# 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,tobias.schneider-a7a}@rub.de

**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 protect 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].

*Finding blocks of gadgets for re-using 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 randomness. We devise a simple tool that depending on the structure of the algorithm identifies blocks which can securely share randomness. Our tool follows a naive brute-force approach, and we leave it as an important question for future work to develop more efficient tools for identifying blocks of gadgets that are suitable for re-using randomness.

*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 (which would be the case for most inexpensive embedded devices), then our

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

new countermeasure would outperform state-of-the-art solutions even up to $t = 21$. We leave it as an important question for future research to design efficient masking schemes with shared randomness when $t > 21$.

### 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 $\rho$;
- *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 3, $\boldsymbol{r}_1$ is the vector $(r_{13}, r_1, r_8, r_7)$. 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* $\mathsf{g}$ *be a gadget with $m$ inputs $(a^{(1)}, \ldots, a^{(m)})$ each composed by $n$ shares and $\Omega$ 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* $\mathsf{g}$ *is called* $(\mathcal{S}, \Omega)$-simulatable *(or $(\mathcal{S}, \Omega)$–*$\mathsf{SIM}$*) 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* $\mathsf{g}$ *is called* $(\mathcal{S}, \Omega)$-Non Interfering *(or $(\mathcal{S}, \Omega)$–*$\mathsf{NI}$*) 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* $\mathsf{g}$ *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 $\Omega$ 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 $(\mathcal{S}, \Omega)$–$\mathsf{SIM}$.

**Lemma 1.** *For every gadget* $\mathsf{g}$ *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 $\Omega$, with $|\Omega| \leq t$,* $\mathsf{g}$ *is $(\mathcal{S}, \Omega)$–*$\mathsf{SIM}$ *if and only if* $\mathsf{g}$ *is $(\mathcal{S}, \Omega)$–*$\mathsf{NI}$*, with respect to the same sets $(\mathcal{S}, \Omega)$.*

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

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 *($t$–*$\mathsf{SNI}$*) 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**
    $r_{i,j} \xleftarrow{\$} \mathbb{F}_{2^n}$;
    $r_{j,i} \leftarrow (r_{i,j} + a_i \odot b_j) + a_j \cdot b_i$;
  **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 $\mathsf{g}$ 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 $\mathsf{g}_1, \ldots, \mathsf{g}_d$ having independent inputs, 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*

1. *their inputs are mutually independent;*
2. *for each set* $\mathcal{P}_i$ *of* $t_i$ *probes on* $\mathsf{g}_i$ *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* $\mathsf{g}_i$ *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 $\mathsf{g}_i$ 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 $\mathsf{g}_1$. 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 $\mathsf{g}_i$ 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).** *For every* $d \in \mathbb{N}$*, consider* $\mathcal{G}_1(\boldsymbol{R}) = \{\mathsf{g}_{1,1}(\boldsymbol{r_1}), \ldots, \mathsf{g}_{1,d}(\boldsymbol{r_d})\}, \ldots, \mathcal{G}_N(\boldsymbol{R}) = \{\mathsf{g}_{N,1}(\boldsymbol{r_1}), \ldots, \mathsf{g}_{N,d}(\boldsymbol{r_d})\}$ $N$ *blocks of gadgets sharing the same random component* $\boldsymbol{R} = (\boldsymbol{r_1}, \ldots, \boldsymbol{r_d})$ *and masking their input into* $n$ *shares. Suppose* $\mathcal{G}_i$ *be* $t$–NI *for each* $i = 1, \ldots, N$*. 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.* First it is easy to see that, since $\mathsf{g}_{1,1}, \ldots, g_{N,1}$ are $t$–SCR then their inputs have independent masks and so the same holds for the inputs of blocks $\mathcal{G}_1, \ldots, \mathcal{G}_N$. Let us next discuss the second property given in the $t$–SCR definition. 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.

