

A Unified Analysis of Refresh Gadgets in the Random Probing Model

Sonia Belaïd¹, Victor Normand^{1,2}, and Matthieu Rivain¹

¹ CryptoExperts, Paris, France

{firstname.lastname}@cryptoexperts.com,

² DIENS, Ecole normale supérieure, PSL University, CNRS, Inria, Paris, France

Abstract. Masking is a standard countermeasure against side-channel attacks on embedded cryptographic implementations. Its security is commonly analyzed in the random probing model, which offers a useful trade-off between realistic leakage assumptions and tractable security proofs. Recent years have seen the emergence of several masking compilers based on compositional security frameworks such as general/cardinal random probing composability (RPC). Most of these approaches rely on dedicated refresh gadgets whose structure is often tightly coupled to the targeted security analysis.

In this work, we investigate the impact of refresh gadgets on the random probing security of masking compilers within the recent general/cardinal RPC frameworks. We formalize two broad families of refresh gadgets, namely direct and zero encoding-based refreshes, define ideal constructions for both families under explicit randomness and uniformity conditions, and extend the compositional analysis of zero encoding-based refreshes. We then introduce an atomic refresh function capturing the core operation underlying most refresh gadgets from the literature. This abstraction allows us to express and analyze existing constructions within a unified framework, derive their cardinal/general RPC envelopes, and obtain analytical formulas for several of them. Finally, we compare ideal and concrete refresh gadgets on masked implementations of AES and Raccoon. Our results provide a systematic comparison of the security-complexity trade-offs achieved by current refresh strategies.

Keywords: Masking, Random Probing Security, Refresh Gadgets

1 Introduction

Cryptographic algorithms are generally considered to be secure in the black-box model. However, in practice, cryptographic implementations are vulnerable to *side-channel attacks* that exploit physical emanations from the device, such as execution time, power consumption or electromagnetic emissions. In particular, the leakage from secret-dependent intermediate computations can often be exploited to recover sensitive information significantly more efficiently than through purely black-box attacks.

In order to thwart side-channel attacks, a widely adopted countermeasure is *masking*. Introduced simultaneously by Goubin and Patarin [19] and Chari et al. [16] in 1999, masking consists of splitting each sensitive value x into n shares such that recovering x requires the knowledge of all the shares.

To reason about the security of masking schemes, the community has introduced various *leakage models*, which formalize the information leaked during a computation. The *t-probing model* was originally introduced in the seminal work of Ishai, Sahai and Wagner [20]. In this model, a circuit is said to be *t-probing secure* if any set of t intermediate values is independent of the secret. Although this model is well suited for security proofs, it does not fully capture the information leakage occurring during the computation. In particular, it fails to model the information that can be accumulated through repeated manipulations of the same internal values, as exploited by horizontal attacks [5]. Alternatively, the *noisy leakage model* [16,25] describes more faithfully the leakage measured on embedded devices. In this model, the adversary learns, for every intermediate variable, the output of a noisy leakage function. Nevertheless, this model is less suitable for formal security proofs, as no proof has currently been established directly in this setting without relying on strong assumptions.

In [18], the authors introduced a reduction from the noisy leakage model to the probing model, together with an intermediate leakage model called the *random probing model*, later formalized in [8]. In this model, each intermediate value leaks independently with some fixed probability, called the *leakage rate*. The random probing model offers a useful compromise between realistic assumptions and tractable security proofs.

Several automatic tools have been developed to verify the random probing security of a masked circuit [8,15,9,23,6,7], but they do not scale well to large circuits or high numbers of shares. A more scalable approach is to rely on masking compilers that assemble small gadgets satisfying local security properties, so that their composition guarantees random probing security.

A first line of works [1,3,2,8,12,13] relies on expansion techniques, iteratively applying masking to already masked circuits in order to tolerate constant leakage rates. While these constructions achieve arbitrary security levels, they often lead to large and costly circuits. Other approaches instead rely on direct composition frameworks, introducing stronger gadget-level notions to derive tighter circuit-level guarantees. In particular, *general random probing composability* (general RPC) [15] characterizes, for every set of output shares, the probability distribution of the input shares required by a leakage simulator. Simpler notions such as *(uniformly) cardinal RPC* [11,10] only track the number of required input and output shares, possibly up to share permutations, leading to more compact analyses. More recently, Berti et al. [14] introduced a compositional framework based on leakage diagrams, providing explicit analytical security bounds as a function of the leakage probability and the number of shares. Jahandideh et al. [21] proposed an alternative algebraic approach based on mutual-information bounds, achieving improved security/performance trade-offs but without directly implying standard simulation-based random probing security.

Most compositional masking frameworks rely on *refresh gadgets*, which re-randomize the shares of a sensitive value in order to reduce dependencies between successive computations. The first such gadget was introduced by Rivain and Prouff [26] in the probing model. Since then, many variants have been proposed, often tailored to a specific compositional framework or leakage setting. For instance, the recent works [11,10] rely on the **RPRefresh** gadget introduced in [11], while the leakage diagram framework of [14] and the algebraic approach of [21] are restricted to the *circular refresh* gadget from [4], whose cyclic structure is particularly convenient for their analyses. Other constructions have also been considered, such as the $\mathcal{O}(n \log n)$ refresh gadget of [5], which appears to provide strong resistance against horizontal attacks. Finally, the recent **perseus** tool [7] showed that, in some cases, even lightweight refresh gadgets such as the half refresh, using only $n/2$ random values, may suffice to achieve security.

In his 2021 PhD thesis [22], Mathieu-Mahias showed that the $\mathcal{O}(n \log n)$ refresh gadget satisfies stronger security properties in the probing model at a lower randomness cost when the random values are first combined into a sharing of zero before being added to the input shares through a single addition layer. This technique is now commonly referred to as *zero encoding-based refresh*. In [11], the authors introduced the *cardinal RPP* notion to characterize the security of such zero encoding-based constructions, from which cardinal/general RPC envelopes can be derived.

Overall, each compositional framework tends to rely on refresh gadgets whose structure is particularly convenient for its security analysis. However, despite the variety of available constructions, there is currently no systematic understanding of how these different refresh strategies compare in terms of random probing security once embedded into a compiled circuit.

Our contributions. In this work, we investigate and compare the impact of refresh gadgets on the random probing security of masking compilers within the recent cardinal/general RPC frameworks, through a unified framework. Our main contributions are the following.

- We formalize two broad families of refresh gadgets: *direct refresh gadgets*, which directly manipulate the input sharing, and *zero encoding-based refresh gadgets*, which first generate a randomized zero-sharing before combining it with the input sharing. Together they capture all the refresh constructions from the literature.
- We extend the composability framework of [11] for zero encoding-based constructions by introducing the notion of *general Random Probing Partitionability* (general RPP).

- We define ideal refresh gadgets for both families under explicit randomness and uniformity conditions. These ideal constructions only expose the minimal set of wires that are structurally unavoidable under these conditions. They provide formal upper bounds on the random probing security achievable by concrete refresh gadgets satisfying these conditions, and serve more generally as best-case reference benchmarks for the corresponding refresh families.
- We compare these ideal gadgets with the **RPreFresh** gadget from [11], currently the only refresh gadget with explicit cardinal RPC envelopes.
- We introduce a generic *atomic refresh function* capturing the core operation underlying most refresh gadgets from the literature. Using this abstraction, we derive cardinal/general RPC properties for direct constructions and cardinal/general RPP properties for zero encoding constructions, together with corresponding composition rules.
- Using this atomic abstraction, we derive analytical envelope formulas for the most widely used refresh gadgets, including the linear refresh of [26], the half refresh [7], the double half refresh inspired from [4], the $\mathcal{O}(n \log n)$ refresh [5], and **RPreFresh** [11]. We also extend the **IronMask** tool [9] to automatically compute general and cardinal RPC envelopes.
- Finally, we apply our framework to masked implementations of AES and Raccoon [17,24], the latter involving only (quasi) linear operations. We compare ideal and concrete refresh gadgets in terms of security/complexity trade-offs under random probing leakage.

The source code and scripts used for our experiments are available at <https://github.com/CryptoExperts/AC26-cardinal-rp-compiler>. The extensions to IronMask developed in this work are available in the IronMask repository: <https://github.com/CryptoExperts/IronMask>.

2 Preliminaries

2.1 Notations

Throughout the paper, $\mathbb{K}$ denotes a finite field or a finite ring. For $n \in \mathbb{N}$, we write $[n] = \{1, \dots, n\}$. We denote for all $i_1, i_2 \in [n]$:

$$[i_1, i_2]_{\circ} := \begin{cases} \{i_1, i_1 + 1, \dots, i_2\} & \text{if } i_1 \leq i_2 \\ \{i_1, i_1 + 1, \dots, n, 1, \dots, i_2\} & \text{if } i_1 > i_2 \end{cases}$$

Partitions. We denote by Π_n the set of partitions of $[n]$. For $P \in \Pi_n$ and $i, j \in [n]$, we write $i \sim_P j$ if there exists a set $I \subset P$ such that $i, j \in I$, *i.e.*, if i and j belong to the same set of P . For $P, Q \in \Pi_n$, we say that P is *thinner* than Q , and write $P \leq Q$ if for all $i, j \in [n]$, $i \sim_P j \Rightarrow i \sim_Q j$. We denote by $\mathcal{P}(n)$ the set of integer partitions of n .

