Random Probing Security: Verification, Composition, Expansion and New Constructions

Sonia Belaïd[1(✉)], Jean-Sébastien Coron[2(✉)], Emmanuel Prouff[3(✉)], Matthieu Rivain[1(✉)], and Abdul Rahman Taleb[1(✉)]

[1] CryptoExperts, Paris, France
{sonia.belaid,matthieu.rivain,abdul.taleb}@cryptoexperts.com
[2] University of Luxembourg, Luxembourg, Luxembourg
jean-sebastien.coron@uni.lu
[3] ANSSI, Paris, France
emmanuel.prouff@ssi.gouv.fr

Abstract. The masking countermeasure is among the most powerful countermeasures to counteract side-channel attacks. Leakage models have been exhibited to theoretically reason on the security of such masked implementations. So far, the most widely used leakage model is the *probing model* defined by Ishai, Sahai, and Wagner at (CRYPTO 2003). While it is advantageously convenient for security proofs, it does not capture an adversary exploiting full leakage traces as, *e.g.*, in horizontal attacks. Those attacks target the multiple manipulations of the same share to reduce noise and recover the corresponding value. To capture a wider class of attacks another model was introduced and is referred to as the *random probing model*. From a leakage parameter p, each wire of the circuit leaks its value with probability p. While this model much better reflects the physical reality of side channels, it requires more complex security proofs and does not yet come with practical constructions.

In this paper, we define the first framework dedicated to the random probing model. We provide an automatic tool, called VRAPS, to quantify the random probing security of a circuit from its leakage probability. We also formalize a composition property for secure random probing gadgets and exhibit its relation to the *strong non-interference* (SNI) notion used in the context of probing security. We then revisit the expansion idea proposed by Ananth, Ishai, and Sahai (CRYPTO 2018) and introduce a compiler that builds a random probing secure circuit from small base gadgets achieving a *random probing expandability* property. We instantiate this compiler with small gadgets for which we verify the expected properties directly from our automatic tool. Our construction can tolerate a leakage probability up to 2^{-8}, against 2^{-25} for the previous construction, with a better asymptotic complexity.

Keywords: Compiler · Masking · Automated verification · Random probing model

D. Micciancio and T. Ristenpart (Eds.): CRYPTO 2020, LNCS 12170, pp. 339–368, 2020.
https://doi.org/10.1007/978-3-030-56784-2_12

1 Introduction

Most cryptographic algorithms are assumed to be secure against *black-box* attacks where the adversary is limited to the knowledge of some inputs and outputs to recover the manipulated secrets. However, as revealed in the late nineties [19], when implemented on physical devices, they become vulnerable to the more powerful *side-channel attacks* which additionally exploit the physical emanations such as temperature, time, power consumption, electromagnetic radiations.

As such attacks may only require cheap equipment and can be easily mounted in a short time interval, the community had to adapt quickly by looking for efficient countermeasures. The most widely deployed approach to counteract side-channel attacks was simultaneously introduced in 1999 by Chari et al. [11] and by Goubin and Patarin [16] and is now called *masking*. Basically, the idea is to split each sensitive variable x of the implementation into n shares such that $n-1$ of them are generated uniformly at random and the last one is computed as the combination of x and all the previous shares according to some group law $*$. When $*$ is the (bitwise) addition, we talk about *linear sharing* (aka Boolean masking). The adversary thus needs to get information on all the shares of x to recover information on the sensitive value. This countermeasure is really simple to implement for linear operations which are simply applied on each share separately. However, things are getting trickier for non-linear operations where it is impossible to compute the result without combining shares.

To reason about the security of masked implementations, the community introduced leakage models. One of the most broadly used is the *probing model*, introduced by Ishai, Sahai, and Wagner [18]. In a nutshell, a circuit is claimed to be t-probing secure if the exact values of any set of t intermediate variables do not reveal any information on the secrets. As leakage traces are assumed to reveal noisy functions of the manipulated data, this model is motivated by the difficulty to recover information from the combination of t variables from their noisy functions in masking schemes (as t grows). Nevertheless, the probing model fails to capture the huge amount of information resulting from the leakage of all manipulated data, and in particular from the repeated manipulation of identical values (see horizontal attacks in [7]). Therefore, after a long sequence of works building and analyzing masking schemes with respect to their security in the probing model [8,14,23], the community is now looking for security in more practical models.

The *noisy leakage model* was originally considered by Chari et al. in [11] and was later formalized by Prouff and Rivain in [22] as a specialization of the *only computation leaks* model [21] in order to better capture the reality of the physical leakage. Informally, a circuit is secure in the noisy leakage model if the adversary cannot recover the secrets from a noisy function of each intermediate variable of the implementation. While realistic, this model is not convenient for security proofs, and therefore masking schemes continued to be verified in the probing model relying on the *not tight* reduction that was formally established by Duc, Dziembowski, and Faust [15].

The latter reduction actually came with an intermediate leakage model, called *random probing model*, to which the security in the noisy leakage model reduces to. In the random probing model, each intermediate variable leaks with some constant leakage probability p. A circuit is secure in this model if there is a negligible probability that these leaking wires actually reveal information on the secrets. It is worth noting that this notion advantageously captures the horizontal attacks which exploit the repeated manipulations of variables throughout the implementation. Classical probing-secure schemes are also secure in the random probing model but the tolerated leakage probability (a.k.a. leakage rate) might not be constant which is not satisfactory from a practical viewpoint. Indeed, in practice the side-channel noise might not be customizable by the implementer.

Only a few constructions [1–3] tolerate a constant leakage probability. These three constructions are conceptually involved and their practical instantiation is not straightforward. The first one from Ajtai et al. and its extension [3] are based on expander graphs. The tolerated probability is not made explicit. The third work [2] is based on multi-party computation protocols and an expansion strategy; the tolerated probability is around 2^{-26} and for a circuit with $|C|$ gates, the complexity is $\mathcal{O}(|C| \cdot \mathsf{poly}(\kappa))$ for some parameter κ but the polynomial is not made explicit.

Following the long sequence of works relying on the probing security, formal tools have recently been built to supervise the development of masking implementations proven secure in the probing model. Namely, verification tools are now able to produce a security proof or identify potential attacks from the description of a masked implementation at up to some masking orders (i.e., <5) [4,10,13]. In the same vein, compilers have been built to automatically generate masked implementations at any order given the high level description of a primitive [5,9,10]. Nevertheless, no equivalent framework has yet been proposed to verify the security of implementations in the random probing model.

Our contributions. In this paper, we aim to fill this huge gap by providing a framework to verify, compose, and build random probing secure circuits from simple gadgets. Our contributions are three-fold.

Automatic verification tool. As a first contribution, we define a verification method that we instantiate in a tool to automatically exhibit the random probing security parameters of any small circuit defined with addition and multiplication gates whose wires leak with some probability p. In a nutshell, a circuit is (p, f)-random probing secure if it leaks information on the secret with probability $f(p)$, where $f(p)$ is the *failure probability function*. From these notations, our tool named VRAPS (for Verifier of Random Probing Security), based on top of a set of rules that were previously defined to verify the probing security of implementations [4], takes as input the description of a circuit and outputs an upper bound on the failure probability function. While it is limited to small circuits by complexity, the state-of-the-art shows that verifying those circuits can be particularly useful in practice (see e.g. the `maskVerif` tool [4]), for instance

to verify gadgets and then deduce global security through composition properties and/or low-order masked implementations. The source code of VRAPS is publicly available.[1]

Composition and expanding compiler. We introduce a composition security property to make gadgets composable in a global random probing secure circuit. We exhibit the relation between this new *random probing composability* (RPC) notion and the *strong non-interference* (SNI) notion which is widely used in the context of probing security [5]. Then, we revisit the modular approach of Ananth, Ishai, and Sahai [2] which uses an expansion strategy to get random probing security from a multi-party computation protocol. We introduce the *expanding compiler* that builds random probing secure circuits from small base gadgets. We formalize the notion of *random probing expandability* (RPE) and show that a base gadget satisfying this notion can be securely used in the expanding compiler to achieve arbitrary/composable random probing security. As a complementary contribution, our verification tool, VRAPS, is extended to verify the newly introduced RPC and RPE properties.

Instantiation. We instantiate the expanding compiler with new constructions of simple base gadgets that fulfill the desired RPE property, which is verified by VRAPS. For a security level κ, our instantiation achieves a complexity of $\mathcal{O}(\kappa^{7.5})$ and tolerates a constant leakage probability $p \approx 0.0045 > 2^{-8}$. In comparison, and as a side contribution, we provide a precise analysis of the construction from [2] and show that it achieves an $\mathcal{O}(\kappa^{8.2})$ complexity for a much lower tolerated leakage probability ($p \approx 2^{-26}$). Finally, we note that our framework probably enables more efficient constructions based on different base gadgets; we leave such optimizations open for future works.

2 Preliminaries

Along the paper, $\mathbb{K}$ shall denote a finite field. For any $n \in \mathbb{N}$, we shall denote $[n]$ the integer set $[n] = [1, n] \cap \mathbb{Z}$. For any tuple $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{K}^n$ and any set $I \subseteq [n]$, we shall denote $\boldsymbol{x}|_I = (x_i)_{i \in I}$. Any two probability distributions D_1 and D_2 are said ε-*close*, denoted $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε, that is

$$\mathsf{SD}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon \ ,$$

where $p_{D_1}(\cdot)$ and $p_{D_1}(\cdot)$ denote the probability mass functions of D_1 and D_2.

2.1 Circuit Compilers

In this paper, an *arithmetic circuit* over a field $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations over $\mathbb{K}$. We consider three types of arithmetic gate:

[1] See https://github.com/CryptoExperts/VRAPS.

– an addition gate, of fan-in 2 and fan-out 1, computes an addition over $\mathbb{K}$,
– a multiplication gate, of fan-in 2 and fan-out 1, computes a multiplication over $\mathbb{K}$,
– a copy gate, of fan-in 1 and fan-out 2, outputs two copies of its input.

A *randomized arithmetic circuit* is equipped with an additional type of gate:

– a random gate, of fan-in 0 and fan-out 1, outputs a fresh uniform random value of $\mathbb{K}$.

A (randomized) arithmetic circuit is further formally composed of input gates of fan-in 0 and fan-out 1 and output gates of fan-in 1 and fan-out 0. Evaluating an ℓ-input m-output circuit C consists in writing an input $\boldsymbol{x} \in \mathbb{K}^\ell$ in the input gates, processing the gates from input gates to output gates, then reading the output $\boldsymbol{y} \in \mathbb{K}^m$ from the output gates. This is denoted by $\boldsymbol{y} = C(\boldsymbol{x})$. During the evaluation process, each wire in the circuit is assigned with a value on $\mathbb{K}$. We call the tuple of all these wire values a *wire assignment* of C (on input $\boldsymbol{x}$).