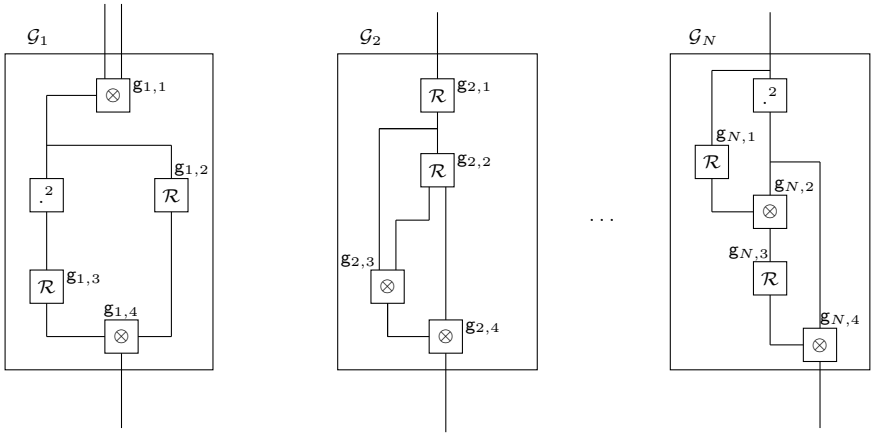


**Fig. 1.** A set of $N$ blocks of gadgets with dimension $d = 4$ each.

- 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.
  - The probes are placed on $\{\mathsf{g}_{1,d}, \ldots, g_{N,d}\}$: in this case, since by hypothesis $\{\mathsf{g}_{1,d}, \ldots, g_{N,d}\}$ 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 $\{\mathsf{g}_{1,d}, \ldots, g_{N,d}\}$ and a set of probes $\mathcal{Q}$ is placed on $\{\{\mathsf{g}_{1,1}, \ldots, \mathsf{g}_{1,d-1}\}, \ldots, \{\mathsf{g}_{N,1}, \ldots, \mathsf{g}_{N,d-1}\}\}$: 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 $\{\mathsf{g}_{1,d}, \ldots, g_{N,d}\}$ and the simulatability of the probes in $\mathcal{Q}$ is guaranteed by the $t$–SCR of $\{\{\mathsf{g}_{1,1}, \ldots, \mathsf{g}_{1,d-1}\}, \ldots, \{\mathsf{g}_{N,1}, \ldots, \mathsf{g}_{N,d-1}\}\}$.

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 for guaranteeing also a sound composition. 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 $\mathcal{G}_i(\boldsymbol{R}) = \{\mathsf{g}_{i,1}(\boldsymbol{r_1}), \ldots, \mathsf{g}_{i,d}(\boldsymbol{r_d})\}$ for each $i = 1, \ldots, N$ and with inputs masked with $n$ shares and such that the gadgets outside such blocks are either linear or $t$–SNI ones. If*

– *the outputs of* $\mathcal{G}_1, \ldots, \mathcal{G}_N$ *are independent*
– $\forall j = 1, \ldots, N$ $\mathcal{G}_j$ *is* $t$–SNI *and*
– $\forall j = 1, \ldots, d$ $\mathsf{g}_{1j}, \ldots, g_{Nj}$ *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 in this section 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 in an appropriate way, 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, when designing such circuits, even if on the one hand the randomness involved in the gadgets can be completely reused, we require on the other hand additional refreshing schemes to guarantee the independence of the inputs and outputs of each block. Notice that independence is needed for ensuring $t$–SCR and, as recalled in Section 2.1, it is satisfied by refreshing via Algorithm 2.

For these reasons, in order to have an actual reduction in the amount of randomness, it is needed to take a couple of precautions when structuring a circuit into blocks of gadgets. First of all, it is necessary to construct these blocks such that the number of the outputs which are inputs of other blocks do not exceed the number of gadgets in the block; otherwise we would require more randomness for refreshing than what was saved by the reusing of randomness within the block. In addition, it is important to find a good trade-off between the dimension of the blocks and the number of them in the circuit.

More formally speaking if $N$ is the number of randomized gadgets of the original circuit, $N_{\mathrm{C}}$ is the number of gadgets which use the same random bits in the restructured circuit and $N_{\mathrm{R}}$ is the number of new refreshing schemes that we need to add to it for guaranteeing the independence of the inputs of the blocks, then the total saving in the randomness of the circuit is given by the difference $N-(N_{\mathrm{C}}+N_{\mathrm{R}})$. To illustrate how this quantity changes according to the different dimension of the blocks let us take a look at some concrete cases. Suppose for simplicity that each block of gadget has only one input and one output. If we divide the circuit into many small blocks, then on the one hand we reuse a small amount of randomness, and so $N_{\mathrm{C}}$ is smaller, on the other hand, since at every block corresponds one output which needs to be refreshed before being input of another block, the number of new randomness involved increases, and then

$N_{\mathrm{R}}$ is bigger. Otherwise, if the circuit is designed in few large blocks of gadgets, then since we have fewer blocks, there are also fewer outputs to be refreshed, therefore the amount of fresh randomness $N_{\mathrm{R}}$ is reduced. On the other hand, more random bits are needed for refreshing for the common randomness in the blocks, and so the amount $N_{\mathrm{C}}$ increases. A more concrete example can be found in Figures 2, 3 and 4, where the same circuit is structured in blocks of gadgets in two different ways .

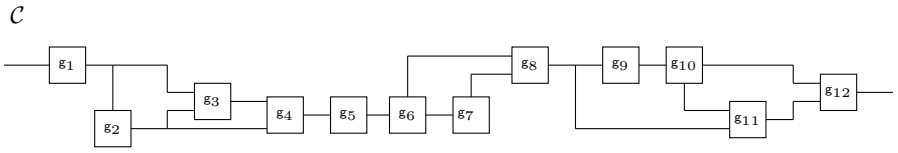


**Fig. 2.** The original circuit $\mathcal{C}$ composed by $N = 12$ randomized gadgets.

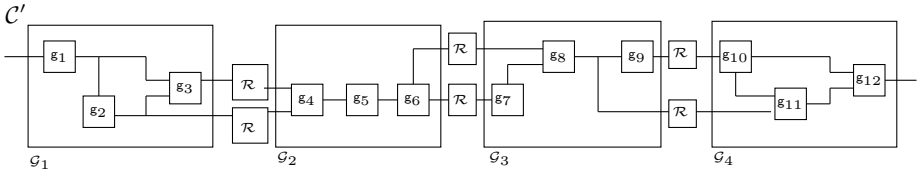


**Fig. 3.** The circuit $\mathcal{C}'$ representing $\mathcal{C}$ structured into 4 blocks of gadgets, where $N = 12, N_{\mathrm{C}} = 3, N_R = 6$ and the saving consists of 3 randomized gadgets.

