently shown by Goudarzi and Rivain in [13]) our implementations still suffice as a proof of concept. The problem of randomness generation affects a majority of implementations independent of the degree of optimization and can pose a bottleneck, especially if no dedicated TRNG is available. Therefore, we argue that our performance results can be transferred to other types of implementations and platforms, and we expect a similar performance improvement if the run time is not completely independent of the randomness generation (e.g., pre-computed randomness).

As shown in Table 2, the implementations with common randomness requires fewer calls to the TRNG for most of the considered t. Only after $t > 9$, the randomness complexity of the fresh multiplications with increased shares becomes too high. The runtime benefit of common randomness strongly depends on the performance of the random number generator. While for the efficient $\texttt{TRNG}_{32}$ our approach leads to faster implementations only until $t = 5$, it is superior until $t = 7$ for the slower $\texttt{TRNG}_8$. The dependency on the performance of the randomness generation is visualized in Figure 7. For $\texttt{TRNG}_8$, the curve is shifted downwards compared to the faster generator. In theory, an even slower randomness generator could move the break-even point to after $t = 9$ for our scenario, i.e., until the implementation with common randomness requires more TRNG calls.

Table 2. Cycle counts of our AES implementations on an ARM Cortex-M4F with $\texttt{TRNG}_{32}$. In addition, we provide the required number of calls to the TRNG for each t.

	Without Common Randomness				*With Common Randomness*			
		TRNG	**Cycle Count**			**TRNG**	**Cycle Count**	
t	n	**Calls**	$\texttt{TRNG}_{32}$	$\texttt{TRNG}_8$	n	**Calls**	$\texttt{TRNG}_{32}$	$\texttt{TRNG}_8$
					2	2	73,441	73,560
1	2	1,200	112,919	187,519	2	605	86,137	99,334
2	3	3,600	308,600	548,477	3	1,415	200,423	230,334
3	4	7,200	496,698	1,089,092	4	2,430	292,579	412,523
4	5	12,000	751,670	1,812,213	7	6,082	721,438	1,151,922
5	6	18,000	1,051,323	2,729,052	7	6,699	754,117	1,255,033
6	7	25,200	1,403,243	3,836,006	13	20,712	2,299,960	3,952,722
7	8	33,600	1,779,403	5,125,072	13	20,726	2,315,160	3,960,749
8	9	43,200	2,286,003	6,603,199	21	50,798	5,656,097	10,165,790
9	10	54,000	2,814,435	8,257,996	21	50,820	5,671,608	10,208,375
10	11	66,000	3,459,684	10,096,735	31	106,705	10,908,480	22,612,297
11	12	79,200	4,046,836	12,112,375	31	106,737	10,919,488	22,630,028

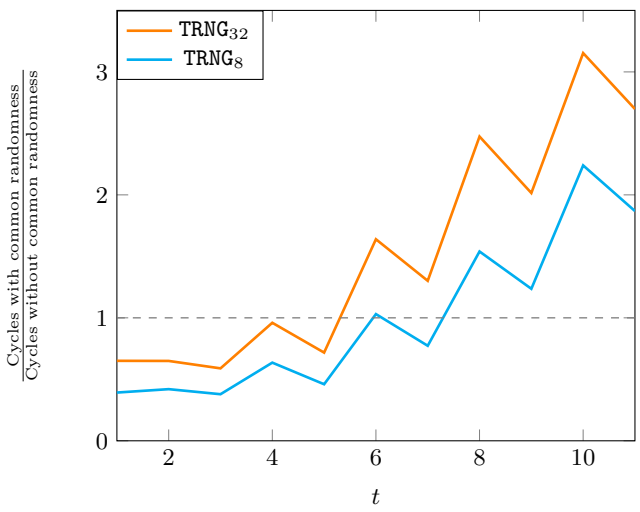

Fig. 7. Ratio between the cycle counts of the AES implementations from Table 2 with and without common randomness for each t.