Definition 1 (Circuit Compiler). *A* circuit compiler *is a triplet of algorithms* $(\mathsf{CC}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*

– CC *(circuit compilation) is a deterministic algorithm that takes as input an arithmetic circuit C and outputs a randomized arithmetic circuit $\widehat{C}$.*
– Enc *(input encoding) is a probabilistic algorithm that maps an input $\boldsymbol{x} \in \mathbb{K}^\ell$ to an encoded input $\widehat{\boldsymbol{x}} \in \mathbb{K}^{\ell'}$.*
– Dec *(output decoding) is a deterministic algorithm that maps an encoded output $\widehat{\boldsymbol{y}} \in \mathbb{K}^{m'}$ to a plain output $\boldsymbol{y} \in \mathbb{K}^m$.*

These three algorithms satisfy the following properties:

– ***Correctness:*** *For every arithmetic circuit C of input length ℓ, and for every $\boldsymbol{x} \in \mathbb{K}^\ell$, we have*

$$\Pr\left(\mathsf{Dec}\big(\widehat{C}(\widehat{\boldsymbol{x}})\big) = C(\boldsymbol{x}) \;\middle|\; \widehat{\boldsymbol{x}} \leftarrow \mathsf{Enc}(\boldsymbol{x})\right) = 1 \ , \ \textit{where } \widehat{C} = \mathsf{CC}(C).$$

– ***Efficiency:*** *For some security parameter $\lambda \in \mathbb{N}$, the running time of $\mathsf{CC}(C)$ is $\mathrm{poly}(\lambda, |C|)$, the running time of $\mathsf{Enc}(\boldsymbol{x})$ is $\mathrm{poly}(\lambda, |\boldsymbol{x}|)$ and the running time of $\mathsf{Dec}\big(\widehat{\boldsymbol{y}}\big)$ is $\mathrm{poly}(\lambda, |\widehat{\boldsymbol{y}}|)$, where $\mathrm{poly}(\lambda, q) = O(\lambda^{k_1} q^{k_2})$ for some constants k_1, k_2.*

2.2 Linear Sharing and Gadgets

In the following, the *n-linear decoding* mapping, denoted LinDec, refers to the function $\bigcup_n \mathbb{K}^n \to \mathbb{K}$ defined as

$$\mathsf{LinDec} : (x_1, \ldots, x_n) \mapsto x_1 + \cdots + x_n \ ,$$

for every $n \in \mathbb{N}$ and $(x_1, \ldots, x_n) \in \mathbb{K}^n$. We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $(\widehat{x}_1, \ldots, \widehat{x}_\ell) \in (\mathbb{K}^n)^\ell$ the n-linear decoding mapping acts as

$$\mathsf{LinDec} : (\widehat{x}_1, \ldots, \widehat{x}_\ell) \mapsto (\mathsf{LinDec}(\widehat{x}_1), \ldots, \mathsf{LinDec}(\widehat{x}_\ell)) \ .$$

Let us recall that for some tuple $\widehat{x} = (x_1, \ldots, x_n) \in \mathbb{K}^n$ and for some set $I \subseteq [n]$, the tuple $(x_i)_{i \in I}$ is denoted $\widehat{x}|_I$.

Definition 2 (Linear Sharing). *Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an* n-linear sharing *of x is a random vector $\widehat{x} \in \mathbb{K}^n$ such that $\mathsf{LinDec}(\widehat{x}) = x$. It is said to be* uniform *if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $\widehat{x}|_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. A* n-linear encoding *is a probabilistic algorithm LinEnc which on input a tuple $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in \mathbb{K}^\ell$ outputs a tuple $\widehat{\boldsymbol{x}} = (\widehat{x}_1, \ldots, \widehat{x}_\ell) \in (\mathbb{K}^n)^\ell$ such that $\widehat{x}_i$ is a uniform n-sharing of x_i for every $i \in [\ell]$.*

In the following, we shall call an (n-*share*, ℓ-*to-m*) *gadget*, a randomized arithmetic circuit that maps an input $\widehat{\boldsymbol{x}} \in (\mathbb{K}^n)^\ell$ to an output $\widehat{\boldsymbol{y}} \in (\mathbb{K}^n)^m$ such that $\boldsymbol{x} = \mathsf{LinDec}(\widehat{\boldsymbol{x}}) \in \mathbb{K}^\ell$ and $\boldsymbol{y} = \mathsf{LinDec}(\widehat{\boldsymbol{y}}) \in \mathbb{K}^m$ satisfy $\boldsymbol{y} = g(\boldsymbol{x})$ for some function g. In this paper, we shall consider gadgets for three types of functions (corresponding to the three types of gates): the addition $g : (x_1, x_2) \mapsto x_1 + x_2$, the multiplication $g : (x_1, x_2) \mapsto x_1 \cdot x_2$ and the copy $g : x \mapsto (x, x)$. We shall generally denote such gadgets G_{add}, G_{mult} and G_{copy} respectively.

Definition 3 (Standard Circuit Compiler). *Let $\lambda \in \mathbb{N}$ be some security parameter and let $n = \text{poly}(\lambda)$. Let G_{add}, G_{mult} and G_{copy} be n-share gadgets respectively for the addition, multiplication and copy over $\mathbb{K}$. The* standard circuit compiler *with* sharing order n *and* base gadgets G_{add}, G_{mult}, G_{copy} *is the circuit compiler* $(\mathsf{CC}, \mathsf{Enc}, \mathsf{Dec})$ *satisfying the following:*

1. *The input encoding* Enc *is an n-linear encoding.*
2. *The output decoding* Dec *is the n-linear decoding mapping* LinDec.
3. *The circuit compilation* CC *consists in replacing each gate in the original circuit by an n-share gadget with corresponding functionality (either G_{add}, G_{mult} or G_{copy}), and each wire by a set of n wires carrying a n-linear sharing of the original wire. If the input circuit is a randomized arithmetic circuit, each of its random gates is replaced by n random gates, which duly produce a n-linear sharing of a random value.*

For such a circuit compiler, the correctness and efficiency directly holds from the correctness and efficiency of the gadgets G_{add}, G_{mult} and G_{copy}.

2.3 Random Probing Leakage

Let $p \in [0, 1]$ be some constant leakage probability parameter. This parameter is sometimes called *leakage rate* in the literature. Informally, the p-random probing model states that during the evaluation of a circuit C each wire leaks its value with probability p (and leaks nothing otherwise), where all the wire leakage events are mutually independent.

In order to formally define the random-probing leakage of a circuit, we shall consider two probabilistic algorithms:

– The *leaking-wires sampler* takes as input a randomized arithmetic circuit C and a probability $p \in [0, 1]$, and outputs a set $\mathcal{W}$, denoted as

$$\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p) \ ,$$

where $\mathcal{W}$ is constructed by including each wire label from the circuit C with probability p to $\mathcal{W}$ (where all the probabilities are mutually independent).

– The *assign-wires sampler* takes as input a randomized arithmetic circuit C, a set of wire labels $\mathcal{W}$ (subset of the wire labels of C), and an input $\boldsymbol{x}$, and it outputs a $|\mathcal{W}|$-tuple $\boldsymbol{w} \in (\mathbb{K} \cup \{\perp\})^{|\mathcal{W}|}$, denoted as

$$\boldsymbol{w} \leftarrow \mathsf{AssignWires}(C, \mathcal{W}, \boldsymbol{x}) \ ,$$

where $\boldsymbol{w}$ corresponds to the assignments of the wires of C with label in $\mathcal{W}$ for an evaluation on input $\boldsymbol{x}$.

We can now formally define the random probing leakage of a circuit.

Definition 4 (Random Probing Leakage). *The* p-random probing leakage *of a randomized arithmetic circuit* C *on input* $\boldsymbol{x}$ *is the distribution* $\mathcal{L}_p(C, \boldsymbol{x})$ *obtained by composing the leaking-wires and assign-wires samplers as*

$$\mathcal{L}_p(C, \boldsymbol{x}) \stackrel{id}{=} \mathsf{AssignWires}(C, \mathsf{LeakingWires}(C, p), \boldsymbol{x}) \ .$$

Remark 1. By convention the output wires of C (*i.e.* the wires incoming output gates) are excluded by the $\mathsf{LeakingWires}$ sampler whereas the input wires of C (*i.e.* the wires connecting input gates to subsequent gates) are included. Namely the output set $\mathcal{W}$ of $\mathsf{LeakingWires}(C, p)$ does not include any output wire label of C. This is because when composing several circuits (or gadgets), the output wires of a circuit are the input wires in a next circuit. This also relates to the widely admitted *only computation leaks* assumption [21]: the processing of a gate leaks information on its input values (and information on the output can be captured through information on the input).

Definition 5 (Random Probing Security). *A randomized arithmetic circuit* C *with* $\ell \cdot n \in \mathbb{N}$ *input gates is* (p, ε)-random probing secure *with respect to encoding* Enc *if there exists a simulator* Sim *such that for every* $\boldsymbol{x} \in \mathbb{K}^{\ell}$*:*

$$\mathsf{Sim}(C) \approx_{\varepsilon} \mathcal{L}_p(C, \mathsf{Enc}(\boldsymbol{x})) \ . \tag{1}$$

A circuit compiler $(\mathsf{CC}, \mathsf{Enc}, \mathsf{Dec})$ *is* (p, ε)-random probing secure *if for every (randomized) arithmetic circuit* C *the compiled circuit* $\widehat{C} = \mathsf{CC}(C)$ *is* $(p, |C| \cdot \varepsilon)$*-random probing secure where* $|C|$ *is the size of original circuit.*

As in [2] we shall consider a *simulation with abort*. In this approach, the simulator first calls the leaking-wires sampler to get a set $\mathcal{W}$ and then either aborts (or fails) with probability ε or outputs the exact distribution of the wire assignment corresponding to $\mathcal{W}$. Formally, for any leakage probability $p \in [0, 1]$, the simulator Sim is defined as

$$\mathsf{Sim}(\widehat{C}) = \mathsf{SimAW}(\widehat{C}, \mathsf{LeakingWires}(\widehat{C}, p)) \tag{2}$$

where SimAW, the *wire assignment simulator*, either returns $\perp$ (simulation failure) or a perfect simulation of the requested wires. Formally, the experiment

$$\mathcal{W} \leftarrow \mathsf{LeakingWires}(\widehat{C}, p)$$
$$out \leftarrow \mathsf{SimAW}(\widehat{C}, \mathcal{W})$$

leads to

$$\Pr[out = \perp] = \varepsilon \quad \text{and} \quad \left(out \mid out \neq \perp\right) \stackrel{\text{id}}{=} \left(\mathsf{AssignWires}(\widehat{C}, \mathcal{W}, \mathsf{Enc}(\boldsymbol{x})) \mid out \neq \perp\right) . \tag{3}$$

It is not hard to see that if we can construct such a simulator SimAW for a compiled circuit $\widehat{C}$, then this circuit is (p, ε)-random probing secure.

3 Formal Verification

In this section we show how to compute the simulation failure probability $f(p)$ as a function of the leakage probability p for the base gadgets. Since even for simple gadgets this tasks would be difficult to perform by hand, we use a formal verification tool to compute $f(p)$.