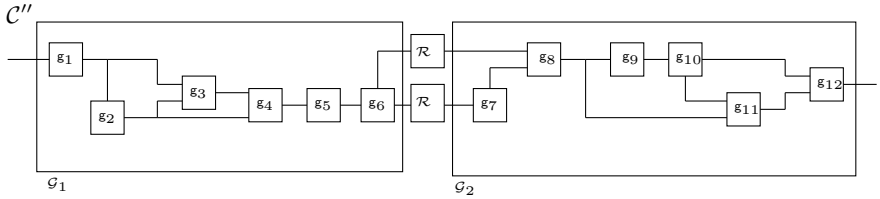


**Fig. 4.** The circuit $\mathcal{C}''$ representing $\mathcal{C}$ structured into 2 blocks of gadgets, where $N = 12, N_{\mathrm{C}} = 6, N_R = 2$ and the saving consists of 4 randomized gadgets.

In Section 4, we will present a naive method to restructure a circuit in such a way that these conditions are satisfied and in order to find an efficient grouping in blocks of gadgets.

### 3.1 A $t$-SCR Multiplication Scheme

In this subsection, we introduce a multiplication scheme, which can be combined with other gadgets sharing the same randomness and remains $t$–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 $\mathsf{Mult}$.

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, $\frac{tn}{2}$ random components $r_i$ need to be added for the multiplication to be $t$–SNI. A subset of $t$ 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. Furthermore, we include additional random elements $rx_{j=1,\dots,n}$ which only occur in one output share each and enable a simple simulation of the gadget even in the presence of shared 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 fulfils $\lfloor \frac{t}{2} \rfloor$-non-completeness becomes a complex task due to the large number of possible combinations for increasing $t$. For $t = 4$, one possible $t$–SCR multiplication is defined in Algorithm 3 and it requires $n = 7$ shares. A complete description of a multiplication algorithm for higher orders fulfilling the properties aforementioned can be found in the full version of the paper.

**Algorithm 3** Mult for order $t = 4$ with $n = 7$ shares.

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

$c_1 = ((((((((((rx_1 + a_1b_1) + r_{13}) + a_1b_2) + a_2b_1) + r_1) + a_1b_3) + a_3b_1) + r_8) + a_2b_3) + a_3b_2) + r_7) + rx_1)$;
$c_2 = ((((((((((rx_2 + a_4b_4) + r_{14}) + a_1b_4) + a_4b_1) + r_2) + a_1b_5) + a_5b_1) + r_9) + a_4b_5) + a_5b_4) + r_1) + rx_2)$;
$c_3 = ((((((((((rx_3 + a_7b_7) + r_8) + a_1b_6) + a_6b_1) + r_3) + a_1b_7) + a_7b_1) + r_{10}) + a_6b_7) + a_7b_6) + r_2) + rx_3)$;
$c_4 = ((((((((((rx_4 + a_2b_2) + r_9) + a_2b_4) + a_4b_2) + r_4) + a_2b_6) + a_6b_2) + r_{11}) + a_4b_6) + a_6b_4) + r_3) + rx_4)$;
$c_5 = ((((((((((rx_5 + a_5b_5) + r_{10}) + a_2b_5) + a_5b_2) + r_5) + a_2b_7) + a_7b_2) + r_{12}) + a_5b_7) + a_7b_5) + r_4) + rx_5)$;
$c_6 = ((((((((((rx_6 + a_3b_3) + r_{11}) + a_3b_4) + a_4b_3) + r_6) + a_3b_7) + a_7b_3) + r_{13}) + a_4b_7) + a_7b_4) + r_5) + rx_6)$;
$c_7 = ((((((((((rx_7 + a_6b_6) + r_{12}) + a_3b_5) + a_5b_3) + r_7) + a_3b_6) + a_6b_3) + r_{14}) + a_5b_6) + a_6b_5) + r_6) + rx_7)$;

Now we present the security analysis of this multiplication scheme and we show that it can be securely composed with the refreshing scheme in Algorithm 2 in blocks of gadgets sharing the same random component. Due to size constraints, we only give a sketch of the proof and refer to the full version of the paper for the complete proof.

**Lemma 4.** *Let* $\mathsf{Mult}_1, \dots, \mathsf{Mult}_N$ *be a set of* $N$ *multiplication schemes as in Algorithm 3, with outputs* $\boldsymbol{c}^{(1)}, \dots, \boldsymbol{c}^{(N)}$*. Suppose that the maskings of the inputs*

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

*Proof.* In the first case, all probes are placed in the same Mult and it is sufficient to show $t$–SNI of Mult. We indicate with $p_{l,m}$ the $m$-th sum of the output $c_l$. We can classify the probes in the following groups.

