New Techniques for Random Probing Security and Application to Raccoon Signature Scheme

Sonia Belaïd, Matthieu Rivain, and Mélissa Rossi*

CryptoExperts, Paris, France
first.last@cryptoexperts.com

Abstract. The random probing model formalizes a leakage scenario where each wire in a circuit leaks with probability p. This model holds practical relevance due to its reduction to the noisy leakage model, which is widely regarded as the appropriate formalization for power and electromagnetic side-channel attacks.

In this paper, we present new techniques for designing efficient masking schemes that achieve tighter random probing security with lower complexity. First, we introduce the notion of *cardinal random probing composability* (Cardinal-RPC), offering a new trade-off between complexity and security for composing masking gadgets. Next, we propose a novel refresh technique based on a simple iterative process: randomly selecting and updating two shares with fresh randomness. While not perfectly secure in the standard probing model, this method achieves arbitrary cardinal-RPC security, making it a versatile tool for constructing random-probing secure circuits. Using this refresh, we develop additional basic gadgets (e.g., linear multiplication, addition, and copy) that satisfy the cardinal-RPC notion. Despite the increased complexity, the gains in security significantly outweigh the overhead, with the number of iterations offering useful flexibility.

To showcase our techniques, we apply them to lattice-based signatures. Specifically, we introduce a new random-probing composable gadget for sampling small noise, a key component in various post-quantum algorithms. To assess security in this context, we generalize the random probing security model to address auxiliary inputs and public outputs. We apply our findings to Raccoon, a masking-friendly signature scheme originally designed for standard probing security. We prove the secure composition of our new gadgets for key generation and signature computation, and show that our masking scheme achieves a superior security-performance tradeoff compared to previous approaches based on random probing expansion. To our knowledge, this is the first fully secure instantiation of a post-quantum algorithm in the random probing model.

Keywords: Post-quantum signature · Masking countermeasure · Random Probing Model · Raccoon Signature Scheme

1 Introduction

Most widely used cryptographic algorithms are considered secure against black-box attacks, where the adversary's knowledge is restricted to certain inputs and outputs. However, as discovered in the late 1990s, their implementations on physical devices can be vulnerable to more sophisticated side-channel attacks. These attacks leverage physical emanations from the device, such as execution time, temperature, power consumption, or electromagnetic emissions during the algorithm's execution. One of the most prevalent techniques to defend against side-channel attacks is masking, introduced independently in 1999 by Chari, Jutla, Rao, and Rohatgi [14], and by Goubin and Patarin [22]. In essence, the approach involves splitting each sensitive variable in an implementation into n shares, with $n-1$ generated uniformly at random and the final share derived from the original value and the previous ones. Masking ensures that an adversary cannot recover the secret without all the shares.

* This work was done while the author was working at Agence Nationale de Sécurité des Systèmes d'Informations, Paris, France.

The probing model. To prove the security of masking schemes, it is essential to model the side-channel leakage an attacker could exploit, which leads to the concept of leakage models. One of the most well-known models is the d-probing model, introduced by Ishai, Sahai, and Wagner in [24], which states that an attacker can gain access to the exact values of d intermediate variables, where d is a security parameter, but nothing more. The underlying intuition is that while all processed data may leak, combining noisy variables requires an exponential number of measurements as their number increases [14].

Most masking approaches consist in carefully composing small gadgets that are individually proven secure. This often necessitates using *refresh gadgets*, which functionally act as the identity function and help reduce share dependencies by introducing additional randomness. However, masking schemes secure in the d-probing model can become prohibitively expensive as the number of shares increases. While the complexity of linear operations scales linearly with the number of shares, the main bottleneck arises from non-linear operations, such as multiplying shared variables in $\mathbb{Z}_q$, which is common in post-quantum schemes. These operations significantly affect overall efficiency, complicating the balance between security and performance as the number of shares grows.

Masking-friendly post-quantum schemes. The use of masking in post-quantum NIST standards comes with a significant performance cost. While the slowdown remains manageable for the Crystals-Kyber key encapsulation mechanism [30], the impact on performance is much more severe for the Crystals-Dilithium signature scheme [26], as highlighted in [4,15]. For other schemes, such as Falcon [28], designing efficient and provable masking in the d-probing model presents even greater challenges. To tackle the security-performance challenge, recent efforts have focused on designing masking-friendly schemes that reconsider the unmasked design to limit the use of non-linear operations. These schemes instead primarily rely on gadgets that can be efficiently masked and provably secure within the d-probing model. A first attempt to modify Falcon's sampler in this purpose [21] was invalidated in [27]. A more successful approach has emerged with the design of Raccoon [16], a masking-friendly analog to Crystals-Dilithium. While Raccoon produces larger signature sizes, it offers a significant speedup—by orders of magnitude for masked implementation. Similarly, a masking-friendly version of Falcon has been introduced in [20]. Both designs abandon the use of complex Gaussian distributions in favor of a simpler noise distribution based on small uniform additions.

The stronger random probing model. Although these schemes have made progress in reconciling security and efficiency, the d-probing model, on which they rely, has raised concerns about its practical relevance [6]. The noisy leakage model, introduced by Prouff and Rivain [29] and inspired by [14], better reflects real-world devices by assuming all data leaks with noise. In 2014, Duc, Dziembowski, and Faust demonstrated that masking schemes that are secure under the conceptually-simpler *random probing model* are also secure against in the noisy leakage model [18].

In this model, each wire leaks with a constant probability p, reflecting side-channel noise in practice. A circuit is secure in this model if the leakage can be simulated without knowledge of the secret, with a negligible failure probability, as originally formalized in [1]. This model enables provable security bounds against attackers, similar to traditional black-box security proofs. Furthermore, the random probing model not only addresses probing attacks but also captures horizontal attacks [6], which exploit repeated manipulations of variables. Overall, this model provides significantly stronger theoretical security guarantees compared to the standard probing model. Several attempts have been made to design masking schemes secure in the random probing model [24,1,3,23,2,8]. However, most of these schemes are either impractical or fail to tolerate a constant leakage probability p. A notable breakthrough was made by Ananth, Ishai, and Sahai [2], who introduced an expansion strategy built upon secure multi-party computation protocols. This approach was later refined by Belaïd et al. [8,10,11], resulting in simpler *random probing expandable* gadgets that achieve arbitrary levels of random probing security for a fixed leakage probability p. Cassiers et al. further developed a tighter composition framework applied to the AES S-box in [13], though it struggles to scale to larger circuits and numbers of shares. In parallel, Berti, Faust, and Orlt [12] introduced a generic compiler based on leakage diagrams, by employing a refresh gadget derived from Dziembowski et al. [19]. While promising, this approach requires specific gadgets with carefully crafted composition properties to achieve efficient secure composition, making its generalization challenging. Jahandideh, Mennink, and Batina [25] proposed an algebraic approach to evaluate random probing security, but it relies on estimations and assumes the adversary targets a single secret, limiting its applicability. As a result, extending these solutions to general secure circuit constructions remains non-trivial. All in all, no concrete instantiations exist for post-quantum schemes, which are expected to become widely adopted in the near future.

Our contributions. The primary goal of this paper is to make random probing security more practical for real-world implementations. We introduce new techniques which we apply to the use case of lattice-based signatures. Specifically, we focus our study on Raccoon [16], whose masking-friendly structure makes it an ideal starting point. Our work addresses all the key steps necessary to achieve this goal, ranging from extending security definitions to designing elementary gadgets and providing a comprehensive composition proof. More specifically, the contributions of this paper are as follows:

1. We first extend existing random-probing security notions, which already include various flavors (*e.g.* for composition and expansion), by adding two key features:
   - We extend the random-probing security framework to handle auxiliary inputs in the circuit. Such inputs are not masked and must hence tolerate some amount of leakage. In Raccoon's masking proof, auxiliary inputs (a.k.a. "unshared inputs") capture the small uniform randomness used for sampling noise. Additionally, we consider the possibility to expose public outputs, such as the public key in the key generation algorithm and the signature in the signing algorithm, as proposed in [5] in the context of standard probing security. The extended random-probing security notions are presented in Section 3.1.
   - We generalize the notion of random probing composability (RPC). Instead of a fixed threshold on the number of input and output shares, we consider all possible cardinalities of leaked output shares and required input shares for the simulation. This refined notion, termed *cardinal-RPC*, is detailed in Section 3.2. We also combine the concepts of auxiliary inputs/public outputs with cardinal-RPC in Section 3.3.
2. A crucial step in designing random probing secure gadgets is defining a refreshing gadget that randomizes the shares before performing the target operation. One of our key contributions is the design of a new random probing-friendly refresh gadget. The concept is simple: start with an encoding of zero and, in each iteration, add and subtract a random value to two randomly selected shares. After a certain number of iterations, the input to be refreshed is share-wise added to the result. As the number of iterations increase, the "quality" of the refreshing improves until a threshold is reached. In Section 4.1, we quantify this quality by introducing the new intermediary notion of *random probing partitioned* (RPP). We use this to prove that our new refresh is cardinal-RPC, with a tunable security advantage depending on the number of iterations. Taking a step forward, we integrate our new refresh design and composability definition into the necessary basic gadgets: addition, copy and linear multiplication (*i.e.* multiplication by a public value). We prove the cardinal-RPC security of these gadgets. These designs and proofs are detailed in Section 4.2.
3. In Section 5, we introduce the noise generation gadget essential for Raccoon. This design leverages new elementary gadgets to create an algorithm that combines small uniform random samples for generating Raccoon's secrets. We employ two approaches for designing the secure gadget: (1) using *random probing expandable* (RPE) gadgets from [8,10,11] and (2) using our new cardinal-RPC composition framework and gadgets. This section demonstrates a significant enhancement in complexity and randomness consumption with our composition proof and new gadgets.
4. Finally, in Section 6, we combine all concepts and gadgets to present a random-probing-secure instantiation of Raccoon, using parameters from the original scheme [16]. We evaluate both scenarios: (1) using expansion and (2) applying our general composition proof. Our detailed complexity analysis demonstrates that our new gadgets significantly enhance efficiency while achieving high levels of security. For example, for the Raccoon-128-16 instance with a leakage probability of 2^{-24}, we achieve 128-bit probing security with a randomness consumption that is between 10 and 20 times lower than with the expansion strategy of [8] (the number of multiplications remaining similar in both approaches, while the number of additions scales with the randomness consumption).

All security and complexity results for our new proposal can be reproduced using the Python script available at https://github.com/CryptoExperts/EC25-random-probing-Raccoon.

2 Preliminaries

2.1 Notations

A probabilistic polynomial time algorithm (PPT) runs in time polynomial in the security parameter. Along the paper, $\mathbb{K}$ shall denote a finite field or a finite ring.[1] Sequences indexes are starting at 1. By convention, when the maximum index is 0, the sequence is empty, for example $x_1, \cdots, x_0 = \emptyset$. For some tuple $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{K}^n$ and for some set $I \subseteq [n]$, the tuple $(x_i)_{i \in I}$ is denoted $\mathbf{x}|_I$. For an integer n, we denote $\mathcal{P}(n)$ the set of partitions of n, which is defined as:

$$\mathcal{P}(n) = \left\{ (n_1, \ldots, n_\ell) \;\middle|\; \ell \in \mathbb{N} \;\wedge\; n_1 \geq n_2 \geq \cdots \geq n_\ell \;\wedge\; n_1 + \cdots + n_\ell = n \right\}.$$

Distributions. Two probability distributions, D_1 and D_2, are *ε-close*, denoted as $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε, namely

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

where $p_{D_1}(\cdot)$ and $p_{D_2}(\cdot)$ represent the probability mass functions of D_1 and D_2, respectively. For two random variables X_1 and X_2, we write $X_1 \overset{\text{id}}{=} X_2$ when $\varepsilon = 0$ *i.e.* when X_1 and X_2 are identically distributed.

Envelopes. In the proofs throughout this paper, exact probabilities are often challenging to compute directly, so we instead rely on upper bounds. The notion of probability envelope will be heavily used in this paper. Specifically, for a random variable X following a discrete distribution $\mathcal{D}$, we define $\mathcal{E}$ as an upper envelope of the distribution. That is, for any value x, the probability that X takes the value x (*i.e.* $\mathbb{P}(X = x)$) is bounded above by $\mathcal{E}(x)$, written as $X \lesssim \mathcal{E}$. As we will consider probability distributions conditioned to different disjoint events, we will often consider collections of probability envelopes with sometimes complex indices. For instance, we consider collections $\boldsymbol{\mathcal{E}} = (\mathcal{E}_{(j_1, \ldots, j_m)})$ indexed by tuples $(j_1, \ldots, j_m) \in [0, n]^m$. In other words, each possible tuple value gives rise to an envelope associated with a distribution indexed by the same tuple.

Sharing. We use the general notation of $[\![x]\!] = (x_1, \ldots, x_n) \in \mathbb{K}^n$ for a sharing of a secret value x.

2.2 Linear Sharing, Circuits, and Gadgets

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

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

for any $n \in \mathbb{N}$ and $(x_1, \ldots, x_n) \in \mathbb{K}^n$. We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $([\![x_1]\!], \ldots, [\![x_\ell]\!]) \in (\mathbb{K}^n)^\ell$ the n-linear decoding mapping acts as

$$\mathsf{LinDec} : ([\![x_1]\!], \ldots, [\![x_\ell]\!]) \mapsto (\mathsf{LinDec}([\![x_1]\!]), \ldots, \mathsf{LinDec}([\![x_\ell]\!])) \ .$$

[1] Along the paper, $\mathbb{K}$ denotes the base structure (ring or field) of the arithmetic circuits. We use the notation $\mathbb{K}$ as those circuits are defined over a field most of the time. But in the context of our application to Raccoon, $\mathbb{K}$ shall be the ring $\mathbb{Z}_q$ with q a non-prime integer.

Definition 1 (Linear Sharing from [8]). *Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an* n-sharing *of x is a random vector $[\![x]\!] \in \mathbb{K}^n$ such that $\mathsf{LinDec}([\![x]\!]) = x$. It is said to be* uniform *if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $[\![x]\!]|_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. An* 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 $[\![\boldsymbol{x}]\!] = ([\![x_1]\!], \ldots, [\![x_\ell]\!]) \in (\mathbb{K}^n)^\ell$ such that $[\![x_i]\!]$ is a uniform n-sharing of x_i for every $i \in [\ell]$.*

An *arithmetic circuit* on a finite field (or finite ring) $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations on $\mathbb{K}$. The circuit is built from a base set of gates $\mathbb{B} = \{g : \mathbb{K}^\ell \to \mathbb{K}^m\}$, such as addition gates $(x_1, x_2) \mapsto x_1 + x_2$, multiplication gates $(x_1, x_2) \mapsto x_1 \cdot x_2$, and copy gates $x \mapsto (x, x)$. A *randomized arithmetic circuit* includes a random gate that outputs a fresh uniform random value from $\mathbb{K}$. For a randomized circuit C, when we write $C(\rho, \cdot)$, this refers to the corresponding de-randomized circuit with the random tape $\rho \in \{0,1\}^*$ as input.

In the following, we shall call an *(n-share, ℓ-to-m) gadget*, a randomized arithmetic circuit that maps an input $[\![\boldsymbol{x}]\!] \in (\mathbb{K}^n)^\ell$ to an output $[\![\boldsymbol{y}]\!] \in (\mathbb{K}^n)^m$ such that $\boldsymbol{x} = \mathsf{LinDec}([\![\boldsymbol{x}]\!]) \in \mathbb{K}^\ell$ and $\boldsymbol{y} = \mathsf{LinDec}([\![\boldsymbol{y}]\!]) \in \mathbb{K}^m$ satisfy $\boldsymbol{y} = g(\boldsymbol{x})$ for some function g. We recall the definition of a circuit compiler in Appendix A (introduced in [8]).

2.3 Random Probing Security

Let $p \in [0, 1]$ be some constant leakage probability parameter, a.k.a. the *leakage rate*. In the p-random probing model, an evaluation of a circuit C leaks the value carried by each wire with a probability p, all the wire leakage events being mutually independent.

As in [8], we formally define the random-probing leakage of a circuit from the two following 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 drawn 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 deterministic arithmetic circuit C, a fixed random tape $\rho \in \{0,1\}^*$, a set of wire labels $\mathcal{W}$ (subset of all the wire labels of C), and a shared input $[\![x]\!]$, and it outputs a $|\mathcal{W}|$-tuple $\boldsymbol{w} \in \mathbb{K}^{|\mathcal{W}|}$, denoted as
$$\boldsymbol{w} \leftarrow \mathsf{AssignWires}(C, \rho, \mathcal{W}, [\![x]\!]) \ ,$$
where $\boldsymbol{w}$ corresponds to the exact assignments of the wires with label in $\mathcal{W}$ for an evaluation of $C(\rho, [\![x]\!])$.

Note that $\boldsymbol{w}$ is deterministically fixed from $C, \rho, \mathcal{W}$ and $[\![x]\!]$. Let us now induce a distribution for $\boldsymbol{w}$ as a random probing leakage with fixed leaked wires.

Definition 2 (Random Probing Leakage for Fixed Leaked Wires). *Let C be a deterministic arithmetic circuit, p be a probability, $\mathcal{W}$ be a set of wire labels and $[\![x]\!]$ be a shared input of C. Let $|\rho|$ be the size of the random tape for C and R be a uniform random variable on $\{0,1\}^{|\rho|}$. We define the leakage for $\mathcal{W}$ as the following random variable*
$$\mathcal{L}_{\mathcal{W}}(C, [\![x]\!]) := \mathsf{AssignWires}(C, R, \mathcal{W}, [\![x]\!]) \ .$$

Following [7], we shall say that a pair of vectors $([\![x]\!], [\![y]\!]) \in (\mathbb{K}^n)^2$ is *admissible* for a randomized 1-to-1 circuit C if there exists a random tape $\rho_{[\![x]\!],[\![y]\!]}$ such that $[\![y]\!] = C(\rho_{[\![x]\!],[\![y]\!]}, [\![x]\!])$.

Definition 3 (Induced Random Probing Leakage for Fixed Leaked Wires). *Let C be a deterministic arithmetic circuit, p be a probability and $\mathcal{W}$ be a set of wire labels. Let $([\![x]\!], [\![y]\!])$ be an admissible pair of vectors for C and $R_{[\![x]\!],[\![y]\!]}$ be a uniform random variable on the set of random tapes ρ such that $[\![y]\!] = C(\rho, [\![x]\!])$. We define the leakage for $\mathcal{W}$ as the following random variable*
$$\mathcal{L}_{\mathcal{W}}(C, [\![x]\!], [\![y]\!]) := \mathsf{AssignWires}(C, R_{[\![x]\!],[\![y]\!]}, \mathcal{W}, [\![x]\!]) \ .$$

Let us now introduce the formal definition of the so-called random probing security (RPS for short).

Definition 4 (Random Probing Security). *A randomized arithmetic circuit C is (p, ϵ)-*random probing secure *with respect to encoding* Enc *if there exists a PPT simulator*[2] Sim *such that for every input x, the distribution of $\mathsf{Sim}(C)$ is ϵ-close to the distribution of $\mathcal{L}_{\mathcal{W}}(C, [\![x]\!])$ where*

$$\begin{aligned} &\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p) && \texttt{Drawing the leaking wires.} \\ &[\![x]\!] \leftarrow \mathsf{Enc}(x). && \texttt{Drawing an encoding of the input.} \end{aligned}$$

Informally, while the random probing allows to consider that a large amount of wires can leak at the same time, the leakage itself is balanced by the probability of such an event. For example, if $\mathcal{W}$ corresponds to the labels of all the shares of the input x, the simulator will not be able to draw a correct sample without knowing x. However, this event happens with a probability p^n (n being the number of shares) which becomes negligible as n grows.

3 Extensions of the Random Probing Security

In this section we introduce the notion of *random probing security with auxiliary inputs and public outputs* (RPS-AI-O) as well as associated notions for the secure composition of gadgets. We also introduce the notion of *cardinal random probing composability* (cardinal-RPC) which provides a new trade-off between complexity and tightness for the composition compared to existing notions. We finally provide general composition results in this framework.

3.1 Random Probing Security with Auxiliary Inputs and Public Outputs

The notion of random probing security with auxiliary inputs and public outputs is a generalization of the RPS notion (Definition 4). It considers a randomized arithmetic circuit with shared inputs and outputs (as in the standard RPS notion) with two extra powers for the simulator.

1. First, the circuit admits additional auxiliary inputs which are not in shared form and for which partial information is potentially leaked. This partial information is captured by letting the simulator requesting a set of coordinates for each auxiliary input. The cardinality of these sets follow a list of distributions which is a parameter of the achieved notion.
2. Secondly, additional (unmasked) outputs are fully given to the simulator.

We introduce these new features in order to relax the RPS and achieve better complexity-security trade-offs for cryptosystems using random nonces that can be partly leaked without compromising the security. A typical example is the masked lattice-based signature scheme Raccoon. In this signature, some random nonces can be exposed to a probing attacker without harming the security. In security proofs, these random nonces are treated not as shares but as additional inputs [17]. This is formalized in the following definition.

[2] In concrete instantiations, the simulator Sim starts with generating a set of leaking wires through, as explicited in other definitions. Specifically, Sim will be decomposed with another PTT algorithm Sim^* as follows

$$\begin{aligned} \mathsf{Sim}(C): \quad &\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p) \\ &\text{out} \leftarrow \mathsf{Sim}^*(C, \mathcal{W}). \end{aligned}$$

Experiment 1: RPS-AI-O

```
$(I_1, \cdots, I_k) \leftarrow \mathsf{Sim}_1(C)$                      The simulator chooses indexes for the auxiliary inputs.
$out \leftarrow \mathsf{Sim}_2(\boldsymbol{a_1}|_{I_1}, \cdots, \boldsymbol{a_k}|_{I_k}, \boldsymbol{z})$   The simulator is given the values of the auxiliary inputs. Next, it returns simulated values for the leaking wires.
$return\ (I_1, \cdots, I_k, out)$
```

Definition 5 (Random Probing Security with Auxiliary Inputs and Public Outputs). *Let $n, \ell, m, k, d, \alpha \in \mathbb{N}$. Let C be a randomized arithmetic circuit with the following input/output partition*[3]

$$C : \overbrace{(\mathbb{K}^n)^\ell}^{\text{masked inputs}} \times \overbrace{(\mathbb{K}^\alpha)^k}^{\text{auxiliary inputs}} \to \overbrace{(\mathbb{K}^n)^m}^{\text{masked outputs}} \times \overbrace{\mathbb{K}^d}^{\text{public outputs}}$$
$$(\llbracket \boldsymbol{x} \rrbracket, \quad \boldsymbol{a}_1, \cdots, \boldsymbol{a}_k) \mapsto \quad (\llbracket \boldsymbol{y} \rrbracket, \quad \boldsymbol{z}).$$

Let $\boldsymbol{\mathcal{E}} = \mathcal{E}_1, \cdots \mathcal{E}_k$ be a set of probability distribution envelopes over the discrete set $[0, \alpha]$.