3.1 Simulation Failure Probability

We first derive an upper bound on the simulation failure probability as a function of the leakage probability p. We consider a compiled circuit $\widehat{C}$ composed of s wires labeled from 1 to s and a simulator SimAW as defined in previous section. For any sub-set $\mathcal{W} \subseteq [s]$ we denote by $\delta_{\mathcal{W}}$ the value defined as follows:

$$\delta_{\mathcal{W}} = \begin{cases} 1 & \text{if } \mathsf{SimAW}(\widehat{C}, \mathcal{W}) = \perp, \\ 0 & \text{otherwise.} \end{cases}$$

The simulation failure probability ε in (3) can then be explicitly expressed as a function of p. Namely, we have $\varepsilon = f(p)$ with f defined for every $p \in [0, 1]$ by:

$$f(p) = \sum_{\mathcal{W} \subseteq [s]} \delta_{\mathcal{W}} \cdot p^{|\mathcal{W}|} \cdot (1-p)^{s-|\mathcal{W}|} \ . \tag{4}$$

Letting c_i be the number of sub-sets $\mathcal{W} \subseteq [s]$ of cardinality i for which $\delta_{\mathcal{W}} = 1$, namely for which the simulation fails, we have $c_i = \sum_{|\mathcal{W}|=i} \delta_{\mathcal{W}}$ and hence (4) simplifies to

$$f(p) = \sum_{i=1}^{s} c_i \cdot p^i \cdot (1-p)^{s-i} \ . \tag{5}$$

For any circuit $\widehat{C}$ achieving t-probing security, the values $\delta_{\mathcal{W}}$ with $|\mathcal{W}| \leq t$ are equal to zero, and therefore the corresponding c_i's are zero, which implies the following simplification:

$$f(p) = \sum_{i=t+1}^{s} c_i \cdot p^i \cdot (1-p)^{s-i} \ .$$

Moreover, by definition, the coefficients c_i satisfy:

$$c_i \leqslant \binom{s}{i} \tag{6}$$

which leads to the following upper-bound for $f(p)$:

$$f(p) \leqslant \sum_{i=t+1}^{s} \binom{s}{i} \cdot p^i \cdot (1-p)^{s-i} \ .$$

An example of the evaluation of $f(p)$ for the 2-share multiplication gadget from [18] is given in the full version of this paper.

3.2 Verification Method

For any compiled circuit $\widehat{C}$ and any simulator defined as in Sect. 2.3, the computation of the function $f(p)$ for any probability p essentially amounts to computing the values of the coefficients c_i's appearing in (5). If no assumption is made on the circuit, this task seems difficult to carry out by hand. Actually, it may be checked that an exhaustive testing of all the possible tuples of wires for a gadget with s wires has complexity lower bounded by 2^s, which gives 2^{21} for a simple gadget like the ISW multiplication gadget with two shares per input. Here, we introduce a verification tool, that we call VRAPS, enabling to automatically test the perfect simulation for any set of wires of size lower than or equal to some threshold β. The role of the latter threshold is simply to control the verification duration (which can be long if the circuit to test is complex). Our tool implicitly defines a simulator that may fail with a probability $\varepsilon = f(p)$ satisfying (5).

The verification tool takes as input the representation of a compiled circuit $\widehat{C}$ and a test parameter β, and outputs the list of coefficients c_1, ..., c_β. It is assumed that $\widehat{C}$ takes as input the n-linear encoding $\mathsf{Enc}(\boldsymbol{x})$ of vector $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ defined in $\mathbb{K}^\ell$. It is moreover assumed that $\widehat{C}$ is composed of s wires respectively denoted by w_1, ..., w_s. In the following, we consider s-tuples in the form of $u = (u_1, \ldots, u_s) \in \{0,1\}^s$ together with the common rule $u' \subset u$ iff for every $i \in [s]$, $u'_i = 1 \Rightarrow u_i = 1$ (in this case u' will be said to be included in u). An s-tuple u for which there exists an assignment of the wires in $\mathcal{W} = \{w_i;\ i \in [s], u_i = 1\}$ such that the simulation fails shall be called a *failure tuple*. Such a tuple shall be said to be *incompressible* if no tuple $t' \subset t$ is a failure tuple. The main idea of the proposed verification tool is to test the simulation failure only on incompressible failure tuples whose Hamming weight ranges from 1 to β. The steps are described in Algorithm 1.

The function listTuples outputs the list of all s-tuples with Hamming weight h with $h \in [s]$. The function eliminateFromSmaller takes as input the list ℓ_h of current tuples of Hamming weight h and the list of incompressible failure tuples ℓ_p. It returns two lists:

- $\ell_h^{f_1}$: the elements of ℓ_h which are not incompressible (*i.e.* which include at least one element from ℓ_p)
- ℓ_h^{p}: the elements of ℓ_h which are incompressible (*i.e.* $\ell_h \backslash \ell_h^{f_1}$)

Algorithm 1. Verification tool

Input: a compiled circuit $\widehat{C}$ with s wires and a threshold $\beta \leqslant s$
Output: a list of β coefficients c_1, ..., c_β

1: $\ell_p \leftarrow []$ ▷ will be used to store a list of failure tuples
2: $\boldsymbol{c} \leftarrow (0, \ldots, 0)$ ▷ will be used to store the output coefficients
3: **for** $h = 1$ to β **do**
4: $\quad \ell_h \leftarrow \mathsf{listTuples}(s,h)$ ▷ list of s-tuples of Hamming weight h
5: $\quad (\ell_h^p, \ell_h^{f_1}) \leftarrow \mathsf{eliminateFromSmaller}(\ell_h, \ell_p)$ ▷ select tuples including an incompressible failure tuple
6: $\quad \ell_h^{f_2} \leftarrow \mathsf{failureTest}(\widehat{C}, \ell_h^p)$ ▷ identify failure tuples in ℓ_h^p
7: $\quad \ell_p \leftarrow \ell_p \cup \ell_h^{f_2}$ ▷ update list of incompressible failure tuples
8: $\quad \boldsymbol{c} \leftarrow \mathsf{updateCoeffs}(\boldsymbol{c}, \ell_h^{f_1} \cup \ell_h^{f_2})$ ▷ update coefficients
9: **end for**
10: **return** $\boldsymbol{c}$

The function failureTest takes as input the second list ℓ_h^p and checks if a perfect simulation can be achieved for each wire family $\mathcal{W}$ corresponding to a tuple in ℓ_h^p. Basically, for each wire family, a sequence of rules taken from maskVerif [4] is tested to determine whether $\mathcal{W}$ can be perfectly simulated. It outputs $\ell_h^{f_2}$, the list of incompressible failure s-tuples of Hamming weight h. In a nutshell, each wire w_i in $\mathcal{W}$ is considered together with the algebraic expression $\varphi_i(\cdot)$ describing its assignment by $\widehat{C}$ as a function of the circuit inputs and the random values returned by the random gates, then the three following rules are successively and repeatedly applied on all the wires families $\mathcal{W}$ (see [4] for further details):

rule 1: check whether all the expressions $\varphi_i(\cdot)$ corresponding to wires w_i in $\mathcal{W}$ contain all the shares of at least one of the coordinates of $\boldsymbol{x}$;
rule 2: for every $\varphi_i(\cdot)$, check whether a random r (*i.e.* an output of a random gate) additively masks a sub-expression e (which does not involve r) and appears nowhere else in the other $\varphi_j(\cdot)$ with $j \neq i$; in this case replace the sum of the so-called sub-expression and r by r, namely $e + r \leftarrow r$;
rule 3: apply mathematical simplifications on the tuple.

Function updateCoeffs takes as input the current array of β coefficients c_i for $1 \leqslant i \leqslant \beta$ and the concatenation of both lists of potential failure tuples $\ell_h^{f_1}$ and $\ell_h^{f_2}$. For each failure tuple, these coefficients are updated.

Implementation. An implementation of Algorithm 1 has been developed in Python. This tool, named VRAPS, has been open sourced at:

https://github.com/CryptoExperts/VRAPS

Further details. The full version of this paper gives more details on the concrete link between VRAPS and maskVerif and provides two possible optimizations to improve the performances of the former. Three examples of multiplication gadgets are also displayed to illustrate the behavior of our verification tool.

4 Composition

This section aims to provide composition properties for random-probing secure gadgets. In a nutshell, we aim to show how to build random probing secure larger circuits from specific random probing secure building blocks.

4.1 Random Probing Composability

We introduce hereafter the *random probing composability* notion for a gadget. In the following definition, for an n-share, ℓ-to-m gadget, we denote by $\boldsymbol{I}$ a collection of sets $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ with $I_1 \subseteq [n]$, $\ldots$, $I_\ell \subseteq [n]$ where $n \in \mathbb{N}$ refers to the number of shares. For some $\widehat{\boldsymbol{x}} = (\widehat{x}_1, \ldots, \widehat{x}_\ell) \in (\mathbb{K}^n)^\ell$, we then denote $\widehat{\boldsymbol{x}}|_{\boldsymbol{I}} = (\widehat{x}_1|_{I_1}, \ldots, \widehat{x}_\ell|_{I_\ell})$ where $\widehat{x}_i|_{I_i} \in \mathbb{K}^{|I_i|}$ is the tuple composed of the coordinates of the sharing $\widehat{x}_i$ of indexes included in I_i.

Definition 6 (Random Probing Composability). *Let $n, \ell, m \in \mathbb{N}$. An n-share gadget $G : (\mathbb{K}^n)^\ell \to (\mathbb{K}^n)^m$ is (t, p, ε)-*random probing composable (RPC) *for some $t \in \mathbb{N}$ and $p, \varepsilon \in [0, 1]$ if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $\widehat{\boldsymbol{x}} \in (\mathbb{K}^n)^\ell$ and for every set collection $J_1 \subseteq [n]$, $\ldots$, $J_m \subseteq [n]$ of cardinals $|J_1| \leq t$, $\ldots$, $|J_m| \leq t$, the random experiment*

$$\begin{aligned} &\mathcal{W} \leftarrow \mathsf{LeakingWires}(G, p) \\ &\boldsymbol{I} \leftarrow \mathsf{Sim}_1^G(\mathcal{W}, \boldsymbol{J}) \\ &\mathit{out} \leftarrow \mathsf{Sim}_2^G\left(\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}\right) \end{aligned}$$

yields

$$\Pr\left((|I_1| > t) \vee \ldots \vee (|I_\ell| > t)\right) \leq \varepsilon \tag{7}$$

and

$$\mathit{out} \overset{id}{=} \left(\mathsf{AssignWires}(G, \mathcal{W}, \widehat{\boldsymbol{x}})\ ,\ \widehat{\boldsymbol{y}}|_{\boldsymbol{J}}\right)$$

where $\boldsymbol{J} = (J_1, \ldots, J_m)$ and $\widehat{\boldsymbol{y}} = G(\widehat{\boldsymbol{x}})$. Let $f : \mathbb{R} \to \mathbb{R}$. The gadget G is (t, f)-RPC if it is $(t, p, f(p))$-RPC for every $p \in [0, 1]$.