(1) $a_i b_j + r_k =: p_{l,1}$
(2) $a_i, b_j, a_i b_j$
(3) $r_k$
(4) $p_{l,m} + a_i b_j =: q$
(5) $p_{l,m} + r_k =: s$
(6) output shares $c_i$

Suppose an adversary corrupts at most $t$ wires $w_1, \ldots, w_t$. We define two sets $I, J$ with $|I| < n$ $|J| < n$ 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 the probes containing a random bit $r_k$, we add $k$ to $K$.
2. Initially $I, J$ are empty and the $w_i$ unassigned.
3. For every wire in the group (1), (2), (4) and (5) add $i$ to $I$ and $j$ to $J$.

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

- For every probe in group (2), then $i \in I$ and $i \in J$ and the values are perfectly simulated.
- For every probe in group (3), $r_k$ can be simulated as a random and independent value.
- For every probe in group (1), if $k \notin K$, we can assign a random independent value to the probe, otherwise, if $r_k$ has already been simulated we can simulate the probe by taking the $r_k$ previously simulated, simulating the shares of $a$ and $b$ by using the needed indices in $I$ and $J$ and performing the inner products and additions as in the real execution of the algorithm.
- For every probe in group (4) if $p_{l,m}$ was already probed, we can compute $q$ by using $p_{l,m}$ and the needed indices of $a$ and $b$ in $I$ and $J$. Otherwise, we can pick $q$ as a uniform and random value.
- For every probe in group (5), if $p_{l,m}$ was already probed and $k \in K$, we can compute $s$ by using $p_{l,m}$ and the already simulated $r_k$. Otherwise, we can pick $s$ as a uniform and random value.

Finally, we simulate the output wires $\boldsymbol{c}_i$ in group (6) using only a number of input shares smaller or equal to the number of internal probes. We have to take into account two cases.

- If the attacker has already observed a partial value of the output shares, we note that by construction, independently of the intermediate elements probed, at least one of the $r_k$ does not enter into the computation of the probed internal values and so $\boldsymbol{c}_i$ can be simulated as a random value.

– If the adversary has observed all the partial sums of $c_i$, then, since these probes have been previously simulated, the simulator now add these simulated values for reconstructing the $c_i$.
– If no partial value fo $c_i$ has been probed. By definition, at least one of the $r_k$ involved in the computation of $\boldsymbol{c}_i$ is not used in any other observed wire. Therefore, $\boldsymbol{c}_i$ can be assigned to a random and independent value.

In the second case, the probes are placed into different $\mathsf{Mult}_i$. However, the number of probes in one particular gadget does not exceed $\lfloor \frac{t}{2} \rfloor$. In this case, security is given by the $\lfloor \frac{t}{2} \rfloor$-non-completeness property of our multiplication schemes.

In the third case, the number of probes for one $\mathsf{Mult}_i$ does exceed $\lfloor \frac{t}{2} \rfloor$. For this, we base our proof strategy on the two observations. First, since all $\mathsf{Mult}_i$ share the same randomness, it is possible to probe the same final output share $c_i$ in two gadgets to remove all random elements and get information about all the input shares used in the computation of $c_i$. Secondly, a probe in any intermediate sum of $c_i$ is randomized by $rx_i$. Therefore, this probe can always be simulated as uniform random if not another probe is placed on $rx_i$ or on a different intermediate sum of $c_i$ (including in a different $\mathsf{Mult}_j$). Therefore, any probe of an intermediate sum of $c_i$ can be reduced to a probe of the final output share $c_i$, since in the latter case one receives information about more or an equal number of input shares with the same number of probes (i.e., two). Therefore, to get information about the maximum number of input shares the probes need to be placed in the same $\lfloor \frac{t}{2} \rfloor$ output shares in two multiplications. Based on the $\lfloor \frac{t}{2} \rfloor$-non-completeness, this can be easily simulated. The remaining probe, given that $t$ is odd, can be simulated as uniform random, since it is either

– an intermediate sum of an unprobed output share $c_k$. This can be simulated as uniform random due to the unprobed $rx_k$.
– an unprobed output share $c_k$. This can be also simulated as uniform random, as by construction there is always at least one random element $r_i$ which is not present in one of the $\lfloor \frac{t}{2} \rfloor$ probed output shares.

For the special case of $t < 4$, it is possible to avoid the extra $rx_i$ per output share. This is based on the limited number of probes. For $t = 2$, 1-non-completeness (for the case of one probe in two multiplications) and $t$–$\mathsf{SNI}$ (for the case of two probes in one multiplication) are sufficient to enable $t$–$\mathsf{SCR}$. The same applies to $t = 3$ as for the last probe there is always one unknown random $r_i$ masking any required intermediate sum. □

In the following lemma we show that the $t$–$\mathsf{SNI}$ refreshing scheme in Algorithm 2 is also $t$–$\mathsf{SCR}$. Due to size constraints, we again only provide a proof sketch and refer to the full version of the paper for the complete proof.