Vectors. For $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{K}^n$ and $I \subseteq [n]$, we denote by $\mathbf{x}|_I = (x_i)_{i \in I}$ the restriction of $\mathbf{x}$ to the coordinates indexed by I .

Distributions. Two probability distributions D_1 and D_2 are ε -close, denoted $D_1 \approx_{\varepsilon} D_2$, if their statistical distance is at most ε , namely $\text{SD}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon$, where $p_{D_1}(\cdot)$ and $p_{D_2}(\cdot)$ denote the probability mass functions of D_1 and D_2 .

Distribution Envelopes. Since exact probabilities are often difficult to compute directly, we follow the approach of [11,10] and rely on upper bounds expressed through the notion of a *probability envelope*. For a discrete random variable $X \sim \mathcal{D}$, we say that $\mathcal{E}$ is an envelope of the distribution if for every value x , $\Pr[X = x] \leq \mathcal{E}(x)$, which we denote by $X \lesssim \mathcal{E}$.

When dealing with distributions conditioned on disjoint events, we use collections of envelopes indexed by structured parameters. For example, we may consider collections $\mathcal{E} = (\mathcal{E}_{(t_1^{\text{out}}, \dots, t_m^{\text{out}})})$ indexed by tuples $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$, or collections $\mathcal{E} = (\mathcal{E}_{(J_1, \dots, J_m)})$ indexed by sets $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$. Each index corresponds to a distinct distribution together with its associated envelope.

2.2 Linear Sharing, Circuits and Gadgets

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 $\llbracket x \rrbracket = (x_1, \dots, x_n) \in \mathbb{K}^n$ such that $x_1 + \dots + x_n = x$. It is uniform if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $\llbracket x \rrbracket|_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. An n -linear encoding is a probabilistic algorithm LinEnc which on input of a tuple $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{K}^\ell$ outputs a tuple $\llbracket \mathbf{x} \rrbracket = (\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \in (\mathbb{K}^n)^\ell$ such that $\llbracket x_i \rrbracket$ is a uniform n -sharing of x_i for every $i \in [\ell]$.

An *arithmetic circuit* over a finite field (or finite ring) $\mathbb{K}$ is a labeled directed acyclic graph whose edges are called *wires* and whose vertices are *arithmetic gates* performing operations over $\mathbb{K}$. The circuit is built from a base set of gates $\mathbb{B} = \{g : \mathbb{K}^\ell \rightarrow \mathbb{K}^m\}$, 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 $x \mapsto (x, x)$. We also consider *probabilistic* (or *randomized*) arithmetic circuits that include *random gates*. Each random gate outputs a fresh value sampled uniformly from $\mathbb{K}$.

Beyond randomness in values, randomness may also appear in the *wiring* of a circuit, as introduced in [10]. A *randomized-wiring arithmetic circuit* consists of a fixed set of gates together with a random choice of connections between them, determined by a random tape ρ . We denote by C^ρ the resulting deterministic circuit. This model naturally captures masked gadgets operating on randomly permuted share indices, such as RPreFresh [11] and the multiplication gadget from [10]. Importantly, the random tape ρ is assumed to be fully leaked to the adversary. Hence, in the random probing model, the simulator receives C^ρ for a random ρ . Unlike randomness in gates, wiring randomness only affects share indices and does not require leakage-resistant implementations.

In the following, we shall call an $(n\text{-share}, \ell\text{-to-}m)$ *gadget*, a randomized arithmetic circuit that maps an input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ to an output $\llbracket \mathbf{y} \rrbracket \in (\mathbb{K}^m)^m$ such that $\mathbf{y} = g(\mathbf{x})$ for some function g .

2.3 Random Probing Security

Let $p \in [0, 1]$ denote the *leakage rate*. In the p -random probing model, the evaluation of an arithmetic circuit C^ρ , instantiated with a random tape ρ , leaks the value carried by each wire independently with probability p . The adversary is also assumed to know the instantiated circuit C^ρ and its wiring, which we model by allowing full leakage of the random tape ρ .

Following [8], the random probing leakage of an arithmetic circuit is formally defined through two probabilistic algorithms.

- The *leaking-wires sampler* takes as input a circuit C^ρ and a leakage rate p , and outputs a set of wire labels $\mathcal{W} \leftarrow \text{LeakingWires}(C^\rho, p)$, constructed by including each wire of C^ρ independently with probability p .
- The *assign-wires sampler* takes as input a circuit C^ρ , a set of wires $\mathcal{W}$, and an input $\mathbf{x}$, and outputs $\mathbf{w} \leftarrow \text{AssignWires}(C^\rho, \mathcal{W}, \mathbf{x})$, where $\mathbf{w}$ contains the values carried by the wires in $\mathcal{W}$ during the evaluation of C^ρ on input $\mathbf{x}$.

Following [10], we recall the notions of *random probing leakage* and *random probing security* for randomized-wiring arithmetic circuits. In the special case where the wiring distribution is a singleton (i.e., the circuit has a fixed wiring), these definitions coincide with the classical ones for arithmetic circuits from [8].

Definition 2 (Random Probing Leakage). Let C be a randomized-wiring arithmetic circuit and $\mathbf{x}$ an input. The p -random probing leakage of C on input $\mathbf{x}$ is the distribution $\mathcal{L}_p(C, \mathbf{x})$ obtained by sampling a random tape ρ and composing the leaking-wires and assign-wires samplers:

$$\mathcal{L}_p(C, \mathbf{x}) \stackrel{\text{id}}{=} (\text{AssignWires}(C^\rho, \text{LeakingWires}(C^\rho, p), \mathbf{x}) \mid \rho \leftarrow \$).$$

Intuitively, unlike the t -probing model, the random probing model allows an arbitrary number of wires to leak, while weighting each leakage pattern by its probability. It also captures repeated manipulations: if the same sensitive value is carried by k wires, it leaks at least once with probability $1 - (1 - p)^k$. Conversely, leaking all n shares of an input occurs only with probability p^n .

Definition 3 (Random Probing Security). A randomized-wiring arithmetic circuit C with $\ell \cdot n$ input gates is (p, ε) -random probing secure with respect to an encoding Enc if there exists a simulator Sim such that for every $\mathbf{x} \in \mathbb{K}^\ell$,

$$Sim(C^\rho) \approx_\varepsilon \mathcal{L}_p(C, Enc(\mathbf{x})).$$

2.4 Random Probing Composability

Proving random probing security directly is generally feasible only for small gadgets and does not scale to large circuits. To overcome this limitation, several composability notions have been introduced, allowing stronger local security properties to imply global random probing security under composition.

Intuitively, these notions characterize which input shares a simulator needs in order to reproduce the internal leakage of a gadget together with some of its output shares. This information can then be propagated through gadget composition, since the outputs of one gadget become inputs to subsequent gadgets. The different RPC notions mainly differ in the granularity with which they describe both sides of this relation: which output shares are considered, and which input shares are required for their simulation.

The first such notion is *threshold random probing composability* (threshold RPC), introduced in [8]. It requires that the random probing leakage of a gadget's internal wires, together with any set of up to t output shares, can be simulated using at most t input shares, except with a simulation failure probability ε . A key advantage of this notion is its *trivial composability*: any circuit composed of threshold RPC gadgets remains threshold RPC, with the overall failure probability being bounded by the sum of the individual failure probabilities. However, this notion lacks tightness, since requiring more than t input shares to simulate a single gadget does not necessarily imply failure in a composed circuit.

To obtain tighter bounds, the notion of *general random probing composability* (general RPC) was introduced in [15] and later used in [11]. In this framework, the simulation failure probability is specified for every combination of output shares to be simulated together with internal leakage from every combination of input shares. This leads to the notion of a *probe distribution table* (PDT), also referred to as a *general RPC envelope*. However, computing such envelopes requires considering exponentially many combinations of input and output shares. In particular, for a circuit with m input sharings and ℓ output sharings, the general RPC envelope contains bounds for $2^{(m+\ell)n}$ distinct probabilities, where n is the number of shares per sharing.

To mitigate this complexity, the notion of *cardinal random probing composability* (cardinal RPC) was introduced in [11]. Instead of distinguishing individual sets of shares, this notion only considers their cardinalities. As a result, a cardinal RPC envelope for a circuit with m input sharings and ℓ output sharings contains $(n+1)^{(m+\ell)}$ bounds, which is polynomial in the number of shares n .