In the above definition, the first-pass simulator Sim_1^G determines the necessary input shares (through the returned collection of sets $\boldsymbol{I}$) for the second-pass simulator Sim_2^G to produce a perfect simulation of the leaking wires defined by the set $\mathcal{W}$ together with the output shares defined by the collection of sets $\boldsymbol{J}$. Note that there always exists such a collection of sets $\boldsymbol{I}$ since $\boldsymbol{I} = ([n], \ldots, [n])$ trivially allows a perfect simulation whatever $\mathcal{W}$ and $\boldsymbol{J}$. However, the goal of Sim_1^G is to return a collection of sets $\boldsymbol{I}$ with cardinals at most t. The idea behind this constraint is to keep the following composition invariant: for each gadget we can achieve a perfect simulation of the leaking wires plus t shares of each output sharing from t shares of each input sharing. We shall call *failure event* the event that at least one of the sets I_1, $\ldots$, I_ℓ output of Sim_1^G has cardinality greater than t. When (t, p, ε)-RPC is achieved, the failure event probability is

upper bounded by ε according to (7). A failure event occurs whenever Sim_2^G requires more than t shares of one input sharing to be able to produce a perfect simulation of the leaking wires (*i.e.* the wires with label in $\mathcal{W}$) together with the output shares in $\widehat{\boldsymbol{y}}|_{\boldsymbol{J}}$. Whenever such a failure occurs, the composition invariant is broken. In the absence of failure event, the RPC notion implies that a perfect simulation can be achieved for the full circuit composed of RPC gadgets. This is formally stated in the next theorem whose proof is given in the full version.

Theorem 1 (Composition). *Let $t \in \mathbb{N}$, $p, \varepsilon \in [0,1]$, and CC be a standard circuit compiler with (t,p,ε)-RPC base gadgets. For every (randomized) arithmetic circuit C composed of $|C|$ gadgets, the compiled circuit $\mathsf{CC}(C)$ is $(p, |C| \cdot \varepsilon)$-random probing secure. Equivalently, the standard circuit compiler CC is (p,ε)-random probing secure.*

4.2 Relation with Standard Probing Composition Notions

We first reformulate the Strong Non-Interference notion introduced in [5] with the formalism used for our definition of the Random Probing Composability.

Definition 7 (Strong Non-Interference (SNI)). *Let n, ℓ and t be positive integers. An n-share gadget $G : (\mathbb{K}^n)^\ell \to \mathbb{K}^n$ is t-SNI if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every set $J \subseteq [n]$ and subset $\mathcal{W}$ of wire labels from G satisfying $|\mathcal{W}| + |J| \leqslant t$, the following random experiment with any $\widehat{\boldsymbol{x}} \in (\mathbb{K}^n)^\ell$*

$$\begin{aligned} \boldsymbol{I} &\leftarrow \mathsf{Sim}_1^G(\mathcal{W}, J) \\ out &\leftarrow \mathsf{Sim}_2^G\left(\widehat{\boldsymbol{x}}|_{\boldsymbol{I}}\right) \end{aligned}$$

yields

$$|I_1| \leqslant |\mathcal{W}|, \ldots, |I_\ell| \leqslant |\mathcal{W}| \tag{8}$$

and

$$out \overset{id}{=} \left(\mathsf{AssignWires}(G, \mathcal{W}, \widehat{\boldsymbol{x}})\ ,\ \widehat{\boldsymbol{y}}|_J\right) \tag{9}$$

where $\boldsymbol{I} = (I_1, \ldots, I_\ell)$ and $\widehat{\boldsymbol{y}} = G(\widehat{\boldsymbol{x}})$.

Then, we demonstrate that gadgets satisfying the t-SNI notion are also random probing composable for specific values that we explicit in the following proposition, whose proof is available in the full version of this paper.

Proposition 1. *Let n, ℓ and t be positive integers and let G be a gadget from $(\mathbb{K}^n)^\ell$ to $\mathbb{K}^n$. If G is t-SNI, then it is also $(t/2, p, \varepsilon)$-RPC for any probability p and ε satisfying:*

$$\varepsilon = \sum_{i=\lfloor \frac{t}{2}+1 \rfloor}^{s} \binom{s}{i} p^i (1-p)^{s-i}\ , \tag{10}$$

where s is the number of wires in G.

4.3 Verification of Gadget Composability

Our random probing verification tool (Algorithm 1) can be easily extended to define a simulator for the (t, p, ε)-random probing composability of a gadget for some t and some p. This essentially amounts to extend Algorithm 1 inputs with a multi-set $\mathcal{O}$ and to modify the failureTest procedure in order to test the simulation for each tuple in the input list ℓ_n^p augmented with the outputs coordinates with indices in $\mathcal{O}$. Then, our extended algorithm is called for every set $\mathcal{O}$ composed of at most t indices in each of the sets $J_1, \ldots, J_m$. The output for the call with input set $\mathcal{O}$ is denoted by $\boldsymbol{c}_{\mathcal{O}} = (c_1^{\mathcal{O}}, \ldots, c_\beta^{\mathcal{O}})$. For our simulator construction, the probability ε satisfies

$$\varepsilon = \sum_{i=1}^{s} c_i \cdot p^i \cdot (1-p)^{s-i},$$

where s denotes the number of wires in the tested gadget. Moreover, the c_i's satisfy $c_i = \max_{\mathcal{O}} c_i^{\mathcal{O}}$.

The full version of this paper provides an illustration of the proposition with the well deployed 3-share ISW multiplication gadget [18].

5 Expansion

Constructing random-probing-secure circuit compilers with a gadget expansion strategy has been proposed by Ananth, Ishai and Sahai in [2]. Such strategy was previously used in the field of multi-party computation (MPC) with different but close security goals [12,17]. Note that such approach is called *composition* in [2] since it roughly consists in composing a base circuit compiler several times. We prefer the terminology of *expansion* here to avoid any confusion with the notion of composition for gadgets as considered in Sect. 4 and usual in the literature – see for instance [5,8,10].

We recall hereafter the general principle of the gadget expansion strategy and provide an asymptotic analysis of the so-called *expanding circuit compiler*. Then we propose an implementation of this strategy which relies on the new notion of *gadget expandability*. In contrast, the construction of [2] relies on a t-out-n secure MPC protocol in the passive security model. The advantage of our notion is that it can be achieved and/or verified by simple atomic gadgets leading to simple and efficient constructions. After introducing the gadget expandability notion, we show that it allows to achieve random-probing security with the expansion strategy. We finally explain how to adapt the verification tool described in Sect. 3 to this expandability notion.

5.1 Expansion Strategy

The basic principle of the gadget expansion strategy is as follows. Assume we have three n-share gadgets G_{add}, G_{mult}, G_{copy} and denote CC the standard circuit compiler for these base gadgets. We can derive three new n^2-share gadgets

by simply applying CC to each gadget: $G^{(2)}_{\text{add}} = \mathsf{CC}(G_{\text{add}})$, $G^{(2)}_{\text{mult}} = \mathsf{CC}(G_{\text{mult}})$ and $G^{(2)}_{\text{copy}} = \mathsf{CC}(G_{\text{copy}})$. Let us recall that this process simply consists in replacing each addition gate in the original gadget by G_{add}, each multiplication gate by G_{mult} and each copy gate by G_{copy}, and by replacing each wire by n wires carrying a sharing of the original wire. Doing so, we obtain n^2-share gadgets for the addition, multiplication and copy on $\mathbb{K}$. This process can be iterated an arbitrary number of times, say k, to an input circuit C:

$$C \xrightarrow{\mathsf{CC}} \widehat{C}_1 \xrightarrow{\mathsf{CC}} \cdots \xrightarrow{\mathsf{CC}} \widehat{C}_k \ .$$

The first output circuit $\widehat{C}_1$ is the original circuit in which each gate is replaced by a base gadget G_{add}, G_{mult} or G_{copy}. The second output circuit $\widehat{C}_2$ is the original circuit C in which each gate is replaced by an n^2-share gadget $G^{(2)}_{\text{add}}$, $G^{(2)}_{\text{mult}}$ or $G^{(2)}_{\text{copy}}$ as defined above. Equivalently, $\widehat{C}_2$ is the circuit $\widehat{C}_1$ in which each gate is replaced by a base gadget. In the end, the output circuit $\widehat{C}_k$ is hence the original circuit C in which each gate has been replaced by a k-expanded gadget and each wire as been replaced by n^k wires carrying an (n^k)-linear sharing of the original wire. The underlying compiler is called *expanding circuit compiler* which is formally defined hereafter.

Definition 8 (Expanding Circuit Compiler). *Let* CC *be the standard circuit compiler with sharing order* n *and base gadgets* G_{add}, G_{mult}, G_{copy}. *The* expanding circuit compiler *with* expansion level k *and base compiler* CC *is the circuit compiler* $(\mathsf{CC}^{(k)}, \mathsf{Enc}^{(k)}, \mathsf{Dec}^{(k)})$ *satisfying the following:*

1. *The input encoding* $\mathsf{Enc}^{(k)}$ *is an* (n^k)*-linear encoding.*
2. *The output decoding* Dec *is the* (n^k)*-linear decoding mapping.*
3. *The circuit compilation is defined as*

$$\mathsf{CC}^{(k)}(\cdot) = \underbrace{\mathsf{CC} \circ \mathsf{CC} \circ \cdots \circ \mathsf{CC}}_{k \text{ times}}(\cdot)$$

The goal of the expansion strategy in the context of random probing security is to replace the leakage probability p of a wire in the original circuit by the failure event probability ε in the subsequent gadget simulation. If this simulation fails then one needs the full input sharing for the gadget simulation, which corresponds to leaking the corresponding wire value in the base case. The security is thus amplified by replacing the probability p in the base case by the probability ε (assuming that we have $\varepsilon < p$). If the failure event probability ε can be upper bounded by some function of the leakage probability: $\varepsilon < f(p)$ for every leakage probability $p \in [0, p_{\max}]$ for some $p_{\max} < 1$, then the expanding circuit compiler with expansion level k shall result in a security amplification as

$$p = \varepsilon_0 \xrightarrow{f} \varepsilon_1 \xrightarrow{f} \cdots \xrightarrow{f} \varepsilon_k = f^{(k)}(p) \ ,$$

which for an adequate function f (*e.g.* $f : p \mapsto p^2$) provides exponential security. In order to get such a security expansion, the gadgets must satisfy a stronger notion than the composability notion introduced in Sect. 4 which we call *random probing expandability*; see Sect. 5.3 below.

5.2 Asymptotic Analysis of the Expanding Compiler

In this section we show that the asymptotic complexity of a compiled circuit $\widehat{C} = \mathsf{CC}^{(k)}(C)$ is $|\widehat{C}| = \mathcal{O}\big(|C| \cdot \kappa^e\big)$ for security parameter κ, for some constant e that we make explicit.