**Lemma 5.** *Let $\mathcal{R}_1, \ldots, \mathcal{R}_N$ be a set of $N$ refreshing schemes, as in Algorithm 2, with inputs $\boldsymbol{a}^{(1)} \ldots, \boldsymbol{a}^{(N)}$ and outputs $\boldsymbol{c}^{(1)} \ldots, \boldsymbol{c}^{(N)}$. Suppose that $\left(a_i^{(1)}\right)_{i=1,\ldots,n}, \ldots, \left(a_i^{(N)}\right)_{i=1,\ldots,n}$ are independent and randomly chosen maskings*

*of the input values and for $k = 1, \ldots, N$ each $\mathcal{R}_k$ uses the same random bits $\left(r_{i,j}\right)_{i,j=1,\ldots,n}$. Then $\mathcal{R}_1, \ldots, \mathcal{R}_N$ are $t$–SCR.*

*Proof.* Since according to Algorithm 2 every output share contains only one single share of the input and since the inputs are encoded in $n > t$ shares, it is not possible to probe all of the input shares of one $\mathcal{R}_i$ with $t$ probes. Therefore, the simulation can be done easily. □

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, since it requires $\frac{n^2}{2}$ random bits. We depict in Algorithm 4 a more efficient refreshing scheme which uses only $\frac{t \cdot n}{2}$ random bits. It essentially consist in multiplying the input value times 1, by means of Algorithm 3 as subroutine. It is easy to see that the security of the scheme relies

**Algorithm 4** Refreshing scheme with optimal amount of randomness

**Input:** shares $a_1, \ldots, a_n$ such that $\bigoplus a_i = a$
**Output:** shares $c_1, \ldots, c_n$ such that $\bigoplus c_i = a$

```
for i = 1 to n do
    u_i = 1;
end for
if n is even then
    u_n = 0;
end if
(c_1, ..., c_n) = Mult(a, (u_1, ..., u_n));
```

on the security of the multiplication algorithm Mult, and therefore Algorithm 4 is $t$–SNI and it can securely share randomness with other multiplication gadgets.

An example of blocks of gadgets using multiplication and refreshing schemes is given in Figure 5, where are depicted two blocks of gadgets of dimension 6 involving the multiplication scheme Mult and the refreshing $\mathcal{R}$ of Algorithm 2 secure even if sharing the same randomness.

## 4 A tool for general circuits

The results from the previous sections essentially show that it is possible to transform a circuit $\mathcal{C}$ in another circuit $\mathcal{C}'$ performing the same operation, but using a reduced amount of randomness. To this end, according to Theorem 1, it is sufficient to group the gadgets composing the circuit $\mathcal{C}$ in blocks $\mathcal{G}_i$ sharing the same component of random bits and having independent inputs, i.e. values refreshed by Algorithm 2. As pointed out in Section 3, the actual efficiency of this procedure is not straightforward, but it is given by the right trade off between the dimension of the blocks and the number of extra refreshing schemes needed in order to guarantee the independence of their inputs.

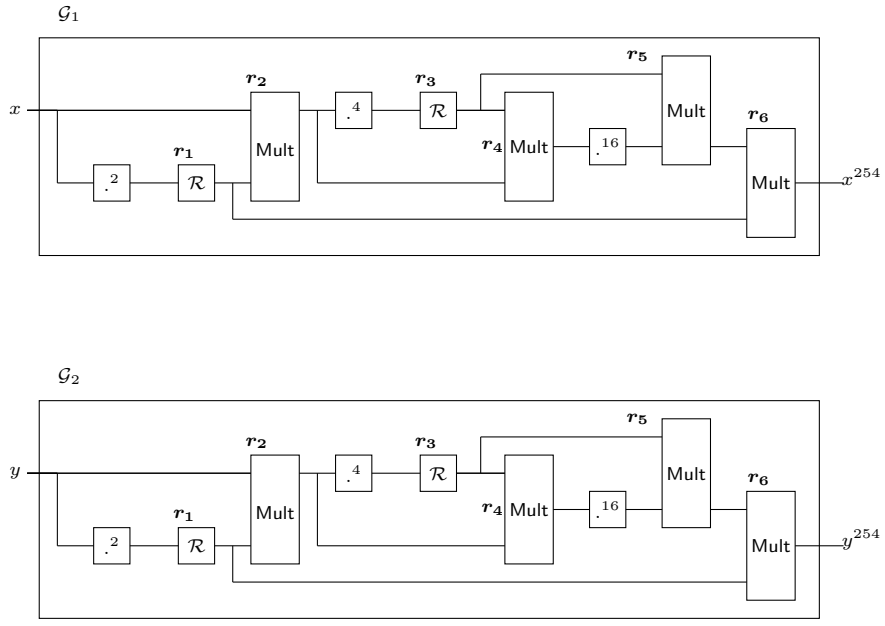


**Fig. 5.** Two blocks of gadgets $\mathcal{G}_1, \mathcal{G}_2$ composed by the same gadgets using the random components $\boldsymbol{r}_i$ with independent inputs $x$ and $y$

In the following we give a tool, depicted in Algorithm 7, which allows to perform this partitioning and amortize the randomness complexity of a given circuit.

A circuit $\mathcal{C}$ is represented as a directed graph where the *nodes* constitute the randomized gadgets and the *edges* are input or output wires of the related gadget, according to the respective direction. In particular, if the same output wire is used as input several time in different gates, it is represented with a number of edges equivalent to the number of times it is used. The linear gates are not represented. The last node is assigned to the label "End" and every intersection node with parallel branches is marked as "Stop".