*The circuit C is $(p, \epsilon, \boldsymbol{\mathcal{E}})$-*random probing secure with auxiliary inputs and public outputs *($(p, \epsilon, \boldsymbol{\mathcal{E}})$-RPS-AI-O) with respect to encoding* Enc *if there exists a PPT stateful two-stage simulator* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that, for every admissible pair $((\llbracket \boldsymbol{x} \rrbracket, \boldsymbol{a}_1, \cdots, \boldsymbol{a}_k), (\llbracket \boldsymbol{y} \rrbracket, \boldsymbol{z}))$, the outputs of Experiment 1 are such that*

1. *$|I_1| \lesssim \mathcal{E}_1, \cdots, |I_k| \lesssim \mathcal{E}_k$, and*
2. *$out \approx_\varepsilon \underbrace{\mathcal{L}_{\mathcal{W}}(C, (\llbracket \boldsymbol{x} \rrbracket, \boldsymbol{a}_1, \cdots, \boldsymbol{a}_k), (\llbracket \boldsymbol{y} \rrbracket, \boldsymbol{z}))}_{\text{Induced random probing leakage}}$ where $\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p)$ and $\llbracket \boldsymbol{x} \rrbracket \leftarrow \mathsf{Enc}(\boldsymbol{x})$.*

3.2 Cardinal Random Probing Composability

The notion of *random probing composability* (RPC) was initially introduced in [8]. In essence, a circuit composed of individual gadgets that satisfy the RPC property will itself be RPC, and, as a result, random probing secure. Specifically, a gadget is (t, p, ϵ)-RPC with some integer $t < n$ and $p, \epsilon \in [0, 1]$ if, given t output shares, the probability that more than t shares of each input are required to simulate the leakage of the gadget –where each wire leaks with probability p– along with the t output shares, is upper bounded by ϵ. Based on this property, a circuit C composed of (t, p, ϵ_i)-RPC gadgets $(G_i)_{1 \le i \le |C|}$ will itself be (t, p, ϵ)-RPC with $\epsilon = 1 - \prod_{1 \le i \le |C|}(1 - \epsilon_i)$. The formal RPC definition and the latter composition theorem are formally recalled in Appendix A.

In this paper, we shall refer to the original RPC notion as *threshold-RPC* because it is defined with respect to a threshold t on the tolerated leakage on the input and output sharings. An alternative notion, based on so-called *probe distribution tables* (PDT) and targeting tighter composition, was put forward in [13]. Instead of fixing a threshold t for both the number of output shares (to be simulated) and the number of input shares (required for the simulations), the PDT of a gadget considers each specific pair of sets of input shares and output shares. Specifically, each cell (I, J) in the table represents the probability that a particular set of input shares I (represented by the row) is required to simulate the leakage of a gadget – with each wire leaking with probability p – along with a corresponding set of output shares J (represented by the column). Based on such PDT for each gadget, the authors of [13] define several composition rules to upper bound the random probing security of the global circuit.

Similarly, we shall refer to the PDT notion as *general-RPC* because it is the most general RPC notion in the sense that it encompasses the original (threshold-)RPC notion as well as the cardinal-RPC notion that

[3] To be more generic, one can consider different dimensions for the auxiliary inputs *i.e.* $\mathbb{K}^{\alpha_1}, \ldots, \mathbb{K}^{\alpha_k}$ instead of $(\mathbb{K}^\alpha)^k$. The definition can be straightforwardly adapted.

Experiment 2: Cardinal-RPC

```
$\mathcal{W} \leftarrow \mathsf{LeakingWires}(G, p)$                                Drawing the leaking wires.
$I_1, \cdots I_\ell \leftarrow \mathsf{Sim}_1(G, \mathcal{W}, J_1, \cdots J_m)$      The simulator is given the indexes of the
                                                                                     leaking outputs. It chooses indexes for
                                                                                     the inputs.
$out_1, out_2 \leftarrow \mathsf{Sim}_2([\![x_1]\!]|_{I_1}, \cdots, [\![x_\ell]\!]|_{I_\ell})$   The simulator is given the shares at desired
                                                                                     indexes and returns simulated values for the
                                                                                     leaking wires ($out_1$) and outputs ($out_2$)
return $(I_1, \cdots I_\ell, out_1, out_2)$
```

we propose here. The definition of general-RPC with our formalism is given in Appendix A. This notion provides tighter composition than the threshold-RPC notion, which can be viewed as a special case of the former. Specifically, the t-threshold-RPC security parameter ϵ of a gadget can be easily recalculated from its PDT. For each column with an output set of cardinality at most t, we compute the sum of the rows corresponding to input sets with cardinality strictly greater than t. The largest of these sums across all eligible columns determines the threshold-RPC security parameter ϵ.

The cardinal-RPC notion provides a practical trade-off between the threshold RPC notion and the general RPC notion. In essence, rather than using a fixed threshold t and for the number of shares required for each output and input, the cardinal-RPC notion considers the possible cardinalities for each of them. This makes it simpler than the general RPC notion, which considers all possible sets of shares for each input and output—a number that grows exponentially with the number of shares. On the other hand, the combinations of cardinalities only grow polynomially with the number of shares.

In the following definition, we use a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. It means that there are $(n+1)^m$ functions $\mathcal{E}_{j_1, \cdots j_m}$ such that

$$\forall j_1, \cdots j_m, i_1, \cdots, i_\ell \in [0, n]^{m+\ell}, \quad \mathcal{E}_{j_1, \cdots j_m}(i_1, \cdots, i_\ell) \in [0, 1] \ .$$

Definition 6 (Cardinal Random Probing Composability). *Let $n, \ell, m \in \mathbb{N}$. Let $\boldsymbol{\mathcal{E}}$ represent a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. An n-share gadget $G : (\mathbb{K}^n)^\ell \to (\mathbb{K}^n)^m$ is $(p, \boldsymbol{\mathcal{E}})$-*cardinal random probing composable ($(p, \boldsymbol{\mathcal{E}})$-cardinal-RPC) *for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for every shared input $[\![\boldsymbol{x}]\!] \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \ldots, J_m)$ where $J_1 \subseteq [n]$, ..., $J_m \subseteq [n]$, the outputs of Experiment 2 are such that*

1. $(|I_1|, \cdots, |I_\ell|) \lesssim \mathcal{E}_{|J_1|, \cdots |J_m|}$, *and*
2. $$(out_1, out_2) \overset{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, [\![\boldsymbol{x}]\!], [\![\boldsymbol{y}]\!])}_{\texttt{Induced Random probing leakage}} \quad , \quad \underbrace{([\![y_1]\!]|_{J_1}, \cdots, [\![y_m]\!]|_{J_m})}_{\texttt{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \mathsf{LeakingWires}(G, p)$ and $[\![\boldsymbol{y}]\!] \leftarrow G([\![\boldsymbol{x}]\!])$.

We emphasize the broadness of Definition 6: every gadget can be cardinal RPC for some probability and envelope. Hence, it only makes sense to claim that a gadget is $(p, \boldsymbol{\mathcal{E}})$-cardinal-RPC for some explicit p and $\boldsymbol{\mathcal{E}}$.

The security advantage for the threshold-RPC property can be derived from the distribution envelope of the cardinal-RPC property in a similar way that it is derived from the PDT of the general RPC notion as explained above. This implication is formally stated in Lemma 1 hereafter.

Although cardinal-RPC provides a looser bound compared to general-RPC, the two notions become equivalent in the case of a uniformly ℓ-to-m cardinal-RPC gadget. A gadget is uniformly cardinal-RPC when, for any $(t_1, \ldots, t_\ell) \in [0, n]^\ell$, all input shares of cardinality $(t_1, \ldots, t_\ell)$—necessary for simulating both

Experiment 3: Cardinal-RPC-AI-O

```
$(\llbracket \boldsymbol{y} \rrbracket, \boldsymbol{z}) \leftarrow G(\llbracket \boldsymbol{x} \rrbracket, \boldsymbol{a})$    Drawing the public outputs.
$\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p)$    Drawing the leaking wires.
$(I_1, \cdots, I_\ell), (L_1, \cdots, L_k) \leftarrow \mathsf{Sim}_1(G, \mathcal{W}, \boldsymbol{z}, J_1, \cdots, J_m)$    The simulator is given the indexes of the leaking outputs. It chooses indexes for the shared and auxiliary inputs.
$out_1, out_2 \leftarrow \mathsf{Sim}_2(\llbracket x_1 \rrbracket|_{I_1}, \cdots, \llbracket x_\ell \rrbracket|_{I_\ell}, \boldsymbol{a_1}|_{L_1}, \cdots, \boldsymbol{a_k}|_{L_k})$    The simulator is given the shares at desired indexes and returns simulated values for the leaking wires and outputs.
return $(out_1, out_2, (I_1, \cdots, I_\ell), (L_1, \cdots, L_k))$
```

the leakage and the specified output—occur with equal probability. Furthermore, this equal probability distribution also applies to outputs of the same cardinality, ensuring symmetry in the simulation. The formal definition is given in Appendix A.

3.3 Cardinal-RPC with Auxiliary Inputs and Public Outputs

We now extend the previously introduced notion of cardinal-RPC to gadgets with auxiliary inputs and public outputs. The following definition is naturally obtained from cardinal-RPC (Definition 6) and RPS-AI-O (Definition 5).

Definition 7 (Cardinal RPC with Auxiliary Inputs and Public Outputs). *Let $n, \ell, m, k, \alpha, d \in \mathbb{N}$. Let $\boldsymbol{\mathcal{E}}$ represent a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. Let $\boldsymbol{\mathcal{E}}'$ represent a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, \alpha]^k$. Let G be a gadget with the following input/output partition*

$$\begin{array}{rccccccc} G: & \overbrace{(\mathbb{K}^n)^\ell}^{\text{masked inputs}} & \times & \overbrace{(\mathbb{K}^\alpha)^k}^{\text{auxiliary inputs}} & \to & \overbrace{(\mathbb{K}^n)^m}^{\text{masked outputs}} & \times & \overbrace{\mathbb{K}^d}^{\text{public outputs}} \\ & (\llbracket \boldsymbol{x} \rrbracket, & & \boldsymbol{a}_1, \cdots, \boldsymbol{a}_k) & \mapsto & (\llbracket \boldsymbol{y} \rrbracket, & & \boldsymbol{z}). \end{array}$$

*The gadget G is $(p, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{E}}')$-*cardinal random probing composable with auxiliary inputs and public outputs *($(p, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{E}}')$-cardinal-RPC-AI-O) for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for every shared input $\llbracket x \rrbracket \in (\mathbb{K}^n)^\ell$, auxiliary input $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k \in (\mathbb{K}^\alpha)^k$, and for every set collection $(J_1, \ldots, J_m)$ where $J_1 \subseteq [n]$, $\ldots$, $J_m \subseteq [n]$, the outputs of Experiment 3 are such that*

1. $(|I_1|, \cdots, |I_\ell|) \lesssim \mathcal{E}_{|J_1|, \cdots |J_m|}$,
2. $(|L_1|, \cdots, |L_k|) \lesssim \mathcal{E}'_{|J_1|, \cdots |J_m|}$, *and*
3.

$$(out_1, out_2) \overset{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, (\llbracket \boldsymbol{x} \rrbracket, \boldsymbol{a}), (\llbracket \boldsymbol{y} \rrbracket, \boldsymbol{z}))}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \cdots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right)$$

where $\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p)$ and $(\llbracket \boldsymbol{y} \rrbracket, \boldsymbol{z}) \leftarrow G(\llbracket \boldsymbol{x} \rrbracket, \boldsymbol{a})$.

Similarly, one can define threshold-RPC-AI-O from the original threshold-RPC definition. See Definition 17 in appendix for details.

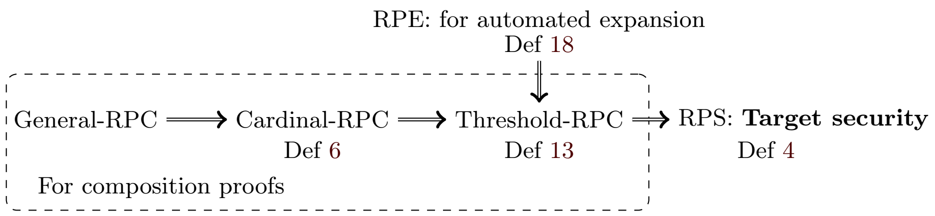


Fig. 1: Schematic implications between security notions

3.4 Summary of the notions

There exist many implications between the presented security notions. The main ones are shown in the schematic graph of Figure 1.

Lemma 1 (Cardinal-RPC $\implies$ Threshold-RPC). *Let $n, \ell, m \in \mathbb{N}$. Let $\boldsymbol{\mathcal{E}}$ represent a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. If an n-share gadget $G : (\mathbb{K}^n)^\ell \to (\mathbb{K}^n)^m$ is $(p, \boldsymbol{\mathcal{E}})$-cardinal-RPC for some $p \in [0, 1]$, then it is (t, p, ϵ)-threshold-RPC for the same probability p and for any integer $t \in [0, n-1]$, with*

$$\epsilon \leq \max_{(J_1,\cdots J_m)\in \boldsymbol{J}(t)} \left(\sum_{(I_1,I_2,\ldots,I_\ell)\in \boldsymbol{I}(t)} \mathcal{E}_{|J_1|,\cdots,|J_m|}(|I_1|, \ldots, |I_\ell|) \right)$$
$$\textit{with } \boldsymbol{I}(t) = [(I_1, I_2, \ldots, I_\ell) \in [0, n]^\ell \textit{ such that } \exists i \in [1, \ell], |I_i| > t],$$
$$\textit{and } \boldsymbol{J}(t) = [(J_1, J_2, \ldots, J_m) \in [0, n]^m \textit{ such that } |J_1| \leq t, |J_2| \leq t, \ldots, |J_m| \leq t].$$

Naturally, an n-share cardinal-RPC gadget is also cardinal-RPC-AI-O with auxiliary inputs of cardinality that is a multiple of n. This implication is established in Lemma 2 and follows directly from the definitions and the application of the law of total probability.

Lemma 2 (Cardinal-RPC-O $\implies$ Cardinal-RPC-AI-O). *Let $n, \ell, m, k, d \in \mathbb{N}$. Let $\boldsymbol{\mathcal{E}}$ represent a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, n]^{\ell+k}$. If a gadget $G : (\mathbb{K}^n)^{\ell+k} \to (\mathbb{K}^n)^m \times \mathbb{K}^d$ is $(p, \boldsymbol{\mathcal{E}})$*-cardinal-RPC-O*, then it is also $(p, \boldsymbol{\mathcal{E}}^x, \boldsymbol{\mathcal{E}}^a)$-cardinal-RPC-AI-O where $\boldsymbol{\mathcal{E}}^x$ and $\boldsymbol{\mathcal{E}}^a$ are defined, for all $j_1, \cdots j_m$, for all $i_1, \cdots, i_\ell \in [0, n]^\ell$, for all $i'_1, \cdots, i'_k \in [0, n]^k$ as*

$$\mathcal{E}^x_{j_1,\cdots j_m}(i_1, \cdots, i_\ell) = \sum_{(t_1,\cdots,t_k)\in[0,n]^k} \mathcal{E}_{j_1,\cdots j_m}(i_1, \cdots, i_\ell, t_1, \cdots, t_k),$$
$$\mathcal{E}^a_{j_1,\cdots j_m}(i'_1, \cdots, i'_k) = \sum_{(t_1,\cdots,t_\ell)\in[0,n]^\ell} \mathcal{E}_{j_1,\cdots j_m}(t_1, \cdots, t_\ell, i'_1, \cdots, i'_k).$$

While the cardinal-RPC-AI-O property is introduced for compositional purposes, it is important to highlight that a cardinal-RPC-AI-O circuit is also threshold-RPC-AI-O (following similar arguments to those of Lemma 1) and RPS-AI-O, as formalized in Lemma 3.

Lemma 3 (Cardinal-RPC-AI-O $\implies$ RPS-AI-O). *Following Definition 7, let $n, \ell, m, k, \alpha \in \mathbb{N}$. Let G be a gadget with the following input/output partition $(\mathbb{K}^n)^\ell \times (\mathbb{K}^\alpha)^k \to (\mathbb{K}^n)^m \times \mathbb{K}^d$. Let $\boldsymbol{\mathcal{E}}$ be a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. Let $\boldsymbol{\mathcal{E}}'$ be a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, \alpha]^k$. If the gadget G is $(p, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{E}}')$-cardinal-RPC-AI-O, then is also $(p, \epsilon, \boldsymbol{\mathcal{E}}^*)$-RPS-AI with*

$$\epsilon = 1 - \sum_{(n_1,n_2,\ldots,n_\ell)\in[0,n-1]^\ell} \mathcal{E}_{(0,\ldots,0)}(n_1, n_2, \ldots, n_\ell) \quad \textit{and} \quad (\mathcal{E}^*_1, \cdots, \mathcal{E}^*_k) = \mathcal{E}'_{(0,\ldots,0)} \ .$$

Proof. Let us assume that there exists a simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ for Experiment 3 and build a simulator $(\mathsf{Sim}_1^*, \mathsf{Sim}_2^*)$ for Experiment 1. These simulators $(\mathsf{Sim}_1^*, \mathsf{Sim}_2^*)$ are defined as follows.

```
$\mathsf{Sim}_1^*(G, \mathcal{W}, \boldsymbol{z})$ :  $(I_1, \cdots, I_\ell), (L_1, \cdots, L_k) \leftarrow \mathsf{Sim}_1(C, \mathcal{W}, \boldsymbol{z}, \emptyset, \cdots, \emptyset)$
    keep $(|I_1|, \cdots, |I_\ell|)$ in the internal state
    return $(L_1, \cdots, L_k)$
```

```
$\mathsf{Sim}_2^*(\boldsymbol{a_1}|_{I_1}, \cdots, \boldsymbol{a_k}|_{I_k})$ :  get $(|I_1|, \cdots, |I_\ell|)$ from the internal state
    for $i \in [1, \ell]$:
        $r_i \leftarrow (\$, \cdots, \$) \in [1, n]^{|I_i|}$
    $out_1, out_2 \leftarrow \mathsf{Sim}_2(r_1, \cdots, r_\ell, \boldsymbol{a_1}|_{I_1}, \cdots, \boldsymbol{a_k}|_{I_k})$
    return $out_2$
```

The simulators $(\mathsf{Sim}_1^*, \mathsf{Sim}_2^*)$ will succeed if the cardinals of $\boldsymbol{I}$ are all in $[0, n-1]$. Indeed, in that case, the shares can be simulated as uniform random values without knowledge of the inputs. The distinguishing advantage ϵ is then

$$\epsilon \leq 1 - \sum_{(n_1, n_2, \ldots, n_\ell) \in [0, n-1]^\ell} \mathcal{E}_{(0, \cdots, 0)}(n_1, n_2, \ldots, n_\ell).$$

The probability envelopes are an application of Condition 2 of Definition 7 with $\boldsymbol{J} = (\emptyset, \cdots, \emptyset)$. □

Property 1 (RPx-AI-O $\Longrightarrow$ RPx). Let RPx be a security notion $\in$ {RPS, threshold-RPC, cardinal-RPC} (see Definitions 4, 13 and 6 respectively). The security RPx is a particular case of RPx-AI-O (see Definitions 5, 17 and 7 respectively) where $d = k = 0$.

Definition 8 (RPx-AI and RPx-O). *Let RPx be a security notion $\in$ {RPS, RPC, threshold-RPC, cardinal-RPC}. We denote by RPx-AI (resp. RPx-O) the particularly case of RPx-AI-O where $d = 0$ (resp. $k = 0$).*

3.5 Composition Results

In the following, to construct cardinal-RPC-AI-O secure circuits, we will begin by considering the auxiliary inputs as regular shared inputs of base cardinal-RPC gadgets, using a padding that we will define later. Next, we will compose these cardinal-RPC gadgets using Lemma 4 and variants.

Lemma 4. *Let $n \in \mathbb{N}$ and $p \in [0, 1]$. Let $G_1 : (\mathbb{K}^n)^2 \to \mathbb{K}^n$ be a $(p, \mathcal{E}^1)$-cardinal-RPC gadget for $\mathcal{E}^1$ a collection of $n+1$ probability envelopes over $[0, n]^2$, and let $G_2 : (\mathbb{K}^n)^2 \to \mathbb{K}^n$ be a $(p, \mathcal{E}^2)$-cardinal-RPC gadget for $\mathcal{E}^2$ a collection of $n+1$ probability envelopes over $[0, n]^2$. The sequential composition $G_3 : (\mathbb{K}^n)^3 \to \mathbb{K}^n$ depicted in Figure 2 is $(p, \boldsymbol{\mathcal{E}}^3)$-cardinal-RPC where $\mathcal{E}^3$ is a collection of $n+1$ probability envelopes over $[0, n]^3$ verifying $\forall \boldsymbol{t}_{in} = (t_{in}^1, t_{in}^2, t_{in}^3) \in [0, n]^3$ and $\forall t_{out} \in [0, n]$,*

$$\mathcal{E}^3_{t_{out}}(\boldsymbol{t}_{in}) = \sum_{i=0}^{n} \mathcal{E}^1_i(t_{in}^1, t_{in}^2) \cdot \mathcal{E}^2_{t_{out}}(t_{in}^3, i).$$

Proof. Let us assume that there exists simulators $(\mathsf{Sim}_1^1, \mathsf{Sim}_2^1)$ for Experiment 2 for G_1 and $(\mathsf{Sim}_1^2, \mathsf{Sim}_2^2)$ for G_2. We aim at building a simulator $(\mathsf{Sim}_1^*, \mathsf{Sim}_2^*)$ for Experiment 2 for G_3.

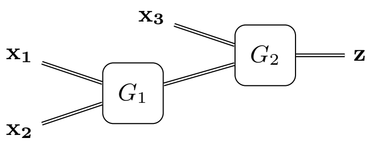


Fig. 2: Composition between $G_1 : (\mathbb{K}^n)^2 \to \mathbb{K}^n$ and $G_2 : (\mathbb{K}^n)^2 \to \mathbb{K}^n$.