Let us denote by $\boldsymbol{N} = (N_a, N_c, N_m, N_r)^\mathsf{T}$ the column vector of gate counts for some base gadget G, where N_a, N_c, N_m, N_r stands for the number of addition gates, copy gates, multiplication gates and random gates respectively. We have three different such vectors $\boldsymbol{N}_{\text{add}} \doteq (N_{\text{add},a}, N_{\text{add},c}, N_{\text{add},m}, N_{\text{add},r})^\mathsf{T}$, $\boldsymbol{N}_{\text{mult}} \doteq (N_{\text{mult},a}, N_{\text{mult},c}, N_{\text{mult},m}, N_{\text{mult},r})^\mathsf{T}$, $\boldsymbol{N}_{\text{copy}} \doteq (N_{\text{copy},a}, N_{\text{copy},c}, N_{\text{copy},m}, N_{\text{copy},r})^\mathsf{T}$ for the gate counts respectively in the base addition gadget G_{add}, in the base multiplication gadget G_{mult} and in the base copy gadgets G_{copy}. Let us define the 4×4 square matrix $\boldsymbol{M}$ as

$$\boldsymbol{M} = \big(\boldsymbol{N}_{\text{add}} \mid \boldsymbol{N}_{\text{copy}} \mid \boldsymbol{N}_{\text{mult}} \mid \boldsymbol{N}_{\text{rand}}\big) \quad \text{with} \quad \boldsymbol{N}_{\text{rand}} = (0,0,0,n)^\mathsf{T} \ ,$$

where the definition $\boldsymbol{N}_{\text{rand}}$ holds from the fact that the standard circuit compiler replaces each random gate by n random gates.

It can be checked that applying the standard circuit compiler with base gadgets G_{add}, G_{mult} and G_{copy} to some circuit C with gate-count vector $\boldsymbol{N}_C$ gives a circuit $\widehat{C}$ with gate-count vector $\boldsymbol{N}_{\widehat{C}} = \boldsymbol{M} \cdot \boldsymbol{N}_C$. It follows that the kth power of the matrix $\boldsymbol{M}$ gives the gate counts for the level-k gadgets as:

$$\boldsymbol{M}^k = \underbrace{\boldsymbol{M} \cdot \boldsymbol{M} \cdots \boldsymbol{M}}_{k \text{ times}} = \big(\boldsymbol{N}^{(k)}_{\text{add}} \mid \boldsymbol{N}^{(k)}_{\text{copy}} \mid \boldsymbol{N}^{(k)}_{\text{mult}} \mid \boldsymbol{N}^{(k)}_{\text{rand}}\big) \quad \text{with} \quad \boldsymbol{N}^{(k)}_{\text{rand}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ n^k \end{pmatrix}$$

where $\boldsymbol{N}^{(k)}_{\text{add}}$, $\boldsymbol{N}^{(k)}_{\text{mult}}$ and $\boldsymbol{N}^{(k)}_{\text{copy}}$ are the gate-count vectors for the level-k gadgets $G^{(k)}_{\text{add}}$, $G^{(k)}_{\text{mult}}$ and $G^{(k)}_{\text{copy}}$ respectively. Let us denote the eigen decomposition of $\boldsymbol{M}$ as $\boldsymbol{M} = \boldsymbol{Q} \cdot \boldsymbol{\Lambda} \cdot \boldsymbol{Q}^{-1}$, we get

$$\boldsymbol{M}^k = \boldsymbol{Q} \cdot \boldsymbol{\Lambda}^k \cdot \boldsymbol{Q}^{-1} \quad \text{with} \quad \boldsymbol{\Lambda}^k = \begin{pmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \lambda_3^k & \\ & & & \lambda_4^k \end{pmatrix}$$

where λ_1, λ_2, λ_3, λ_4 are the eigenvalues of $\boldsymbol{M}$. We then obtain an asymptotic complexity of

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot (\lambda_1^k + \lambda_2^k + \lambda_3^k + \lambda_4^k)\big) = \mathcal{O}\big(|C| \cdot \max(\lambda_1, \lambda_2, \lambda_3, \lambda_4)^k\big)$$

for a compiled circuit $\widehat{C} = \mathsf{CC}^{(k)}(C)$ (where the constant in the $\mathcal{O}(\cdot)$ depends on $\boldsymbol{Q}$ and shall be fairly small).

Interestingly, if multiplication gates are solely used in the multiplication gadget (*i.e.* $N_{\text{add},m} = N_{\text{copy},m} = 0$) which is the case in the constructions we consider

in this paper, it can be checked that (up to some permutation) the eigenvalues satisfy

$$(\lambda_1, \lambda_2) = \mathsf{eigenvalues}(\boldsymbol{M}_{ac}), \quad \lambda_3 = N_{\mathrm{mult},m}^k \quad \text{and} \quad \lambda_4 = n^k$$

where $\boldsymbol{M}_{ac}$ is the top left 2×2 block matrix of $\boldsymbol{M}$ *i.e.*

$$\boldsymbol{M}_{ac} = \begin{pmatrix} N_{\mathrm{add},a} & N_{\mathrm{copy},a} \\ N_{\mathrm{add},c} & N_{\mathrm{copy},c} \end{pmatrix} .$$

We finally get

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot N_{\max}^k\big) \quad \text{with} \quad N_{\max} = \max(\mathsf{eigenvalues}(\boldsymbol{M}_{ac}), N_{\mathrm{mult},m}) . \quad (11)$$

In order to reach some security level $\varepsilon = 2^{-\kappa}$ for some target security parameter κ and assuming that we have a security expansion $p \to f^{(k)}(p)$, the expansion level k must be chosen so that $f^{(k)}(p) \leq 2^{-\kappa}$. In practice, the function f is of the form

$$f : p \mapsto \sum_{i \geq d} c_i \, p^i \leq (c_d + \mathcal{O}(p)) \, p^d .$$

where $\mathcal{O}(p)$ is to be interpreted as p tends to 0. In the rest of this paper, we shall say that such a function has *amplification order* d.

The upper bound $f(p) \leq c'_d p^d$ with $c'_d = c_d + \mathcal{O}(p)$ implies $f^{(k)}(p) < (c'_d p)^{d^k}$. Hence, to satisfy the required security $f^{(k)}(p) \leq 2^{-\kappa}$ while assuming $c'_d \, p < 1$, the number k of expansions must satisfy:

$$k \geqslant \log_d(\kappa) - \log_d(-\log_2(c'_d \, p)) .$$

We can then rewrite (11) as

$$|\widehat{C}| = \mathcal{O}\big(|C| \cdot \kappa^e\big) \quad \text{with} \quad e = \frac{\log N_{\max}}{\log d} . \quad (12)$$

5.3 Random Probing Expandability

In the evaluation of random probing composability, let us recall that the failure event in the simulation of a gadget means that more that t shares from one of its inputs are necessary to complete a perfect simulation. For a gadget to be expandable we need slightly stronger notions than random probing composability. As first requirement, a two-input gadget should have a failure probability which is independent for each input. This is because in the base case, each wire as input of a gate leaks independently. On the other hand, in case of failure event in the child gadget, the overall simulator should be able to produce a perfect simulation of the full output (that is the full input for which the failure occurs). To do so, the overall simulator is given the clear output (which is obtained from the simulation of the base case) plus any set of $n-1$ output shares. This means that whenever the set J is of cardinal greater than t, the gadget simulator can replace it by any set J' of cardinal $n-1$.

Definition 9 (Random Probing Expandability). *Let $f : \mathbb{R} \to \mathbb{R}$. An n-share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ is* (t, f)-random probing expandable (RPE) *if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} &\mathcal{W} \leftarrow \mathsf{LeakingWires}(G, p) \\ &(I_1, I_2, J') \leftarrow \mathsf{Sim}_1^G(\mathcal{W}, J) \\ &out \leftarrow \mathsf{Sim}_2^G(\mathcal{W}, J', \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events $\mathcal{F}_1 \equiv \big(|I_1| > t\big)$ and $\mathcal{F}_2 \equiv \big(|I_2| > t\big)$ verify*

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \tag{13}$$

   with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),
2. *J' is such that $J' = J$ if $|J| \leq t$ and $J' \subseteq [n]$ with $|J'| = n - 1$ otherwise,*
3. *the output distribution satisfies*

$$out \overset{id}{=} \big(\mathsf{AssignWires}(G, \mathcal{W}, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_{J'}\big) \tag{14}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$.

The RPE notion can be simply extended to gadgets with 2 outputs: the Sim_1^G simulator takes two sets $J_1 \subseteq [n]$ and $J_2 \subseteq [n]$ as input and produces two sets J_1' and J_2' satisfying the same property as J' in the above definition (w.r.t. J_1 and J_2). The Sim_2^G simulator must then produce an output including $\widehat{z}_1|_{J_1'}$ and $\widehat{z}_2|_{J_1'}$ where $\widehat{z}_1$ and $\widehat{z}_2$ are the output sharings. The RPE notion can also be simply extended to gadgets with a single input: the Sim_1^G simulator produces a single set I so that the failure event $(|I| > t)$ occurs with probability lower than ε (and the Sim_2^G simulator is then simply given $\widehat{x}|_I$ where $\widehat{x}$ is the single input sharing). For the sake of completeness, and since we only focus in $2 \to 1$ and $1 \to 2$ gadgets in this paper, the RPE definition for the $1 \to 2$ case is given in the full version of this paper.

It is not hard to check that the above expandability notion is stronger that the composability notion introduced in Sect. 4. Formally, we have the following reduction:

Proposition 2. *Let $f = \mathbb{R} \to \mathbb{R}$ and $n \in \mathbb{N}$. Let G be an n-share gadget. If G is (t, f)-RPE then G is (t, f')-RPC, with $f'(\cdot) = 2 \cdot f(\cdot)$.*

Proof. We consider a (t, f)-RPE n-share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$. The $(t, 2 \cdot f)$-random composability property is directly implied by the (t, f)-random probing expandability by making use of the exact same simulators and observing that

$$\Pr\big((|I_1| > t) \vee (|I_2| > t)\big) \leq \Pr(|I_1| > t) + \Pr(|I_2| > t) = 2 \cdot \varepsilon.$$

The case of $1 \to 2$ gadgets is even more direct. □

5.4 Expansion Security

Definition 9 of random probing expandability is valid for base gadgets. For level-k gadgets $G^{(k)} = \mathsf{CC}^{(k-1)}(G)$ where $G \in \{G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}\}$ is a base gadget, we provide a generalized definition of random probing expandability.

Adequate subsets of $[n^k]$. We first define the notion of "adequate" subsets of $[n^k]$, instead of only bounded subsets. For this we define recursively a family $S_k \in \mathcal{P}([n^k])$, where $\mathcal{P}([n^k])$ denotes the set of all subsets of $[n^k]$, as follows:

$$S_1 = \{I \in [n],\ |I| \leq t\}$$
$$S_k = \{(I_1, \ldots, I_n) \in (S_{k-1} \cup [n^{k-1}])^n, I_j \in S_{k-1}\ \forall\ j \in [1,n] \text{ except at most } t\}$$

In other words, a subset I belongs to S_k if among the n subset parts of I, at most t of them are full, while the other ones recursively belong to S_{k-1}; see the full version for an illustration with $n = 3$ and $t = 1$.

Generalized definition of random probing expandability. We generalize Definition 9 as follows. At level k the input sets I_1 and I_2 must belong to S_k, otherwise we have a failure event. As in Definition 9, the simulation is performed for an output subset J' with $J' = J$ if $J \in S_k$, otherwise $J' = [n^k] \setminus \{j^\star\}$ for some $j^\star \in [n^k]$.