The idea at the basis of our algorithm is quite primitive. We empirically noticed that for a circuit composed by $N$ randomized gadgets a balanced choice for the dimension of the blocks of gadgets can be the *central divisors* ($d_1$ and $d_2$ in the algorithms) of $N$, where if for instance $N = 12$ and then the vector of its divisors is $(1, 2, 3, 4, 6, 12)$, with *central divisors* we identify the values 3 and 4. Therefore, we aim at dividing the circuit in $d_1$ blocks of gadgets of dimension $d_2$ (and vice versa). We start taking the fist $d_1$ nodes and we verify that the number of outputs do not exceed the one of randomized gadgets in the block. Indeed, if it would be so, since each output needs to be refreshed before being input of another block, then the number of reused random bits is inferior to the one of new random bits which need to be refreshed. In case the condition is not verified

the algorithm adds a new node, i.e. a new randomized gadget, to the block and check again the property, until it is verified. Then it takes again the next $d_1$ nodes and repeats the procedure. At last, we compare the saved randomness respectively when the algorithm tries to divide the circuit in $d_1$ blocks and in $d_2$ blocks and we output the transformed circuit with the best amortizing rate.

More technically, at first we give the subroutine in Algorithm 5, which chooses two divisors of a given integer. With $\boldsymbol{V}$ we indicate the vector composed by all the divisors of a given number $N$ (which in the partitioning algorithm will be the number of the randomized gadgets of a circuit) and with $|\boldsymbol{V}|$ the length of $\boldsymbol{V}$, i.e. the number of its elements.

**Algorithm 5** Divisors

**Input:** positive integer $N$
**Output:** divisors $d_1$ and $d_2$
$\boldsymbol{V} \leftarrow$ divisors of $N$ (by look up table);
$n \leftarrow |\boldsymbol{V}|$ ;
**if** $n$ is even **then**
  $i \leftarrow \frac{n}{2}$;
**else**
  $i \leftarrow \frac{n-1}{2}$;
**end if**
$d_1 \leftarrow \boldsymbol{V}[i]$;
$d_2 \leftarrow \boldsymbol{V}[i+1]$;
**return** $d_1, d_2$

Algorithm 6 constructs a block of gadgets $\mathcal{G}$ of dimension at least $d$, such that the number of extra refreshing needed does not exceed the number of randomized gadgets in the block. In the algorithm, the integers $m_o^{(j)}$ and $m_g^{(j)}$ represent respectively the number of output edges and the amount of nodes contained in the block of gates $\mathcal{G}_j$.

The procedure Partition in Algorithm 7 partitions a circuit $\mathcal{C}$ in sub-circuits $\mathcal{G}_i$ followed by a refreshing gate $\mathcal{R}$ per each output edge. In the algorithm, $\boldsymbol{O}$ and $\boldsymbol{M}$ are two vectors such that the $j$-th position represents respectively the number of output wires and the amount of nodes of the block $\mathcal{G}_j$. With $\mathcal{R}$ it is indicated the refreshing scheme of Algorithm 2. The integers $n_R$ and $n'_R$ count the total number of refreshing gadgets needed in the first and second partition of $\mathcal{C}$ respectively. The integers $n_G$ and $n'_G$ count the total number of gadgets (multiplications and refreshing) which need to refresh the random bits once in the circuit. The integers $n_{TOT}$ and $n'_{TOT}$ represent the total amount of randomness needed, computed as the number of gadgets which need fresh random bits once. By comparing these two values, the algorithm decides which is the best partition in terms of amortized randomness. In particular the notation $\boldsymbol{O}[i] \cdot \mathcal{R}$ means that the block $\mathcal{G}_i$ is followed by $\boldsymbol{O}[i]$ refreshing schemes (one per output edge).

**Algorithm 6** FindBlock

**Input:** circuit $\mathcal{C}$ performing a function $f(x)$, first node $v_j$ of the block, $d$
**Output:** block of gates $\mathcal{G}_i$, node $v$, integers $m_o^{(j)}, i$

$\mathcal{G}_j \leftarrow \{v_{j+1}, v_{j+2}, \ldots, v_{j+d}\}$;
$i \leftarrow 0$;
**while** $m_o^{(j)} \geq m_g^{(j)}$ **do**
  $i \leftarrow i + 1$;
  **if** $v_{j+d+i} \neq$ "Stop" **then**
    $\mathcal{G}_j \leftarrow \mathcal{G}_j \cup \{v_{j+d+i}\}$;
  **else**
    **end while**
    **return** $\mathcal{G}_j$, $v_{j+d+i}, m_o^{(j)}, i-1$;
  **end if**
**end while**
**return** $\mathcal{G}_j$, $v_{j+d+i+1}, m_o^{(j)}, i$;

**Algorithm 7** Partition

**Input:** circuit $\mathcal{C}$ performing a function $f(x)$, $N$ total number of randomized gadgets
**Output:** circuit $\mathcal{C}'$ performing a function $f(x)$ with a reduced amount of randomness