These composability notions were later adapted to randomized-wiring arithmetic circuits in [10]. In the special case where the wiring distribution is a singleton (i.e., the circuit has a fixed wiring), the resulting definitions coincide with their classical counterparts. For completeness, we recall the formal definition of general RPC in this setting. The definitions of threshold RPC and cardinal RPC are given in Appendix A.

Definition 4 (General RPC). Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^g$ be a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and defined over $\mathcal{P}([n])^\ell$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^g)$ -general RPC for some $p \in [0, 1]$ if there exists a two-stage simulator (Sim_1, Sim_2) such that for every shared input $\llbracket x \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \dots, J_m)$, the outputs of Experiment 1 are such that

1. $(I_1, \dots, I_\ell) \lesssim \mathcal{E}_{J_1, \dots, J_m}^g$,
2. $(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\text{AssignWires}(G^\rho, R, \mathcal{W}, \llbracket x \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right)$, where $\mathcal{W} \leftarrow \text{LeakingWires}(G^\rho, p)$
and $\llbracket y \rrbracket \leftarrow G^\rho(\llbracket x \rrbracket)$.

$\rho \leftarrow \$$ $\mathcal{W} \leftarrow \text{LeakingWires}(G^\rho, p)$ $I_1, \dots, I_\ell \leftarrow \text{Sim}_1(G^\rho, \mathcal{W}, J_1, \dots, J_m)$ $out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket_{I_1}, \dots, \llbracket x_\ell \rrbracket_{I_\ell})$ $\text{return } (I_1, \dots, I_\ell, out_1, out_2)$	<i>Sampling the random tape for the wires.</i> <i>Sampling the leaking wires.</i> <i>Sim₁ determines the indexes of the</i> <i>required inputs.</i> <i>Sim₂ returns simulated values of the</i> <i>internal leaking wires out₁ and the</i> <i>output leaking wires out₂</i>
---	---

3 Families of Refresh Gadgets and Composition

A *refresh gadget* takes as input a sharing and outputs a new sharing of the same secret, while injecting fresh randomness to re-randomize the shares.

In this section, we review the two main families of refresh gadgets used in masked implementations and highlight their structural differences. We show that the first family naturally fits within existing compositional frameworks such as cardinal and general Random Probing Composability (RPC). For the second family, we introduce a new security notion, called *general Random Probing Partitioning* (general RPP), which refines and extends the cardinal RPP notion from [11]³, together with the corresponding composition results.

3.1 Families of Refresh Gadgets

We distinguish between the two families of refresh gadgets used in masked implementations, depending on how fresh randomness is injected into the sharing: (i) randomness can be injected directly and incrementally into the shares (*direct refresh*), or (ii) randomness can first be structured as an explicit sharing of zero and then added sharewise to the input (*zero-encoding-based (ZE-based) refresh*).

Direct Refresh. A *direct refresh* injects fresh randomness into the shares incrementally, without first constructing an explicit zero-sharing through a dedicated zero-encoding primitive. Correctness is ensured by construction, through pairs of signed random values that cancel out globally.

Definition 5 (Direct Refresh). Let $n, m \in \mathbb{N}$. And let $\{(i_k, j_k, b_k)\}_{k=1}^{2m}$ be a sequence of triplets where $i_k \in [n]$, $j_k \in [m]$, and $b_k \in \{0, 1\}$ such that for every pair $(j, b) \in [m] \times \{0, 1\}$, there exists a unique k such that $(j_k, b_k) = (j, b)$. The direct refresh gadget defined with respect to this sequence is a probabilistic algorithm that, on input of a sharing $\llbracket x \rrbracket$, proceeds as follows:

1. Sample m independent random values $r_1, \dots, r_m \xleftarrow{\$} \mathbb{K}$.
2. Let $\llbracket y \rrbracket = \llbracket x \rrbracket$. For $k = 1$ to $2m$, update the sharing $\llbracket y \rrbracket$ as

$$y_{i_k} \leftarrow y_{i_k} + (-1)^{b_k} \cdot r_{j_k}.$$

3. Return $\llbracket y \rrbracket$.

Direct refresh gadgets were historically the first constructions proposed in the literature [26,5,4]. In his 2021 thesis, Mathieu-Mahias [22] showed that an optimized variant of the refresh gadget from [5] fails to satisfy the intended probing security guarantees in its original form. The desired security property is, however, restored when the construction is modified so that randomness is first generated via a zero-encoding primitive. This observation motivated the development of alternative refresh strategies.

³ In [11], this notion was simply referred to as RPP; we use the term *cardinal RPP* to distinguish it from general RPP.

Zero Encoding-based Refresh. Instead of injecting randomness incrementally, ZE-based refresh gadgets first generate an explicit sharing of zero, which is then added sharewise to the input. This separation between randomness generation and its injection into the shares can yield stronger compositional security guarantees. Today, all known refresh gadgets can be expressed either as direct refreshes or as zero encoding-based refreshes. Formally, zero encoding-based refresh gadgets relies on the following primitive.

Definition 6 (Zero Encoding from [11]). Let $n \in \mathbb{N}$. A zero encoding algorithm, denoted ZeroEnc , is a probabilistic algorithm sampling an n -sharing of 0. In other words, $\llbracket z \rrbracket \leftarrow \text{ZeroEnc}()$ such that $\sum z_i = 0$ (with probability 1).

Definition 7 (Zero Encoding-based Refresh). Let $n \in \mathbb{N}$. A Zero Encoding-based Refresh is a refresh gadget that relies on a zero encoding primitive (Definition 6). On input $\llbracket x \rrbracket = (x_1, \dots, x_n)$, it samples $\llbracket z \rrbracket \leftarrow \text{ZeroEnc}()$ and outputs $\llbracket y \rrbracket = (y_1, \dots, y_n)$ defined by

$$y_i = x_i + z_i \quad \text{for all } 1 \leq i \leq n.$$

Correctness follows immediately from the fact that the zero encoding gadget outputs an n -sharing of zero.

Remark 1. We stress that, in full generality, a refresh gadget might not strictly fit into either of these two families but be a hybrid of both approaches. In particular, some randomness may be summed before being injected into the shares, and this process might be repeated several times. However, to the best of our knowledge, all known refresh gadgets can be expressed as either direct refresh or zero encoding-based refresh as defined above.

3.2 Random Probing Partitioning Framework

In the following, we generalize the cardinal RPP property introduced in [11], designed to quantify the security of zero encoding primitives and derive the corresponding cardinal RPC envelopes of ZE-based refresh gadgets. More precisely, we refine it into a more expressive notion, called *general Random Probing Partitioning* (general RPP), enabling finer leakage analyses, tighter composition of zero encoding primitives, and derivation of general RPC envelopes. The associated composition theorem is formalized in Lemma 2.

Intuitively, RPP is an intermediate notion that characterizes the quality of a zero encoding after internal leakage. For a given leakage pattern, the output zero-sharing can be decomposed into a leakage-dependent component, accounting for the randomness revealed by the leakage, and a residual fresh component. The latter is described by a partition of the output shares: each block identifies a subset of shares that still jointly forms a fresh sharing of zero. The RPP envelope then upper bounds the probability of obtaining each such partition. This information can subsequently be translated into an RPC envelope when the zero encoding is used inside a ZE-based refresh gadget.

Extension of the RPP Property. For completeness, we recall in Appendix A.3 the original notion of *cardinal RPP*. Cardinal RPP only records the sizes of the blocks in the resulting partition. We introduce the finer notion of *general RPP*, which additionally records the specific share indices contained in each block. As for general versus cardinal RPC, this additional structural information enables more precise leakage analyses.

To this end, we first introduce a generalized partition zero encoding, denoted $\text{GeneralPartitionZeroEnc}$, which takes as input a partition of the index set $[n]$ (instead of an integer partition of n , as in CPZE).

Definition 8 (General Partition ZeroEncoding). For $P = (I_1, \dots, I_\ell) \in \Pi_n$, we define $\text{GeneralPartitionZeroEnc}(P)$ (GPZE) as a distribution producing a sharing $\llbracket z \rrbracket$ where, for every $j \in [1, \ell]$, $\llbracket z^j \rrbracket = (z_i)_{i \in I_j}$ is a fresh $|I_j|$ -sharing of 0 containing the shares of indices in I_j . In other words, a sample

$$\llbracket z \rrbracket \leftarrow \text{GPZE}(I_1, \dots, I_\ell)$$

is uniformly random over $\mathbb{K}^n$ subject to $\sum_{i \in I_1} z_i = \dots = \sum_{i \in I_\ell} z_i = 0$.