Definition 10 (Random Probing Expandability with $\{S_k\}_{k\in\mathbb{N}}$). *Let $f : \mathbb{R} \to \mathbb{R}$ and $k \in \mathbb{N}$. An n^k-share gadget $G : \mathbb{K}^{n^k} \times \mathbb{K}^{n^k} \to \mathbb{K}^{n^k}$ is* (S_k, f)-random probing expandable (RPE) *if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^{n^k} \times \mathbb{K}^{n^k}$, for every set $J \in S_k \cup [n^k]$ and for every $p \in [0,1]$, the random experiment*

$$\mathcal{W} \leftarrow \mathsf{LeakingWires}(G, p)$$
$$(I_1, I_2, J') \leftarrow \mathsf{Sim}_1^G(\mathcal{W}, J)$$
$$out \leftarrow \mathsf{Sim}_2^G(\mathcal{W}, J', \widehat{x}|_{I_1}, \widehat{y}|_{I_2})$$

ensures that

1. *the failure events $\mathcal{F}_1 \equiv \big(I_1 \notin S_k\big)$ and $\mathcal{F}_2 \equiv \big(I_2 \notin S_k\big)$ verify*

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \tag{15}$$

   with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),
2. *the set J' is such that $J' = J$ if $J \in S_k$, and $J' = [n^k] \setminus \{j^\star\}$ for some $j^\star \in [n^k]$ otherwise,*
3. *the output distribution satisfies*

$$out \overset{id}{=} \big(\mathsf{AssignWires}(G, \mathcal{W}, (\widehat{x}, \widehat{y}))\ ,\ \widehat{z}|_{J'}\big) \tag{16}$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$.

The notion of random probing expandability from Definition 10 naturally leads to the statement of our main theorem; the proof is given in the full version.

Theorem 2. *Let $n \in \mathbb{N}$ and $f : \mathbb{R} \to \mathbb{R}$. Let G_{add}, G_{mult}, G_{copy} be n-share gadgets for the addition, multiplication and copy on $\mathbb{K}$. Let CC be the standard circuit compiler with sharing order n and base gadgets G_{add}, G_{mult}, G_{copy}. Let $\mathsf{CC}^{(k)}$ be the expanding circuit compiler with base compiler CC. If the base gadgets G_{add}, G_{mult} and G_{copy} are (t, f)-RPE then, $G^{(k)}_{add} = \mathsf{CC}^{(k-1)}(G_{add})$, $G^{(k)}_{mult} = \mathsf{CC}^{(k-1)}(G_{mult})$, $G^{(k)}_{copy} = \mathsf{CC}^{(k-1)}(G_{copy})$ are $(S_k, f^{(k)})$-RPE, n^k-share gadgets for the addition, multiplication and copy on $\mathbb{K}$.*

The random probing security of the expanding circuit compiler can then be deduced as a corollary of the above theorem together with Proposition 2 (RPE $\Rightarrow$ RPC reduction) and Theorem 1 (composition theorem).

Corollary 1. *Let $n \in \mathbb{N}$ and $f : \mathbb{R} \to \mathbb{R}$. Let G_{add}, G_{mult}, G_{copy} be n-share gadgets for the addition, multiplication and copy on $\mathbb{K}$. Let CC be the standard circuit compiler with sharing order n and base gadgets G_{add}, G_{mult}, G_{copy}. Let $\mathsf{CC}^{(k)}$ be the expanding circuit compiler with base compiler CC. If the base gadgets G_{add}, G_{mult} and G_{copy} are (t, f)-RPE then $\mathsf{CC}^{(k)}$ is $(p, 2 \cdot f^{(k)}(p))$-random probing secure.*

5.5 Relaxing the Expandability Notion

The requirement of the RPE property that the failure events $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent might seem too strong. In practice it might be easier to show or verify that some gadgets satisfy a weaker notion. We say that a gadget is (t, f)-*weak random probing expandable* (wRPE) if the failure events verify $\Pr(\mathcal{F}_1) \leq \varepsilon$, $\Pr(\mathcal{F}_2) \leq \varepsilon$ and $\Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) \leq \varepsilon^2$ instead of (13) in Definition 9. Although being easier to achieve and to verify this notion is actually not much weaker as the original RPE. We have the following reduction of RPE to wRPE; see the full version for the proof.

Proposition 3. *Let $f = \mathbb{R} \to [0, 0.14]$. Let $G : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n$ be an n-share gadget. If G is (t, f)-wRPE then G is (t, f')-RPE with $f'(\cdot) = f(\cdot) + \frac{3}{2} f(\cdot)^2$.*

Assume that we can show or verify that a gadget is wRPE with the following failure event probabilities

$$\Pr(\mathcal{F}_1) = f_1(p) \ , \quad \Pr(\mathcal{F}_2) = f_2(p) \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = f_{12}(p) \ ,$$

for every $p \in [0, 1]$. Then the above proposition implies that the gadget is (p, f)-RPE with

$$f : p \mapsto f_{\max}(p) + \frac{3}{2} f_{\max}(p)^2 \quad \text{with} \quad f_{\max} = \max(f_1, f_2, \sqrt{f_{12}}) \ .$$

We shall base our verification of the RPE property on the above equation as we describe hereafter.

5.6 Verification of Gadget Expandability

We can easily adapt our automatic tool to verify the weak random probing expandability for base gadgets (Definition 9). Basically, the verification is split into two steps that we first describe for the case of addition and multiplication gadgets with fan-in 2 and fan-out 1.

In a first step, our tool computes the function f to check the (t, f)-wRPE property for output sets of shares of cardinal at most t. For 2-input gadgets, this step leads to the computation of coefficients c_i corresponding to three failure events $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_1 \wedge \mathcal{F}_2$ as defined above but restricted to output sets of shares of cardinal less than t. The process is very similar to the verification of random probing composability but requires to separate the failure events counter into failure events for the first input ($|\mathcal{I}_1| > t$), for the second input ($|\mathcal{I}_2| > t$) or for both ($(|\mathcal{I}_1| > t) \wedge (|\mathcal{I}_2| > t)$). In the following, we denote the three functions formed from the corresponding coefficients as $f_1^{(1)}$, $f_2^{(1)}$, and $f_{12}^{(1)}$.

Then, in a second step, our tool verifies that there exists at least one set of $n-1$ shares for each output, such that the simulation failure is limited by $f(p)$ for some probability $p \in [0, 1]$. In that case, it still loops on the possible output sets of shares (of cardinal $n-1$) but instead of computing the maximum coefficients, it determines whether the simulation succeeds for at least one of such sets. A failure event is recorded for a given tuple if no output sets of cardinal $n-1$ can be simulated together with this tuple from at most t shares of each input. As for the first verification step, we record the resulting coefficients for the three failure events to obtain functions $f_1^{(2)}$, $f_2^{(2)}$, and $f_{12}^{(2)}$.

From these two steps, we can deduce f such that the gadget is (t, f)-wRPE:

$$\forall p \in [0, 1], f(p) = \max(f_1(p), f_2(p), \sqrt{f_{12}(p)})$$

with

$$f_\alpha(p) = \max(f_\alpha^{(1)}(p), f_\alpha^{(2)}(p)) \qquad \text{for} \quad \alpha \in \{1, 2, 12\}$$

The computation of f for a gadget to satisfy (t, f)-weak random probing expandability is a bit trickier for copy gadgets which produce two outputs. Instead of two verification steps considering both possible ranges of cardinals for the output set of shares J, we need to consider four scenarios for the two possible features for output sets of shares J_1 and J_2. In a nutshell, the idea is to follow the first verification step described above when both J_1 and J_2 have cardinal equal or less than t and to follow the second verification step described above when both J_1 and J_2 have greater cardinals. This leads to functions $f^{(1)}$ and $f^{(2)}$. Then, two extra cases are to be considered, namely when ($|J_1| \leq t$) and ($|J_2| > t$) and the reverse when ($|J_1| > t$) and ($|J_2| \leq t$). To handle these scenarios, our tool loops over the output sets of shares of cardinal equal or less than t for the first output, and it determines whether there exists a set of $n-1$ shares of the second output that a simulator can perfectly simulate with the leaking wires and the former set. This leads to function $f^{(12)}$ and reversely to

function $f^{(21)}$. From these four verification steps, we can deduce f such that the copy gadget is (t, f)-wRPE:

$$\forall p \in [0,1], f(p) = \max(f^{(1)}(p), f^{(2)}(p), f^{(12)}(p), f^{(21)}(p)).$$

Once gadgets have been proven (t, f)-weak RPE, they are also proven to be (t, f')-RPE from Proposition 3 with $f' : p \mapsto f(p) + \frac{3}{2}f(p)^2$. Examples of such computations for 3-share gadgets are provided in Sect. 6.

6 New Constructions

In this section, we exhibit and analyze $(1, f)$-wRPE gadgets for the addition, multiplication, and copy (on any base field $\mathbb{K}$) to instantiate the expanding circuit compiler. These gadgets are sound in the sense that their function f has amplification order strictly greater than one. As explained in previous sections, an amplification order strictly greater than one guarantees that there exists a probability $p_{max} \in [0,1]$ such that $\forall p \leq p_{max}, f(p) \leq p$, which is necessary to benefit from the expansion. For 2-input gadgets, f is defined as the maximum between f_1, f_2, and $\sqrt{f_{12}}$. Therefore, the constraint on the amplification order also applies to the functions f_1, f_2, and $\sqrt{f_{12}}$. For the function f_{12}, this means that the amplification order should be strictly greater than two.

We start hereafter with an impossibility result, namely there are no (2-share, 2-to-1) $(1, f)$-RPE gadgets such that f has an amplification order greater than one. Then, we provide concrete instantiations of addition, multiplication, and copy gadgets based on 3 shares which successfully achieve $(1, f)$-RPE for amplification order greater than one and can be used in the expansion compiler.

6.1 About 2-Share Gadgets

Consider a gadget G with a 2-share single output $\boldsymbol{z} = (z_0, z_1)$ and two 2-share inputs $\boldsymbol{x} = (x_0, x_1)$ and $\boldsymbol{y} = (y_0, y_1)$. We reasonably assume that the latter are the outputs of gates with fan-in at most two (and not direct input shares). For G to be $(1, f)$-RPE with f of amplification order strictly greater than one, then f_{12} must be of amplification strictly greater than two. In other words, we should be able to exhibit a simulator such that one share of each input is enough to simulate anyone of the output shares and an arbitrary couple of leaking wires. But the output wire z_0 and both input gates of the second output share z_1 represent the full output and require the knowledge of both inputs to be simulated. Therefore, f_{12} has a non-zero coefficient in p and is thus not of amplification order strictly greater than two. We thus restrict our investigation to n-share gadgets, with $n \geq 3$ to instantiate our compiler.

In the upcoming gadget descriptions, notice that variables r_i are fresh random values, operations are processed with the usual priority rules, and the number of implicit copy gates can be deduced from the occurrences of each intermediate variable such that n occurrences require $n-1$ implicit copy gates. Also, the function expression below each gadget corresponds to the function obtained from our

verification tool when verifying weak random probing expandability. It implies that the gadget is (t, f)-wRPE for t usually equal to one except when defined otherwise. A more complete description of each function (with more coefficients) is available in the full version of this paper.