$d_1, d_2 \leftarrow \mathsf{Divisors}(N)$;
$i \leftarrow 1$;
$\mathcal{G}_1, v, \boldsymbol{O}[1], \boldsymbol{M}[1] \leftarrow \mathsf{FindBlock}(\mathcal{C}, v_1, d_1)$;
**while** $v \neq$ "End" **do**
  $i \leftarrow i + 1$;
  $\mathcal{G}_i, v, \boldsymbol{O}[i], \boldsymbol{M}[i] \leftarrow \mathsf{FindBlock}(\mathcal{C}, v, d_1)$
**end while**
$n_R = \boldsymbol{O}[1] + \cdots + \boldsymbol{O}[i]$;
$n_G = \max(\boldsymbol{M}[1], \ldots, \boldsymbol{M}[i])$;
$n_{TOT} = n_R + n_G$;
$k \leftarrow 1$;
$\mathcal{G}'_1, v', \boldsymbol{O}'[1], \boldsymbol{M}'[1] \leftarrow \mathsf{FindBlock}(\mathcal{C}, v_1, d_2)$;
**while** $v \neq$ "End" **do**
  $k \leftarrow k + 1$;
  $\mathcal{G}'_i, v', \boldsymbol{O}'[k], \boldsymbol{M}'[k] \leftarrow \mathsf{FindBlock}(\mathcal{C}, v', d_2)$;
**end while**
$n'_R = \boldsymbol{O}'[1] + \cdots + \boldsymbol{O}'[k]$;
$n'_G = \max(\boldsymbol{M}'[1], \ldots, \boldsymbol{M}'[k])$;
$n'_{TOT} = n'_R + n'_G$;
**if** $n_{TOT} \leq n'_{TOT}$ **then**
  $\mathcal{C}' \leftarrow (\mathcal{G}_1, \boldsymbol{O}[1] \cdot \mathcal{R}, \ldots, \mathcal{G}_i, \boldsymbol{O}[i] \cdot \mathcal{R})$;
**else**
  $\mathcal{C}' \leftarrow (\mathcal{G}'_1, \boldsymbol{O}'[1] \cdot \mathcal{R}, \ldots, \mathcal{G}'_i, \boldsymbol{O}'[k] \cdot \mathcal{R})$;
**end if**
**return** $\mathcal{C}'$

We conclude this section by emphasizing that our algorithm is not designed to provide the optimal solution (as in finding the grouping which requires the least amount of randomness). Nevertheless, it can help to decompose an arbitrary circuit without a regular structure and serve as a starting point for further optimizations. However, for circuits with an obvious structure (e.g., layers for symmetric ciphers) which contain easily-exploitable regularities to group the gadgets, the optimal solutions can be usually found by hand.

## 5 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 8 and Algorithm 9. Notice that when parts of the operations are written in parenthesizes, then this means that these operations are executed first.

**Algorithm 8** 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 9** 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. $b = (c_1 d_1 + c_1 d_2 + r_1, c_2 d_1 + c_2 d_2 - r_1)$;
(iv) the first input is in category (3), i.e. $a = (c_1 d_1 + c_1 d_2 + r_0, c_2 d_1 + c_2 d_1 - r_0)$, 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. $a = (c_1 d_1 + c_1 d_2 + r_1, c_2 d_1 + c_2 d_2 - r_1)$ and $b = (c'_1 d'_1 + c'_1 d'_2 + r_0, c'_2 d'_1 + c'_2 d'_2 - r_0)$.

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 10 and 11.

**Algorithm 10** 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 $r_{1-i}$

$c_1 \leftarrow a_1 b_1 + (a_1 b_2 + r_{1-i})$;
$c_2 \leftarrow a_2 b_1 + (a_2 b_2 - r_{1-i})$;

**Algorithm 11** 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 $r_{1-i}$

$\delta \leftarrow -r_{1-i}$;
$\delta \leftarrow \delta + r_i b_1$;
$\delta \leftarrow \delta + r_i b_2$;
$c_1 \leftarrow a_1 b_1 + (a_1 b_2 - \delta)$;
$c_2 \leftarrow a_2 b_1 + (a_2 b_2 + \delta)$;

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

**Lemma 6.** *Let $a$ and $b$ be two input values of Algorithm 10 or of Algorithm 11. 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 12 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 12 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 12** 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 7.** *Let $a$ be an input value of the form* (3) *depending on a random bit $r_i \in R$ for Algorithm 12. 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 $a = (c_1 d_1 + c_1 d_2 + r_0, c_2 d_1 + c_2 d_1 - r_0)$. Then the output $e = \mathcal{R}'(a)$ is:

$$e_1 = (c_1 d_1 + c_1 d_2 + r_0 + r_1) - r_0 = c_1 d_1 + c_1 d_2 + r_1$$
$$e_2 = (c_2 d_1 + c_2 d_1 - r_0 - r_1) + r_0 = c_2 d_1 + c_2 d_1 - r_1$$

completing the proof. □

Lastly, in Algorithm 13 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

**Algorithm 13** 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 $r_k$

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

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 8, 10 and 11 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.