Let us define $(\mathsf{Sim}_1^*, \mathsf{Sim}_2^*)$ as follows.

$$\begin{aligned}\mathsf{Sim}_1^*(G, \mathcal{W}, J) : \quad & \text{Separate } \mathcal{W} \text{ in } \mathcal{W}_1, \mathcal{W}_2 \text{ for leaking wires in } G_1 \text{ and } G_2 \\ & (I_1^2, I_2^2) \leftarrow \mathsf{Sim}_1^2(C, \mathcal{W}_2, J) \\ & (I_1^1, I_2^1) \leftarrow \mathsf{Sim}_1^1(C, \mathcal{W}_1, I_2^2) \\ & \text{return } (I_1^1, I_2^1, I_1^2).\end{aligned}$$

$$\begin{aligned}\mathsf{Sim}_2^*([\![x_1]\!]|_{I_1}, [\![x_2]\!]|_{I_2}, [\![x_3]\!]|_{I_3}) : \quad & out_1^1, out_2^1 \leftarrow \mathsf{Sim}_2^1\big([\![x_1]\!]|_{I_1}, [\![x_2]\!]|_{I_2}\big) \\ & out_1^2, out_2^2 \leftarrow \mathsf{Sim}_2^2\big([\![x_3]\!]|_{I_3}, out_2^1\big) \\ & \text{return } (out_1^1, out_1^2), out_2^2\end{aligned}$$

Let us compute the distribution envelope. We denote by $I_{i,j} = |I_j^i|$ and $O = |J|$ where I_j^i and J are defined in Sim_1^*. By definition, $\forall \boldsymbol{t}_{\text{in}} = (t_{\text{in}}^1, t_{\text{in}}^2, t_{\text{in}}^3) \in [0, n]^3$, $\mathcal{E}^3_{t_{\text{out}}}(\boldsymbol{t}_{\text{in}})$ is an upper bound on the following probability

$$p_c = \mathbb{P}(I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2 \wedge I_{2,1} = t_{\text{in}}^3 | O = t_{\text{out}}).$$

From the law of total probability and the Bayes formula, we have

$$\begin{aligned} p_c &= \sum_{i=0}^{n} \mathbb{P}\left(\left(I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2 \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i\right) | O = t_{\text{out}}\right) \\ &= \sum_{i=0}^{n} \mathbb{P}\left(\left(I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2\right) \mid \left(O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i\right)\right) \\ &\qquad \cdot \mathbb{P}\left(\left(I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i\right) | O = t_{\text{out}}\right)\end{aligned}$$

We now introduce the random variable $\mathcal{Y}$ which represents the subset of output shares from G_1 that are required for the subsequent simulation:

$$\begin{aligned} p_c = \sum_{i=0}^{n} \sum_{Y \subseteq [0,n]} & \mathbb{P}\left(\left(I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2\right) \mid \left(O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i \wedge \mathcal{Y} = Y\right)\right) \\ & \cdot \mathbb{P}\left((\mathcal{Y} = Y) | O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i\right) \\ & \cdot \mathbb{P}\left(\left(I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i\right) | O = t_{\text{out}}\right)\end{aligned}$$

The events $I_{1,1} = t_{\text{in}}^1$ and $I_{1,2} = t_{\text{in}}^2$ in the first term depend only on the output set $\mathcal{Y}$ (and its cardinality) and not on the subsequent leakage anymore, hence we can simplify the expression:

$$\begin{aligned} p_c = \sum_{i=0}^{n} \sum_{\substack{Y \subseteq [0,n] \\ |Y| = i}} & \mathbb{P}\left(\left(I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2\right) \mid (\mathcal{Y} = Y)\right) \\ & \cdot \mathbb{P}\left((\mathcal{Y} = Y) | O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i\right) \\ & \cdot \mathbb{P}\left(\left(I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i\right) | O = t_{\text{out}}\right)\end{aligned}$$

Algorithm 1 $\mathsf{RPZeroEnc}^\gamma$

```
Output: $[\![z]\!] = (z_1, \ldots, z_n) \in \mathbb{K}^n$ such that $z_1 + \cdots + z_n = 0$
1. $[\![z]\!] = (0, 0, \ldots, 0)$ {$n$ zeros}
2. for $i = 1$ to $\gamma$ do
3.     Select two distinct indices $i_1, i_2 \in [1, n]$ uniformly at random
4.     $r \xleftarrow{\$} \mathbb{K}$
5.     $z_{i_1} \leftarrow z_{i_1} + r$
6.     $z_{i_2} \leftarrow z_{i_2} - r$
7. end for
8. return $[\![z]\!]$
```

Algorithm 2 $\mathsf{RPRefresh}^\gamma([\![a]\!])$

```
Input: $[\![a]\!] = (a_1, \ldots, a_n) \in \mathbb{K}^n$
Output: $[\![s]\!] = (s_1, \ldots, s_n) \in \mathbb{K}^n$ such that $s_1 + \cdots + s_n = a_1 + \cdots + a_n$
1. $[\![z]\!] \leftarrow \mathsf{RPZeroEnc}^\gamma()$
2. for $i = 1$ to $n$ do
3.     $s_i \leftarrow a_i + z_i$
4. end for
5. return $[\![s]\!]$
```

To obtain an upper bound, we use the worst-case scenario for the first term, which corresponds to the envelope function of G_1, and we bound the third term using the envelope function of G_2, denoted by $\mathcal{E}^2_{t_\text{out}}(t^3_\text{in}, i)$:

$$p_c \leq \sum_{i=0}^{n} \mathcal{E}^1_i(t^1_\text{in}, t^2_\text{in}) \cdot \mathcal{E}^2_{t_\text{out}}(t^3_\text{in}, i) \cdot \sum_{\substack{Y \subseteq [0,n] \\ |Y| = i}} \mathbb{P}\left((\mathcal{Y} = Y) \,|O = t_\text{out} \wedge I_{2,1} = t^3_\text{in} \wedge I_{2,2} = i\right)$$

The last summation term equals one, which concludes the proof. □

4 New Random Probing Composable Gadgets

Now that we have defined our new security notions, we introduce cardinal-RPC *base* gadgets, which will be used to construct random probing secure circuits. We begin with our new refresh algorithm in Section 4.1, which is based on a novel zero encoding and will serve as the foundation for our elementary linear operations—addition, multiplication by a constant, and copy— detailed in Section 4.2

4.1 A New Versatile Cardinal-RPC Refresh Gadget

Definition 9 (ZeroEncoding). *Let $n \in \mathbb{N}$. A zero-encoding algorithm, denoted* $\mathsf{ZeroEnc}$*, is a probabilistic algorithm sampling an n-sharing of* 0*. In other words, $[\![z]\!] \leftarrow \mathsf{ZeroEnc}()$ such that $\sum z_i = 0$ (with probability 1).*

We here introduce our new refresh gadget, referred to as $\mathsf{RPRefresh}$, for *random-pair refresh*, which is described in Algorithm 2. This gadget is based on the general construction which consists in refreshing the input sharing by the addition of a sharing of 0. The novelty of our refresh gadget lies in the underlying gadget $\mathsf{RPZeroEnc}$ to generate a sharing of 0, which is detailed in Algorithm 1. This gadget is parameterized by a number γ of iterations, which influences both its complexity and its security (up to a certain limit).

The process begins with the initialization of a constant all-0 sharing. During each iteration, two share indices are generated uniformly at random[4] and a fresh random value is respectively added to and subtracted from the corresponding shares. After all iterations have been completed, the sharing of zero produced by $\mathsf{RPZeroEnc}$ is added to the input value in $\mathsf{RPRefresh}$, share by share.

While its conceptual idea is simple, this refreshing technique never outputs a perfectly fresh sharing of zero because of the probabilistic nature of the refreshed indexes. In a first step, let us discuss the quality of the output while leaving probing security apart.

[4] We deviate here from the classical circuit model by introducing the generation of random indices. However, this can be interpreted as sampling a specific circuit (instantiated with our uniform indices) that we fully disclose to the attacker.

Definition 10 (Partition ZeroEncoding). *For $P = (n_1, \dots, n_\ell) \in \mathcal{P}(n)$ an integer partition, we define* $\mathsf{PartitionZeroEnc}(P)$ *as a particular type of distribution producing a sharing $[\![z]\!] = [\![z_1]\!] \parallel \dots \parallel [\![z_\ell]\!]$ where $[\![z_i]\!]$ is a fresh n_i-sharing of 0 for every $i \in [1, \ell]$. In other words,*

$$\mathsf{PartitionZeroEnc}(n_1, \dots, n_\ell) = \underbrace{z_1, \cdots, z_{n_1}}_{\sum z_i = 0}, \underbrace{z_{n_1+1}, \cdots, z_{n_1+n_2}}_{\sum z_i = 0}, \cdots, \underbrace{z_{n-n_\ell+1}, \cdots, z_n}_{\sum z_i = 0}.$$

The two extreme cases in Definition 10 are (worst case) for $P = (1, \dots, 1)$ where $\mathsf{PartitionZeroEnc}(1, \dots, 1) = (0, \dots, 0)$ (the poorest output quality), and (ideal case) for the singleton $P = (n)$ where $\mathsf{PartitionZeroEnc}(n)$ is a uniform sharing of 0 (the best output quality).

Lemma 5. *Let $n, \gamma \in \mathbb{N}$. Let ρ be a random tape. There exists a partition $P_{\gamma,\rho} \in \mathcal{P}(n)$, a permutation function $\sigma_{\gamma,\rho} : [1, n] \to [1, n]$ and another random tape $\rho'_{\gamma,\rho}$ such that*

$$\mathsf{RPZeroEnc}^\gamma(\rho) = \sigma_{\gamma,\rho}(\mathsf{PartitionZeroEnc}(P_{\gamma,\rho}, \rho'_{\gamma,\rho}))$$

where $\mathsf{RPZeroEnc}$ *is depicted in Algorithm 1.*

This lemma follows from the definition of $\mathsf{RPZeroEnc}$. Let $\gamma \in \mathbb{N}$ and the random tape ρ be fixed parameters. Here is an intuition of an iterative construction of the partition, denoted P, and the permutation, denoted σ. Before the first iteration, the variable $[\![z]\!] = (0, \cdots, 0)$ is exactly the output of

$$\mathsf{RPZeroEnc}(\underbrace{1, 1, \cdots, 1}_{n \text{ times}}) .$$

Suppose now that the generated indices in the first iteration are $(i_1, i_2) = (1, 2)$, then we have $[\![z]\!] = (r_1, -r_1, \dots, 0)$ for $r_1 \leftarrow \$$ which is identically distributed to

$$\mathsf{RPZeroEnc}(2, \underbrace{1, \cdots, 1}_{n-2 \text{ times}}) .$$

If the generated indices were different from $(1, 2)$, we would have a distribution identical to $\sigma(\mathsf{PartitionZeroEnc}(2, 1, \cdots, 1))$ for the same partition but with some coordinate permutation σ. Now for the second iteration, let $r_2 \leftarrow \$$,

$$\begin{aligned}
&\text{if } (i_1, i_2) = (1, 2), \ [\![z]\!] = (r_1 + r_2, r_1 - r_2, 0 \dots, 0),\\
&\text{if } (i_1, i_2) = (2, 3), \ [\![z]\!] = (r_1 \quad, -r_1 + r_2, r_2, 0 \dots, 0),\\
&\text{if } (i_1, i_2) = (3, 4), \ [\![z]\!] = (r_1 \quad, -r_1 \quad, r_2, -r_2, 0 \dots, 0).
\end{aligned}$$

In these cases, $[\![z]\!]$ will be statistically indistinguishable to

$$\begin{aligned}
&\mathsf{RPZeroEnc}(2, \underbrace{1, \cdots, 1}_{n-2 \text{ times}}) \text{ if } (i_1, i_2) = (1, 2),\\
&\mathsf{RPZeroEnc}(3, \underbrace{1, \cdots, 1}_{n-3 \text{ times}}) \text{ if } (i_1, i_2) = (2, 3),\\
&\mathsf{RPZeroEnc}(2, 2, \underbrace{1, \cdots, 1}_{n-4 \text{ times}}) \text{ if } (i_1, i_2) = (3, 4).
\end{aligned}$$

All the cases are captured with a coordinate permutation in a similar way. On and on, one can construct a partition iteratively depending on the randomness of $\mathsf{RPZeroEnc}$. The permutation σ captures the possibility of $\mathsf{RPZeroEnc}$ shuffling the order of the refreshed indices. As the number of iterations accumulates, the distribution of $[\![z]\!]$ becomes increasingly less partitioned, approaching the ideal partition $P = (n)$. This iterative construction is formalized in a more general case (in the presence of leakage) in Lemma 6.

Experiment 4: Random Probing Partition

```
$\mathcal{W} \leftarrow \mathsf{LeakingWires}(\mathsf{ZeroEnc}, p)$          Drawing the leaking wires.
$\rho \leftarrow \$$                                      Drawing a random tape.
$[\![z]\!] \leftarrow \mathsf{ZeroEnc}(\rho)$                       Drawing $[\![z]\!]$ an encoding of $0$.
$leak \leftarrow \mathsf{AssignWires}(\mathsf{ZeroEnc}, \rho, \mathcal{W}, \emptyset)$    Computing the leaked values.
$[\![w]\!] \leftarrow \mathsf{Sim}(\mathcal{W}, leak)$                  The leakages are given to the simulator. It
                                          outputs an encoding of $0$ whose randomness
                                          is "weakened" due to the leakage.
$(P, \sigma) := \phi(\mathcal{W})$                        Computing a partition and a permutation
                                          corresponding to the leaking wires.
$[\![u]\!] \leftarrow \sigma\big(\mathsf{PartitionZeroEnc}(P)\big)$        Drawing an encoding of $0$ corresponding to the
                                          "remaining" randomness to compensate
                                          the randomness that was given in the leakage.
return $(P, [\![z]\!], [\![w]\!], [\![u]\!])$
```

Let us now address the random probing security of $\mathsf{RPZeroEnc}$. We will show below in Theorem 1 that $\mathsf{RPZeroEnc}$ is cardinal-RPC (we recall that this security notion was presented in Definition 6). To be able to establish the cardinal-RPC property, one needs to take into account the possible leaked iterations in the quality of the refreshing. Naturally, the more leaked randomness, the less qualitative the refreshing is. We introduce the intermediary notion of *random probing partitioned* (RPP) to characterize this quality with respect to the number of iterations and the leakage probability. We later prove the cardinal-RPC security of the refresh (Theorem 1) from the RPP security of the zero-encoding gadget.

Definition 11 (Random-Probing-Partition). *Let $n \in \mathbb{N}$ and $\mathcal{P}(n)$ be the set of partitions of n. Let $\mathcal{E}$ be a distribution envelope over $\mathcal{P}(n)$. An n-share zero-encoding gadget* $\mathsf{ZeroEnc} : \emptyset \to \mathbb{K}^n$ *is $(p, \mathcal{E})$*-random-probing-partitioned *(RPP) for some $p \in [0, 1]$ if there exists a deterministic algorithm ϕ and a PPT simulator* Sim *such that the outputs of the random experiment 4 satisfy:*

1. *$P \lesssim \mathcal{E}$, and*
2. *$[\![z]\!]$ and $[\![w]\!] + [\![u]\!]$ are identically distributed.*

Intuition of Definition 11. Note that Definition 11 is different from other security notions introduced in this paper. It does not ensure that the random probing leakage can be simulated. The simulator is instead given the leakage and the leaking wires. The idea of Definition 11 is to show that the output of $\mathsf{ZeroEnc}$ is such that

$$[\![z]\!] \approx \underbrace{[\![w]\!]}_{\texttt{leaked 0-encoding}} + \underbrace{[\![u]\!]}_{\texttt{fresh partitioned 0-encoding}} .$$

The sharing $[\![u]\!]$ hence captures the "remaining" randomness of the output sharing $[\![z]\!]$ given the internal leakage of the gadget. Let us further stress that $[\![u]\!]$ follows the random distribution $\sigma\big(\mathsf{PartitionZeroEnc}(P)\big)$ which randomness is highly related to the partition P. The envelope $\mathcal{E}$ upper-bounds the distribution of the partition P defined by $(P, \sigma) = \phi(\mathsf{LeakingWires}(\mathsf{ZeroEnc}, p))$.

The rationale for characterizing $\mathsf{RPZeroEnc}$ under the RPP notion is the following. Assume that, after a number of non-leaking iterations, we get a distribution $[\![z]\!] = \sigma(\mathsf{PartitionZeroEnc}(P))$ for some partition P and coordinate permutation σ. If, during a subsequent iteration, the value of the random r leaks, the output $[\![z']\!]$ of this iteration can be expressed as: $[\![z']\!] = [\![z]\!] + [\![w]\!]$ where $[\![w]\!]$ is a vector with one coordinate set to r, one to $-r$, and the rest to 0. Given the leakage of r, one can perfectly simulate this $[\![w]\!]$. This gives the intuition behind the RPP property of $\mathsf{RPZeroEnc}$: non-leaking iterations contribute to reducing the partition structure of $[\![u]\!] = \sigma(\mathsf{PartitionZeroEnc}(P))$, while leaking iterations result in updates to the $[\![w]\!]$ component, accumulating the leaked randomness. We note that the leakage can also concern z_{i_1} or z_{i_2} which results in an update of both $[\![u]\!]$ (or (P, σ)) and $[\![w]\!]$.

Before formally establishing the RPP security of $\mathsf{RPZeroEnc}$ in the next lemma, we introduce the concept of *transition set* of a partition. Assume the output of an iteration corresponds to a partition P. The transition set $\mathcal{T}_j(P)$ contains all partitions P' that can be obtained after an iteration with $i_1 = i_2 = j$ while the transition set $\mathcal{T}_{j_1,j_2}(P)$ contains all partitions P' that can be obtained when $i_1 = j_1 \neq i_2 = j_2$. These transition sets are identified in Table 1 and Table 2 , considering the different possible scenarios in terms of leakage. In these tables, expressions such as $(z_{i_1} \oplus z_{i_2}) \wedge \overline{r}$ in the leakage column should be interpreted as a scenario where either z_{i_1} or z_{i_2} leaks (but not both) and r does not leak.

Lemma 6 ($\mathsf{RPZeroEnc}$ **is RPP).** *Let n, γ in $\mathbb{N}$ and $p \in [0, 1]$. The n-share zero-encoding gadget $\mathsf{RPZeroEnc}^\gamma$ is $(p, \mathcal{E}^\gamma)$-RPP for some distribution envelope $\mathcal{E}^\gamma$ defined over $\mathcal{P}(n)$ constructively as follows. For $\gamma = 0$, we simply have $\mathcal{E}^0(\mathbf{1}) = 1$ and $\mathcal{E}^0(P) = 0$ for all $P \in \mathcal{P}(n)\backslash\{\mathbf{1}\}$, where $\mathbf{1} = (1, 1, \ldots, 1)$. Then, for all $\gamma > 0$, we define $\mathcal{E}^{\gamma+1}$ from $\mathcal{E}^\gamma$ as follows:*

1. *for all $P \in \mathcal{P}(n)$, we initiate $\mathcal{E}^{\gamma+1}(P)$ to 0,*
2. *for all $P \in \mathcal{P}(n)$, for all $j \in \{1, \ldots, |P|\}$, for all $P' \in \mathcal{T}_j(P)$ from Table 1,*
$$\mathcal{E}^{\gamma+1}(P') = \mathcal{E}^{\gamma+1}(P') + \mathbb{P}(P \to P') \cdot \mathcal{E}^\gamma(P),$$
3. *for all $P \in \mathcal{P}(n)$, for all $j_1, j_2 \in \{1, \ldots, |P|\}$ with $j_1 < j_2$, for all $P' \in \mathcal{T}_{j_1,j_2}(P)$ from Table 2,*
$$\mathcal{E}^{\gamma+1}(P') = \mathcal{E}^{\gamma+1}(P') + \mathbb{P}(P \to P') \cdot \mathcal{E}^\gamma(P).$$

Proof. The $\mathsf{RPZeroEnc}$ algorithm begins with a shared vector initialized to $(0, \ldots, 0)$. Since all shares are constant and known, and based on Definition 11, the only possible partition with strictly positive probability is the vector of ones.

At iteration $\gamma + 1$, two indices i_1 and i_2 are drawn uniformly at random from $\{1, \ldots, n\}$. These indices can either belong to the same part n_j of a partition $P = (n_1, n_2, \ldots, n_k)$ with probability

$$p_s = \frac{n_j \cdot (n_j - 1)}{n(n-1)}$$

or to two different parts n_{j_1} and n_{j_2} with probability

$$p_d = 2 \cdot \frac{n_{j_1} \cdot n_{j_2}}{n(n-1)}.$$

Next, a random value r is drawn uniformly from $\mathbb{K}$ and added to the two elements z_{i_1} and z_{i_2}. If neither of these elements leaks and both indices belong to the same part n_j, then the partition remains unchanged. However, if all these three elements are leaking and both indices belong to the same part n_j, then z_{i_1} and z_{i_2} are excluded from their zero encodings, forming two new independent parts. This scenario corresponds to the first row of Table 1 and occurs with probability

$$p_s \cdot p^2 \cdot (1 - (1-p)^3),$$

where z_{i_1} and z_{i_2} each leak with probability p, and the value r, being manipulated three times, leaks with probability $1 - (1 - p)^3$.

All possible leakage scenarios, whether the indices are drawn from the same part or from different parts, are detailed in Tables 1 and 2. This demonstrates the construction of our envelope. $\square$

Table 1: Intermediate probabilities for Lemma 6, starting from a partition $P = (n_1, \ldots, n_k)$ and when the drawn indices i_1 and i_2 belong to the same subset of cardinal n_j where $p_s := \frac{n_j \cdot (n_j - 1)}{n(n-1)}$.

Leakage	New partition $P' \in \mathcal{T}_j(P)$	Probability $\mathbb{P}(P \to P')$
$z_{i_1} \wedge z_{i_2} \wedge r$	$(1, 1, n_1, \ldots, n_{j-1}, n_j - 2, n_{j+1}, \ldots, n_k)$	$p_s \cdot p^2 \cdot (1 - (1-p)^3)$
$z_{i_1} \wedge z_{i_2} \wedge \overline{r}$	$(2, n_1, \ldots, n_{j-1}, n_j - 2, n_{j+1}, \ldots, n_k)$	$p_s \cdot p^2 \cdot (1-p)^3$
$(z_{i_1} \oplus z_{i_2}) \wedge r$	$(1, n_1, \ldots, n_{j-1}, n_j - 1, n_{j+1}, \ldots, n_k)$	$2 \cdot p_s \cdot (p - p^2) \cdot (1 - (1-p)^3)$
$\big((z_{i_1} \oplus z_{i_2}) \wedge \overline{r}\big)$ $\vee(\overline{z_{i_1}} \wedge \overline{z_{i_2}})$	$(n_1, \ldots, n_k)$	$2 \cdot p_s \cdot p \cdot (1-p)^4$ $+ p_s \cdot (1-p)^2$

Table 2: Intermediate probabilities for Lemma 6, starting from a partition $P = (n_1, \ldots, n_k)$ and when the drawn indices i_1 and i_2 belong to two blocks of cardinals n_{j_1} and n_{j_2}, where $p_d := 2 \cdot \frac{n_{j_1} \cdot n_{j_2}}{n(n-1)}$.