For the special case of $t = 1$, we presented a solution (cf. Section 4) with constant randomness independent of the circuit size. Following the aforementioned procedure, we realized an 1-probing secure AES implementation with only two TRNG calls. Overall, the implementation using the constant randomness scheme requires less cycles than the one with common randomness. In general, however, this advantage strongly depends on the performance of the TRNG as the implementation with constant randomness requires additional operations to ensure security.

6 Conclusion

Since the number of shares n for our t–SCR multiplication grows in $\mathcal{O}(t^2)$ and our multiplication algorithm (resp. $\mathcal{R}$) requires $\mathcal{O}(n^2)$ (resp. $\mathcal{O}(nt)$) random elements, the practicability of our proposed methodology becomes limited for increasing t. Nevertheless, our case study showed that for small t our approach results in significant performance improvement for the masked implementations. The improvement factor could potentially be even larger, if we replace our efficient TRNG with a common PRNG. Additionally, an improved multiplication or $\mathcal{R}$ with a smaller randomness complexity, e.g., $\mathcal{O}(t^2)$, could lead to better performances even for $t \geq 10$ and is an interesting starting point for future work. This would be of interest as with time larger security orders might be required to achieve long-term security.

Another interesting aspect for future work is designing an automatic application of our methodology to an arbitrary circuit. While we showed how to use

our methodology on a circuit with a layered structure which contains easily-exploitable regularities, further research might be able to derive an algorithm which finds the optimal grouping for any given design. This would help to create a compiler which automatically applies masking to an unprotected architecture in the most efficient way removing the requirement for a security-literate implementer and reducing the chance for human error.

Acknowledgments. Sebastian Faust and Clara Paglialonga are partially funded by the Emmy Noether Program FA 1320/1-1 of the German Research Foundation (DFG). Tobias Schneider is partially funded by European Unions Horizon 2020 program under project numbers 645622 PQCRYPTO and 724725 SWORD. This work is also partially supported by the VeriSec project 16KIS0634 - 16KIS0602 from the Federal Ministry of Education and Research (BMBF).

References

1. Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
2. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, and Benjamin Grégoire. Compositional verification of higher-order masking: Application to a verifying masking compiler. Technical report, Cryptology ePrint Archive, Report 2015/506, 2015.
3. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *CHES 2016*, pages 23–39, 2016.
4. Sonia Belaïd, Fabrice Benhamouda, Alain Passelgue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. Cryptology ePrint Archive, Report 2016/211, 2016. `http://eprint.iacr.org/2016/211`.
5. Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and first-order DPA resistant implementations of keccak. In *CARDIS*, volume 8419 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2013.
6. Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In *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 II*, pages 326–343, 2014.
7. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO'99*, pages 398–412, 1999.
8. Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-order threshold implementation of the AES s-box. In *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS*

2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers, pages 259–272, 2015.

9. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In *International Workshop on Fast Software Encryption*, pages 410–424. Springer, 2013.
10. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 423–440. Springer, 2014.
11. Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 135–156, 2010.
12. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In *CHES'99*, pages 158–172, 1999.
13. Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In *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*, pages 567–597, 2017.
14. Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust pseudorandom generators. In *International Colloquium on Automata, Languages, and Programming*, pages 576–588. Springer, 2013.
15. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.
16. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.
17. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 764–783, 2015.
18. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 413–427, 2010.

Appendix

A Construction of t–SCR Multiplication Schemes

As noted in Section 3.2, our approach to construct a t–SCR multiplication schemes consists of two separate steps. Firstly, we find a $\lfloor \frac{t}{2} \rfloor$-non-complete mapping of the input shares to the output shares and, secondly, we include random components to achieve the required t–SNI property. Especially, the first step becomes a non-trivial task for increasing t. In the following, we present a basic algorithm that constructs a t–SCR multiplication for a given t and give concrete instantiations for up to $t = 11$.