Experiment 2: General Random Probing Partition

$\mathcal{W} \leftarrow \text{LeakingWires}(\text{ZeroEnc}, p)$ $\rho \leftarrow \$$ $\llbracket z \rrbracket \leftarrow \text{ZeroEnc}(\rho)$ $\text{leak} \leftarrow \text{AssignWires}(\text{ZeroEnc}, \rho, \mathcal{W}, \emptyset)$ $\llbracket w \rrbracket \leftarrow \text{Sim}(\mathcal{W}, \text{leak})$ $P := \phi(\mathcal{W})$ $\llbracket u \rrbracket \leftarrow \text{GPZE}(P)$ $\llbracket z' \rrbracket \leftarrow \llbracket w \rrbracket + \llbracket u \rrbracket$ $\text{return } (P, \llbracket z \rrbracket, \llbracket z' \rrbracket)$	Drawing the leaking wires. Drawing a random tape. Computing $\llbracket z \rrbracket$ a sharing of 0 from ρ . Assigning the leaked values. Computing $\llbracket w \rrbracket$, a sharing of 0 corresponding to the leaked part of $\llbracket z \rrbracket$. Computing a partition corresponding to the leaking wires. Drawing an encoding of 0 corresponding to the unrevealed randomness. Summing the (un)revealed randomness together.
---	--

Definition 9 (General Random Probing Partition). Let $n \in \mathbb{N}$ and Π_n the set of partitions of $[n]$. Let $\mathcal{E}$ be a distribution envelope over Π_n . An n -share zero encoding gadget $\text{ZeroEnc} : \emptyset \rightarrow \mathbb{F}^n$ is $(p, \mathcal{E})$ general random probing partitioned (gRPP) for some $p \in [0, 1]$ if there exists a deterministic algorithm ϕ and a PPT simulator Sim such that the output of Experiment 2 satisfy :

1. $P \lesssim \mathcal{E}$
2. $\llbracket z \rrbracket$ and $\llbracket z' \rrbracket$ are identically distributed.

From General RPP to General RPC. Before stating our first theorem on the general RPC property of ZE-based refresh gadgets, we recall the artificial construction ZeroEnc_+ introduced in [11]. Informally, ZeroEnc_+ extends a zero encoding gadget ZeroEnc by allowing each output share to be individually leaked or manipulated after the execution of ZeroEnc . Equivalently, from the leakage perspective, each output share of ZeroEnc is additionally subject to independent leakage with probability p , on top of the internal leakages already considered.

This extension is useful for our analysis. Indeed, when a zero encoding is used within a ZE-based refresh gadget, its output shares become internal wires of the overall circuit and may therefore leak. Consequently, in order to derive the general RPC property of the refresh gadget, we rely on the general RPP property of ZeroEnc_+ .

Hence, we need a way to derive the general RPP envelope of ZeroEnc_+ from that of ZeroEnc . This is achieved in Lemma 1. Informally, starting from a partition $P \in \Pi_n$, we consider every subset $J \subseteq [n]$ of output shares that may additionally leak in ZeroEnc_+ . The corresponding partition is then refined by isolating the leaking shares into singleton blocks, and weighted by the probability that exactly the shares in J leak. Aggregating these contributions over all partitions and all subsets J yields the general RPP envelope of ZeroEnc_+ . The complete proof is provided in Appendix D.1.

Lemma 1 (General RPP of ZeroEnc_+). Let $n \in \mathbb{N}$. Let ZeroEnc be an n -share zero encoding gadget that is $(p, \mathcal{E})$ -gRPP, where $\mathcal{E}$ is defined over Π_n . Then ZeroEnc_+ is $(p, \mathcal{E}_+)$ -gRPP where $\mathcal{E}_+$ is build as follows :

1. For all $P' \in \Pi_n$, set $\mathcal{E}_+(P') \leftarrow 0$.
2. For all $P = (I_1, \dots, I_\ell) \in \Pi_n$ and for all $J = \{j_1, \dots, j_k\} \subseteq [n]$, update $\mathcal{E}_+(P')$ as

$$\mathcal{E}_+(P') \leftarrow \mathcal{E}_+(P') + \mathcal{E}(P) \cdot p^k \cdot (1-p)^{n-k}$$

where

$$P' = (I_1 \setminus J, \dots, I_\ell \setminus J, \{j_1\}, \dots, \{j_k\}) .$$

Using any **ZeroEnc** gadget with its general RPP envelope and then Lemma 1, we can derive the general RPC property of the corresponding ZE-based refresh gadget. Intuitively, RPP captures the residual freshness structure of the zero encoding after leakage, whereas RPC translates this information, once the zero encoding is added to the input sharing, into the input shares required by the simulator to reproduce the internal leakage together with a given set of output shares. In this sense, RPP provides the intermediate compositional information needed to analyze the complete refresh gadget. The formal conversion is stated in Theorem 1. The proof, given in Appendix D.2, follows the same strategy as Theorem 1 of [11], adapted to the general partition setting.

Theorem 1 (General RPC of ZE-based Refresh Gadgets). *Let $n \in \mathbb{N}$ and let Ref be an n -share zero encoding based refresh gadget based on the zero encoding primitive **ZeroEnc**. Assume that the extension ZeroEnc_+ (i.e., **ZeroEnc** with the additional manipulation of its output shares) is $(p, \mathcal{E}^{\text{ZeroEnc}_+})$ -general random probing partitioned. Then Ref is $(p, \mathcal{E})$ -general RPC, where for all $I, J \subseteq [n]$,*

$$\mathcal{E}_J(I) = \sum_{I' \subseteq I} q(I', J) \cdot p^{|I \setminus I'|} \cdot (1-p)^{n-|I|}$$

and

$$q(I', J) = \sum_{P \in \Pi_n : I' = \bigcup_{S \in P, S \subseteq J} S} \mathcal{E}^{\text{ZeroEnc}_+}(P)$$

Under similar arguments, we can derive the cardinal RPC envelope of ZE-based refresh gadgets.

Corollary 1 (Cardinal RPC of ZE-based Refresh Gadgets). *Let $n \in \mathbb{N}$ and let Ref be an n -share zero encoding based refresh gadget based on the zero encoding primitive **ZeroEnc**. Assume that the extension ZeroEnc_+ (i.e., **ZeroEnc** with the additional manipulation of its output shares) is $(p, \mathcal{E}^{\text{ZeroEnc}_+})$ -general random probing partitioned. Then Ref is $(p, \mathcal{E})$ -cardinal RPC, where for all $t^{in}, t^{out} \in [n]$,*

$$\mathcal{E}_{t^{out}}(t^{in}) = \sum_{t_1^{in}=0}^{t^{in}} q(t_1^{in}, t^{out}) p^{t^{in}-t_1^{in}} (1-p)^{n-t^{in}} \binom{n-t_1^{in}}{t^{in}-t_1^{in}}$$

and

$$q(t_1^{in}, t^{out}) = \max_{\{J \mid |J|=t^{out}\}} \left(\sum_{\substack{P \in \Pi_n \\ |S \in P, S \subseteq J| = t_1^{in}}} \mathcal{E}^{\text{ZeroEnc}_+}(P) \right)$$

RPP Composition Framework. The composition of direct refresh gadgets can be handled naturally within the cardinal and general RPC frameworks. In contrast, these frameworks do not provide a direct way to reason about the composition of zero encoding blocks prior to their addition to the input sharing.

Lemma 2 fills this gap by showing that the general RPP property is preserved under sharewise addition of zero encoding blocks, and by explicitly characterizing the resulting envelope.

Lemma 2 (General RPP of Sharewise Addition of ZeroEnc Blocks). *Let $n \in \mathbb{N}$. Let $\text{ZeroEnc}_+^{(1)}$ and $\text{ZeroEnc}_+^{(2)}$ be two independent n -share zero-encoding gadgets (with additional manipulation of the output shares) that are respectively $(p, \mathcal{E}^{(1)})$ -gRPP and $(p, \mathcal{E}^{(2)})$ -gRPP, where $\mathcal{E}^{(1)}$ and $\mathcal{E}^{(2)}$ are distributions over Π_n . Let $\text{ZeroEnc}^\oplus$ be the composed gadget defined by sampling $\llbracket z \rrbracket^{(1)} \leftarrow \text{ZeroEnc}^{(1)}()$ and $\llbracket z \rrbracket^{(2)} \leftarrow \text{ZeroEnc}^{(2)}()$ (independently) and outputting the sharewise sum*

$$\llbracket z \rrbracket = \llbracket z \rrbracket^{(1)} + \llbracket z \rrbracket^{(2)} \in \mathbb{F}^n.$$

Then $\text{ZeroEnc}^\oplus$ is $(p, \mathcal{E}^\oplus)$ -gRPP, where the envelope $\mathcal{E}^\oplus$ is built as follows:

1. For all $R \in \Pi_n$, $\mathcal{E}^\oplus(R) \leftarrow 0$.
2. For all partitions $(P, Q) \in \Pi_n^2$, we have

$$\mathcal{E}^\oplus(P \vee Q) = \mathcal{E}^\oplus(P \vee Q) + \mathcal{E}^{(1)}(P) \cdot \mathcal{E}^{(2)}(Q)$$

where $P \vee Q = \min\{R \in \Pi_n \mid P \leq R \text{ and } Q \leq R\}^4$.