Leakage	New partition $P' \in \mathcal{T}_{j_1,j_2}(P)$	Probability $\mathbb{P}(P \to P')$
$z_{i_1} \wedge z_{i_2} \wedge r$	$(1, 1, n_1, \ldots, n_{j_1-1}, n_{j_1} - 1, n_{j_1+1}, \ldots, n_{j_2-1}, n_{j_2} - 1, n_{j_2+1}, \ldots, n_k)$	$p_d \cdot p^2 \cdot (1 - (1-p)^3)$
$z_{i_1} \wedge z_{i_2} \wedge \overline{r}$	$(2, n_1, \ldots, n_{j_1-1}, n_{j_1} - 1, n_{j_1+1}, \ldots, n_{j_2-1}, n_{j_2} - 1, n_{j_2+1}, \ldots, n_k)$	$p_d \cdot p^2 \cdot (1-p)^3$
$z_{i_1} \wedge \overline{z_{i_2}} \wedge r$	$(1, n_1, \ldots, n_{j_1-1}, n_{j_1} - 1, n_{j_1+1}, \ldots, n_k)$	$p_d \cdot p \cdot (1-p) \cdot (1 - (1-p)^3)$
$\overline{z_{i_1}} \wedge z_{i_2} \wedge r$	$(1, n_1, \ldots, n_{j_2-1}, n_{j_2} - 1, n_{j_2+1}, \ldots, n_k)$	$p_d \cdot p \cdot (1-p) \cdot (1 - (1-p)^3)$
$z_{i_1} \wedge \overline{z_{i_2}} \wedge \overline{r}$	$(n_1, \ldots, n_{j_1-1}, n_{j_1} - 1, n_{j_1+1}, \ldots, n_{j_2-1}, n_{j_2} + 1, n_{j_2+1}, \ldots, n_k)$	$p_d \cdot p \cdot (1-p)^4$
$\overline{z_{i_1}} \wedge z_{i_2} \wedge \overline{r}$	$(n_1, \ldots, n_{j_1-1}, n_{j_1} + 1, n_{j_1+1}, \ldots, n_{j_2-1}, n_{j_2} - 1, n_{j_2+1}, \ldots, n_k)$	$p_d \cdot p \cdot (1-p)^4$
$\overline{z_{i_1}} \wedge \overline{z_{i_2}} \wedge r$	$(n_1, \ldots, n_k)$	$p_d \cdot (1-p)^2 \cdot (1 - (1-p)^3)$
$\overline{z_{i_1}} \wedge \overline{z_{i_2}} \wedge \overline{r}$	$(n_1, \ldots, n_{j_1-1}, n_{j_1+1}, \ldots, n_{j_2-1}, n_{j_1} + n_{j_2}, n_{j_2+1}, \ldots, n_k)$	$p_d \cdot (1-p)^5$

In order to simplify the security analysis of RPRefresh, we now introduce an artificial extension of RPZeroEnc, called RPZeroEnc$_+$, in which all the output shares are individually manipulated after γ iterations. In terms of leakage, this means that all the output shares of RPZeroEnc can now leak with probability p, in addition to the internal leakage already accounted for in Lemma 6.

Lemma 7 **(RPZeroEnc$_+$ is RPP).** *Let n, γ in $\mathbb{N}$ and $p \in [0,1]$. The n-share zero-encoding gadget* RPZeroEnc$_+^{n,\gamma}$ *is $(p, \mathcal{E}^\gamma)$-RPP for some distribution envelope $\mathcal{E}^\gamma$ defined over $\mathcal{P}(n)$ constructively as follows. The distribution envelope $\mathcal{E}^\gamma$ is initialized to 0 for all the possible partitions. Then, for all k, for all partition $P = (n_1, \ldots, n_k)$ in $\mathcal{P}(n)$, for all $(d_1, \ldots, d_k)$ such that $d_1 \leq n_1, \ldots, d_k \leq n_k$,*

$$\mathcal{E}^\gamma(P') = \mathcal{E}^\gamma(P') + \mathcal{E}^\gamma_{\mathsf{RPZeroEnc}}(P) \cdot \prod_{i=1}^{k} \binom{n_i}{d_i} \cdot p^{d_i} \cdot (1-p)^{n_i - d_i}$$

with $\mathcal{E}^\gamma_{\mathsf{RPZeroEnc}}$ the distribution envelope for the RPP property of RPZeroEnc *given in Lemma 6 and $P' = (1, \ldots, 1, n_1 - d_1, \ldots, n_k - d_k)$.*

Proof. Let us consider a partition $P = (n_1, \ldots, n_k)$ for some $k \in [1, n]$. We recall that the elements n_i for $i \in [1, k]$ represent the cardinals of shares of $[\![z]\!]$ which form a perfect zero-encoding given the probed wires.

We begin by computing the intermediate probability p_{int} that for each part n_i of P, exactly $d_i (\leq n_i)$ shares of $[\![z]\!]$ leak. Since each share of $[\![z]\!]$ leaks independently with probability p, we apply the binomial distribution to obtain:

$$p_{\text{int}} = \prod_{i=1}^{k} \binom{n_i}{d_i} \cdot p^{d_i} \cdot (1-p)^{n_i - d_i}.$$

As a result, from partition P, we derive a new partition $P' = (1, \ldots, 1, n_1 - d_1, \ldots, n_k - d_k)$ with probability p_{int}.

Since the probability of obtaining P at the output of $\mathsf{RPZeroEnc}^{\gamma}$ is independent of p_{int}, the joint probability is simply the product of these two probabilities.

Finally, the probability of obtaining P' at the output of $\mathsf{RPZeroEnc}_{+}^{n,\gamma}$ is the sum of the probabilities to have a partition P at the output of $\mathsf{RPZeroEnc}^{\gamma}$ ($\mathcal{E}^{\gamma}_{\mathsf{RPZeroEnc}}(P)$) and a final leakage that transform P into P'. ☐

We now introduce the major statement of this section.

Theorem 1 **(**$\mathsf{RPRefresh}$ **is Cardinal-RPC).** *Let $n, \gamma \in \mathbb{N}$, $p \in [0,1]$. The n-share gadget $\mathsf{RPRefresh}^{\gamma}$ is $(p, (\mathcal{E}_t)_{t \in [0,n]})$-cardinal-RPC with $(\mathcal{E}_t)_{t\in[0,n]}$ defined over $[0,n]$ as follows. For all $t_{in}, t_{out} \in [0,n]$,*

$$\mathcal{E}_{t_{out}}(t_{in}) = \sum_{i=0}^{t_{in}} q(i, t_{out}) \cdot \binom{n-i}{t_{in}-i} \cdot p^{t_{in}-i} \cdot (1-p)^{n-t_{in}}$$

where $q(i, t_{out})$ is defined constructively as follows:

1. *For all $i, t_{out} \in [0,n]$, initialize $q(i, t_{out}) \leftarrow 0$.*
2. *For every $P = (n_1, \ldots, n_k) \in \mathcal{P}(n)$, for every $d_1 \in [0, n_1]$, $\ldots$, $d_k \in [0, n_k]$, update*

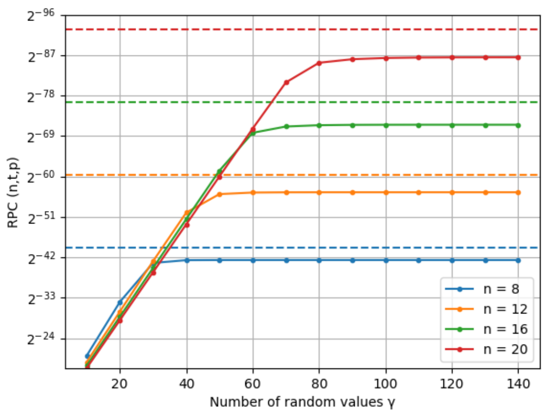
$$q(i, t_{out}) \leftarrow q(i, t_{out}) + \mathcal{E}^{\gamma}_{\mathsf{RPZeroEnc}_{+}}(P) \cdot \frac{\binom{n_1}{d_1} \cdots \binom{n_k}{d_k}}{\binom{n}{t_{out}}}$$

for $t_{out} = d_1 + \cdots + d_k$, $i = n_1 \cdot \mathbb{1}_{d_1 = n_1} + \cdots + n_k \cdot \mathbb{1}_{d_k = n_k}$ and $\mathcal{E}^{\gamma}_{\mathsf{RPZeroEnc}_{+}}$ taken from Lemma 7.

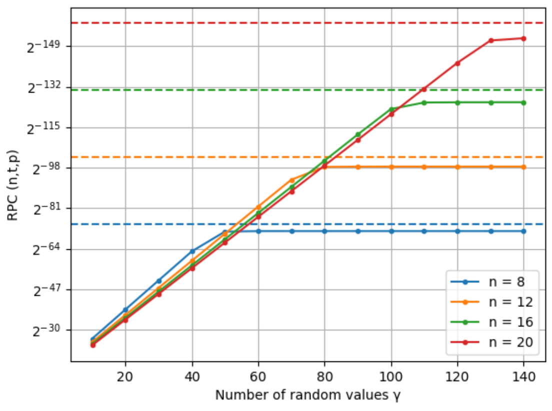
Intuitively, $q(i, t_{\text{out}})$ upper bounds the probability that i input shares are required to simulate the leakage on the internal variables of $\mathsf{RPRefresh}$ (without considering the input shares) along with the given t_{out} output shares.

Proof. From Lemmas 6 and 7, we have the probabilities associated to each partition of zero encoding cardinals at the output of $\mathsf{RPZeroEnc}_{+}^{\gamma}$. In the cardinal-RPC property, the attacker is been given t_{out} output shares. When the indices of some of these output shares recover all the indices of a part of the partition at the output of $\mathsf{RPZeroEnc}_{+}^{\gamma}$, then the attacker obtains the corresponding input shares. Conversely, if the indices of the output shares given to the attacker do not entirely recover the indices of a part of the partition at the output of $\mathsf{RPZeroEnc}_{+}^{\gamma}$, the corresponding input shares are perfectly masked and hence not required for the simulation. We can compute the related probability with the distribution of a multihypergeometric law. Namely, for a partition $P = (n_1, \ldots, n_k)$ for some $k \in [1, n]$, given t_{out} output shares, the probability that i input shares are required to simulate the leakage on the internal variables of $\mathsf{RPRefresh}$ (without considering the leakage on the input shares) is given by

$$q(i, t_{\text{out}}) = \sum_{\substack{P=(n_1,\ldots,n_k)\\ P \in \mathcal{P}(n)}} \mathcal{E}^{\gamma}_{\mathsf{RPZeroEnc}_{+}}(P) \sum_{\substack{(d_1,\ldots,d_k)\\ d_1+\cdots+d_k = t_{\text{out}}\\ n_1 \cdot \mathbb{1}_{d_1=n_1} + \cdots + n_k \cdot \mathbb{1}_{d_k=n_k} = i}} \frac{\binom{n_1}{d_1} \cdots \binom{n_k}{d_k}}{\binom{n}{t_{\text{out}}}}$$



(a) $p = 2^{-10}$



(b) $p = 2^{-16}$

Fig. 3: Computed Threshold-RPC-AI security advantage ε of RPRefresh with fixed $(p, n, \frac{n}{2})$ for $n \in \{8, 12, 16, 20\}$ and $p \in \{p = 2^{-10}, p = 2^{-16}\}$ as a function of the number of iterations γ. The dotted lines corresponds to the maximum achievable security advantage.

in which each d_i represent the number of given output shares whose indices belong to the i^{th} encoding. When $i \leq t_{\text{in}}$, this probability is to be multiplied by the probability to recover the remaining input shares according to their leakage. Namely, the probability to recover j input shares among $n - i$ is given by

$$\binom{n-i}{j} \cdot p^j \cdot (1-p)^{n-i-j},$$

which concludes the proof. □

Complexity of the envelope computation. The above results provide explicit formulas to compute the envelopes characterizing the cardinal-RPC security of our refresh. Following Lemma 6, evaluating those formulas has a complexity $\mathcal{O}(\gamma \cdot |\mathcal{P}(n)| \cdot n^2)$ by iterating over the γ iterations of the refresh, the $|\mathcal{P}(n)|$ partitions of n, and the different pairs of elements within a partition (whose number is lower than n^2). According to the Hardy-Ramanujan asymptotic formula, the number of integer partitions of n can be approximated as

$$|\mathcal{P}(n)| \approx \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right)$$

which is subexponential in n. For instance, $|\mathcal{P}(8)| = 22$, $|\mathcal{P}(16)| = 231$, $|\mathcal{P}(32)| = 8\,349$, $|\mathcal{P}(64)| < 2^{21}$. In practice, using our (non-optimized) Python implementation of this computation results in, *e.g.*, less than 40 milliseconds for $(n, \gamma) = (8, 50)$, around 9 seconds for $(n, \gamma) = (16, 100)$, around 3 minutes for $(n, \gamma) = (20, 120)$ on a regular laptop.

Figure 3 illustrates the threshold-RPC security advantage of RPRefresh for various numbers of shares n, with $t = \frac{n}{2}$, and for two leakage probabilities, $p = 2^{-10}$ (Figure 3a) and $p = 2^{-16}$ (Figure 3b), as a function of the number of random values γ. Those results have been obtained and validated by two independent implementations (from two different developers) of the envelope evaluation. We observe that, after a sufficient number of random values, the security advantage stabilizes, reaching a steady state. This steady state is logically lower bounded by the probability of observing $t + 1$ input shares from a simple random probing leakage of these shares (i.e., each share leaks with probability p), which is given by $\binom{n}{t+1} \cdot p^{t+1} \cdot (1-p)^{n-t-1}$. In the figure, this probability is represented by dotted lines in colors corresponding to n.

4.2 Simple Sharewise Cardinal-RPC Gadgets

With RPRefresh now defined, we proceed to construct a set of foundational gadgets that will serve as building blocks for many common schemes. The proposed constructions follow a very natural blueprint for defining a sharewise (linear multiplication, addition, or copy) gadget from a refresh gadget, called Refresh, such as already proposed in [10,11]. Our main contribution here is to prove the cardinal-RPC security of these gadgets from that of the underlying refresh.

Linear Multiplication Gadget. The linear multiplication gadget was defined in [11] with a linear multiplication preceding the refreshing. For consistency with the other simple gadgets, we define it with a refreshing of the shares preceding the linear multiplication with the constant: $\mathrm{G}_{\text{cmult}}(\llbracket x \rrbracket, c) = \mathsf{Refresh}(\llbracket x \rrbracket) \cdot c$.

Algorithm 3 $\mathrm{G}_{\text{cmult}}(\llbracket x \rrbracket, c)$ adapted from [11]

```
Input: $\llbracket x \rrbracket = (x_1, x_2, \ldots, x_n)$, constant $c \in \mathbb{K}$
Output: $\llbracket z \rrbracket = (z_1, z_2, \ldots, z_n)$ such that $z_1 + z_2 + \cdots + z_n = c \cdot (x_1 + x_2 + \cdots + x_n)$
 1. $\llbracket z \rrbracket \leftarrow \mathsf{Refresh}(\llbracket x \rrbracket)$
 2. for $i = 1$ to $n$ do
 3.     $z_i \leftarrow z_i \cdot c$
 4. end for
 5. return $\llbracket z \rrbracket$
```

Lemma 8 ($\mathrm{G}_{\text{cmult}}$ is Cardinal-RPC). *Let $n \in \mathbb{N}$. Let $\mathsf{Refresh}\colon \mathbb{K}^n \to \mathbb{K}^n$ be an n-share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t\in[0,n]})$-cardinal-RPC for some collection of envelopes $(\mathcal{E}_t^R)_{t\in[0,n]}$ and some probability $p \in [0,1]$. Then, $\mathrm{G}_{\text{cmult}}$ is $(p, (\mathcal{E}_t^{\text{cmult}})_{t\in[0,n]})$-cardinal-RPC where, for all $t_{out} \in [0,n]$, $\mathcal{E}_{t_{out}}^{\text{cmult}}$ is defined, for all $t_{in} \in [0,n]$, as:*

$$\mathcal{E}_{t_{out}}^{\text{cmult}}(t_{in}) = \sum_{i=0}^{n-t_{out}} \binom{n-t_{out}}{i} \cdot p^i \cdot (1-p)^{n-t_{out}-i} \cdot \mathcal{E}_{t_{out}+i}^R(t_{in}).$$

Proof. The only additional wires that can potentially leak in $\mathrm{G}_{\text{cmult}}$ compared to Refresh are the output shares of Refresh. The probability that t_{in} input shares are required to simulate the leakage and the final output shares is therefore obtained by computing the product of the leakage probability of i such wires with the probability of Refresh, considering an output composed of the global output plus these extra leaking wires. The probability that i output shares of Refresh leak, out of the $n - t_{\text{out}}^{\text{cmult}}$ output shares that remain unknown to the attacker, is given by

$$\binom{n-t_{\text{out}}}{i} \cdot p^i \cdot (1-p)^{n-t_{\text{out}}-i},$$

which completes the proof. □

Addition Gadget. G_{add} is the simple addition gadget from [10]. It performs $\mathrm{G}_{\text{add}}(\llbracket x \rrbracket, \llbracket y \rrbracket) = \mathsf{Refresh}(\llbracket x \rrbracket) + \mathsf{Refresh}(\llbracket y \rrbracket)$.

Algorithm 4 $\mathrm{G}_{\text{add}}(\llbracket x \rrbracket, \llbracket y \rrbracket)$ from [10]

```
Input: $\llbracket x \rrbracket = (x_1, x_2, \ldots, x_n)$ and $\llbracket y \rrbracket = (y_1, y_2, \ldots, y_n)$
Output: $\llbracket z \rrbracket = (z_1, z_2, \ldots, z_n)$ such that $z_1 + z_2 + \cdots + z_n = (x_1 + y_1) + (x_2 + y_2) + \cdots + (x_n + y_n)$
 1. $\llbracket x' \rrbracket \leftarrow \mathsf{Refresh}(\llbracket x \rrbracket)$
 2. $\llbracket y' \rrbracket \leftarrow \mathsf{Refresh}(\llbracket y \rrbracket)$
 3. for $i = 1$ to $n$ do $z_i \leftarrow x'_i + y'_i$
 4. return $\llbracket z \rrbracket$
```

Lemma 9 ($\mathbf{G_{add}}$ is Cardinal-RPC). *Let $n \in \mathbb{N}$. Let $\mathsf{Refresh}\colon \mathbb{K}^n \to \mathbb{K}^n$ be an n-share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t\in[0,n]}))$-cardinal-RPC for some collection envelopes $(\mathcal{E}_t^R)_{t\in[0,n]}$ and some probability $p \in [0,1]$. Then, G_{add} is $(p, (\mathcal{E}_t^{\mathrm{add}})_{t\in[0,n]})$-cardinal-RPC where, for all $t_{out} \in [0,n]$, $\mathcal{E}_{t_{out}}^{\mathrm{add}}$ is defined, for all $\boldsymbol{t}_{in} = (t_{in,1}, t_{in,2}) \in [0,n]^2$, as*

$$\mathcal{E}_{t_{out}}^{\mathrm{add}}(\boldsymbol{t}_{in}) = \sum_{i_1=0}^{n-t_{out}} \binom{n-t_{out}}{i_1} \cdot p^{i_1} \cdot (1-p)^{n-t_{out}-i_1} \cdot \mathcal{E}_{t_{out}+i_1}^R(t_{in,1}) \cdot \sum_{i_2=0}^{n-t_{out}} \binom{n-t_{out}}{i_2} \cdot p^{i_2} \cdot (1-p)^{n-t_{out}-i_2} \cdot \mathcal{E}_{t_{out}+i_2}^R(t_{in,2}).$$

Proof. We assume (this is the worst case) that all the t_{out} output shares given to the attacker require the corresponding $2 \cdot t_{\mathrm{out}}$ output shares of the two instances of $\mathsf{Refresh}$ to be simulated. As for $\mathrm{G}_{\mathrm{cmult}}$, the output shares to simulate for each instance of $\mathsf{Refresh}$ are completed by the possible leakage at the input of the addition. In that case, the probabilities for both instances of $\mathsf{Refresh}$ to require $t_{\mathrm{in},\alpha}$ input shares (for $\alpha \in \{1,2\}$) are independent given the total number of output shares to simulate with the leakage and we obtain the above envelope. □

In general, the addition gadget $\mathrm{G}_{\mathrm{add}}$ can involve two distinct refresh gadgets, each with a different envelopes collection. Lemma 9 can be easily updated to account for this by getting use of the first envelopes collection in the first sum and the second envelopes collection in the second sum.

Copy Gadget. $\mathrm{G}_{\mathrm{copy}}$ is the simple copy gadget from [10]. It computes $\mathrm{G}_{\mathrm{copy}}(\llbracket x \rrbracket) = (\mathsf{Refresh}(\llbracket x \rrbracket), \mathsf{Refresh}(\llbracket x \rrbracket))$.