A.1 Basic Algorithm

In the first step, we have to find a $\lfloor \frac{t}{2} \rfloor$-non-complete mapping of the input shares to the output shares. This problem can be described as follows. Given a set containing the indices of all n input shares as

$$\mathcal{M}_{all} = \{1, 2, \ldots, n-1, n\},$$

we have to find subsets $\mathcal{M}_{i=1,\ldots,n}$ with cardinality l such that every element of $\mathcal{M}_{all}$ shares *exactly* one subset with every other element of $\mathcal{M}_{all}$. Each of these subsets refers to one output share and determines which l input shares are leaked, i.e., the l elements of the subset. Therefore, this construction automatically ensures $\lfloor \frac{t}{2} \rfloor$-non-completeness and that correctness can be achieved, since every term of the cross-product of the input shares can be distributed to a designated output share. In the following, we denote the set of the subsets $\mathcal{M}_i$ as $\mathcal{M}$ and describe a basic algorithm to construct it.

In our algorithm, $\mathcal{M}$ is accessed similar to an $n \times l$ array with n rows and l columns, e.g., $\mathcal{M}[i, j]$ refers to the j-th element of the i-th subset. At the start, every entry is initialized with zero. Our algorithm traverses the array entry-by-entry by first processing every column of the current row before jumping to the first unassigned column of the next row. In each processing step, the algorithm checks which element of $\mathcal{M}_{all}$ can be used without creating a conflict with the existing elements of the row, and assigns the first valid value to the entry. This process is illustrated in Steps 1 and 2 of Figure 8, where $\mathcal{M}[4, 3]$ and $\mathcal{M}[5, 3]$ are completed. If at some point there are no valid values for the current entry, the algorithm backtracks and changes the previous entry to the next possible valid value. The same applies if all possible values have been processed before.

To increase the performance of the algorithm, it is possible to initialize $\mathcal{M}$ with preassigned values. In particular, we can fill the first l rows easily without creating conflicts. The same applies to the first two columns of the remaining rows. Initialization reduces the search space, but requires an adjustment to the traversing algorithm to ignore these preassigned entries. The result of the initialization for $t = 4, n = 7$ is depicted in the left part of Figure 8.

A pseudo-code representation of the whole process is given in Algorithm 15. We assume that the functions Freq and PossibleEntries are efficiently implemented. This can be achieved with additional arrays which are constantly updated for every new entry. To increase the comprehensibility of Algorithms 15 and 16, we explicitly omit these extra steps and instead provide a description of the two functions' behavior:

- Freq($\mathcal{M}$): Returns the number of occurrence of each element of $\mathcal{M}_{all}$ in $\mathcal{M}$.
- PossibleEntries($\mathcal{M}, row$): Returns a list of all valid values of the current row.

Algorithm 15 returns a $\lfloor \frac{t}{2} \rfloor$-non-complete mapping as depicted in the right part of Figure 8. To derive the multiplication scheme, it is necessary to sum the related terms of the cross-product for each output share, i.e., for output share c_i it is required to sum $a_j b_k + a_k b_j$ for $\forall j, k \in \mathcal{M}_i$ with $j \neq k$. The terms $a_j b_j, j = 1, \ldots, n$ can be assigned to every output shares that includes a particular j. For simplicity, we distributed these terms equally over all output shares. We only provide the leaking indices $\mathcal{L}$ for every order in Tables 3 to 5. Furthermore, we adjusted the order of indices to ensure that the first index of c_i is also present in c_{i-1} for the later-discussed security property. The generic multiplication based on $\mathcal{L}$ for $t \geq 6$ is given in Algorithm 17.

While the algorithm returns a $\lfloor \frac{t}{2} \rfloor$-non-complete mapping for $t = 3$, it does not lead to a correct t–SCR multiplication scheme. As noted before, in this special case the number of shares is not bounded by the non-completeness, but rather by the t–SNI requirement of $t < n$.

After finding a $\lfloor \frac{t}{2} \rfloor$-non-complete mapping for a given t, it is necessary to distribute the random components required for t–SNI equally over the output shares. To this end, we group the terms for each output share into equally sized groups and add the random components with increasing indices. Each of the $\frac{n^2}{2}$ components needs to be added twice during the whole process. To avoid collisions of combinations of random components, we shift each random column by a different offset except for the original columns. We do not provide an explicit algorithm to distribute the randomness for $t < 6$. Instead, we briefly describe the intuition behind the distribution for these cases.