6.2 Addition Gadgets

The most classical masked addition schemes are sharewise additions which satisfy the simpler probing security property. Basically, given two input n-sharings $\boldsymbol{x}$ and $\boldsymbol{y}$, such an addition computes the output n-sharing $\boldsymbol{z}$ as $z_1 \leftarrow x_1 + y_1$, $z_2 \leftarrow x_2 + y_2, \ldots, z_n \leftarrow x_n + y_n$. Unfortunately, such elementary gadgets do not work in our setting. Namely consider an output set of shares J of cardinality t. Then, for any n, there exists sets $\mathcal{W}$ of leaking wires of cardinality one such that no set I of cardinality $\leq t$ can point to input shares that are enough to simulate both the leaking wire and the output shares of indexes in J. For instance, given a set $J = \{1, \ldots, t\}$, if $\mathcal{W}$ contains x_{t+1}, then no set I of cardinal $\leq t$ can define a set of input shares from which we can simulate both the leaking wire and $z_1, \ldots, z_t$. Indeed, each z_i for $1 \leq i \leq t$ requires both input shares x_i and y_i for its simulation. Thus, a simulation set I would contain at least $\{1, \ldots, t\}$ and $t + 1$ for the simulation of the leaking wire. I would thus be of cardinal $t + 1$ which represents a failure event in the random probing expandability definition. As a consequence, such a n-share addition gadget could only be (t, f)-RPE with f with a first coefficient c_1 as defined in Sect. 3 strictly positive. In other words, f would be of amplification order one such that $\forall p \in [0, 1]$, $f(p) \geq p$.

In the following, we introduce two 3-share addition gadgets. From our automatic tool, both are $(1, f)$-wRPE with f of amplification order strictly greater than one. Basically, in our first addition gadget G^1_{add}, both inputs are first refreshed with a circular refreshing gadget as originally introduced in [6]:

$$
\begin{aligned}
G^1_{\text{add}} : z_0 &\leftarrow x_0 + r_0 + r_1 + y_0 + r_3 + r_4 \\
z_1 &\leftarrow x_1 + r_1 + r_2 + y_1 + r_4 + r_5 \qquad\qquad f_{max}(p) = \sqrt{10}p^{3/2} + \mathcal{O}(p^2) \\
z_2 &\leftarrow x_2 + r_2 + r_0 + y_2 + r_5 + r_3
\end{aligned}
$$

The second addition gadget G^2_{add} simply rearranges the order of the refreshing variables:

$$
\begin{aligned}
G^2_{\text{add}} : z_0 &\leftarrow x_0 + r_0 + r_4 + y_0 + r_1 + r_3 \\
z_1 &\leftarrow x_1 + r_1 + r_5 + y_1 + r_2 + r_4 \qquad\qquad f_{max}(p) = \sqrt{69}p^2 + \mathcal{O}(p^3) \\
z_2 &\leftarrow x_2 + r_2 + r_3 + y_2 + r_0 + r_5
\end{aligned}
$$

In each gadget, $\boldsymbol{x}$ and $\boldsymbol{y}$ are the input sharings and $\boldsymbol{z}$ the output sharing; f_{max} additionally reports the maximum of the first non zero coefficient (as defined in Sect. 3) of the three functions f_1, f_2, and f_{12}, as defined in the previous section, obtained for the random probing expandability automatic verifications. A further definition of these functions can be found in the full version of this paper. Note that both gadgets G^1_{add} and G^2_{add} are built with 15 addition gates and 6 implicit copy gates.

6.3 Multiplication Gadget

We start by proving an impossibility result: no 3-share multiplication gadget composed of direct products between input shares satisfies $(1, f)$-RPE with amplification order strictly greater than one. Consider such a gadget G with two 3-input sharings $\boldsymbol{x}$ and $\boldsymbol{y}$ whose shares are directly multiplied together. Let $(x_i \cdot y_j)$ and $(x_k \cdot y_\ell)$ be two such products such that $i, j, k, \ell \in [3]$ and $i \neq k$ and $j \neq \ell$. If both results are leaking, then the leakage can only be simulated using the four input shares. Namely, $\{i, k\} \subseteq I_1$ and $\{j, \ell\} \subseteq I_2$. This scenario represents a failure since cardinals of I_1 and I_2 are both strictly greater than one. As a consequence, function f_{12} which records the failures for both inputs is defined with a coefficient c_2 at least equal to one. Hence f_{12} is not of amplification greater than two and f cannot be of amplification order greater than one. Regular 3-share multiplication gadgets consequently cannot be used as base gadgets of our compiler.

To circumvent this issue, we build a 3-share multiplication gadget G^1_{mult} whose both inputs are first refreshed, before any multiplication is performed:

$$\begin{aligned} u_0 &\leftarrow x_0 + r_5 + r_6; & u_1 &\leftarrow x_1 + r_6 + r_7; & u_2 &\leftarrow x_2 + r_7 + r_5 \\ v_0 &\leftarrow y_0 + r_8 + r_9; & v_1 &\leftarrow y_1 + r_9 + r_{10}; & v_2 &\leftarrow y_2 + r_{10} + r_8 \end{aligned}$$

$$\begin{aligned} z_0 &\leftarrow (u_0 \cdot v_0 + r_0) + (u_0 \cdot v_1 + r_1) + (u_0 \cdot v_2 + r_2) \\ z_1 &\leftarrow (u_1 \cdot v_0 + r_1) + (u_1 \cdot v_1 + r_4) + (u_1 \cdot v_2 + r_3) \\ z_2 &\leftarrow (u_2 \cdot v_0 + r_2) + (u_2 \cdot v_1 + r_3) + (u_2 \cdot v_2 + r_0) + r_4 \end{aligned}$$

$$f_{max}(p) = \sqrt{83}p^{3/2} + \mathcal{O}(p^2)$$

6.4 Copy Gadget

We exhibit a 3-share $(1, f)$-wRPE copy gadget G^1_{copy} with f of amplification order strictly greater than one:

$$\begin{aligned} v_0 &\leftarrow u_0 + r_0 + r_1; \; w_0 \leftarrow u_0 + r_3 + r_4 \\ v_1 &\leftarrow u_1 + r_1 + r_2; \; w_1 \leftarrow u_1 + r_4 + r_5 \qquad\qquad f_{max}(p) = 33p^2 + \mathcal{O}(p^3) \\ v_2 &\leftarrow u_2 + r_2 + r_0; \; w_2 \leftarrow u_2 + r_5 + r_3 \end{aligned}$$

It simply relies on two calls of the circular refreshing from [6] on the input. This last gadget is made of 6 addition gates and 9 implicit copy gates.

6.5 Complexity and Tolerated Probability

Following the asymptotic analysis of Sect. 5.2, our construction yields the following instantiation of the matrix $\boldsymbol{M}$

$$\boldsymbol{M} = \begin{pmatrix} 15 & 12 & 28 & 0 \\ 6 & 9 & 23 & 0 \\ 0 & 0 & 9 & 0 \\ 6 & 6 & 11 & 3 \end{pmatrix} \tag{17}$$

with

$$\boldsymbol{M}_{ac} = \begin{pmatrix} 15 & 12 \\ 6 & 9 \end{pmatrix} \quad \text{and} \quad N_{\text{mult},m} = 9 \ .$$

The eigenvalues of $\boldsymbol{M}_{ac}$ are 3 and 21, which gives $N_{\max} = 21$. We also have a random probing expandability with function f of amplification order $d = \frac{3}{2}$. Hence we get

$$e = \frac{\log N_{\max}}{\log d} = \frac{\log 21}{\log 1.5} \approx 7.5$$

which gives a complexity of $|\widehat{C}| = \mathcal{O}\big(|C| \cdot \kappa^{7.5}\big)$. Finally, it can be checked from the coefficients of the RPE functions given in the full version of this paper that our construction tolerates a leakage probability up to

$$p_{\max} \approx 0.0045 > 2^{-8} \ .$$

This corresponds to the maximum value p for which we have $f(p) < p$ which is a necessary and sufficient condition for the expansion strategy to apply with (t, f)-RPE gadgets.

The full version of this paper displays the new gate count vectors for each of the compiled gadgets $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$ by computing the matrix $\boldsymbol{M}^k$ and plots the values taken by the function f such that the base gadgets are (t, f)-RPE. For instance, for level $k = 9$ the number of gates in the compiled gadgets is around 10^{12} and assuming a leakage probability of $p = 0.0045$ (which is the maximum we can tolerate), we achieve a security of $\varepsilon \approx 2^{-76}$.

7 Comparison with Previous Constructions

In this section, we compare our scheme to previous constructions. Specifically, we first compare it to the well-known Ishai-Sahai-Wagner (ISW) construction and discuss the instantiation of our scheme from the ISW multiplication gadget. Then we exhibit the asymptotic complexity (and tolerated leakage probability) of the Ananth-Ishai-Sahai compiler and compare their results to our instantiation.

7.1 Comparison with ISW

The classical ISW construction [18] is secure in the t-probing model when the adversary can learn any set of t intermediate variables in the circuit, for $n = 2t + 1$ shares. This can be extended to t probes per gadget, where each gadget corresponds to a AND or XOR gate in the original circuit. Using Chernoff bound, security in the t-probing model per gadget implies security in the p-random probing model, where each wire leaks with probability p, with $p = \mathcal{O}(t/|G|)$, where $|G|$ is the gadget size. Since in ISW each gadget has complexity $\mathcal{O}(t^2)$, this gives $p = \mathcal{O}(1/t)$. Therefore, in the p-random probing model, the ISW construction is only secure against a leakage probability $p = \mathcal{O}(1/n)$, where the number of shares n must grow linearly with the security parameter κ in order to achieve security $2^{-\kappa}$. This means that ISW does not achieve security under a

constant leakage probability p; this explains why ISW is actually vulnerable to horizontal attacks [7], in which the adversary can combine information from a constant fraction of the wires.

ISW-based instantiation of the expanding compiler. In our instantiation, we choose to construct a new 3-share multiplication gadget instead of using the ISW multiplication gadget from [18]. In fact, ISW first performs a direct product of the secret shares before adding some randomness, while we proved in Sect. 6 that no such 3-share multiplication gadget made of direct products could satisfy $(1, f)$-RPE with amplification order strictly greater than one. Therefore the ISW gadget is not adapted for our construction with 3 shares.

Table 1 displays the output of our tool when run on the ISW gadget for up to 7 shares with different values for t. It can be seen that an amplification order strictly greater than one is only achieved for $t > 1$, with 4 or more shares. And an order of 3/2 is only achieved with a minimum of 4 shares for $t = 2$, whereas we already reached this order with our 3-share construction for $t = 1$. If we use the 4-share ISW gadget with appropriate 4-share addition and copy gadgets instead of our instantiation, the overall complexity of the compiler would be greater, while the amplification order would remain the same, and the tolerated leakage probability would be worse (recall that our instantiation tolerates a maximum leakage probability $p \approx 2^{-8}$, while 4-share ISW tolerates $p \approx 2^{-9.83}$). Clearly, the complexity of the 4-share ISW gadget $(N_a, N_c, N_m, N_r) = (24, 30, 16, 6)$ is higher than that of our 3-share multiplication gadget $(N_a, N_c, N_m, N_r) = (28, 23, 9, 11)$. In addition, using 3-share addition and copy gadgets (as in our case) provides better complexity than 4-share gadgets. Hence to reach an amplification order of 3/2, a 4-share construction with the ISW gadget would be more complex and would offer a lower tolerated leakage probability.