Proof. Since the envelopes of $\text{ZeroEnc}_+^{(1)}$ and $\text{ZeroEnc}_+^{(2)}$ are known, the possible leakages on each branch have already been accounted for independently. In particular, let $P \in \Pi_n$ and $Q \in \Pi_n$ denote the partitions obtained at the outputs of $\text{ZeroEnc}_+^{(1)}$ and $\text{ZeroEnc}_+^{(2)}$, respectively.

When the two sharings are added sharewise, the resulting partition must reflect the relations induced by both P and Q . Concretely, for all $i, j \in [n]$, if $i \sim_P j$ then $i \sim_R j$, and if $i \sim_Q j$ then $i \sim_R j$. Therefore, the resulting partition R must be the smallest one that satisfies this condition. The event that the first gadget produces partition P and the second gadget produces partition Q occurs with probability $\mathcal{E}^{(1)}(P) \cdot \mathcal{E}^{(2)}(Q)$, since the two gadgets are independent. Summing over all possible pairs of partitions (P, Q) yields the envelope $\mathcal{E}^\oplus$. $\square$

Lemma 2 provides the general RPP envelope of $\text{ZeroEnc}^\oplus$. In practice, however, we often require the envelope of $\text{ZeroEnc}_+^\oplus$ (i.e., with the additional manipulation of the output shares), for instance when composing the gadget again or when computing the general RPC envelope of a zero-encoding-based refresh gadget. In this case, Lemma 1 can be applied to derive the envelope of $\text{ZeroEnc}_+^\oplus$ from the general RPP envelope of $\text{ZeroEnc}^\oplus$.

4 Ideal Refresh Gadgets

In this section, we introduce *ideal refresh gadgets* that serve as reference benchmarks for the two refresh families defined in Section 3.1. The purpose of these abstractions is to model the strongest decoupling between the input and output sharings that can realistically be achieved by direct and zero encoding-based refresh constructions.

To do so, the idealization abstracts away the internal computations of a refresh gadget and retains only its *boundary wires*, namely the wires that are structurally unavoidable under some randomness and uniformity conditions considered below. Only these boundary wires are assumed to leak, so that the ideal gadgets expose the smallest leakage surface compatible with these conditions.

We formalize the considered class through the following assumption.

Assumption 1 (Minimal Randomness and Uniformity) *An ideal refresh gadget re-randomizes all shares while preserving a uniform sharing distribution independent of the input sharing. Formally, for an input sharing $\llbracket x \rrbracket$ and output sharing $\llbracket y \rrbracket$ of the same value, the distribution of $\llbracket y \rrbracket$ is uniform over all valid sharings of the encoded value and independent of the particular input sharing $\llbracket x \rrbracket$. In the case of ZE-based refresh gadgets, the generated zero-sharing is assumed to be uniformly distributed over all sharings of zero.*

Moreover, each output share y_i depends on strictly more than one fresh random variable generated during the refresh. In particular, each y_i depends on at least two fresh random values.

Assumption 1 is not a structural requirement of either refresh family and is not intended to characterize all refresh gadgets. In particular, constructions such as half refresh (see Section 5), in which each output share depends on a single fresh random value, fall outside its formal scope.

We deliberately adopt the stronger two-fresh-randoms-per-share condition because we believe that it better captures the best-case behavior of a refresh gadget: strong re-randomization should make each output share depend on sufficient fresh randomness so as to maximize the decoupling between the input and output sharings. A more general idealization allowing only one fresh random contribution per output share could also

⁴ Computing $P \vee Q$ can be done efficiently using a *Union-Find* data structure.

be considered, and the same analysis methodology would apply. However, it would yield a looser benchmark, since the additional gate-input leakage induced by combining several fresh random contributions would no longer be accounted for.

Consequently, under Assumption 1, the ideal constructions defined below provide formal upper bounds on the random probing security achievable by concrete refresh gadgets satisfying the assumption.

For refresh gadgets that do not satisfy Assumption 1, these bounds should instead be understood as ideal reference benchmarks rather than formally proven upper bounds. Nevertheless, our design intuition is that the ideal abstractions capture the strongest realistic decoupling achievable within the two refresh families, and we therefore expect constructions with weaker per-share re-randomization not to outperform these ideal benchmarks.

4.1 Ideal Constructions and Envelopes

Under Assumption 1, the ideal refresh gadgets corresponding to the two families are illustrated in Figure 1. They contain only the minimal boundary wires that are structurally required under the assumption.

- (a) The ideal *direct refresh* (Figure 1a) contains $4n$ internal wires. In particular, it comprises (i) at least one wire carrying each input share and (ii) for each output share, a two-input gate producing this output share (with two input wires). This structure is minimal under Assumption 1. Indeed, each output share must depend on at least two independent fresh random variables. A direct connection between an input share and its corresponding output share, or the use of a single random contribution per output share (*e.g.*, a single wire or a one-input gate before the output), would result in an output share depending on at most one fresh random variable, thereby falling outside the scope of Assumption 1.
- (b) The ideal *zero-encoding-based refresh* (Figure 1b) requires, for each zero-encoding output share, a two-input gate combining two independent random contributions. This structure is minimal under Assumption 1: replacing this gate by a single wire (or removing it altogether) would yield an output share depending on at most one, or no, fresh random variable, thereby violating the minimal randomness requirement. The figure also includes the wires corresponding to the final addition between the zero-encoding shares and the input sharing.

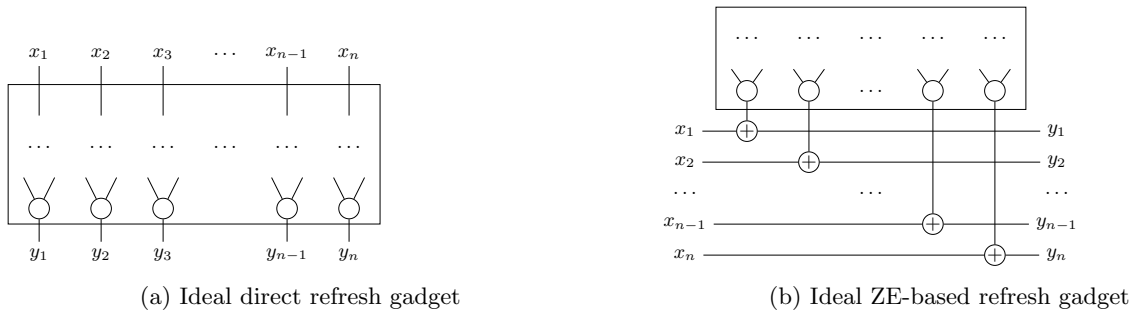


Fig. 1: Ideal refresh gadgets

In both cases, the figures show only the *minimal wires* that any construction satisfying Assumption 1 must contain. These wires represent intermediate values that may leak in the random probing model and therefore induce intrinsic structural constraints, which directly translate into upper bounds on security induced by real gadgets. The resulting cardinal and general RPC envelopes, derived in Lemmas 3, 4, 5 and 6, consequently provide conditional upper bounds on the security achievable by concrete refresh gadgets satisfying Assumption 1.

Lemma 3 (Cardinal RPC of Ideal Direct Refresh Gadget). *Let $n \in \mathbb{N}$. The ideal direct refresh gadget is $(p, \mathcal{E})$ -cardinal RPC such that, for all $t^{in}, t^{out} \in \{0, \dots, n\}$:*

$$\mathcal{E}_{t^{out}}(t^{in}) = \begin{cases} \binom{n}{t^{in}} p^{t^{in}} (1-p)^{n-t^{in}} (1-p^{2(n-t^{out})}) & \text{if } t^{in} < n \text{ and } t^{out} < n, \\ 0 & \text{if } t^{out} = n \text{ and } t^{in} < n, \\ p^n + (1-p^n) p^{2(n-t^{out})} & \text{if } t^{in} = n. \end{cases} \quad (1)$$

Proof (sketch). We reason from the wire structure of the ideal direct refresh gadget depicted in Figure 1a. We model leakages in the random probing setting as follows: each of the n input-share wires leaks independently with probability p . Moreover, since each output share is produced by a two-input gate, we account for the (at least) two wires in input of this gate, but we abstract away the internal computations of the gadget (which are unspecified in the ideal model). The complete proof is given in Appendix D.3.

Similarly, we derive the cardinal RPC envelope of the ideal ZE-based refresh. The proof is given in Appendix D.4.