$t = 2$ Since the adversary has only two probes, there are two possible scenarios we have to consider:

1. $(\mathsf{g} = 2, \mathsf{g}' = 0)$ This attack vector is prevented by the t–SNI property of the multiplication which implies probing security.
2. $(\mathsf{g} = 1, \mathsf{g}' = 1)$ The best case for an adversary is probing the same output share in both gadgets, i.e., receiving the maximum number of input shares. This attack is prevented by the non-completeness property of the gadget which ensures that every output share only contains two of three input shares.

Therefore, the randomness distribution for $t = 2$ is very straight-forward as it is sufficient to have t–SNI with non-completeness to achieve t–SCR.

$t = 3$ For this adversary, it is again enough to consider two possible scenarios:

1. $(\mathsf{g} = 3, \mathsf{g}' = 0)$ This attack vector is prevented by the t–SNI property of the multiplication which implies probing security.
2. $(\mathsf{g} = 2, \mathsf{g}' = 1)$ The best case for an adversary is probing the same output share in both gadgets, i.e., receiving the maximum number of input shares for two shared probes. Based on the distribution of cross-product terms this would result in two shares probed of each input. An adversary would need to probe the remaining two shares of one of the inputs with only one probe. However, given the t–SNI randomness distribution and the fact that the $a_i b_i$ are added at last to the output shares, this is not possible, as she can only probe one term $a_i b_j$, i.e., only one share of each input, because the remaining possibilities which include two unknown shares of one input are masked by unknown randomness.

$t = 4$ For this adversary, it is enough to consider the following three possible scenarios:

1. $(\mathsf{g} = 4, \mathsf{g}' = 0)$ This attack vector is prevented by the t–SNI property of the multiplication which implies probing security.
2. $(\mathsf{g} = 2, \mathsf{g}' = 2)$ Again the best case for an adversary is to probe the same pair of output shares in the both gadgets. This leads the maximum number of shares probed. Since every output share leaks only about three input shares thanks to our non-complete distribution, the adversary would only receive

six out of seven shares of each input in the worst case. In practice this attack would even result in only five input shares given that every output shares exactly one input share with any other output share.

3. ($\mathsf{g} = 3, \mathsf{g}' = 1$) With the shared probe and one normal probe, it is again the best case for an adversary to probe the same output share to gain information about the maximum number of input shares, i.e., three. Given the randomness distribution, the adversary can either probe the terms of the cross-product directly (which leak about one share of each input) or the intermediate sums of the output shares. However, these are partly masked by unprobed randomness and, therefore, would require additional probes on this randomness. In the first scenario, the adversary could only probe two of the four unknown shares. In the second scenario, the adversary could probe the intermediate sum after the first addition with a random element which was already probed with the shared probes. However, this only results in one new share. An alternative approach would be to probe an intermediate sum after the fourth addition. Again we assume that the first random element of this output share was already probed before. However, this time we add another unknown random which needs to be probed as well with the second remaining probe. Therefore, in total the adversary can only access up to two unknown input shares.

$t = 5$ For this adversary, it is enough to consider the following three possible scenarios:

1. ($\mathsf{g} = 5, \mathsf{g}' = 0$) This attack vector is prevented by the t–SNI property of the multiplication which implies probing security.
2. ($\mathsf{g} = 3, \mathsf{g}' = 2$) Same as for $t = 4$, the adversary could probe five of the seven input shares with two shared and two normal probes. However, now there is still one additional probe remaining. Given the computation order, the adversary has two choices for the last probe. In the first scenario, she can probe an intermediate term $a_i b_j$ or $a_i b_i$. However, this provides only one of the two remaining input shares. In the second scenario, the adversary probes one of the $a_i b_i$ terms which also only leaks about one input share. In the second scenario, the adversary probes an intermediate sum of the output shares. Thanks to the randomness distribution based on rotation, the maximum number of random terms per output share, which were leaked by the first four probes, is two. Therefore, the adversary could only probe $(a_i b_j + r_k) + (a_j b_i + r_l)$ for known r_k, r_l. This would indeed leak about two input shares and possibly enabling the adversary to successfully attack the scheme. Therefore, we distributed the randomness and cross-product terms in a fashion that prevents this attack. In particular, to probe the intermediate sum of c_i, the adversary needs to probe c_{i-1} and c_{i+1} (except for c_7) with the shared probes. Otherwise, an unprobed randomness trivially prevents any attack. The cross-products are distributed in way that output shares c_{i-1}, c_i, c_{i+1} (except c_6, c_7, c_1) only leak about six input shares at maximum preventing any attack based on the intermediate sum of c_i. In the case $c_i = c_7$, we slightly adjusted the order of the random terms such that the adversary needs to probe c_1, c_2 to attack the intermediate sum of c_7. Since the tuple c_1, c_2, c_7 is again missing one of the input shares, it cannot be used for an attack.
3. ($\mathsf{g} = 4, \mathsf{g}' = 1$) With one shared and the corresponding normal probe, the adversary can access three input shares and six random elements. With the remaining probes the adversary can probe either the cross-products $a_i b_j$ directly (which only amounts to three of the four unknown shares) or intermediate sums until the second term (otherwise she adds too many unknown randomness terms). This situation corresponds to the scenario of attack (a) analyzed in the proof of Proposition 2, which shows that this attack would give to the adversary the knowledge of at most other three input shares, of the four needed to recover the secret inputs.

$t \geq 6$ For the general construction, we need to distinguish between two cases:

1. ($\mathsf{g} = t, \mathsf{g}' = 0$) This attack vector is prevented by the t–SNI property of the multiplication which implies probing security.

2. ($\mathsf{g} = t - \delta, \mathsf{g}' = \delta$) In the case of $\delta = \lfloor \frac{t}{2} \rfloor$, the security can be easily derived from the non-completeness property of the multiplication. Based on the relationship between l, n, and t given in Section 3.2, we can derive the maximum number of shares which can be leaked by the shared probes as

$$n_{probed} = l + (\delta - 1)(l - 1) = n - \lfloor \frac{t}{2} \rfloor. \tag{4}$$

For even t, this property is already sufficient given that $n_{probed} < n$. However, for odd t, there is still one remaining probe. With this, the adversary needs to probe the remaining $\lfloor \frac{t}{2} \rfloor$ shares targeting an intermediate sum which uses only known random elements. Since we compute from left to right, the adversary needs to know the first random elements used to compute the output shares. The distribution of the cross-product terms is chosen in a way that, if the adversary knows the first random element r_k (from the shared probes) of the intermediate sum $a_i b_j + r_k + a_j b_i + r_m$, then the input shares of index i are already known. Therefore, the adversary needs to probe a sum of $2\lfloor \frac{t}{2} \rfloor - 1$ cross-product terms to access $\lfloor \frac{t}{2} \rfloor$ unknown input shares. This process would require knowing $2\lfloor \frac{t}{2} \rfloor - 2$ random elements of an unprobed output share, which is not possible (for $t \geq 6$) given that the adversary has only $\lfloor \frac{t}{2} \rfloor$ shared probes (and each output share shares only one random element with every other output share).
The other cases of δ can be reduced to this one. Using the remaining probes for cross-product terms is not sufficient as they only leak about only share each. Therefore, it is necessary for an adversary to target the intermediate sums. For these sums there are two possible cases: (a) all the randoms are known from the shared probes (b) at least one random is not known. For (a), it is the most efficient strategy to target either sums with one (leaks about one unknown share) or two (leaks about two unknown shares) known randoms. This gives the best ratio from shared probes to newly probed shares, as the number of randoms for the intermediate sums grow with a factor of two ($2i - 2$ to probe i shares). In our considered cases $t \geq 6$, the number of new shares which can be probed with the strategy is always smaller than the required number. Therefore, this does not lead to a successful attack. In the case of (b), the adversary needs to either directly probe the unknown randomness or a sum which contains it. However, given that the random elements are only reused in sums with more than t other random element, probing such a sum would lead to more than one new unknown random element. Therefore, probing r_i directly would be the only solution for these sums. However, in this case it would be more beneficial to probe the complete output share and use the second probe in the parallel multiplication to probe all random element and cross-product terms at once. This strategy can, therefore, be reduced to the case of $\delta + 1$.