Algorithm 5 $\mathrm{G}_{\mathrm{copy}}(\llbracket x \rrbracket)$ from [10]

```
Input: $\llbracket x \rrbracket = (x_1, x_2, \ldots, x_n)$
Output: $\llbracket z^1 \rrbracket = (z_1^1, z_2^1, \ldots, z_n^1)$ and $\llbracket z^2 \rrbracket = (z_1^2, z_2^2, \ldots, z_n^2)$ such that $z_1^1 + z_2^1 + \cdots + z_n^1 = z_1^2 + z_2^2 + \cdots + z_n^2 = x_1 + x_2 + \cdots + x_n$
1. $\llbracket z^1 \rrbracket \leftarrow \mathsf{Refresh}(\llbracket x \rrbracket)$
2. $\llbracket z^2 \rrbracket \leftarrow \mathsf{Refresh}(\llbracket x \rrbracket)$
3. return $(\llbracket z^1 \rrbracket, \llbracket z^2 \rrbracket)$
```

Lemma 10 ($\mathbf{G_{copy}}$ is Cardinal-RPC). *Let $n \in \mathbb{N}$. Let $\mathsf{Refresh}\colon \mathbb{K}^n \to \mathbb{K}^n$ be an n-share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t\in[0,n]}))$-cardinal-RPC for some collection of envelopes $(\mathcal{E}_t^R)_{t\in[0,n]}$ and some probability $p \in [0,1]$. Then, G_{copy} is $(p, (\mathcal{E}_{\boldsymbol{t}}^{\mathrm{copy}})_{\boldsymbol{t}\in[0,n]^2})$-cardinal-RPC where, for all $\boldsymbol{t}_{out} \in [0,n]^2$, $\mathcal{E}_{\boldsymbol{t}_{out}}^{\mathrm{copy}}$ is defined, for all $t_{in} \in [0,n]$, as:*

$$\mathcal{E}_{\boldsymbol{t}_{out}}^{\mathrm{copy}}(t_{in}) = \sum_{0\le i\le n} \mathcal{E}_{t_{out,1}}^R(i) \cdot \mathcal{E}_{t_{out,2}}^R(t_{in} - i).$$

Proof. From the property of $\mathsf{Refresh}$, when t_{out}^R output shares are given to the attacker, the probability that t_{in}^R input shares are required for the simulation of both these output shares and the leakage is upper bounded by

$$\mathcal{E}_{t_{\mathrm{out}}}^R(t_{\mathrm{in}}^R).$$

In the context of $\mathrm{G}_{\mathrm{copy}}$, we have two instances of $\mathsf{Refresh}$. The probability that t_{in} input shares are required for the simulation of the leakage and the output shares for $\mathrm{G}_{\mathrm{copy}}$ is thus computed from the probability that one instance of $\mathsf{Refresh}$ requires i input shares for its own simulation and the second one $t_{\mathrm{in}} - i$, hence the final result. □

Algorithm 6 $\mathrm{G}_{\mathrm{decode}}(\llbracket x \rrbracket)$

```
Input: $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n)$
Output: $x = x_1 + x_2 + \dots + x_n$
 1. $\llbracket t \rrbracket \leftarrow \mathsf{Refresh}(\llbracket x \rrbracket)$
 2. $x \leftarrow \mathsf{ShareSum}(\llbracket t \rrbracket)$
 3. return $x$
```

Algorithm 7 $\mathsf{ShareSum}(\llbracket x \rrbracket)$

```
Input: $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n)$
Output: $x = x_1 + x_2 + \dots + x_n$
 1. if $n = 2$ then return $x_1 + x_2$
 2. if $n = 3$ then return $\mathsf{ShareSum}((x_1, x_2)) + x_3$
 3. $y \leftarrow \mathsf{ShareSum}(x_1, x_2, \dots, x_{\lfloor \frac{n}{2} \rfloor})$
 4. $z \leftarrow \mathsf{ShareSum}(x_{\lfloor \frac{n}{2} \rfloor + 1}, x_{\lfloor \frac{n}{2} \rfloor + 2}, \dots, x_n)$
 5. return $x = y + z$
```

The cardinal-RPC advantage of $\mathrm{G}_{\mathrm{copy}}$ is tighter when the gadget is instantiated with uniformly cardinal-RPC refresh gadgets, as shown in Corollary 1[5]. Similar to the addition gadget, $\mathrm{G}_{\mathrm{copy}}$ can also be instantiated with two distinct refresh gadgets, and the security results can be easily adapted accordingly.

Corollary 1. *Let $n \in \mathbb{N}$. Let* $\mathsf{Refresh}$*: $\mathbb{K}^n \to \mathbb{K}^n$ be an n-share $(p, (\mathcal{E}_t^R)_{t \in [0,n]})$-uniformly cardinal-RPC refresh gadget for some collection of envelopes $(\mathcal{E}_t^R)_{t \in [0,n]}$ and some probability $p \in [0,1]$ (See Definition 16). Then, G_{copy} is $(p, (\mathcal{E}_{\boldsymbol{t}}^{\mathrm{copy}})_{\boldsymbol{t} \in [0,n]^2})$-cardinal-RPC where, for all $\boldsymbol{t}_{out} \in [0,n]^2$, $\mathcal{E}_{\boldsymbol{t}_{out}}^{\mathrm{copy}}$ is defined as:*

$$\forall t_{in} \in [0,n], \quad \mathcal{E}_{\boldsymbol{t}_{out}}^{\mathrm{copy}}(t_{in}) = \sum_{\substack{0 \le i,j \le n \\ i,j \le t_{in} \le i+j}} \mathcal{E}_{t_{out,1}}^R(i) \cdot \mathcal{E}_{t_{out,2}}^R(j) \cdot \frac{\binom{i}{j-(t_{in}-i)} \cdot \binom{n-i}{t_{in}-i}}{\binom{n}{j}}.$$

Proof. We use the property of uniformly cardinal-RPC gadgets for which the sets of input shares required for the simulation are equiprobable for equal cardinals. Specifically, we can compute the probability that two sets of input shares of sizes i and j, required to simulate the leakage of both refresh gadgets and the outputs, represent t_{in} input shares. The number of common elements between both sets follows an hypergeometric law. Namely, the probability to have $i + j - t_{\mathrm{in}}$ common elements is given by

$$\frac{\binom{i}{j-(t_{\mathrm{in}}-i)} \cdot \binom{n-i}{t_{\mathrm{in}}-i}}{\binom{n}{j}},$$

hence the final result. □

4.3 Output Decoding Gadget

An output decoding gadget reconstructs values from their shares and outputs them in plain form. It is involved for variables that should be unmasked before being returned such as the public key in the key generation algorithm or the signature in the signing algorithm. In the probing model, it typically consists of a refresh gadget followed by a straightforward addition of all the shares. To our knowledge, it has not yet been analyzed within the random probing model. In this work, we propose combining a refresh gadget with a recursive addition of the shares (yielding a binary tree of additions), as presented in Algorithm 6.

Since the output of this gadget is intended to be revealed publicly, its security requirements differ from those of the previous gadgets. Specifically, in the context of composition, we are interested in the *cardinal RPC with public output* (Cardinal-RPC-O). This notion is a special case of the Cardinal-RPC-AI-O notion of Definition 7.

Theorem 2 ($\mathsf{ShareSum}$ **is Cardinal-RPC-O).** *Let $n \in \mathbb{N}$.* $\mathsf{ShareSum}$ *from Algorithm 7 is $(p, \mathcal{E})$-cardinal-RPC-O where $\mathcal{E}$ is defined from the distribution $\mathcal{D}^{(n)}$ over $[0,n]^2$ as:*

[5] Note that the refresh gadget instantiated in Section 4.1 does not achieve this uniform property. Hence, the tightness of Corollary 1 cannot be obtained in Section 6.

1. Sample $(r, s) \leftarrow \mathcal{D}^{(n)}$
2. Return s

The distribution $\mathcal{D}^{(n)}$ *is recursively defined as follows. For every* $n > 1$*:*

1. Sample $(r_1, s_1) \leftarrow \mathcal{D}^{(\lfloor n/2 \rfloor)}$
2. Sample $(r_2, s_2) \leftarrow \mathcal{D}^{(\lceil n/2 \rceil)}$
3. Return

$$(r, s) = \begin{cases} \big(n,\ \min(r_1 + r_2, s_1 + \lceil n/2 \rceil, s_2 + \lfloor n/2 \rfloor)\big) & \textit{with proba. } p \\ \big(r_1 + r_2,\ \min(r_1 + r_2, s_1 + \lceil n/2 \rceil, s_2 + \lfloor n/2 \rfloor)\big) & \textit{with proba. } 1 - p \end{cases}$$

And for $n = 1$*:*

$$\begin{cases} \textit{Return } (1, 1) \textit{ with proba. } p \\ \textit{Return } (0, 0) \textit{ with proba. } 1 - p \end{cases}$$

Proof. Consider the circuit defined by ShareSum which is represented as a binary tree, where the n leaves are the input shares of the circuits and the root the output. Each node leaks with probability p and we want to identify the number of input shares which are necessary for a perfect simulation of the leakage, given that the output x is provided to the simulator. By definition of ShareSum, we have the following invariant: an addition node in the tree with n' incoming leaves has two incoming sub-trees, the left one having $\lfloor n'/2 \rfloor$ leaves and the right one having $\lceil n'/2 \rceil$ leaves. The simulation strategy is illustrated on Figure 4.

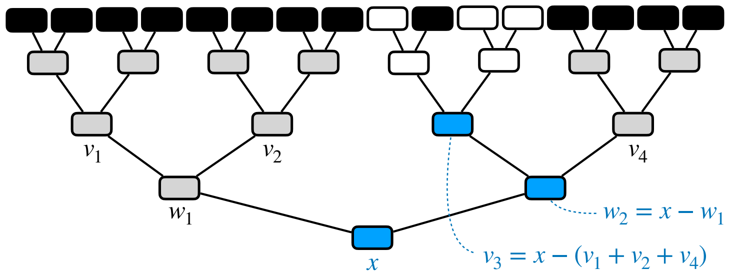


Fig. 4: Illustration of the simulation strategy for the ShareSum gadget. White nodes do not leak and hence do not require simulation. Black leaves either leak or are required for the simulation gray nodes. All the gray nodes can be simulated from the leaves (whether they leak or not). All the blue nodes are "simulated from the root", meaning they are simulated from the output x plus some of the gray nodes. In this example, one needs $n - 3$ input shares for the simulation.

We say that a node can be "simulated from the root" whenever its parent node can be simulated from the root and its sibling node is the root of a sub-tree for which all the leaves are input of the simulation. In the example of Figure 4, w_2 and v_3 satisfy this definition. This make them simulatable from the root x and some gray nodes which are themselves simulatable from the given leaves. In practice, the pattern represented in Figure 4 can be used for the simulation with any blue path going from the root to one intermediate node (v_3 in the example). The sibling sub-trees of this path (gray nodes in the example) are all fully simulated from their leaves which are required to the simulation. The remaining sub-tree with highest blue node as root is simulated from the required subset of its leaves, depending on the leaking nodes.

To identify the minimal set of input shares required for the simulation one can define a recursive walk from the root to the leaves as follows. Let us denote $T = (R, T_1, T_2)$ any sub-tree with root R, left sub-tree T_1 and right sub-tree T_2. It $T = (R, \emptyset, \emptyset)$, then R is a leaf. We denote $\mathsf{leak}(R)$ the event that the node R leaks (which occurs with probability p) and $\mathsf{leaves}(T)$ the set of leaves of a sub-tree T. We further denote b, a Boolean value indicating whether the root of the current sub-tree T can be simulated from the root (which depends on what happened before in the walk). We obtain the following recursive function:

```
NeededLeaves($T, b$):
1. $(R, T_1, T_2) = T$
2. If $T_1 = T_2 = \emptyset$ ($R$ is a leaf),
   (a) If leak($R$) ($R$ leaks), return $\{R\}$.
   (b) If $\neg$leak($R$) ($R$ does not leak), return $\emptyset$.
3. Let $\mathcal{L}_1$ = NeededLeaves($T_1, 1$) $\cup$ leaves($T_2$) (make $T_2$ a gray sub-tree, only possible if $b = 1$).
4. Let $\mathcal{L}_2$ = NeededLeaves($T_2, 1$) $\cup$ leaves($T_1$) (make $T_1$ a gray sub-tree, only possible if $b = 1$).
5. Let $\mathcal{L}_3$ = NeededLeaves($T_1, 0$) $\cup$ NeededLeaves($T_2, 0$) (identify leaves to simulate $T_1$ and $T_2$ sub-trees, only possible if $R$ does
   not leak).
6. If $b = 0$ ($R$ cannot be simulated from the root):
   (a) If leak($R$), return leaves($T$) (all the leaves are necessary to simulate $R$).
   (b) If $\neg$leak($R$), return $\mathcal{L}_3$ (leaves to simulate $T_1$ and $T_2$).
7. If $b = 1$ ($R$ can be simulated from the root): return either $\mathcal{L}_1$, $\mathcal{L}_2$ or $\mathcal{L}_3$, the set with the smallest cardinality.
```

By calling $\mathsf{NeededLeaves}(T, 1)$ on the full tree T, we obtain the smallest set of leaves that are necessary to simulate all the leaking nodes of the tree according to the simulation strategy exemplified in Figure 4. Consider T as the random variable representing a sub-tree where each node leaks with probability p. Define $\mathcal{R} = \mathsf{NeededLeaves}(T, 0)$ and $\mathcal{S} = \mathsf{NeededLeaves}(T, 1)$. We now argue that the pair $(|\mathcal{R}|, |\mathcal{S}|)$ follows the distribution $\mathcal{D}^{(n_T)}$ defined in Theorem 2, where n_T is the number of leaves of the sub-tree T. The base case is trivial: $(|\mathcal{R}|, |\mathcal{S}|) = (1, 1)$ with probability p (leaf leakage) and $(0, 0)$ otherwise. The general case follows from the description of $\mathsf{NeededLeaves}$, observing that in the definition of $\mathcal{D}^{(n)}$, we have $(r_1, s_1) = (|\mathcal{R}_1|, |\mathcal{S}_1|)$ and $(r_2, s_2) = (|\mathcal{R}_2|, |\mathcal{S}_2|)$ where:

$$(\mathcal{R}_i, \mathcal{S}_i) \sim \big(\mathsf{NeededLeaves}(T_i, 0), \mathsf{NeededLeaves}(T_i, 1)\big) \quad \forall i \in \{1, 2\} \ .$$

□

From Lemma 4, $\mathrm{G}_{\text{decode}}$ is therefore naturally Cardinal-RPC-O. The distribution $\mathcal{D}^{(n)}$ in the above theorem can be efficiently computed through recursion, providing a precise evaluation of the Cardinal-RPC-O security for our decoding gadget.

5 Random Probing Secure Noise Generation

In this section, we provide a random-probing composable version of the noise generation gadget introduced in Raccoon [16]. In such a context, the base ring of the considered arithmetic circuit is $\mathbb{K} = \mathbb{Z}_q$ for some integer q. The principle of the noise generation gadget is to sample uniform random values over a subset of $\mathbb{Z}_q$ and to add them to the input sharing, thereby introducing the noise required in the underlying Module Learning with Errors (MLWE) instance. In our context, these random values are captured as auxiliary inputs and the noise generation gadget simply aims at summing them to the sharings while minimizing the leakage on these values. We evaluate the security and complexity of two constructions –either in chain or in tree structure– when this sum is instantiated with our new addition gadget introduced in Section 4.2.

5.1 Noise Generation Gadget

Algorithm 8 generically describes the gadget responsible for noise addition. Essentially, it relies on a parsing procedure $\mathsf{DivideAI}$, depicted in Algorithm 9, which splits the auxiliary inputs into blocks of n elements from $\mathbb{K}$ and appends zeros to the last block, if necessary. Subsequently, a macro gadget, G_{sum}, is invoked to operate on these blocks of size n. This can be instantiated either with $\mathrm{G}_{\text{sum-chain}}$ from Algorithm 10 or with $\mathrm{G}_{\text{sum-tree}}$ from Algorithm 11. The input sharing is finally added to the output of G_{sum} to produce the final result.

As stated in Lemma 11, $\mathsf{AddNoiseTo}$ is immediately cardinal-RPC if both algorithms G_{sum} and G_{add} are cardinal-RPC, as indicated by the composition result in Lemma 4.

Algorithm 8 AddNoiseTo($[\![x]\!]$, $\mathbf{a}$)

```
Input: $[\![x]\!] = (x_1, x_2, \ldots, x_n)$, auxiliary inputs $\mathbf{a} \in \mathbb{K}^\alpha$
Output: $[\![y]\!] = (y_1, y_2, \ldots, y_n)$ such that $y = x + a_1 + a_2 + \cdots + a_\alpha$
 1. $(\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b}_{\lceil \frac{\alpha}{n} \rceil}) \leftarrow$ DivideAI$(\mathbf{a}, n)$
 2. $\mathbf{z} \leftarrow \mathrm{G_{sum}}(\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b}_{\lceil \frac{\alpha}{n} \rceil})$
 3. return $\mathrm{G_{add}}([\![x]\!], \mathbf{z})$
```

Algorithm 9 DivideAI($\mathbf{a}$, n)

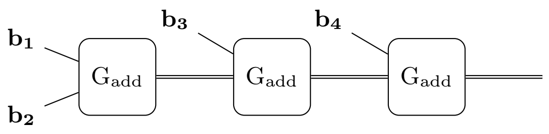
```
Input: Auxiliary inputs $\mathbf{a} \in \mathbb{K}^\alpha$, $n$
Output: $\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b}_{\lceil \frac{\alpha}{n} \rceil} \in \mathbb{K}^n$
 1. for $i = 1$ to $\lceil \frac{\alpha}{n} \rceil - 1$ do
 2.     $\mathbf{b_i} \leftarrow (a_{(i-1)\cdot n+1}, \ldots, a_{i \cdot n})$
 3. end for
 4. $\mathbf{b}_{\lceil \frac{\alpha}{n} \rceil} \leftarrow (a_{(\lceil \frac{\alpha}{n} \rceil - 1)\cdot n+1}, \ldots, a_\alpha, 0, \ldots, 0)$
 5. return $(\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b}_{\lceil \frac{\alpha}{n} \rceil})$
```

Algorithm 10 $\mathrm{G_{sum\text{-}chain}}(\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b}_\ell)$

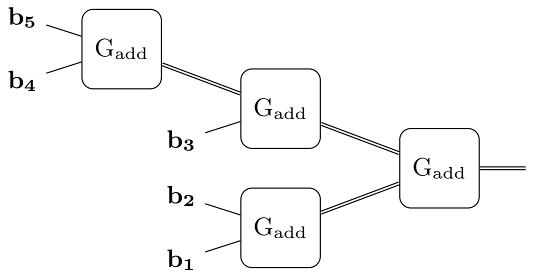
```
Input: $(\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b}_\ell) \in (\mathbb{K}^n)^\ell$
Output: $\mathbf{z} \in \mathbb{K}^n$
 1. $\mathbf{t} \leftarrow \mathbf{b_1}$
 2. FOR $i = 2$ to $\ell$, $\mathbf{t} \leftarrow \mathrm{G_{add}}(\mathbf{t}, \mathbf{b_i})$.
 3. return $\mathbf{t}$
```

Algorithm 11 $\mathrm{G_{sum\text{-}tree}}(\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b}_\ell)$

```
Input: $(\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b}_\ell) \in (\mathbb{K}^n)^\ell$
Output: $\mathbf{z} \in \mathbb{K}^n$
 1. if $\ell = 1$ return $\mathbf{b_1}$
 2. return $\mathrm{G_{add}}(\mathrm{G_{sum\text{-}tree}}(\mathbf{b_1}, , \ldots, \mathbf{b}_{\lfloor \ell/2 \rfloor}),$
    $\mathrm{G_{sum\text{-}tree}}(\mathbf{b}_{\lfloor \ell/2 \rfloor+1}, \mathbf{b_2}, \ldots, \mathbf{b}_\ell))$
```



(a) $\mathrm{G_{sum\text{-}chain}}$



(b) $\mathrm{G_{sum\text{-}tree}}$

Fig. 5: Noise generation gadget.

Lemma 11 (AddNoiseTo **is Cardinal-RPC**)**.** *Let $n, \alpha \in \mathbb{N}$ and $p \in [0,1]$. Let $G_{sum}\colon (\mathbb{K}^n)^{\lceil \frac{\alpha}{n} \rceil - 1} \to \mathbb{K}^n$ be $(p, (\mathcal{E}_t^{G_{sum}})_{t\in[0,n]})$-cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{G_{sum}})_{t\in[0,n]}$ defined over $[0,n]^{\lceil \frac{\alpha}{n} \rceil - 1}$ and $G_{add}\colon (\mathbb{K}^n)^2 \to \mathbb{K}^n$ be $(p, (\mathcal{E}_t^{G_{add}})_{t\in[0,n]})$-cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{G_{add}})_{t\in[0,n]}$ defined over $[0,n]^2$. Then,* AddNoiseTo*, defined in Algorithm 8, is $(p, (\mathcal{E}_t^{\mathtt{ADN}})_{t\in[0,n]})$-cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{\mathtt{ADN}})_{t\in[0,n]}$ defined over $[0,n]\times[0,\alpha]$ such that for all $t \in [0,n]$, for all $\boldsymbol{t}_{in} = (t_{in}^x, t_{in,1}^a, \ldots, t_{in,\lceil \frac{\alpha}{n} \rceil}^a) \in [0,n]^{\lceil \frac{\alpha}{n} \rceil + 1}$,*

$$\mathcal{E}_t^{\mathtt{ADN}}(t_{in}^x, t_{in,1}^a + \ldots + t_{in,\lceil \frac{\alpha}{n} \rceil}^a) = \sum_{i=0}^{n} \mathcal{E}_i^{G_{sum}}(t_{in,1}^a, \ldots, t_{in,\alpha}^a) \cdot \mathcal{E}_t^{G_{add}}(t_{in}^x, i).$$

In the following, we explore two structures for generating noise: a chain structure, illustrated in Figure 5a, and a binary tree structure, shown in Figure 5b. The respective algorithms for these structures are provided in Algorithms 10 and 11. Lemmas 12 and 13 establish their cardinal-RPC properties.

Lemma 12 is proven by recursively applying the composition result from Lemma 4. Similarly, Lemma 13 is established by applying the same composition result recursively on each branch of the tree, starting from the output nodes.