### 5.1 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 6. 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.

## 6 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 a block of gadgets $\mathcal{G}_{i=1,\ldots,200}$ using the same random components and each

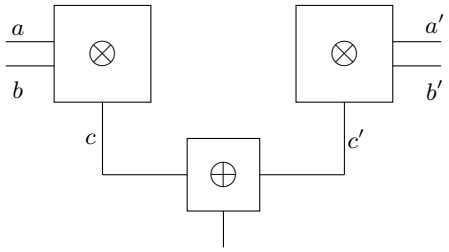


**Fig. 6.**

of these inversions is followed by a refresh $\mathcal{R}_{i=1,\ldots,200}$. For the implementation 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 4 for $t > 3$, and increase the number of shares accordingly. 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 | 21 | 21 |
| 5 | 6 | 15 | 15 | 7 | 25 | 25 |
| 6 | 7 | 21 | 21 | 13 | 52 | 52 |
| 7 | 8 | 28 | 28 | 13 | 59 | 59 |
| 8 | 9 | 36 | 36 | 21 | 105 | 105 |
| 9 | 10 | 45 | 45 | 21 | 116 | 116 |
| 10 | 11 | 55 | 55 | 31 | 186 | 186 |
| 11 | 12 | 66 | 66 | 31 | 202 | 202 |

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 all considered $t$. Only after $t \geq 22$, the randomness complexity of the additional refreshes $\mathcal{R}_{i=1,\ldots,200}$ 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 $\texttt{TRNG}_8$[2]. 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 $t = 23$ for our scenario, i.e., until the implementation with common randomness requires more TRNG calls.

For the special case of $t = 1$, we presented a solution (cf. Section 5) 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 more cycles than the one with common randomness, mostly due to additional operations in the multiplication, addition, and refresh algorithms. This is especially apparent for the key addition layer which is 40% slower. In general, however, the approach with constant randomness could lead to better performances for implementations with many TRNG calls and a slower source of randomness.

## 7 Conclusion

Since the number of shares $n$ for our $t$–SCR multiplication grows in $\mathcal{O}(t^2)$ and $\mathcal{R}$ requires $\mathcal{O}(nt)$ random elements, the practicability our proposed method-

[2] For $t = 1$, our implementation with common randomness is faster for $\texttt{TRNG}_8$ than for $\texttt{TRNG}_{32}$. This is due to the small number of TRNG calls and the extra logic required to access the randomness buffer of $\texttt{TRNG}_{32}$.

**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** | $\texttt{TRNG}_{32}$ | $\texttt{TRNG}_{8}$ |
| 1 | 2 | 1,200 | 112,919 | 187,519 | 2 | 206 | 70,262 | 70,196 |
| 2 | 3 | 3,600 | 308,600 | 548,477 | 3 | 618 | 173,490 | 199,063 |
| 3 | 4 | 7,200 | 496,698 | 1,089,092 | 4 | 1,236 | 309,844 | 412,887 |
| 4 | 5 | 12,000 | 751,670 | 1,812,213 | 7 | 4,326 | 737,260 | 1,206,558 |
| 5 | 6 | 18,000 | 1,051,323 | 2,729,052 | 7 | 5,150 | 808,412 | 1,358,560 |
| 6 | 7 | 25,200 | 1,403,243 | 3,836,006 | 13 | 10,712 | 1,973,885 | 3,134,628 |
| 7 | 8 | 33,600 | 1,779,403 | 5,125,072 | 13 | 12,154 | 2,147,190 | 3,467,553 |
| 8 | 9 | 43,200 | 2,286,003 | 6,603,199 | 21 | 21,630 | 4,647,611 | 7,017,148 |
| 9 | 10 | 54,000 | 2,814,435 | 8,257,996 | 21 | 23,896 | 4,877,985 | 7,498,022 |
| 10 | 11 | 66,000 | 3,459,684 | 10,096,735 | 31 | 38,316 | 8,282,630 | 12,467,274 |
| 11 | 12 | 79,200 | 4,046,836 | 12,112,375 | 31 | 41,612 | 8,640,018 | 13,211,240 |

ology 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 $\mathcal{R}$ with a smaller randomness complexity, e.g., $\mathcal{O}(t^2)$, could lead to better performances even for $t \geq 22$ 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 the automatic application of our methodology to an arbitrary circuit. While we provide a basic heuristic approach in Section 4, 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 Scheider is partially funded by European Unions Horizon 2020 program under project number 645622 PQCRYPTO. This work is also partially supported by the VeriSec project 16KIS0634 - 16KIS0602 from the Federal Ministry of Education and Research (BMBF).

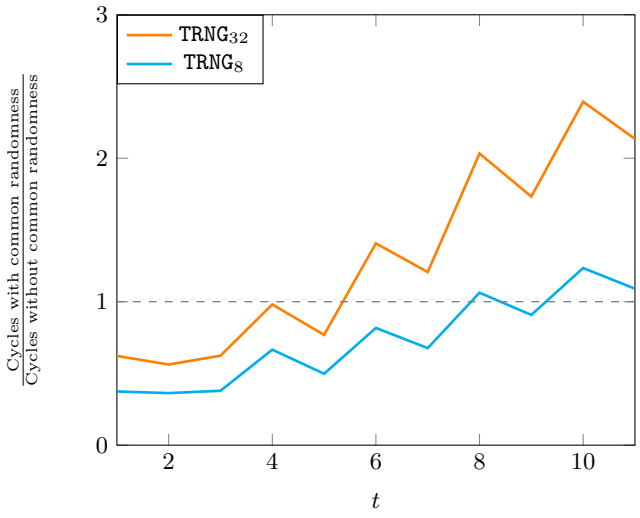


**Fig. 7.** Ratio between the cycle counts of the AES implementations from Table 2 with and without common randomness for each $t$.

## 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.