Algorithm 15 FindMapping

Input: number of probes t.
Output: $\lfloor \frac{t}{2} \rfloor$-non-complete mapping of input shares $\mathcal{M}$.

```
l = ⌊t/2⌋ + 1;
n = ⌊t/2⌋^2 + ⌊t/2⌋ + 1;
(M, row, col) = InitMapping(t, l, n)
O[n, l] = {0}
while min(Freq(M)) ≠ l do
    E = PossibleEntries(M, row)
    if |E| < O[row, col] + 1 then
        O[row, col] = 0
        (row, col) = PreviousEntry(row, col)
        O[row, col] + +
        M[row, col] = 0
    else
        M[row, col] = E[O[row, col] + 1]
        (row, col) = NextEntry(row, col)
    end if
end while
```

Algorithm 16 InitMapping

Input: t, l, n.
Output: $\mathcal{M}, row, col$.

```
M[n, l] = {0}
M[1, 1] = 1
for j = 1 to l do
    for i = 1 to l − 1 do
        M[(j − 1)(l − 1) + i + 1, 1] = j
        M[j, i + 1] = (j − 1)(l − 1) + i + 1
        if j == 2 then
            for k = 1 to l − 1 do
                M[j + k(l − 1) + i − 1, 2] = (j − 1)(l − 1) + i + 1
            end for
        end if
    end for
end for
row = l + 1
col = 3
```

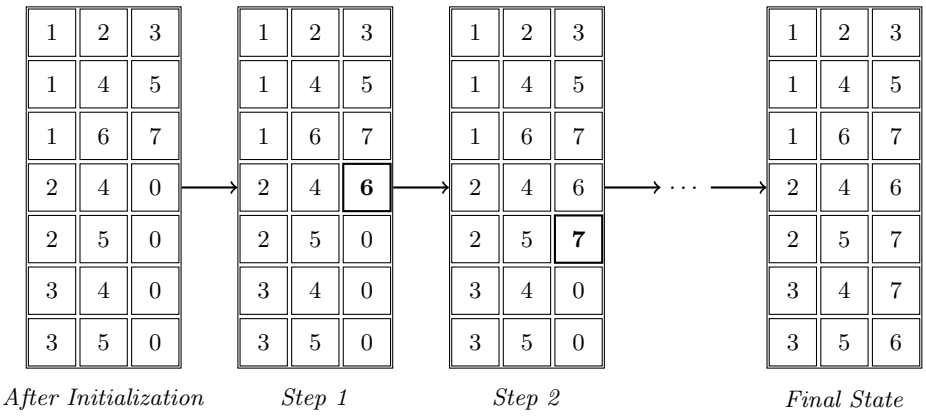

Fig. 8. The evolution of the mapping $\mathcal{M}$ for $t = 4$ and $n = 7$ from the initialization to the final $\lfloor \frac{t}{2} \rfloor$-non-complete distribution of the input shares.

A.2 t–SCR Multiplication for $t \geq 6$

Algorithm 17 Generic t–SCR multiplication algorithm for $t \geq 6$.

Input: shares $(a_i)_{1 \leq i \leq n}$ and $(b_i)_{1 \leq i \leq n}$, such that $\bigoplus_i a_i = a$ and $\bigoplus_i b_i = b$, $\mathcal{L}$, l.
Output: shares $(c_i)_{1 \leq i \leq n}$, such that $\bigoplus_i c_i = a \cdot b$.