Lemma 12 ($\mathbf{G_{sum\text{-}chain}}$ is Cardinal-RPC). *Let $n, \ell \in \mathbb{N}$ and $p \in [0,1]$. Let $G_{add}\colon (\mathbb{K}^n)^2 \to \mathbb{K}^n$ be an n-share $(p, (\mathcal{E}_t^{G_{add}})_{t\in[0,n]})$-cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{G_{add}})_{t\in[0,n]}$ defined over $[0,n]^2$. Gadget $G_{sum\text{-}chain}\colon (\mathbb{K}^n)^\ell \to \mathbb{K}^n$, instantiated with G_{add}, is $(p, (\mathcal{E}_t^{\mathit{SNC}})_{t\in[0,n]})$-cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{\mathit{SNC}})_{t\in[0,n]}$ defined over $[0,n]^\ell$ as follows:*

$$\forall \boldsymbol{t}_{in} = (t_{in}^1, t_{in}^2, \dots, t_{in}^\ell) \in [0,n]^\ell,$$
$$\mathcal{E}_t^{\mathit{SNC}}(\boldsymbol{t}_{in}) = \sum_{i_3=0}^{n} \mathcal{E}_{i_3}^{G_{add}}(t_{in}^1, t_{in}^2) \cdot \ \cdots \ \cdot \sum_{i_\ell=0}^{n} \mathcal{E}_{i_\ell}^{G_{add}}(i_{\ell-1}, t_{in}^{\ell-1}) \cdot \mathcal{E}_t^{G_{add}}(i_\ell, t_{in}^\ell) \ .$$

Lemma 13 ($\mathbf{G_{sum\text{-}tree}}$ is Cardinal-RPC). *Let $n, \ell \in \mathbb{N}$ and $p \in [0,1]$. Let $G_{add}\colon (\mathbb{K}^n)^2 \to \mathbb{K}^n$ be an n-share $(p, (\mathcal{E}_t^{G_{add}})_{t\in[0,n]})$-cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{G_{add}})_{t\in[0,n]}$ defined over $[0,n]^2$. Gadget $G_{sum\text{-}tree}\colon (\mathbb{K}^n)^\ell \to \mathbb{K}^n$, instantiated with G_{add}, is $(p, (\mathcal{E}_t^{SPT(\ell)})_{t\in[0,n]})$-cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{SPT(\ell)})_{t\in[0,n]}$ recursively defined over $[0,n]^\ell$ as follows. For all $\boldsymbol{t}_{in} = (t_{in}^1, t_{in}^2, \dots, t_{in}^\ell) \in [0,n]^\ell$,*

$$\textit{if } \ell = 3, \quad \mathcal{E}_t^{SPT(3)}(t_{in}^1, t_{in}^2, t_{in}^3) = \sum_{i=0}^{n} \mathcal{E}_i^{G_{add}}(t_{in,1}, t_{in,2}) \cdot \mathcal{E}_t^{G_{add}}(i, t_{in,3})$$
$$\textit{otherwise}, \ \mathcal{E}_t^{SPT(\ell)}(\boldsymbol{t}_{in}) = \sum_{i_1=0}^{n}\sum_{i_2=0}^{n} \mathcal{E}_{i_1}^{SPT(\lfloor \ell/2 \rfloor)}(t_{in}^1, \dots, t_{in}^{\lfloor \ell/2 \rfloor}) \cdot \mathcal{E}_{i_2}^{SPT(\lceil \ell/2 \rceil)}(t_{in}^{\lfloor \ell/2 \rfloor+1}, \dots, t_{in}^\ell) \cdot \mathcal{E}_t^{G_{add}}(i_1, i_2)$$

with the sub-tree $SPT(\alpha)$ being $(p, (\mathcal{E}_t^{SPT(\alpha)})_{t\in[0,n]})$-cardinal-RPC for $3 \le \alpha \le n$ for some envelope $(\mathcal{E}_t^{SPT(\alpha)})_{t\in[0,n]}$ and with $SPT(2) = G_{add}$.

5.2 Instantiation from New Gadgets

In a first attempt, we can use the expansion strategy from [8] to construct a noise generation gadget based on a small addition gadget, as detailed in Appendix B.2. In this section, we illustrate how to leverage our new addition gadget $\mathrm{G_{add}}$ to achieve the same objective. Table 3 displays the threshold-RPC advantage of $\mathrm{G_{add}}$ from its cardinal-RPC envelopes (see Lemma 9) while fixing the leakage probability, the number of shares, and the value of t. The complexity is assessed in terms of the number of additions and random values required[6]. As anticipated, the complexities for the same leakage probability and security threshold are more favorable than those obtained through the expansion strategy.

Building on these findings, we can evaluate the RPC security of AddRepNoise for specific probabilities and sizes α of the auxiliary input. With cardinal-RPC gadgets, the composition is tighter, and the requirement for auxiliary inputs is considered globally. For example, with $\alpha = 75$ auxiliary inputs and a leakage probability $p = 2^{-24}$, one option is to combine three $\mathrm{G_{add}}$ gadgets, each with 25 shares, to set up AddNoiseTo. If we fix the number of iterations γ to 25 for each refresh, the resulting RPC advantage is upper bounded by 2^{-182} both for the chain and the tree structure, when 10 shares from the main input and 30 auxiliary inputs are required for the simulation. As for the complexity, the noise gadget requires 150 random values and implements 525 additions. This is to be compared with the 1200 randoms, 2475 additions and 1200 copies required by the AddNoiseTo gadget obtained from the expansion strategy (and which achieves an RPC advantage of 2^{-169}).

Remark 1. Our experiments show that the tree structure generally provides a slightly better security advantage compared to the chain structure, while maintaining the same single-thread complexity. We believe

[6] We omit the count of random values used for selecting indices in the refresh gadget, as they differ in nature from the random produced by the random gates of the circuit. Indeed, these random indices are from $[1,n]$ and not from $\mathbb{K}$. For the considered application (Raccoon), or other lattice-based schemes, elements from $[1,n]$ are expected to be (significantly) smaller than elements from $\mathbb{K}$. Moreover, the random index values in our refresh can be fully revealed to the adversary without compromising the security (which relies on their uniformity but not on their secrecy) unlike the random values of $\mathbb{K}$ produced by random gates which leak with probability p.

Table 3: RPC security advantage and complexity of $\mathrm{G}_{\mathrm{add}}$.

Leakage probability p	2^{-12}	2^{-16}	2^{-20}	2^{-24}
# Shares n and # leaking shares t	(25,13)	(20,10)	(16,8)	(12,6)
# Randoms	240	220	220	200
# Additions	555	500	488	436
RPC advantage	2^{-128}	2^{-128}	2^{-134}	2^{-129}

this is due to the higher average number of dependencies per input in the tree structure, and their balance. Furthermore, both structures offer the potential to explore different instantiations for the refresh gadgets—*i.e.* varying the number of iterations—based on their position, a direction we leave for future research.

6 Random Probing Secure Implementation of Raccoon

We recall the key generation and signature algorithms of Raccoon in Algorithms 12 and 14 from [16]. We refer to [16] for an explicit definition of the subroutines and their rationale. The parameters relevant to our work are presented in Table 4. Specifically, parameters n_R (denoted n in [16]) and q define the quotient ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^{n_R}+1)$, while k and ℓ define the MLWE matrix dimensions. Like in the proof of Raccoon in the d-probing model [17], we do not consider that the noise samples (r_i) and (r'_i) are generated within AddRepNoise algorithm; instead, they are provided as auxiliary inputs. We also artificially de-correlate the number of shares, here denoted n, from the parameter d, fixed in the specifications.

6.1 Random Probing Secure Construction

We denote by G^{vec} the natural extension of a gadget G, originally operating on elements of $\mathbb{Z}_q$, to a gadget operating on elements of $\mathcal{R}_q$ and which applies the same operation component-wise, repeating it n_R times. We also introduce the following core gadgets:

– $\mathrm{G}_{\mathrm{cpmult}}$: This gadget multiplies a uniform public polynomial c^i_{poly} in $\mathcal{R}_q$ with a shared secret vector $[\![s^i]\!]$ in $(\mathcal{R}_q^n)$, for $i \in [1,\ell]$. It requires n_R^2 n-share linear multiplications on elements of $\mathbb{Z}_q$, $n_R(n_R-1)$ n-share additions on elements of $\mathbb{Z}_q$ and $n_R(n_R-1)$ n-share copies on elements of $\mathbb{Z}_q$.
– $\mathrm{G}_{\mathrm{cmmult}}$: This gadget performs a multiplication between a uniform public matrix A in $\mathcal{R}_q^{k\times\ell}$ with a shared secret vector $[\![s]\!]$ in $(\mathcal{R}_q^\ell)^n$. It requires $\ell k n_R^2$ n-share linear multiplications on elements of $\mathbb{Z}_q$, $n_R(n_R-1)\ell k + n_R(\ell-1)k$ n-share additions on elements of $\mathbb{Z}_q$ and $n_R(n_R-1)\ell k + n_R(k-1)\ell$ n-share copies on elements of $\mathbb{Z}_q$.

To obtain a random-probing secure construction, masked gadgets secure in the probing model can be seamlessly replaced by random-probing composable gadgets. The random-probing Raccoon equivalent is illustrated in Algorithms 13 and 15. Let us provide an intuition for the random probing version of the key generation. The original algorithm (Alg. 12) starts with a fresh zero-encoding (line 3) and adds the auxiliary inputs $(r_i)_{1\le i\le\ell}$ with intermediate refreshes (inside AddRepNoise in line 4). In our design, the refresh gadgets are already included in the elementary gadgets, so we simply sum the auxiliary inputs $(r_i)_{1\le i\le\ell}$ split in convenient packs using DivideAI and the $\mathrm{G}_{\mathrm{sum}}$ gadget (see Section 5). Next, the resulting sharing $[\![s]\!]$ is copied: one copy is used for the public key and the other one for the secret key. After the multiplication step, we use AddNoiseTo (presented in Alg. 8) to incorporate the auxiliary inputs $(r'_i)_{1\le i\le k}$ and finally decode and output the result. The same reasoning can be similarly applied to the signature algorithm. In a nutshell, compared to the original d-probing secure algorithms, most intermediate refresh gadgets can be removed and copy gadgets are added whenever a variable needs to be reused. We stress that *our random-probing Raccoon is functionally equivalent to the original d-probing version.*

Algorithm 12 d-Probing-KeyGen

```
Auxiliary Inputs:
$(r_i)_{1\le i\le \ell} \in (\{0,1\}^{u_t})^{d\cdot \text{rep}\cdot \ell\cdot n_R}$
$(r'_i)_{1\le i\le k} \in (\{0,1\}^{u_t})^{d\cdot \text{rep}\cdot k\cdot n_R}$

Output: Key pair (vk, sk)
 1. $\mathsf{seed} \leftarrow \{0,1\}^\kappa$
 2. $A \leftarrow \mathsf{ExpandA}(\mathsf{seed})$
 3. $[\![s]\!] \leftarrow \ell \times \mathsf{ZeroEncoding}(d)$
 4. $[\![s]\!] \leftarrow \mathsf{AddRepNoise}([\![s]\!], (r_i)_{1\le i\le \ell})$
 5. $[\![t]\!] \leftarrow A \cdot [\![s]\!]$
 6. $[\![t]\!] \leftarrow \mathsf{AddRepNoise}([\![t]\!], (r'_i)_{1\le i\le k})$
 7. $t \leftarrow \mathsf{Decode}([\![t]\!])$
 8. return $(\mathsf{vk} \leftarrow (\mathsf{seed}, t), \mathsf{sk} \leftarrow (t, [\![s]\!]))$
```

Algorithm 13 RP-KeyGen

```
Auxiliary Inputs:
$(r_i)_{1\le i\le \ell} \in (\{0,1\}^{u_t})^{d\cdot \text{rep}\cdot \ell\cdot n_R}$
$(r'_i)_{1\le i\le k} \in (\{0,1\}^{u_t})^{d\cdot \text{rep}\cdot k\cdot n_R}$

Output: Key pair (vk, sk)
 1. $\mathsf{seed} \leftarrow \{0,1\}^\kappa$
 2. $A \leftarrow \mathsf{ExpandA}(\mathsf{seed})$
 3. $\mathbf{r_b} \leftarrow \mathsf{DivideAI}^{\mathsf{vec}}((r_i)_{1\le i\le \ell})$
 4. $[\![s]\!] \leftarrow \mathrm{G_{sum}}^{\mathsf{vec}}(\mathbf{r_b})$
 5. $([\![s]\!], [\![s_t]\!]) \leftarrow \mathrm{G_{copy}}^{\mathsf{vec}}([\![s]\!])$
 6. $[\![t]\!] \leftarrow \mathrm{G_{cmmult}}(A, [\![s_t]\!])$
 7. $[\![t]\!] \leftarrow \mathsf{AddNoiseTo}^{\mathsf{vec}}([\![t]\!], (r'_i)_{1\le i\le k})$
 8. $t \leftarrow \mathrm{G_{decode}}^{\mathsf{vec}}([\![t]\!])$
 9. return $(\mathsf{vk} \leftarrow (\mathsf{seed}, t), \mathsf{sk} \leftarrow (t, [\![s]\!]))$
```

Algorithm 14 d-Probing-Sign

```
Auxiliary Inputs:
$(r_i)_{1\le i\le \ell} \in (\{0,1\}^{u_t})^{d\cdot \text{rep}\cdot \ell\cdot n_R}$
$(r'_i)_{1\le i\le k} \in (\{0,1\}^{u_t})^{d\cdot \text{rep}\cdot k\cdot n_R}$

Input: $\mathsf{sk} = (\mathsf{vk}, [\![s]\!])$, $\mathsf{msg} \in \{0,1\}^*$
Output: $\mathsf{sig} = (c_{\mathsf{hash}}, h, z)$
 1. $(\mathsf{vk}, [\![s]\!]) \leftarrow \mathsf{sk}, (\mathsf{seed}, t) \leftarrow \mathsf{vk}$
 2. $\mu \leftarrow H(H(\mathsf{vk})\|\mathsf{msg})$
 3. $A \leftarrow \mathsf{ExpandA}(\mathsf{seed})$
 4. $[\![r]\!] \leftarrow \ell \times \mathsf{ZeroEncoding}(d)$
 5. $[\![r]\!] \leftarrow \mathsf{AddRepNoise}([\![r]\!], (r_i)_{1\le i\le \ell})$
 6. $[\![w]\!] \leftarrow A \cdot [\![r]\!]$
 7. $[\![w]\!] \leftarrow \mathsf{AddRepNoise}([\![w]\!], (r'_i)_{1\le i\le k})$
 8. $w \leftarrow \mathsf{Decode}([\![w]\!])$
 9. $c_{\mathsf{hash}} \leftarrow \mathsf{ChalHash}(w, \mu)$
10. $c_{\mathsf{poly}} \leftarrow \mathsf{ChalPoly}(c_{\mathsf{hash}})$
11. $[\![s]\!] \leftarrow \mathsf{Refresh}([\![s]\!])$
12. $[\![r]\!] \leftarrow \mathsf{Refresh}([\![r]\!])$
13. $[\![z]\!] \leftarrow c_{\mathsf{poly}} \cdot [\![s]\!] + [\![r]\!]$
14. $[\![z]\!] \leftarrow \mathsf{Refresh}([\![z]\!])$
15. $z \leftarrow \mathsf{Decode}([\![z]\!])$
16. $h \leftarrow w - A \cdot z - c_{\mathsf{hash}} \cdot t$
17. $\mathsf{sig} \leftarrow (c_{\mathsf{hash}}, h, z)$
18. if $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ then
19.     goto Line 4
20. end if
21. return sig
```

Algorithm 15 RP-Sign

```
Auxiliary Inputs:
$(r_i)_{1\le i\le \ell} \in (\{0,1\}^{u_t})^{d\cdot \text{rep}\cdot \ell\cdot n_R}$
$(r'_i)_{1\le i\le k} \in (\{0,1\}^{u_t})^{d\cdot \text{rep}\cdot k\cdot n_R}$

Input: $\mathsf{sk} = (\mathsf{vk}, [\![s]\!])$, $\mathsf{msg} \in \{0,1\}^*$
Output: $\mathsf{sig} = (c_{\mathsf{hash}}, h, z)$
 1. $(\mathsf{vk}, [\![s]\!]) \leftarrow \mathsf{sk}, (\mathsf{seed}, t) \leftarrow \mathsf{vk}$
 2. $\mu \leftarrow H(H(\mathsf{vk})\|\mathsf{msg})$
 3. $A \leftarrow \mathsf{ExpandA}(\mathsf{seed})$
 4. $\mathbf{r_b} \leftarrow \mathsf{DivideAI}^{\mathsf{vec}}((r_i)_{1\le i\le \ell})$
 5. $[\![r]\!] \leftarrow \mathrm{G_{sum}}^{\mathsf{vec}}(\mathbf{r_b})$
 6. $([\![r]\!], [\![r_c]\!]) \leftarrow \mathrm{G_{copy}}^{\mathsf{vec}}([\![r]\!])$
 7. $[\![w]\!] \leftarrow \mathrm{G_{cmmult}}(A, [\![r]\!])$
 8. $[\![w]\!] \leftarrow \mathsf{AddNoiseTo}^{\mathsf{vec}}([\![w]\!], (r'_i)_{1\le i\le k})$
 9. $w \leftarrow \mathrm{G_{decode}}^{\mathsf{vec}}([\![w]\!])$
10. $c_{\mathsf{hash}} \leftarrow \mathsf{ChalHash}(w, \mu)$
11. $c_{\mathsf{poly}} \leftarrow \mathsf{ChalPoly}(c_{\mathsf{hash}})$
12. $[\![z]\!] \leftarrow \mathrm{G_{add}}^{\mathsf{vec}}(\mathrm{G_{cpmult}}([\![s]\!], c_{\mathsf{poly}}), [\![r_c]\!])$
13. $z \leftarrow \mathrm{G_{decode}}^{\mathsf{vec}}([\![z]\!])$
14. $h \leftarrow w - A \cdot z - c_{\mathsf{hash}} \cdot t$
15. $\mathsf{sig} \leftarrow (c_{\mathsf{hash}}, h, z)$
16. if $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ then
17.     goto Line 4
18. end if
19. return sig
```

Table 4: Some of Raccoon's parameters (see [16] for the full description).

Param.	Raccoon-128-16
q	549824583172097
n_R	512
k	5
ℓ	4
d	16
rep	2

Table 5: Parameters introduced for the random probing security.

Param.	Description
p	Probability of leaking a wire (resulting from the noise level of the device).
ϵ	Threshold-RPC-AI-O advantage, indicates the security level (high for weak security, low for strong security).
n	Number of shares ($= d$ or $\geq d$ if expansion, often higher to reach better security levels).
t	Threshold RPC parameter (set to $n/2$).
γ	Number of iterations of the refresh algorithm, allowing optimizations of the security/performance trade-off.

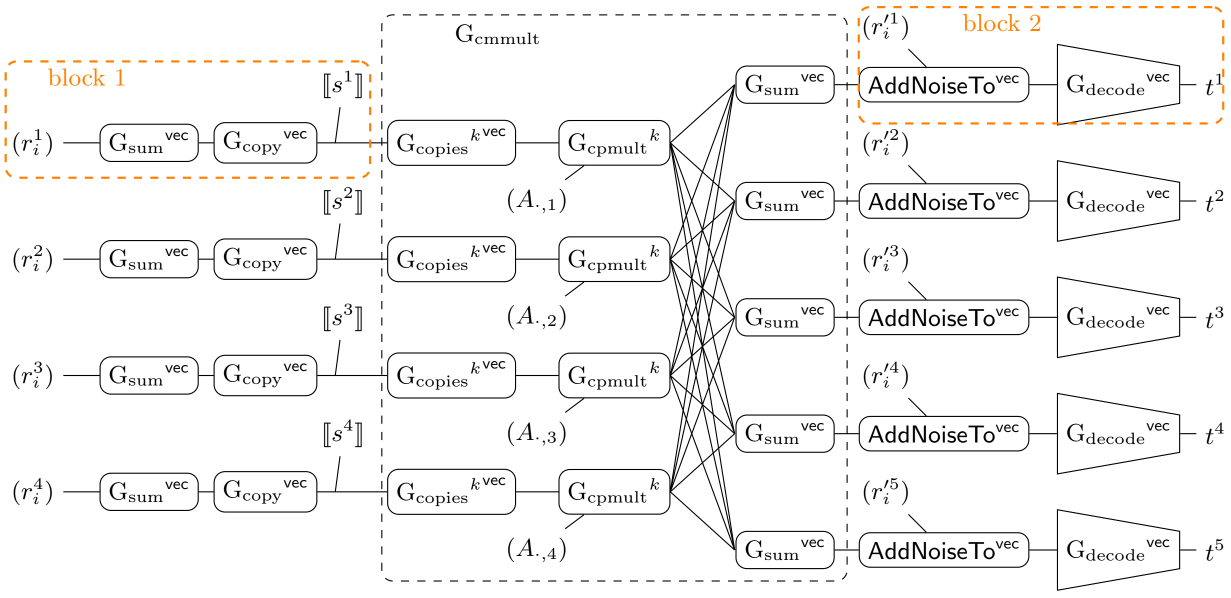


Fig. 6: RP-KeyGen for Raccoon-128 with $\ell = 4$ and $k = 5$

Instantiation with our new gadgets. We now instantiate Algorithms 13 and 15 with our new cardinal-RPC(-AI-O) secure gadgets defined in Section 4.2. Figures 6 and 7 (in Appendix) visually illustrate the processes for Raccoon-128. We denote by $\mathrm{G_{copies}}^k$ the combination of copy gadgets arranged in a binary tree structure to produce k copies of the same sharing, and by $\mathrm{G_{cpmult}}^k$ the application of the $\mathrm{G_{cpmult}}$ gadget to perform a polynomial multiplication on $\mathcal{R}_q$ for k separate inputs.

With these new gadgets, our schemes can be instantiated in various ways, with multiple parameters subject to variation—namely, the probability p, the RPC threshold t, and the number of iterations γ for each refresh instance. All these new parameters are summarized in Table 5. We opt for aligning to the Raccoon-128-16 configuration, which operates with 16-sharings. The corresponding instantiation is detailed in Table 4, while the full set of parameters can be found in [16, Table 2]. The scheme is proven EUF-CMA secure even when up to $d-1 = 15$ values of r_i and r_i' per vector and polynomial coefficient are exposed to the attacker (see [17, Security of Small Raccoon in Sec. 7]). Hence, at least 17ℓ small uniforms per polynomial coefficient must remain secret throughout both key generation and signature processes. The Raccoon-128-16 configuration is particularly well-suited because using 16-sharings requires two inputs per coefficient to incorporate the small uniforms. This results in a single addition gadget per coefficient in the first $\mathrm{G_{sum}}$ and two addition gadgets per coefficient in the AddNoiseTo gadget.

Table 6: Threshold-RPC-AI-O advantages and complexities of Raccoon's key generation and signature when instantiated with our new gadgets. The values for γ (chosen to be different for block 1, block 2 and G_{cmmult}) are fixed to minimize ε (which impacts the overall randomness consumption).

	Key Generation				**Signature**			
p	2^{-24}	2^{-20}	2^{-16}	2^{-12}	2^{-24}	2^{-20}	2^{-16}	2^{-12}
	$\varepsilon \leq 2^{-128}$				$\varepsilon \leq 2^{-128}$			
# additions	1.82e9	2.08e9	-	-	3.44e9	5.33e9	-	-
# linear mult.	8.39e7	8.39e7	-	-	1.01e8	1.01e8	-	-
# randoms	6.57e8	7.88e8	-	-	1.42e9	2.36e9	-	-
	$\varepsilon \leq 2^{-80}$				$\varepsilon \leq 2^{-80}$			
# additions	1.29e9	1.29e9	1.55e9	-	2.44e9	2.49e9	2.81e9	-
# linear mult.	8.39e7	8.39e7	8.39e7	-	1.01e8	1.01e8	1.01e8	-
# randoms	3.94e8	3.94e8	5.26e8	-	9.19e8	9.45e8	1.10e9	-
	$\varepsilon \leq 2^{-64}$				$\varepsilon \leq 2^{-64}$			
# additions	1.03e9	1.03e9	1.29e9	1.56e9	2.02e9	2.07e9	2.39e9	2.70e9
# linear mult.	8.39e7	8.39e7	8.39e7	8.39e7	1.01e8	1.01e8	1.01e8	1.01e8
# randoms	2.63e8	2.63e8	3.94e8	5.26e8	7.09e8	7.35e8	8.92e8	1.05e9

Theorems 3 establishes the threshold-RPC-AI-O security of Raccoon's key generation and Raccoon's signature processes as defined in Algorithms 13 and 15, following Definition 17. The proofs rely on the cardinal-RPC(-AI)(-O) envelopes of gadgets and their compositions, and is provided in Appendix C.