Lemma 4 (Cardinal RPC of Ideal ZE-based Refresh Gadget). *Let $n \in \mathbb{N}$. The ideal zero encoding-based refresh gadget is $(p, \mathcal{E})$ -cardinal RPC such that, for all $t^{in}, t^{out} \in \{0, \dots, n\}$:*

$$\begin{aligned} \mathcal{E}_{t^{out}}(t^{in}) = & (1 - p'^{n-t^{out}}) \cdot \sum_{i=0}^{\min(t^{in}, t^{out})} \binom{t^{out}}{i} \cdot p'^i \cdot (1-p')^{t^{out}-i} \cdot \binom{n-i}{t^{in}-i} \cdot p^{t^{in}-i} \cdot (1-p)^{n-t^{in}} \\ & + \mathbb{1}_{t^{in} \geq t^{out}} \cdot p'^{n-t^{out}} \cdot \binom{n-t^{out}}{t^{in}-t^{out}} \cdot p^{t^{in}-t^{out}} \cdot (1-p)^{n-t^{in}} \end{aligned} \quad (2)$$

where

$$p' = p + (1-p) \cdot p^2.$$

Lemma 5 (General RPC of Ideal Direct Refresh Gadget). *Let $n \in \mathbb{N}$. The ideal direct refresh gadget is $(p, \mathcal{E})$ -general RPC such that, for all $I, J \in [n]$:*

$$\mathcal{E}_J(I) = \begin{cases} p^{|I|} (1-p)^{n-|I|} (1-p^{2(n-|J|)}) & \text{if } I \neq [n] \text{ and } J \neq [n], \\ 0 & \text{if } I \neq [n] \text{ and } J = [n], \\ p^n + (1-p^n) p^{2(n-|J|)} & \text{if } I = [n]. \end{cases} \quad (3)$$

Proof. The proof follows the same reasoning as in Lemma 3. The only difference is that we explicitly specify the input and output share sets rather than working with their cardinalities. As a consequence, the binomial factor $\binom{n}{t^{in}}$ in the first term disappears. $\square$

Lemma 6 (General RPC of Ideal ZE-based Refresh Gadget). *Let $n \in \mathbb{N}$. The ideal zero encoding-based refresh gadget is $(p, \mathcal{E})$ -general RPC such that, for all $I, J \in [n]$:*

$$\begin{aligned} \mathcal{E}_J(I) = & (1 - p'^{n-|J|}) \left(\sum_{i=0}^{|I \cap J|} \binom{|I \cap J|}{i} p'^i (1-p')^{|J|-i} p^{|I|-i} (1-p)^{n-|I|} \right) \\ & + \mathbb{1}_{J \subseteq I} \cdot p'^{n-|J|} p^{|I|-|J|} (1-p)^{n-|I|} \end{aligned} \quad (4)$$

with

$$p' = p + (1-p) \cdot p^2.$$

Proof. The proof follows the same reasoning as in Lemma 4, except that the binomial terms disappear as we work with explicitly identified sets rather than their cardinalities. $\square$

In Appendix C, we further analyze the effect of repeated refresh compositions on random probing security. More precisely, we study the asymptotic behavior of the corresponding threshold RPC bounds as the number of compositions increases. This highlights a fundamental difference between direct and zero encoding-based refreshes: repeated direct refreshes create multiple independent opportunities to recover the secret shares, causing the threshold RPC advantage to eventually converge towards 1, whereas zero encoding-based refreshes avoid this accumulation phenomenon and remain essentially stable under composition.

We leave the study of optimal refresh composition strategies for future work. Nevertheless, our analysis already suggests that, when the goal is to amplify the refreshing effect through repeated compositions, zero encoding-based constructions (or alternatively hybrid approaches combining both families) appear more suitable than purely direct refresh compositions.

4.2 How Close Is the Random-Pair Refresh to Ideal?

In this subsection, we experimentally analyze the “closeness to ideal” of the *random-pair refresh* gadget (**RPrefresh**) from [11]. Namely, we apply this refresh gadget within the random probing composition framework to protect a one-stage butterfly network, and we compare the resulting security guarantees with those obtained when using ideal refresh gadgets instead.

Butterfly networks are common building blocks in lattice-based cryptography relying on efficient number-theoretic transforms, and they are therefore a natural candidate for our analysis. A one-stage butterfly network takes two inputs and produces two outputs. It first multiplies the second input by a fixed constant, and then computes both the sum and the difference of this product with the first input. The corresponding circuit is illustrated in Figure 2, where c^* denotes a fixed constant multiplication. For its masked version, the constant multiplication, addition, and subtraction gadgets are implemented as follows: the inputs are first refreshed and the arithmetic operation is then performed sharewisely.

We choose to consider the **RPrefresh** gadget since it is, to the best of our knowledge, the only refresh gadget for which explicit expressions of both the cardinal RPC envelope and the general RPC envelope are available. This gadget is parameterized by a number of iterations γ . In each iteration, it samples a random pair of indices (i, j) and refreshes the corresponding shares as $(x_i, x_j) \leftarrow (x_i + r, x_j - r)$ for some random r . For our experiments, we assume that this γ is big enough to obtain the best possible security level (specifically we fixed $\gamma = 500$).

Figures 2 (right) present the threshold RPC security of a one-stage butterfly network, obtained for each leakage rate $p \in \{2^{-5}, 2^{-10}, 2^{-15}\}$ and for an increasing number of shares, according to the following evaluation methodology. First, the cardinal RPC envelopes of the constant multiplication, addition, and subtraction gadgets are derived from the underlying refresh cardinal RPC envelopes (either **RPrefresh** from [11] or one of the ideal refresh gadgets). These envelopes are then composed to obtain the cardinal RPC envelope of the entire network. Finally, from this global envelope, we extract the threshold RPC advantage, optimized over the choice of t .

As illustrated in Figure 2, the refresh gadgets are ranked as follows: the ideal direct refresh achieves the best security bounds, followed by the ideal ZE-based refresh, and finally **RPrefresh**. Although ZE-based constructions are expected to provide stronger guarantees, their ideal version yields slightly weaker bounds because the analysis explicitly accounts for a larger number of intermediate wires, which marginally degrades the resulting envelope.

Besides this ranking, we observe from the figure that, although slightly lower, the security achieved by the **RPrefresh** gadget is still very close to the ideal bounds. This suggests that the **RPrefresh** gadget from [11] is close to optimal in terms of random probing security (under Assumption 1), when parameterized with a sufficiently large number of iterations γ . We note that the number of iterations γ can be optimized while keeping the same security level by looking at the convergence towards the bound as γ increases. See for instance [11, Figure 3].

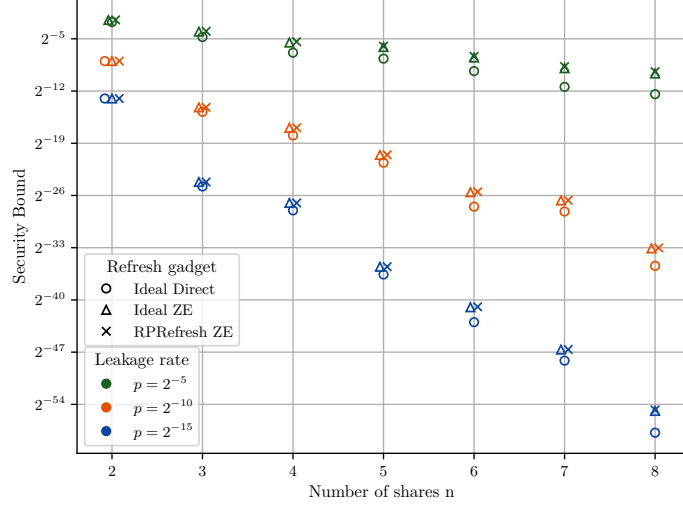
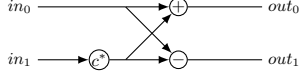


Fig. 2: 1-stage butterfly network and the resulting threshold RPC security

5 Concrete Refresh Gadgets

In the previous section, we introduced ideal refresh gadgets in order to characterize reference upper bounds on random probing security for the subclass of direct and zero encoding-based refresh gadgets satisfying Assumption 1. We now turn to concrete refresh constructions used in practice.

To this end, we introduce an atomic function that serves as a fundamental building block for many direct and ZE-based refresh gadgets. Using this abstraction, we derive analytical formulas for the general RPC envelopes of several refresh gadgets commonly used in masked implementations. In addition, we extend the IronMask tool [9] to automatically compute both general and cardinal RPC envelopes, allowing us to validate our theoretical formulas experimentally.

5.1 Atomic Function

Most refresh gadgets proposed in the literature (both direct and zero encoding-based) can be expressed as the composition of several instances of a simple *atomic function*. This function takes as input two indices of input shares, and adds and subtracts a fresh random value to the shares corresponding to these indices. In the following, we formalize this atomic function and express refresh gadgets from the literature as compositions of such atomic operations. We then characterize the corresponding cardinal/general RPC envelopes for direct refresh gadgets and the cardinal/general RPP properties for zero encoding-based constructions.