$p = \frac{n \cdot (n-1)}{2}$
for $i = 1$ to n **do**
  $col = 0$
  for $j = 1$ to $l - 1$ **do**
    for $k = j + 1$ to l **do**
      if $(i + col \cdot n) > p$ **then**
        $offset = 0$
      else
        $offset = 1$
      end if
      $c_i = c_i + a_{\mathcal{L}(i,j)} \cdot b_{\mathcal{L}(i,k)} + r_{(((i + offset \cdot (col - (n-1)/2 + 1) - 1 \mod n) + col \cdot n \mod p) + 1)}$
      $col = col + 1$
      $c_i = c_i + a_{\mathcal{L}(i,k)} \cdot b_{\mathcal{L}(i,j)} + r_{(((i + offset \cdot (col - (n-1)/2 + 1) - 1 \mod n) + col \cdot n \mod p) + 1)}$
      $col = col + 1$
    end for
  end for
  $c_i = c_i + a_{\mathcal{L}(i,l+1)} \cdot b_{\mathcal{L}(i,l+1)}$
end for

Table 3. $\mathcal{L}$ for $t = 6/7$.

c_1	4	2	3	1	1
c_2	1	5	6	7	5
c_3	1	8	9	10	9
c_4	1	11	12	13	12
c_5	11	5	8	2	2
c_6	2	6	9	12	6
c_7	2	7	10	13	7
c_8	13	5	9	3	3
c_9	3	6	10	11	10
c_{10}	3	7	8	12	8
c_{11}	12	5	10	4	4
c_{12}	4	6	8	13	13
c_{13}	4	7	9	11	11

Table 4. $\mathcal{L}$ for $t = 8/9$.

c_1	5	2	3	4	1	1
c_2	1	6	7	8	9	6
c_3	1	10	11	12	13	11
c_4	1	14	15	16	17	17
c_5	1	18	19	20	21	20
c_6	18	6	10	14	2	2
c_7	2	7	11	15	19	7
c_8	2	8	12	16	20	8
c_9	2	9	13	17	21	9
c_{10}	21	6	11	16	3	3
c_{11}	3	7	10	17	20	10
c_{12}	3	8	13	14	19	13
c_{13}	3	9	12	15	18	12
c_{14}	12	6	4	17	19	4
c_{15}	4	7	13	16	18	16
c_{16}	4	8	10	15	21	15
c_{17}	4	9	11	14	20	14
c_{18}	20	6	13	15	5	5
c_{19}	5	7	12	14	21	21
c_{20}	5	8	11	17	18	18
c_{21}	5	9	10	16	19	19

Table 5. $\mathcal{L}$ for $t = 10/11$.

c_1	6	2	3	4	5	1	1
c_2	1	7	8	9	10	11	7
c_3	1	12	13	14	15	16	13
c_4	1	17	18	19	20	21	21
c_5	1	22	23	24	25	26	26
c_6	1	27	28	29	30	31	28
c_7	27	7	12	17	22	2	2
c_8	2	8	13	18	23	28	8
c_9	2	9	14	19	24	29	9
c_{10}	2	10	15	20	25	30	10
c_{11}	2	11	16	21	26	31	11
c_{12}	31	7	13	19	25	3	3
c_{13}	3	8	14	20	26	27	14
c_{14}	3	9	15	21	22	28	15
c_{15}	3	10	16	17	23	29	16
c_{16}	3	11	12	18	24	30	12
c_{17}	30	7	14	21	23	4	4
c_{18}	4	8	15	17	24	31	17
c_{19}	4	9	16	18	25	27	18
c_{20}	4	10	12	19	26	28	19
c_{21}	4	11	13	20	22	29	20
c_{22}	29	7	15	18	26	5	5
c_{23}	5	8	16	19	22	30	22
c_{24}	5	9	12	20	23	31	23
c_{25}	5	10	13	21	24	27	24
c_{26}	5	11	14	17	25	28	25
c_{27}	28	7	16	20	24	6	6
c_{28}	6	8	12	21	25	29	29
c_{29}	6	9	13	17	26	30	30
c_{30}	6	10	14	18	22	31	31
c_{31}	6	11	15	19	23	27	27