Theorem 3. *Raccoon-128-16 signature scheme displayed in Algorithms 13 and 15 is* $(p, t = 8, n-1 = 15, \varepsilon)$*-threshold-RPC-AI-O with the pairs* (p, ε) *provided in Table 6.*

6.2 Comparison with the expansion technique and the probing secure implementation

Algorithms 13 and 15 can also be instantiated with the expanded gadgets from [8] (note that G_{decode} was not defined in [8]). Details on the expansion tools used to generate a random-probing secure Raccoon algorithm are provided in Appendix B.3. By applying new gadgets and leveraging composition proofs, we achieve significant improvements in both complexity and randomness consumption, as illustrated in Table 7.

Finally, to compare with the original probing-secure version of Raccoon, we calculate the probability of observing more than t variables in both algorithms, which gives a proven upper bound on its random probing security advantage. For $n = 16$ shares and a leakage probability of 2^{-24}, we obtain an advantage of almost 1 –meaning no proven security– for the key generation and for the signature. This highlights the benefit of our constructions to achieve random probing security compared to standard probing secure schemes.

Table 7: Security and complexity of the key generation and the signature of Raccoon with 128 bits of (black-box) security in its original probing version and for both random probing secure scenarios with $p = 2^{-24}$.

	Key Generation			Signature		
	Original	Expansion	New Gadgets	Original	Expansion	New Gadgets
# shares	16	27	16	16	27	16
# additions	8.49e7	2.94e10	1.82e9	1.02e8	3.53e10	3.44e9
# linear mult.	8.39e7	1.42e8	8.39e7	1.01e8	1.70e8	1.01e8
# randoms	3.60e5	1.46e10	6.57e8	5.57e5	1.76e10	1.42e9
Security RPS/C	1	2^{-116}	2^{-132}	1	2^{-116}	2^{-130}

7 Conclusion and future work

In conclusion, this work proposes new techniques towards stronger and tighter random probing security while providing all necessary components, from security definitions to concrete constructions, to show that random-probing security *is achievable* for a post-quantum scheme, namely Raccoon. We provide the first performance results for a random-probing secure post-quantum algorithm. We hope our design and notions will serve as building blocks for more efficient random-probing secure constructions and inspire "random-probing-friendly" designs.

Another interesting avenue for future research could be to investigate if other recent techniques proposed in [19,12,25] could lead to different trade-offs for random probing secure versions of Raccoon. This would require extending the compilers proposed in these works to support composition with auxiliary input and/or public output, designing a secure decoding gadget within these frameworks, and comparing with our approach.

Acknowledgments. This work was co-funded by the European Union (ERC, AMAskZONE, 101077506) and Bpifrance (RESQUE project, France national quantum strategy). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

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. pp. 715–724. ACM Press, San Jose, CA, USA (Jun 6–8, 2011)
2. Ananth, P., Ishai, Y., Sahai, A.: Private circuits: A modular approach. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part III. Lecture Notes in Computer Science, vol. 10993, pp. 427–455. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 19–23, 2018)
3. Andrychowicz, M., Dziembowski, S., Faust, S.: Circuit compilers with $O(1/\log(n))$ leakage rate. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 586–615. Springer Berlin Heidelberg, Germany, Vienna, Austria (May 8–12, 2016)
4. Azouaoui, M., Bronchain, O., Cassiers, G., Hoffmann, C., Kuzovkova, Y., Renes, J., Schneider, T., Schönauer, M., Standaert, F.X., van Vredendaal, C.: Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(4), 58–79 (2023)
5. Barthe, G., Belaïd, S., Espitau, T., Fouque, P.A., Grégoire, B., Rossi, M., Tibouchi, M.: Masking the GLP lattice-based signature scheme at any order. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 354–384. Springer, Cham, Switzerland, Tel Aviv, Israel (Apr 29 – May 3, 2018)
6. 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.) Cryptographic Hardware and Embedded Systems – CHES 2016. Lecture Notes in Computer Science, vol. 9813, pp. 23–39. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–19, 2016)
7. Belaïd, S., Cassiers, G., Rivain, M., Taleb, A.R.: Unifying freedom and separation for tight probing-secure composition. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part III. Lecture Notes in Computer Science, vol. 14083, pp. 440–472. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 20–24, 2023)
8. Belaïd, S., Coron, J.S., Prouff, E., Rivain, M., Taleb, A.R.: Random probing security: Verification, composition, expansion and new constructions. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part I. Lecture Notes in Computer Science, vol. 12170, pp. 339–368. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 17–21, 2020)
9. Belaïd, S., Mercadier, D., Rivain, M., Taleb, A.R.: IronMask: Versatile verification of masking security. In: 2022 IEEE Symposium on Security and Privacy. pp. 142–160. IEEE Computer Society Press, San Francisco, CA, USA (May 22–26, 2022)

10. Belaïd, S., Rivain, M., Taleb, A.R.: On the power of expansion: More efficient constructions in the random probing model. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021, Part II. Lecture Notes in Computer Science, vol. 12697, pp. 313–343. Springer, Cham, Switzerland, Zagreb, Croatia (Oct 17–21, 2021)
11. Belaïd, S., Rivain, M., Taleb, A.R., Vergnaud, D.: Dynamic random probing expansion with quasi linear asymptotic complexity. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part II. Lecture Notes in Computer Science, vol. 13091, pp. 157–188. Springer, Cham, Switzerland, Singapore (Dec 6–10, 2021)
12. Berti, F., Faust, S., Orlt, M.: Provable secure parallel gadgets. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(4), 420–459 (2023)
13. Cassiers, G., Faust, S., Orlt, M., Standaert, F.X.: Towards tight random probing security. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part III. Lecture Notes in Computer Science, vol. 12827, pp. 185–214. Springer, Cham, Switzerland, Virtual Event (Aug 16–20, 2021)
14. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) Advances in Cryptology – CRYPTO'99. Lecture Notes in Computer Science, vol. 1666, pp. 398–412. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999)
15. Coron, J.S., Gérard, F., Trannoy, M., Zeitoun, R.: Improved gadgets for the high-order masking of Dilithium. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(4), 110–145 (2023)
16. del Pino, R., Espitau, T., Katsumata, S., Maller, M., Mouhartem, F., Prest, T., Rossi, M., Saarinen, M.: Raccoon. Tech. rep., National Institute of Standards and Technology (2023)
17. del Pino, R., Katsumata, S., Prest, T., Rossi, M.: Raccoon: A masking-friendly signature proven in the probing model. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part I. Lecture Notes in Computer Science, vol. 14920, pp. 409–444. Springer, Cham, Switzerland, Santa Barbara, CA, USA (Aug 18–22, 2024)
18. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. Lecture Notes in Computer Science, vol. 8441, pp. 423–440. Springer Berlin Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014)
19. Dziembowski, S., Faust, S., Zebrowski, K.: Simple refreshing in the noisy leakage model. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019, Part III. Lecture Notes in Computer Science, vol. 11923, pp. 315–344. Springer, Cham, Switzerland, Kobe, Japan (Dec 8–12, 2019)
20. Esgin, M.F., Espitau, T., Niot, G., Prest, T., Sakzad, A., Steinfeld, R.: sfPlover: Masking-friendly hash-and-sign lattice signatures. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024, Part VII. Lecture Notes in Computer Science, vol. 14657, pp. 316–345. Springer, Cham, Switzerland, Zurich, Switzerland (May 26–30, 2024)
21. Espitau, T., Fouque, P.A., Gérard, F., Rossi, M., Takahashi, A., Tibouchi, M., Wallet, A., Yu, Y.: Mitaka: A simpler, parallelizable, maskable variant of falcon. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part III. Lecture Notes in Computer Science, vol. 13277, pp. 222–253. Springer, Cham, Switzerland, Trondheim, Norway (May 30 – Jun 3, 2022)
22. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Koç, Çetin Kaya., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems – CHES'99. Lecture Notes in Computer Science, vol. 1717, pp. 158–172. Springer Berlin Heidelberg, Germany, Worcester, Massachusetts, USA (Aug 12–13, 1999)
23. Goudarzi, D., Joux, A., Rivain, M.: How to securely compute with noisy leakage in quasilinear complexity. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018, Part II. Lecture Notes in Computer Science, vol. 11273, pp. 547–574. Springer, Cham, Switzerland, Brisbane, Queensland, Australia (Dec 2–6, 2018)
24. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) Advances in Cryptology – CRYPTO 2003. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2003)
25. Jahandideh, V., Mennink, B., Batina, L.: An algebraic approach for evaluating random probing security with application to AES. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2024**(4), 657–689 (2024)
26. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2022)
27. Prest, T.: A key-recovery attack against mitaka in the t-probing model. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 13940, pp. 205–220. Springer, Cham, Switzerland, Atlanta, GA, USA (May 7–10, 2023)
28. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2022)

29. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.X. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2010. Lecture Notes in Computer Science, vol. 6225, pp. 413–427. Springer Berlin Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–20, 2010)
30. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D., Ding, J.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2022)

Experiment 5: Threshold-RPC

```
$\mathcal{W} \leftarrow \mathsf{LeakingWires}(G, p)$                  Drawing the leaking wires.
$I_1, \cdots I_\ell \leftarrow \mathsf{Sim}_1(G, \mathcal{W}, J_1, \cdots J_m)$     The simulator is given the indexes of the
                                                 leaking outputs. It chooses indexes for
                                                 the inputs.
$out_1, out_2 \leftarrow \mathsf{Sim}_2(\llbracket x_1 \rrbracket|_{I_1}, \cdots, \llbracket x_\ell \rrbracket|_{I_\ell})$   The simulator is given the shares at desired
                                                 indexes and returns simulated values for the
                                                 leaking wires ($out_1$) and outputs ($out_2$)
return $(I_1, \cdots I_\ell, out_1, out_2)$
```

A Security definitions

Definition 12 (Circuit Compiler from [8]). *A* circuit compiler *is a triplet of algorithms* (CC, Enc, 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 $\llbracket \boldsymbol{y} \rrbracket \in \mathbb{K}^{m'}$ to a plain output $\boldsymbol{y} \in \mathbb{K}^m$,*

which 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*

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

- ***Efficiency:** For some security parameter $\kappa \in \mathbb{N}$, the running time of $\mathsf{CC}(C)$ is $\mathrm{poly}(\kappa, |C|)$, the running time of $\mathsf{Enc}(\boldsymbol{x})$ is $\mathrm{poly}(\kappa, |\boldsymbol{x}|)$ and the running time of $\mathsf{Dec}(\llbracket \boldsymbol{y} \rrbracket)$ is $\mathrm{poly}(\kappa, |\llbracket \boldsymbol{y} \rrbracket|)$, where $\mathrm{poly}(\kappa, \ell) = \mathcal{O}(\kappa^{e_1} \ell^{e_2})$ for some constants e_1, e_2.*

Definition 13 ((Threshold) Random Probing Composability from [8]). *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 $p, \epsilon \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for every masked input $\llbracket \boldsymbol{x} \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $J_1 \subseteq [n]$, $\ldots$, $J_m \subseteq [n]$ of cardinals $|J_1| \le t$, $\ldots$, $|J_m| \le t$, the outputs of Experiment 5 are such that*

1. *$Pr\big((|I_1| > t) \vee \ldots \vee (|I_\ell| > t)\big) \le \varepsilon$*
2. $$(out_1, out_2) \overset{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}\left(G, \llbracket \boldsymbol{x} \rrbracket, \llbracket \boldsymbol{y} \rrbracket\right)}_{\texttt{Induced Random probing leakage}} \quad , \quad \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \cdots, \llbracket y_m \rrbracket|_{J_m})}_{\texttt{Output leakage}} \right)$$

   where $\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p)$ and $\llbracket \boldsymbol{y} \rrbracket \leftarrow G(\llbracket \boldsymbol{x} \rrbracket)$.

Definition 14. *Let $f : \mathbb{R} \to \mathbb{R}$. The gadget G is (t, f)-RPC if it is $(t, p, f(p))$-RPC for every $p \in [0, 1]$.*

Theorem 4 (Composition from [8]). *Let $t \in \mathbb{N}$, $p, \epsilon \in [0, 1]$, 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 $(p, |C| \cdot \epsilon)$-random probing secure. Equivalently, the standard circuit compiler* CC *is (p, ϵ)-random probing secure.*

Experiment 6: General-RPC

```
$\mathcal{W} \leftarrow \mathsf{LeakingWires}(G, p)$                        Drawing the leaking wires.
$I_1, \cdots I_\ell \leftarrow \mathsf{Sim}_1(G, \mathcal{W}, J_1, \cdots J_m)$    The simulator is given the indexes of the
                                                   leaking outputs. It chooses indexes for
                                                   the inputs.
$out_1, out_2 \leftarrow \mathsf{Sim}_2(\llbracket x_1 \rrbracket|_{I_1}, \cdots, \llbracket x_\ell \rrbracket|_{I_\ell})$   The simulator is given the shares at desired
                                                   indexes and returns simulated values for the
                                                   leaking wires ($out_1$) and outputs ($out_2$)
return $(I_1, \cdots I_\ell, out_1, out_2)$
```

Definition 15 (General Random Probing Composability from [13]). *Let $n, \ell, m \in \mathbb{N}$. Let $\boldsymbol{\mathcal{E}}$ represent a collection of probability envelopes indexed by $(J_1, \cdots J_m) \in \mathbb{K}^{|J_1|} \times \ldots \times \mathbb{K}^{|J_m|}$, where $|J_i| \in [0, n]$ for every $i \in [1, m]$. These envelopes are defined over tuples of size ℓ of elements in $\mathbb{K}^i$ for $i \in [0, n]$. An n-share gadget $G : (\mathbb{K}^n)^\ell \to (\mathbb{K}^n)^m$ is $(p, \boldsymbol{\mathcal{E}})$-*general random probing composable (general-RPC for short) *for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for every shared input $\llbracket x \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \ldots, J_m)$, the outputs of Experiment 6 are such that*

1. $(I_1, \cdots, I_\ell) \lesssim \mathcal{E}_{J_1, \cdots, J_m}$,
2. $$(out_1, out_2) \overset{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, \llbracket \boldsymbol{x} \rrbracket, \llbracket \boldsymbol{y} \rrbracket)}_{\textit{Induced Random probing leakage}} \quad , \quad \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \cdots, \llbracket y_m \rrbracket|_{J_m})}_{\textit{Output leakage}} \right),$$

   where $\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p)$ and $\llbracket \boldsymbol{y} \rrbracket \leftarrow G(\llbracket \boldsymbol{x} \rrbracket)$.

Definition 16 (Uniformly Cardinal Random Probing Composability). *Let $n, \ell, m \in \mathbb{N}$. Let $\boldsymbol{\mathcal{E}}^u$ represent a collection of probability envelopes indexed by $(j_1, \cdots j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. An n-share gadget $G : (\mathbb{K}^n)^\ell \to (\mathbb{K}^n)^m$ is* uniformly $(p, \boldsymbol{\mathcal{E}}^u)$-cardinal random probing composable (uniformly cardinal-RPC for short) *for some $p \in [0, 1]$ if it is $(p, \boldsymbol{\mathcal{E}}^g)$-general-RPC and if for all $(J_1, \ldots, J_m)$, $(J'_1, \ldots, J'_m) \in \mathbb{K}^{|J_1|} \times \ldots \times \mathbb{K}^{|J_m|}$ such that*

$$(|J_1|, \ldots, |J_m|) = (|J'_1|, \ldots, |J'_m|)$$

and for all $(I_1, \ldots, I_\ell)$, $(I'_1, \ldots, I'_\ell) \in \mathbb{K}^{|I_1|} \times \ldots \times \mathbb{K}^{|I_\ell|}$ such that

$$(|I_1|, \ldots, |I_\ell|) = (|I'_1|, \ldots, |I'_\ell|),$$

$$\textit{we have } \mathcal{E}^g_{J_1, \cdots, J_m}(I_1, \ldots, I_l) = \mathcal{E}^g_{J'_1, \cdots, J'_m}(I'_1, \ldots, I'_l).$$

The collection of uniformly cardinal-RPC envelopes $\boldsymbol{\mathcal{E}}^u$ is then naturally defined such that

$$\mathcal{E}^u_{|J_1|, \ldots, |J_m|}(|I_1|, \ldots, |I_\ell|) = \mathcal{E}^g_{J_1, \cdots, J_m}(I_1, \ldots, I_l).$$

Definition 17 ((Threshold) Random Probing Composability with Auxiliary Inputs and Public Outputs).

Let $n, \ell, m, k, \alpha, d \in \mathbb{N}$. Let G be a gadget with the following input/output partition

$$\begin{array}{ccccccc} G: & \overbrace{(\mathbb{K}^n)^\ell}^{\textit{masked inputs}} & \times & \overbrace{(\mathbb{K}^\alpha)^k}^{\textit{auxiliary inputs}} & \to & \overbrace{(\mathbb{K}^n)^m}^{\textit{masked outputs}} & \times & \overbrace{\mathbb{K}^d}^{\textit{public outputs}} \\ & (\llbracket \boldsymbol{x} \rrbracket, & & \boldsymbol{a}_1, \cdots, \boldsymbol{a}_k) & \mapsto & (\llbracket \boldsymbol{y} \rrbracket, & & \boldsymbol{z}). \end{array}$$

The gadget G is (p, t, t', ε)-threshold random probing composable with Auxiliary Inputs and Public Outputs for some $p \in [0, 1]$, $t, t' \in \mathbb{N}$ and $\varepsilon > 0$ if there exists a PPT stateful two-stage simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such*

Experiment 7: Threshold-RPC-AI-O

```
$(*, \boldsymbol{z}) \leftarrow C(\mathsf{Enc}(\boldsymbol{x}), \boldsymbol{a})$    Drawing the public outputs.
$\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p)$    Drawing the leaking wires.
$(I_1, \cdots, I_\ell), (L_1, \cdots, L_k) \leftarrow \mathsf{Sim}_1(C, \mathcal{W}, \boldsymbol{z}, J_1, \cdots, J_m)$    The simulator is given the indexes of the leaking outputs. It chooses indexes for the shared and auxiliary inputs.
$out_1, out_2 \leftarrow \mathsf{Sim}_2([\![x_1]\!]|_{I_1}, \cdots, [\![x_\ell]\!]|_{I_\ell}, \boldsymbol{a_1}|_{L_1}, \cdots, \boldsymbol{a_k}|_{L_k})$    The simulator is given the shares at desired indexes and returns simulated values for the leaking wires and outputs.
return $(out_1, out_2, (I_1, \cdots, I_\ell), (L_1, \cdots, L_k))$
```

that for every shared input $[\![x]\!] \in (\mathbb{K}^n)^\ell$*, auxiliary input* $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k \in (\mathbb{K}^\alpha)^k$*, and for every set collection* $(J_1, \ldots, J_m)$ *where* $J_1 \subseteq [n], \ldots, J_m \subseteq [n]$*, the outputs of Experiment 7 are such that*

1. $Pr\big((|I_1| > t) \vee \ldots \vee (|I_\ell| > t)\big) \leq \varepsilon$
2. $Pr\big((|L_1| > t') \vee \ldots \vee (|L_k| > t')\big) \leq \varepsilon$
3. *and*

$$(out_1, out_2) \overset{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, ([\![\boldsymbol{x}]\!], \boldsymbol{a}), ([\![\boldsymbol{y}]\!], \boldsymbol{z}))}_{\texttt{Induced Random probing leakage}}, \underbrace{([\![y_1]\!]|_{J_1}, \cdots, [\![y_m]\!]|_{J_m})}_{\texttt{Output leakage}} \right)$$

where $\mathcal{W} \leftarrow \mathsf{LeakingWires}(C, p)$ *and* $[\![\boldsymbol{y}]\!], \boldsymbol{z} \leftarrow G([\![\boldsymbol{x}]\!], \boldsymbol{a})$.

B Expansion Strategy

B.1 Random Probing Expandability

In [2], an *expansion* approach was introduced to build a random-probing secure circuit compiler from a secure multi-party protocol. This approach was later revisited in [8] with the formalization of the notion of *expanding compiler* [8].

The principle of the expanding compiler is to recursively apply a base compiler, denoted CC and which simply consists in replacing each gate in the input circuit by the corresponding gadget. Assume we have n-share gadgets G_g for each gate g in a basis $\mathbb{B}$. The base compiler CC simply consists in replacing each gate g in these gadgets by G_g and by replacing each wire by n wires carrying a sharing of the value. We thus obtain n^2-share gadgets by simply applying CC to each gadget: $G_g^{(2)} = \mathsf{CC}(G_g)$. 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 g is replaced by a base gadget G_g. The second output circuit $\widehat{C}_2$ is the original circuit C in which each gate is replaced by an n^2-share gadget $G_g^{(2)}$. 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 has been replaced by n^k wires carrying an (n^k)-linear sharing of the original wire.

Experiment 8: RPE

```
$\mathcal{W} \leftarrow \mathsf{LeakingWires}(G, p)$                Drawing the leaking wires.
$(I_1, I_2, J') \leftarrow \mathsf{Sim}_1(G, \mathcal{W}, J)$       The simulator is given the indexes of the
                                                leaking outputs $(J)$. It chooses indexes for
                                                the inputs $(I_1, I_2)$.
$out_1, out_2 \leftarrow \mathsf{Sim}_2([\![x_1]\!]|_{I_1}, [\![x_2]\!]|_{I_2})$   The simulator is given the shares at desired
                                                indexes and returns simulated values for the
                                                leaking wires $(out_1)$ and outputs $(out_2)$.
return $(I_1, I_2, J', out_1, out_2)$.
```

The expanding compiler achieves random probing security if the base gadgets verify a property called *random probing expandability* [8]. We recall hereafter the original definition of the random probing expandability (RPE) property for 2-to-1 gadgets.