Definition 10 (Atomic Refresh Function). *Let $n \in \mathbb{N}$. For any pair of distinct indices $i, j \in [n]$, the atomic refresh function $\text{Atomic}_{i,j}$ is the randomized transformation that takes as input a sharing $\llbracket x \rrbracket = (x_1, \dots, x_n) \in \mathbb{F}^n$ and proceeds as follows:*

$$r \xleftarrow{\$} \mathbb{F},$$

$$x'_k \leftarrow \begin{cases} x_i + r & \text{if } k = i, \\ x_j - r & \text{if } k = j, \\ x_k & \text{otherwise.} \end{cases}$$

It outputs the sharing $(x'_1, \dots, x'_n)$.

Using this primitive, we can describe a broad class of refresh gadgets as compositions of atomic refresh functions. We first consider direct refresh gadgets.

Definition 11 (Atomic Direct Refresh Gadget). Let $n, m \in \mathbb{N}$. A atomic direct refresh gadget is a direct refresh gadget that can be expressed as a composition of atomic refresh functions. Given an input sharing $\llbracket x \rrbracket^{(0)} = (x_1, \dots, x_n) \in \mathbb{F}^n$, the gadget applies a sequence of m atomic refresh functions:

$$\llbracket x \rrbracket^{(t)} \leftarrow \text{Atomic}_{i_t, j_t}(\llbracket x \rrbracket^{(t-1)}) \quad \text{for } t = 1, \dots, m,$$

where $(i_t, j_t) \in [n]^2$ with $i_t \neq j_t$, and where each call to Atomic_{i_t, j_t} samples an independent random value $r_t \xleftarrow{\$} \mathbb{F}$. The output sharing is $\llbracket x \rrbracket^{(m)}$.

We next consider zero encoding constructions.

Definition 12 (Atomic ZE-Based Refresh Gadget). Let $n, m \in \mathbb{N}$. An atomic zero encoding gadget is a zero encoding gadget that can be expressed as a composition of atomic refresh functions. It initializes the sharing $\llbracket z \rrbracket^{(0)} = (0, \dots, 0) \in \mathbb{F}^n$, and then applies a sequence of atomic refresh functions:

$$\llbracket z \rrbracket^{(t)} \leftarrow \text{Atomic}_{i_t, j_t}(\llbracket z \rrbracket^{(t-1)}) \quad \text{for } t = 1, \dots, m,$$

where $(i_t, j_t) \in [n]^2$ with $i_t \neq j_t$, and where each call to Atomic_{i_t, j_t} samples an independent random value $r_t \xleftarrow{\$} \mathbb{F}$. The output zero-sharing is $\llbracket z \rrbracket^{(m)}$.

The definitions above show that both types of gadgets can be expressed as compositions of atomic refresh functions. Consequently, analyzing the security of such constructions reduces to understanding how the corresponding envelopes evolve when composing several instances of the atomic refresh function.

To this end, we first characterize the security properties of the atomic function itself. In particular, we determine the general RPC envelope of an atomic function when applied to an input sharing, and the general RPP envelope when the atomic function is viewed as a zero encoding block. We then study how these envelopes evolve through composition.

Remark 2. Atomic refresh gadgets are characterized by a sequence of index pairs $(i_1, j_1), \dots, (i_m, j_m)$, where each pair corresponds to an atomic function. Some known gadgets from the literature, which we analyze hereafter, correspond to specific sequences. We note that any atomic direct refresh gadget gives rise to a zero-encoding equivalent and any atomic zero-encoding gadget gives rise to a direct equivalent, both sharing the same sequence of index pairs. Specifically, the atomic direct refresh gadget defined as:

$$\llbracket x \rrbracket \mapsto \text{Atomic}_{i_m, j_m} \circ \dots \circ \text{Atomic}_{i_1, j_1}(\llbracket x \rrbracket)$$

has zero-encoding equivalent defined as:

$$\llbracket x \rrbracket \mapsto \text{Atomic}_{i_m, j_m} \circ \dots \circ \text{Atomic}_{i_1, j_1}(\llbracket 0 \rrbracket) + \llbracket x \rrbracket$$

where $\llbracket 0 \rrbracket = (0, \dots, 0) \in \mathbb{F}^n$, and vice versa.

Lemma 7 (General RPC of Atomic Refresh Function). Let $n \in \mathbb{N}$, let $i, j \in [n]$ with $i \neq j$. Then $\text{Atomic}_{i, j}$ is $(p, \mathcal{E})$ -general RPC, where $\mathcal{E}$ is defined as follows.

	$J \cap \{i, j\} = \emptyset$	$J \cap \{i, j\} = \{i\}$	$J \cap \{i, j\} = \{j\}$	$J \cap \{i, j\} = \{i, j\}$
$I = J \setminus \{i, j\}$	$(1-p)^2$	$(1-p)^5$	$(1-p)^5$	0
$I = \{i\} \cup J \setminus \{j\}$	$p(1-p)$	$1-p-(1-p)^5$	$p(1-p)^4$	0
$I = \{j\} \cup J \setminus \{i\}$	$p(1-p)$	$p(1-p)^4$	$1-p-(1-p)^5$	0
$I = J \cup \{i, j\}$	p^2	$p(1-(1-p)^4)$	$p(1-(1-p)^4)$	1
$I \text{ other}$	0	0	0	0

The proof of Lemma 7 is given in Appendix D.5 and consists in recording the leakage events for all wires and computing the corresponding probabilities.

Lemma 8 (General RPP of Atomic ZE Block). *Let $n \in \mathbb{N}$, let $i, j \in [n]$ with $i \neq j$, and consider the zero encoding block $\text{ZeroEnc}_{i,j}^{\text{Atomic}}$ defined by*

$$\llbracket z \rrbracket = (z_1, \dots, z_n) \quad \text{with} \quad z_i = r, \quad z_j = -r, \quad z_k = 0 \quad \text{for all } k \notin \{i, j\},$$

where $r \xleftarrow{\$} \mathbb{F}$. Then $\text{ZeroEnc}_{i,j}^{\text{Atomic}}$ is $(p, \mathcal{E})$ -general RPP, where, for all $P \in \Pi_n$,

$$\mathcal{E}(P) = \begin{cases} 1 - (1 - p)^3, & \text{if } P = (\{1\}, \dots, \{n\}), \\ (1 - p)^3, & \text{if } P = (\{i, j\}, (\{k\})_{k \in [n] \setminus \{i, j\}}), \\ 0, & \text{otherwise.} \end{cases}$$

Proof. We start from the trivial zero-sharing $(0, \dots, 0)$, which is characterized by the partition $(\{1\}, \dots, \{n\})$ with probability 1. When applying $\text{Atomic}_{i,j}$, the only dependency introduced by the gadget is between z_i and z_j , which share the unknown value r . If none of the wires carrying r leaks, which happens with probability $(1 - p)^3$, the shares i and j are linked and the partition becomes

$$(\{i, j\}, (\{k\})_{k \in [n] \setminus \{i, j\}}).$$

Otherwise, r becomes known and both z_i and z_j are individually determined, yielding the original partition $(\{1\}, \dots, \{n\})$ with probability $1 - (1 - p)^3$. No other partition can arise. $\square$

Hence, for a refresh gadget obtained by composing atomic refresh functions, we can derive its general envelopes compositionally. For direct-based refresh gadgets, the general RPC envelope is obtained by composing the general RPC envelopes of the atomic refresh functions using Lemma 7. For zero encoding-based refresh gadgets, the general RPP envelope is obtained by composing the general RPP envelopes of the atomic zero-encoding blocks (given by Lemma 8) using Lemma 2. This provides a generic method to compute the general RPC envelopes of all known refresh gadgets from the literature.

5.2 Analytical Envelope Formulas for Existing Refresh Gadgets

We now exhibit analytical cardinal/general RPC formulas for the most widely used gadgets of the literature.

Linear Refresh Gadget from [26]. The linear refresh gadget introduced in [26] adds a fresh random value to each share with index from 1 to $n - 1$ and subtracts it from the last share with index n . This gadget is an atomic direct refresh (see Definition 11), as it can be expressed as a composition of atomic functions as follows:

$$\llbracket x \rrbracket \mapsto \text{Atomic}_{n-1,n} \circ \dots \circ \text{Atomic}_{1,n}(\llbracket x \rrbracket).$$

The general RPC property of the linear refresh gadget can be established by composing the atomic general RPC envelope of Lemma 7. As per Remark 2, we can also consider the zero-encoding equivalent of this gadget based on the same sequence of atomic functions used to generate a zero-sharing.