Proposition 3 (order $t \geq 6$). *Let $\mathsf{Mult}_1^t, \ldots, \mathsf{Mult}_N^t$ be a set of N multiplication schemes as in Algorithm 17, with inputs $(\boldsymbol{a}^{(1)}, \boldsymbol{b}^{(1)}), \ldots, (\boldsymbol{a}^{(N)}, \boldsymbol{b}^{(N)})$ and outputs $\boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(N)}$, where $t \geq 6$. Suppose that the maskings of the inputs are independent and uniformly chosen and that for $k = 1, \ldots, N$ each Mult_k^t uses the same random bits $\left(r_i\right)_{i=1,\ldots,tn/2}$. Then $\mathsf{Mult}_1^t, \ldots, \mathsf{Mult}_N^t$ are t–SCR and in particular Algorithm 17 is t–SNI.*

Proof. The procedure to prove the t–SCR is exactly the same to the one of the previous proofs, so we proceed directly to prove t–SNI. We now show the t–SNI property of Mult^t.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be a set of t observations respectively on the internal and on the output wires, where $|\mathcal{I}| = t_1$ and in particular $t_1 + |\mathcal{O}| \leq t$. We classify the internal wires in the following groups:

(1) $a_i, b_j, a_i b_j$
(2) r_k
(3) $a_i b_i + r_k := p$
(4) $p + a_u b_v := q$,
(5) $q + r_h =: s$,

Suppose an adversary corrupts at most t wires $w_1, \ldots, w_t$. We define two sets I, J with $|I| < t_1$ $|J| < t_1$ such that the values of the wires w_h can be perfectly simulated given the values $(a_i)_{i \in I}$, $(b_i)_{i \in J}$. The procedure to construct the sets is the following:

1. We first define a set K such that for all probes in group (2), (3), (4) or (5) we add the index k of each random bit r_k to K.
2. Initially I, J are empty and the w_i unassigned.
3. For every wire in group (1) or a combination of this, if $i \notin I$ add i to I and if $j \notin J$ add j to J.
4. For every wire in group (3) such that all the indices of the random bits appear in K at least two times, for every share a_i observed, if $i \notin I$, add i to I and for every share b_j observed, if $j \notin J$, add j to J.
5. For every wire in group (4), if $u \notin I$ add u to I and if $v \notin J$ add v to J.

We point out that the sets just constructed are such that $|I| \leq t_1$ and $|J| \leq t_1$. Indeed, for each probe in group (1) and (3) at most one index is added. For each probe in (4) and (5), we add more than one index to I and J only if the attacker's probes are of the form

$$p_1 := r_1, \ldots, p_h := r_i, p_{h+1} := a_i b_j + r_1 + \cdots + r_i \text{ or } p_{h+1} := a_i b_j + r_1 + \cdots + r_i + a_u b_v$$

with $h \leq t - 1$. In any case, it trivially always holds that $|I| \leq t_1$ and $|J| \leq t_1$. Now we simulate the wires w_h using only the values $(a_i)_{i \in I}$ and $(b_i)_{i \in J}$.

- We start the simulation with a preliminary phase in which for every $k \in K$ assign r_k to a random and independent value.
- For every probe in group (1), then $i \in I$ and $i \in J$ and so the values are perfectly simulated.
- For every probe in group (3), then, since $k \in K$ and r_k has been simulated in the preliminary phase as a random and independent value, we can compute w_h by using r_k and the indexes of the inputs which are, by construction, in the sets I and J;
- For every probe in group (4), then, if p has been probed, since $u \in I$ and $v \in J$, then it can be simulated by using p and the shares a_u and b_v; otherwise, if for every random bit r_k the index k appears at least two times in K, then we can compute w_h by using r_k and the indexes of the inputs which are, by construction, in the sets I and J; finally, if there exists at least one random bit r_k such that $k \in K$ only one time, then we can simulate w_h as a random and independent value.
- For every probe in group (5), then, if q has been probed, since r_h has been simulated in the preliminary phase, then it can be simulated by using q and r_k; otherwise, if for every random bit r_k the index k appears at least two times in K, then we can compute w_h by using r_k and the indexes of the inputs which are, by construction, in the sets I and J; finally, if there exists at least one random bit r_k such that $k \in K$ only one time, then we can simulate w_h as a random and independent value.

Finally, the output shares are as in group (4) and therefore the simulation of the probes on outputs are performed using at most t_1 shares of the inputs as well, completing the proof. □