Definition 18 (Random Probing Expandability from [8]). *Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n-share 2-to-1 gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is* (t, f)-random probing expandable ((t, f)-RPE for short) *if there there exists a PPT stateful two-stage simulator* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *such that for every admissible pair $(([\![x_1]\!], [\![x_2]\!]), [\![y]\!]) \in (\mathbb{K}^n \times \mathbb{K}^n) \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the outputs of Experiment 8 are such that*

1. $\Pr(|I_1| > t) = \Pr(|I_2| > t) = f(p)$ *and* $\Pr((|I_1| > t) \wedge (|I_2| > t)) = f(p)^2$,
2. *$J' \subseteq [n]$ is such that $J' = J$ if $|J| \leq t$ and $|J'| = n - 1$ otherwise,*
3. *and*

$$(out_1, out_2) \overset{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}\left(G, ([\![x_1]\!], [\![x_2]\!]), [\![y]\!]\right)}_{\texttt{Induced Random probing leakage}}, \quad \underbrace{[\![y]\!]|_{J'}}_{\texttt{Output leakage}} \right)$$

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

The first condition of Definition 18 defines two events, called "failure events" $\mathcal{F}_1 \equiv \big(|I_1| > t\big)$ and $\mathcal{F}_2 \equiv \big(|I_2| > t\big)$. In particular, condition 1 implies that $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent.
The second condition may seem artificial as the simulator is able to modify the indexes of leaking outputs. This is actually necessary to capture some specific events (when $|J| \geq t$) in the proofs, we refer to [8] for more details.

Remark 2. The RPE notion can be simply extended to gadgets with 2 outputs: the Sim_1 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 simulator must then produce an output with the knowledge of $[\![y_1]\!]|_{J_1'}$ and $[\![y_2]\!]|_{J_1'}$ where $[\![y_1]\!]$ and $[\![y_2]\!]$ are the output sharings.

The RPE notion can also be simply extended to gadgets with a single input: the Sim_1 simulator produces a single set I so that the failure event $(|I| > t)$ occurs with probability ε (and the Sim_2 simulator is then simply given $[\![x]\!]|_I$ where $[\![x]\!]$ is the single input sharing). We refer the reader to [8] for the formal definitions of these variants.

B.2 Instantiation of Noise Generation from RPE

In this section, we rely on the expansion strategy from [8] (and recalled in Section B.1) to build a noise generation gadget from an addition gadget. Specifically, based on the analysis and instantiations from [10], we begin with the following small addition and copy gadgets of 3 and 5 shares, which we then expand to increase their original security level for various leakage probabilities:

$$\begin{aligned} {\mathrm{G}_{\mathrm{add}}}^3 &: (\mathbb{K}^3)^2 \to \mathbb{K}^3 \\ ((a_1,a_2,a_3),(b_1,b_2,b_3)) &\mapsto (c_1,c_2,c_3) \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow (r_1 + a_1) + (r_3 + b_1) \\ c_2 &\leftarrow (r_2 + a_2) + (r_4 + b_2) \\ c_3 &\leftarrow (r_1 + r_2 + a_3) + (r_3 + r_4 + b_3) \end{aligned}$$

$$\begin{aligned} {\mathrm{G}_{\mathrm{copy}}}^3 &: \mathbb{K}^3 \to (\mathbb{K}^3)^2 \\ (a_1,a_2,a_3) &\mapsto ((c_1,c_2,c_3),(d_1,d_2,d_3) \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow r_1 + a_1 & d_1 &\leftarrow r_3 + a_1 \\ c_2 &\leftarrow r_2 + a_2 & d_2 &\leftarrow r_4 + a_2 \\ c_3 &\leftarrow r_1 + r_2 + a_3 & d_3 &\leftarrow r_3 + r_4 + a_3 \end{aligned}$$

and

$$\begin{aligned} {\mathrm{G}_{\mathrm{add}}}^5 &: (\mathbb{K}^5)^2 \to \mathbb{K}^5 \\ ((a_i)_{1\leq i\leq 5},(b_i)_{1\leq i\leq 5}) &\mapsto (c_i)_{1\leq i\leq 5} \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow (r_1 + r_2 + a_1) + (r_6 + r_7 + b_1) \\ c_2 &\leftarrow (r_2 + r_3 + a_2) + (r_7 + r_8 + b_2) \\ c_3 &\leftarrow (r_3 + r_4 + a_3) + (r_8 + r_9 + b_3) \\ c_4 &\leftarrow (r_4 + r_5 + a_4) + (r_9 + r_{10} + b_4) \\ c_5 &\leftarrow (r_5 + r_1 + a_5) + (r_{10} + r_6 + b_5) \end{aligned}$$

$$\begin{aligned} {\mathrm{G}_{\mathrm{copy}}}^5 &: (\mathbb{K}^5)^2 \to \mathbb{K}^5 \\ (a_i)_{1\leq i\leq 5} &\mapsto ((c_i)_{1\leq i\leq 5},(d_i)_{1\leq i\leq 5}) \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow r_1 + r_2 + a_1 & d_1 &\leftarrow r_6 + r_7 + a_1 \\ c_2 &\leftarrow r_2 + r_3 + a_2 & d_2 &\leftarrow r_7 + r_8 + a_2 \\ c_3 &\leftarrow r_3 + r_4 + a_3 & d_3 &\leftarrow r_8 + r_9 + a_3 \\ c_4 &\leftarrow r_4 + r_5 + a_4 & d_4 &\leftarrow r_9 + r_{10} + a_4 \\ c_5 &\leftarrow r_5 + r_1 + a_5 & d_5 &\leftarrow r_{10} + r_6 + a_5 \end{aligned}$$

Using `IronMask` from [9], we obtain the RPE advantage of both gadgets for $t = 1$ with 3 shares and $t = 2$ with 5 shares and for any probability p. For 3 shares, we have:

$$f_3(p) \approx 171p^2 + 36.4p^2\sqrt{p} + 72p^3 + 364.9p^3\sqrt{p} + 12648p^4 + \mathcal{O}(p^5)$$

with a maximum leakage probability (*i.e.* the maximum p for which $f(p) < p$) of $2^{-7.4}$, and for 5 shares, we have:

$$f_5(p) \approx 2870p^3 + 476.4p^3\sqrt{p} + 75p^4 + 9230.2p^5 + \mathcal{O}(p^6)$$

with a maximum leakage probability (*i.e.* the maximum p for which $f(p) < p$) of $2^{-5.77}$. Table 8 displays the levels of expansion k that are required for gadgets ${\mathrm{G}_{\mathrm{add}}}^3$ and ${\mathrm{G}_{\mathrm{add}}}^5$ and different probabilities to reach 128 bits of security, with the corresponding complexities. The latter are computed using the expanding compiler from [8].

As expected, the resulting complexities for the same leakage probability and security threshold are better for ${\mathrm{G}_{\mathrm{add}}}^5$. While we could theoretically further increase the number of shares before the expansion, we are limited by the complexity of the automatic tools (here `IronMask`).

Table 8: Security and complexity of expansions of gadgets ${\mathrm{G}_{\mathrm{add}}}^3$ and ${\mathrm{G}_{\mathrm{add}}}^5$.

	Gadget ${\mathrm{G}_{\mathrm{add}}}^3$					Gadget ${\mathrm{G}_{\mathrm{add}}}^5$				
leakage prob. p	2^{-8}	2^{-12}	2^{-16}	2^{-20}	2^{-24}	2^{-8}	2^{-12}	2^{-16}	2^{-20}	2^{-24}
expansion level k	8	5	4	4	3	4	3	3	2	2
# shares n	6561	243	81	81	27	625	125	125	25	25
# leaking shares t	2187	81	27	27	9	250	50	50	10	10
# randoms	854M	253k	17k	17k	1k	500k	14k	14k	400	400
# additions	1709M	506k	34k	34k	2k	1001k	29k	29k	825	825
# copies	854M	253k	17k	17k	1k	500k	14k	14k	400	400
advantage $f^k(p)$	2^{-152}	2^{-154}	2^{-145}	2^{-209}	2^{-140}	2^{-188}	2^{-175}	2^{-283}	2^{-134}	2^{-170}

Using the expansion method, the resulting $\mathrm{G}_{\mathrm{add}}$ gadgets are only RPC for a fixed t once the level of expansion is determined (according to the probability and the target security level). Therefore, we rely on the composition theorem from [8] to compute the RPC security of AddNoiseTo for selected probabilities and size α of the auxiliary input. Specifically, the security advantage of AddNoiseTo is upper bounded by the product of the security advantage of $\mathrm{G}_{\mathrm{add}}$ with the number of such gadgets. The whole gadget requires t (with its post-expansion value) regular input shares (of $[\![x]\!]$) and $t \cdot \lceil \frac{\alpha}{n} \rceil$ auxiliary inputs for the simulation of the leakage and t output shares. For instance, with $\alpha = 75$ auxiliary inputs and a leakage probability $p = 2^{-24}$, we need to combine 3 gadgets $\mathrm{G}_{\mathrm{add}}$ to form a gadget AddNoiseTo. The resulting RPC advantage is upper bounded by 2^{-169}, and 10 shares from the main input and 30 auxiliary inputs are required for the simulation. As for the complexity, the noise gadget requires 1200 randoms and implements 2475 additions and 1200 copies. In this setting, the selection of $\mathrm{G}_{\text{sum-chain}}$ or $\mathrm{G}_{\text{sum-tree}}$ does not affect neither the security, nor the complexity (except when parallelization is activated).

B.3 Instantiation of Raccoon from RPE

To implement the key generation and the signature with the expansion strategy, we miss a linear multiplication gadget, $\mathrm{G}_{\mathrm{cmult}}$, for which we give instantiations below for 3 and 5 shares. We simply follow the structures of $\mathrm{G}_{\mathrm{add}}$ and $\mathrm{G}_{\mathrm{copy}}$ with a refresh of the inputs preceding the linear multiplication.

To implement key generation and signature using the expansion strategy, we need a linear multiplication gadget, $\mathrm{G}_{\mathrm{cmult}}$. Below, we provide specific instantiations for 3 and 5 shares. The design mirrors the structures of $\mathrm{G}_{\mathrm{add}}$ and $\mathrm{G}_{\mathrm{copy}}$, with the inputs refreshed before performing the linear multiplication.

$$
\begin{aligned}
{\mathrm{G}_{\mathrm{cmult}}}^3 &: \mathbb{K}^3 \times \mathbb{K} \to \mathbb{K}^3 \\
&((a_1, a_2, a_3), \alpha) \mapsto (c_1, c_2, c_3)
\end{aligned}
$$

$$
\begin{aligned}
c_1 &\leftarrow \alpha \cdot (r_1 + a_1) \\
c_2 &\leftarrow \alpha \cdot (r_2 + a_2) \\
c_3 &\leftarrow \alpha \cdot (r_1 + r_2 + a_3)
\end{aligned}
$$

$$
\begin{aligned}
{\mathrm{G}_{\mathrm{cmult}}}^5 &: \mathbb{K}^5 \times \mathbb{K} \to \mathbb{K}^5 \\
&((a_1, a_2, a_3, a_4, a_5), \alpha) \mapsto (c_1, c_2, c_3, c_4, c_5)
\end{aligned}
$$

$$
\begin{aligned}
c_1 &\leftarrow \alpha \cdot (r_1 + r_2 + a_1) \\
c_2 &\leftarrow \alpha \cdot (r_2 + r_3 + a_2) \\
c_3 &\leftarrow \alpha \cdot (r_3 + r_4 + a_3) \\
c_4 &\leftarrow \alpha \cdot (r_4 + r_5 + a_4) \\
c_5 &\leftarrow \alpha \cdot (r_5 + r_1 + a_5)
\end{aligned}
$$

We use `IronMask` from [9] to compute the new RPE advantages for $\mathrm{G}_{\mathrm{cmult}}$ combined with $\mathrm{G}_{\mathrm{add}}$ and $\mathrm{G}_{\mathrm{copy}}$. They are identical to those obtained for $\mathrm{G}_{\mathrm{add}}$ without $\mathrm{G}_{\mathrm{cmult}}$ in Section B.2.

In Table 8, we identify a single configuration where the number of shares n and the number of leaking shares t correspond to an existing version in Raccoon (version 16), specifically considering the total number of small uniforms r_i and r'_i and the number of them that can leak. We denote by m_r the number of small uniforms that are added to each coefficient in $\mathbb{Z}_q$ and by m_s the number of small uniforms that cannot be revealed for each coefficient in $\mathbb{Z}_q$. Using our expansion strategy, the small uniforms must be captured inside a n-share variable, that is, there exists $\alpha \in \mathbb{N}^*$ such that

$$(\alpha - 1)n < m_r \leq \alpha n.$$

Then, we must ensure that the number of input shares that are required for the leakage simulation is lower than $m_r - m_s$. This constraint can be written as follows:

$$(\alpha - 1)t + \min(t, n - (\alpha n - m_r)) \leq m_r - m_s.$$

It can be observed that the only column in Table 8) satisfying this inequality corresponds to the use of gadget ${\mathrm{G}_{\mathrm{add}}}^3$ with $n = 27$ shares and an expansion level $k = 3$. For each coefficient in $\mathbb{Z}_q$, 27 small uniforms can be packed into a first sharing, while the remaining 5 small uniforms are packed into a second sharing, supplemented with 22 zeros. The RPC advantage of Raccoon's construction under this configuration then represents the probability that $t + 5 = 14$ small uniforms are required to simulate the leakage and t outputs, with $32 - 14 = 18$ small uniforms remaining secret, exceeding the expected 17 ones.

With these parameters in Raccoon-128, we count almost 16 million base gadgets for key generation and nearly 19 million base gadgets for the signature. Since no implementation is available for the decoding gadget using the expansion strategy, its impact on complexity and security is omitted. Consequently, the security advantage is upper-bounded by 2^{-116} in both cases, for a leakage probability of 2^{-24}. This is calculated by multiplying the atomic security advantage of a single gadget (*i.e.* 2^{-140} from Table 8) by the total number of gadgets. The complexity with this expansion strategy, including the number of logical gates and intermediate random values, is provided in Table 7 and can be compared to the original scheme with 16 shares.

C Proofs of Raccoon's Random Probing Security

C.1 Security Proof for Raccoon's Key Generation

Proof (Proof of Theorem 3). Due to the complexity of the key generation circuit, calculating a global cardinal-RPC-AI envelope is computationally impractical. Even before considering the internal wires, we already have $(\ell + k)n_R = 4.608$ n-share inputs and $\ell n_R = 2.048$ outputs, for which enumerating all possible values in $[0, n]$ would be too expensive. As a result, we opt to decompose the scheme into sub-components, each of which is individually proven to be cardinal-RPC-AI-O. For these sub-components, we calculate the corresponding threshold-RPC-AI-O, using a fixed threshold $t = \frac{n}{2}$. Finally, we sum the threshold-RPC-AI-O advantages of these blocks to determine the overall threshold-RPC-AI-O security of the complete scheme.

Specifically, we identify three different types of blocks:

- Block 1, illustrated on Figure 6, consists of $\mathrm{G}_{\mathrm{sum}}$ and $\mathrm{G}_{\mathrm{copy}}$ gadgets. We first compute the cardinal-RPC-AI envelope of the combination of both gadgets $\mathrm{G}_{\mathrm{sum}}$ and $\mathrm{G}_{\mathrm{copy}}$, then the threshold-RPC-AI advantage for an attacker which gets $t = \frac{n}{2}$ shares of each output and when the auxiliary input r_i^j is constrained to have a global cardinality of at most $t' = 15$. The resulting threshold-RPC-AI advantage is then multiplied by $n_R = 512$ and $\ell = 4$ (following the composition property of [8]) to obtain an upper bound on the threshold-RPC-AI advantage for the left part of the circuit, whose outputs will be used as inputs for $\mathrm{G}_{\mathrm{cmmult}}$.
- Gadget $\mathrm{G}_{\mathrm{cmmult}}$ is first split into three blocks. The ℓn_R $\mathbb{Z}_q$ n-share outputs of the left block are first each copied k times. We compute the cardinal-RPC envelope of the corresponding ${\mathrm{G}_{\mathrm{copies}}}^k$ gadget from which we derive its threshold-RPC advantage with t output shares and when we restrict the cardinality of the input shares to t as well. Then each of the ℓk copies in $\mathcal{R}_q$ goes through a linear polynomial multiplication with gadget $\mathrm{G}_{\mathrm{cpmult}}$. Gadget $\mathrm{G}_{\mathrm{cpmult}}$ consists in the generation of n_R copies of n_R elements in $\mathbb{Z}_q$. We are able to tightly compute the threshold-RPC advantage of the combination of $\mathrm{G}_{\mathrm{copy}}$ gadgets

assembled in a tree to compute n_R copies based on the cardinal-RPC envelopes of each block and by restricting the cardinal of the output shares to t and the cardinal of the input shares to at most t. Then, n_R^2 linear multiplications in $\mathbb{Z}_q$ are performed for which we have the cardinal-RPC envelope from Section 4.2 and from which we can derive the threshold-RPC advantage when restricting the input and output cardinal with respect to t. Finally, we have n_R groups of n_R elements of $\mathbb{Z}_q$ that are summed together. We use gadget $\mathrm{G}_{\mathrm{sum}}$ with the tree structure to compute additions of 4 shared variables (we need $128 + 32 + 8 + 4 + 1 = 173$ of them). From the cardinal-RPC envelopes of $\mathrm{G}_{\mathrm{sum}}$, we derive their threshold-RPC advantage with t and we sum them to obtain the global threshold-RPC advantage of all the additions. Summing all the computed threshold-RPC advantages (following the composition property of [8]) gives us an upper bound the threshold-RPC advantage of $\mathrm{G}_{\mathrm{cmmult}}$.

– Lastly, we compute the global threshold-RPC-AI-O advantage for the right block, whose inputs are the outputs of $\mathrm{G}_{\mathrm{cmmult}}$. We begin by determining the cardinal-RPC-AI-O envelope for the combination of AddNoiseTo and $\mathrm{G}_{\mathrm{decode}}$, then we compute the threshold-RPC-AI-O advantage when we restrict the input shared cardinality to be less or equal to t and the cardinality of the auxiliary inputs $r_i'^j$ to be at most 15. Similar to the process in block 1, we then multiply the resulting threshold-RPC-AI advantage by $n_R = 512$ and $k = 5$ (in line with the composition property from [8]), yielding an upper bound on the threshold-RPC-AI-O advantage for this right block.

All these advantages are computed for different probabilities and for instantiations of RPRefresh with different values of γ for each of the three blocks. The code is provided in Supplementary Material. The resulting threshold-RPC-AI-O advantages are displayed with the corresponding complexities in Table 7. □

C.2 Security Proof for Raccoon's Signature

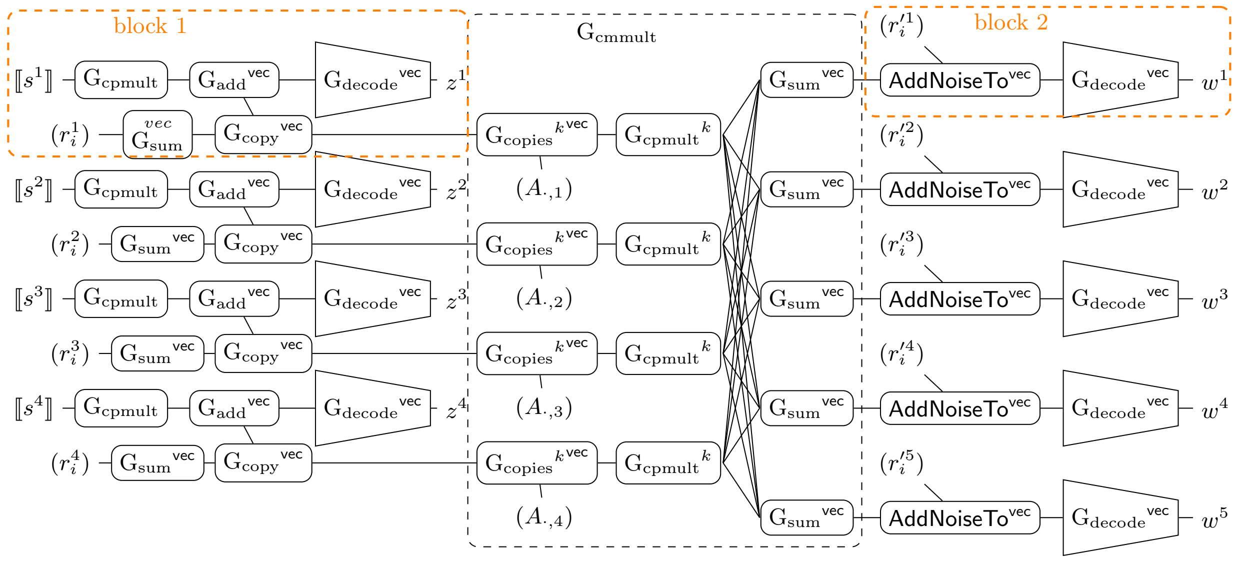

block 1
$\mathrm{G}_{\mathrm{cmmult}}$
block 2
$[\![s^1]\!]$
$\mathrm{G}_{\mathrm{cpmult}}$
$\mathrm{G}_{\mathrm{add}}$ vec
$\mathrm{G}_{\mathrm{decode}}$ vec
z^1
(r_i^1)
$\mathrm{G}_{\mathrm{sum}}^{vec}$
$\mathrm{G}_{\mathrm{copy}}$ vec
$\mathrm{G}_{\mathrm{copies}}$ k vec
$\mathrm{G}_{\mathrm{cpmult}}$ k
$(A_{\cdot,1})$
$(A_{\cdot,2})$
$(A_{\cdot,3})$
$(A_{\cdot,4})$
$\mathrm{G}_{\mathrm{sum}}$ vec
AddNoiseTo vec
$(r_i'^1)$
w^1
w^2
w^3
w^4
w^5
z^2
z^3
z^4

Fig. 7: RP-Sign for Raccoon-128 with $\ell = 4$ and $k = 5$

Proof. The proof for Raccoon's signature is very close to the proof for Raccoon's key generation, except for the computation of the threshold-RPC-AI-O advantage of the first block. Specifically, we tightly compose the envelopes of all the gadgets in block 1, except $\mathrm{G}_{\mathrm{cpmult}}$, and derive the corresponding threshold-RPC-AI-O advantage for t. We separately compute the threshold-RPC advantage of $\mathrm{G}_{\mathrm{cpmult}}$ for t, as in the proof of

Theorem 3. Following [8], we sum the advantages of both parts (the first one being multiplied by ℓn_R and the advantage of $\mathrm{G}_{\mathrm{cpmult}}$ being multiplied by ℓ) to obtain the global threshold-RPC-AI-O of block 1. The script to compute the global threshold-RPC-AI-O of Raccoon's signature is provided in Supplementary Material and a few values are provided in Table 7. $\square$