For higher amplification orders, the ISW gadgets with more than 4 shares or other gadgets can be studied. This is a open construction problem as many gadgets can achieve different amplification orders and be globally compared.

7.2 Complexity of the Ananth-Ishai-Sahai Compiler

The work from [2] provides a construction of circuit compiler (the AIS compiler) based on the expansion strategy described in Sect. 5 with a (p, ε)-composable security property, analogous to our (t, f)-RPE property. To this purpose, the authors use an (m, c)-multi-party computation (MPC) protocol Π. Such a protocol allows to securely compute a functionality shared among m parties and tolerating at most c corruptions. In a nutshell, their composable circuit compiler consists of multiple layers: the bottom layer replaces each gate in the circuit by a circuit computing the (m, c)-MPC protocol for the corresponding functionality (either Boolean addition, Boolean multiplication, or copy). The next $k-1$ above layers apply the same strategy recursively to each of the resulting gates. As this application can eventually have exponential complexity if applied to a whole circuit C directly, the top layer of compilation actually applies the k bottom layers

Table 1. Complexity, amplification order and maximum tolerated leakage probability of the ISW multiplication gadgets. Some leakage probabilities were not computed accurately by VRAPS for performances reasons. An interval on these probabilities is instead given by evaluating lower and upper bound functions f_{inf} and f_{sup} of $f(p)$.

# shares	Complexity $(N_{\text{a}}, N_{\text{c}}, N_{\text{m}}, N_{\text{r}})$	t	Amplification order	$\log_2$ of maximum tolerated leakage probability
3	(12, 15, 9, 3)	1	1	–
4	(24, 30, 16, 6)	1	1	–
		2	3/2	−9.83
5	(40, 50, 25, 10)	1	1	–
		2	3/2	−11.00
		3	2	−8.05
6	(60, 75, 36, 15)	1	1	–
		2	3/2	−13.00
		3	2	[−9.83, −7.87]
		4	2	[−9.83, −5.92]
7	(84, 105, 49, 21)	1	1	–
		2	3/2	[−16.00, −14.00]
		3	2	[−12.00, −7.87]
		4	5/2	[−12.00, −2.27]
		5	2	[−12.00, −3.12]

to each of the gates of C independently and then stitches the inputs and outputs using the correctness of the XOR-encoding property. Hence the complexity is in

$$\mathcal{O}(|C| \cdot N_{\text{g}}^k) \ , \tag{18}$$

where $|C|$ is the number of gates in the original circuit and N_{g} is the number of gates in the circuit computing Π. The authors of [2] prove that such compiler satisfies (p, ε)-composition security property, where p is the tolerated leakage probability and ε is the simulation failure probability. Precisely:

$$\varepsilon = N_{\text{g}}^{c+1} \cdot p^{c+1} \tag{19}$$

Equations (18) and (19) can be directly plugged into our asymptotic analysis of Sect. 5.2, with N_g replacing our $N_{\max}$ and where $c+1$ stands for our amplification order d. The obtained asymptotic complexity for the AIS compiler is

$$\mathcal{O}\big(|C| \cdot \kappa^e\big) \quad \text{with} \quad e = \frac{\log N_{\text{g}}}{\log c + 1} \ . \tag{20}$$

This is to be compared to $e = \frac{\log N_{\max}}{\log d}$ in our scheme. Moreover, this compiler can tolerate a leakage probability

$$p = \frac{1}{N_{\text{g}}^2} \ .$$

The authors provide an instantiation of their construction using an existing MPC protocol due to Maurer [20]. From their analysis, this protocol can be implemented with a circuit of $N_g = (4m - c) \cdot \left(\binom{m-1}{c}^2 + 2m\binom{m}{c}\right)$ gates. They instantiate their compiler with this protocol for parameters $m = 5$ parties and $c = 2$ corruptions, from which they get $N_g = 5712$. From this number of gates, they claim to tolerate a leakage probability $p = \frac{1}{5712^2} \approx 2^{-25}$ and our asymptotic analysis gives a complexity of $\mathcal{O}\big(|C| \cdot \kappa^e\big)$ with $e \approx 7.87$ according to (20). In the full version of this paper, we give a detailed analysis of the Maurer protocol [20] in the context of the AIS compiler instantiation. From our analysis, we get the following number of gates for the associated circuit:

$$N_g = (6m - 5) \cdot \left(\binom{m-1}{c}^2 + m(2k-2) + 2k^2\right) \quad \text{where} \quad k = \binom{m}{c}.$$

Using the parameters $m = 5$ and $c = 2$ from the AIS compiler instantiation [2], we get $N_g = 8150$. This yields a tolerated leakage probability of $p \approx 2^{-26}$ and an exponent $e = \log 8150 / \log 3 \approx 8.19$ in the asymptotic complexity $\mathcal{O}\big(|C| \cdot \kappa^e\big)$ of the AIS compiler.

These results are to be compared to the $p \approx 2^{-8}$ and $e \approx 7.5$ achieved by our construction. In either case ($N_g = 5712$ as claimed in [2] or $N_g = 8150$ according to our analysis), our construction achieves a slightly better complexity while tolerating a much higher leakage probability. We stress that further instantiations of the AIS scheme (based on different MPC protocols) or of our scheme (based on different gadgets) could lead to better asymptotic complexities and/or tolerated leakage probabilities. This is an interesting direction for further research.

8 Implementation Results

We developed an implementation in python of a compiler CC, that given three n-share gadgets $G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}$ and an expansion level k, outputs the compiled gadgets $G^{(k)}_{\text{add}}$, $G^{(k)}_{\text{copy}}$, $G^{(k)}_{\text{mult}}$, each as a C function. The source code (with an example of AES implementation) is publicly available at:

https://github.com/CryptoExperts/poc-expanding-compiler

The variables' type is given as a command line argument. Table 2 shows the complexity of the compiled gadgets from Sect. 6 using the compiler with several expansion levels k, as well as their execution time in milliseconds when run in C on randomly generated 8-bit integers. All implementations were run on a laptop computer (Intel(R) Core(TM) i7-8550U CPU, 1.80GHz with 4 cores) using Ubuntu operating system and various C, python and sage libraries. For the generation of random variables, we consider that an efficient external random number generator is available in practice, and so we simply use the values of an incremented counter variable to simulate random gates.

Table 2. Complexity and execution time (in ms, on an Intel i7-8550U CPU) for compiled gadgets $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$ from Sect. 6 implemented in C.

k	# shares	Gadget	Complexity $(N_{\text{a}}, N_{\text{c}}, N_{\text{m}}, N_{\text{r}})$	Execution time
1	3	$G_{\text{add}}^{2(1)}$	(15, 6, 0, 6)	$1,69.10^{-4}$
		$G_{\text{copy}}^{1(1)}$	(12, 9, 0, 6)	$1,67.10^{-4}$
		$G_{\text{mult}}^{1(1)}$	(28, 23, 9, 11)	$5,67.10^{-4}$
2	9	$G_{\text{add}}^{2(2)}$	(297, 144, 0, 144)	$2,21.10^{-3}$
		$G_{\text{copy}}^{1(2)}$	(288, 153, 0, 144)	$2,07.10^{-3}$
		$G_{\text{mult}}^{1(2)}$	(948, 582, 81, 438)	$9,91.10^{-3}$
3	27	$G_{\text{add}}^{2(3)}$	(6183, 3078, 0, 3078)	$9,29.10^{-2}$
		$G_{\text{copy}}^{1(3)}$	(6156, 3105, 0, 3078)	$9,84.10^{-2}$
		$G_{\text{mult}}^{1(3)}$	(23472, 12789, 729, 11385)	$3,67.10^{-1}$

It can be observed that both the complexity and running time grow by almost the same factor with the expansion level, with multiplication gadgets being the slowest as expected. Base gadgets with $k = 1$ roughly take 10^{-4} ms, while these gadgets expanded 2 times ($k = 3$) take between 10^{-2} and 10^{-1} ms. The difference between the linear cost of addition and copy gadgets, and the quadratic cost of multiplication gadgets can also be observed through the gadgets' complexities.

References

1. Ajtai, M.: Secure computation with information leaking to an adversary. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd Annual ACM Symposium on Theory of Computing, San Jose, CA, USA, 6–8 June 2011, pp. 715–724. ACM Press (2011)
2. Ananth, P., Ishai, Y., Sahai, A.: Private circuits: a modular approach. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 427–455. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_15
3. Andrychowicz, M., Dziembowski, S., Faust, S.: Circuit compilers with $O(1/\log(n))$ leakage rate. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 586–615. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_21
4. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.-A., Grégoire, B., Strub, P.-Y.: Verified proofs of higher-order masking. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 457–485. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_18
5. Barthe, G., et al.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S., (eds.) ACM CCS 2016: 23rd Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016, pp. 116–129. ACM Press (2016)
6. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F.-X., Strub, P.-Y.: Parallel implementations of masking schemes and the bounded moment leakage model. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 535–566. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_19

7. Battistello, A., Coron, J.-S., Prouff, E., Zeitoun, R.: Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 23–39. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_2
8. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Private multiplication over finite fields. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 397–426. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63697-9_14
9. Belaïd, S., Dagand, P.É., Mercadier, D., Rivain, M., Wintersdorff, R.: Tornado: automatic generation of probing-secure masked bitsliced implementations. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 311–341. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_11
10. Belaïd, S., Goudarzi, D., Rivain, M.: Tight private circuits: achieving probing security with the least refreshing. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 343–372. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_12
11. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26
12. Cohen, G., et al.: Efficient multiparty protocols via log-depth threshold formulae - (extended abstract). In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 185–202. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_11
13. Coron, J.-S.: Formal verification of side-channel countermeasures via elementary circuit transformations. In: Preneel, B., Vercauteren, F. (eds.) ACNS 2018. LNCS, vol. 10892, pp. 65–82. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93387-0_4
14. Coron, J.-S., Prouff, E., Rivain, M., Roche, T.: Higher-order side channel security and mask refreshing. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 410–424. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43933-3_21
15. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: from probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_24
16. Goubin, L., Patarin, J.: DES and differential power analysis the "duplication" method. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_15
17. Hirt, M., Maurer, U.M.: Player simulation and general adversary structures in perfect multiparty computation. J. Cryptol. **13**(1), 31–60 (2000). https://doi.org/10.1007/s001459910003
18. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
19. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
20. Maurer, U.: Secure multi-party computation made simple (invited talk). In: Cimato, S., Persiano, G., Galdi, C. (eds.) SCN 2002. LNCS, vol. 2576, pp. 14–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36413-7_2

21. Micali, S., Reyzin, L.: Physically observable cryptography (extended abstract). In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24638-1_16
22. Prouff, E., Rivain, M.: Masking against side-channel attacks: a formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_9
23. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_28