Half Refresh Gadget from [5,7]. The half refresh gadget, as named in [7], corresponds to the first layer of the $n \log(n)$ refresh gadget introduced in [5]. In this layer, a fresh random value is added/subtracted to each consecutive pair of shares. For simplicity, we describe the gadget only for an even number of shares n , although it can be naturally generalized to the odd case. In its direct refresh form, the gadget can be expressed as a composition of atomic functions applied to the input sharing:

$$\llbracket x \rrbracket \mapsto \text{Atomic}_{n-1,n} \circ \dots \circ \text{Atomic}_{1,2}(\llbracket x \rrbracket).$$

For this gadget, the zero-encoding and direct variants coincide, so it suffices to analyze its general and cardinal RPC properties. Owing to this particularly simple structure, we are not only able to derive the envelopes through the composition of atomic functions, but also to obtain explicit and more compact analytical formulas for the corresponding general and cardinal RPC envelopes, given below.

Lemma 9 (General RPC of Half Refresh). *Let $n \in \mathbb{N}$ be even and $p \in [0, 1]$. For any subsets $I, J \subseteq [n]$, define:*

- The set of paired indices in J : $\mathcal{C}_J := \{(i, i+1) \in [n]^2 \mid i \in J, i+1 \in J, i \equiv 1 \pmod{2}\}$
- The set of unpaired indices: $\mathcal{D}_J := J \setminus \bigcup_{(i, i+1) \in \mathcal{C}_J} \{i, i+1\}$
- The counts: $k_1 := |\mathcal{C}_J|, k_2 := |I \cap \mathcal{D}_J|, k_3 := |\mathcal{D}_J|$

Then the half refresh satisfies the $(p, \mathcal{E})$ -general RPC property with:

$$\mathcal{E}_J(I) = \begin{cases} 0 & \text{if } \mathcal{C}_J \not\subseteq I \\ \sum_{\ell=0}^{k_2} \binom{k_2}{\ell} (1 - (1-p)^3)^\ell (1-p)^{3(k_3-\ell)} p^{|I|-2k_1-\ell} (1-p)^{n-|I|} & \text{otherwise.} \end{cases}$$

Proof. For each odd index $i \in [n]$, the output shares are of the form $x_i + r$ where the random r is shared between the two output shares y_i and y_{i+1} . Consequently, if the indices $\{i, i+1\} \subseteq J$, i.e. $(i, i+1) \in \mathcal{C}_J$, we have $y_i + y_{i+1} = x_i + x_{i+1}$ and we need the knowledge of x_i and x_{i+1} to simulate $y_i + y_{i+1}$. Therefore, if $\mathcal{C}_J \not\subseteq I$, then a sum of two output shares $y_i + y_{i+1}$ cannot be simulated with the knowledge of the input shares whose indices are in I . Thus, $\mathcal{E}_J(I) = 0$. Otherwise, when $\mathcal{C}_J \subseteq I$, the input whose indices are in $\mathcal{C}_J$ are given with probability 1 to the simulator, namely $2k_1$ input shares. To require the knowledge of the remaining input shares whose indices are in $I \setminus \mathcal{C}_J$, there are two alternatives :

- Either the associated randomness r leaks for indices in $\mathcal{D}_J \cap I$. Each index in $I \cap \mathcal{D}_J$ leaks independently with probability $1 - (1-p)^3$. Let ℓ be the number of indices for which leakage occurs.
- Either leak directly the input share x_k with probability p , while the remaining $n - |I|$ input shares do not leak.

Combining these events yields the claimed formula. $\square$

Similarly, we can establish the cardinal RPC envelope as stated in the following corollary whose proof is given in Appendix D.6.

Corollary 2 (Cardinal RPC of Half Refresh). *Let $n \in \mathbb{N}$ even, let $p \in 0, 1$. **HalfRefresh** is $(p, \mathcal{E})$ cardinal RPC such that for all $t^{in} \in [n]$, all $t^{out} \in [n]$, we have*

$$\mathcal{E}_{t^{out}}(t^{in}) = \max_{k_1 \in \mathcal{K}} \left(\sum_{\ell=0}^{\min(t^{in}-2k_1, k_3)} \binom{k_3}{\ell} (1-q)^\ell q^{k_3-\ell} \binom{n-k_4}{t^{in}-k_4} p^{t^{in}-k_4} (1-p)^{n-t^{in}} \right)$$

where $\mathcal{K} = \{k_1 \mid \max(0, t^{out} - \frac{n}{2}) \leq k_1 \leq \lfloor \frac{t^{out}}{2} \rfloor\}$ and $2k_1 \leq t^{in}$, $k_3 := t^{out} - 2k_1$, $k_4 := 2k_1 + \ell$ and $q := (1-p)^3$.

Remark 3. Lemma 9 only states analytical formulas for even values of n . The odd case extends naturally by requiring the last input share whenever the last output share needs to be simulated or when it leaks. Similarly, Corollary 2 also extends straightforwardly to odd n .

Double Half Refresh Gadget inspired by [4]. The circular refresh gadget introduced in [4] was originally defined as a direct refresh gadget. However, it does not match the atomic direct refresh definition (Definition 11), since the fresh random values are interleaved. In its natural zero encoding-based form, the gadget is strictly equivalent to an atomic ZE-based refresh gadget defined as follows:

$$\llbracket x \rrbracket \mapsto \text{Atomic}_{n,1} \circ \dots \circ \text{Atomic}_{2,3} \circ \text{Atomic}_{n-1,n} \circ \dots \circ \text{Atomic}_{1,2}(\llbracket 0 \rrbracket) + \llbracket x \rrbracket.$$

We refer to this construction as the *double half refresh* gadget, as it corresponds to the composition of two half refresh layers. As per Remark 2, we can also consider the direct equivalent of this gadget based on the same sequence of atomic functions directly applied to the input sharing.

As for the half refresh, we are able to derive an explicit analytical formula for the general and cardinal RPP envelopes of its zero encoding block. The full proof is given in Appendix D.7. In a nutshell, it relies on the cyclic structure of $\text{ZeroEnc}^{\text{Double half}}$. We associate to the leakage pattern a graph whose vertices are the shares and whose edges correspond to non-leaking random variables $r_{i,i+1}$. Two shares belong to the same part of the partition if and only if they remain connected through a path in this graph. Hence, the induced partition is exactly given by the connected components of the cycle after removing the edges corresponding to leaking randoms. Since each edge is removed independently with probability q , the probability of obtaining a given partition only depends on the number of missing edges, which directly yields the claimed formula.

Lemma 10 (General RPP of $\text{ZeroEnc}^{\text{Double half}}$). *Let $n \in \mathbb{N}^*$ be even and let $p \in [0, 1]$. We denote $q = (1 - p)^3$. Then $\text{ZeroEnc}^{\text{Double half}}$ is $(p, \mathcal{E})$ -general RPP, where for all $R \in \Pi_n$, the function $\mathcal{E}$ is defined as follows.*

If $R = \{\{1, 2, 3, \dots, n\}\}$ then

$$\mathcal{E}(R) = q^n + nq^{n-1}(1 - q).$$

Else if

$$R = \{[i_1 + 1, i_2]_{\circ}, [i_2 + 1, i_3]_{\circ}, \dots, [i_k + 1, i_1]_{\circ}\},$$

where $k \geq 2$ and $i_1, \dots, i_k \in [n]$, then

$$\mathcal{E}(R) = (1 - q)^k q^{n-k}$$

Otherwise, $\mathcal{E}(R) = 0$.

Corollary 3 (Cardinal RPP of $\text{ZeroEnc}^{\text{Double half}}$). *Let $n \in \mathbb{N}^*$ be even and let $p \in [0, 1]$. We denote $q = (1 - p)^3$. Then $\text{ZeroEnc}^{\text{Double half}}$ is $(p, \mathcal{E})$ -cardinal RPP, where for all $P \in \mathcal{P}(n)$ we have:*

$$\mathcal{E}(P) = \begin{cases} q^n + nq^{n-1}(1 - q) & \text{if } P = (n). \\ \frac{n}{k} \cdot \frac{k!}{\prod_s m_s!} (1 - q)^k q^{n-k} & \text{if } P = (\ell_1, \ell_2, \dots, \ell_k). \end{cases}$$

where m_s denotes the multiplicity of the size s among the block sizes $\ell_1, \dots, \ell_k$.

Proof. The first case corresponds to $R = \{\{1, \dots, n\}\}$ in the general RPP envelope. The second case corresponds to

$$R = \{[1, \ell_1]_{\circ}, [\ell_1 + 1, \ell_1 + \ell_2]_{\circ}, \dots, [(\sum_{i=1}^{k-1} \ell_i) + 1, (\sum_{i=1}^{k-1} \ell_i) + \ell_k]_{\circ}\}$$

in the general RPP envelope. Furthermore, cyclic shifts of this partition are also possible and occurs with the same probability. The number of cyclic consecutive partitions with prescribed block-size profile $(\ell_1, \dots, \ell_k)$ is given by $\frac{n}{k} \cdot \frac{k!}{\prod_s m_s!}$, which follows from a multinomial counting argument modulo cyclic rotations. The claim therefore follows. $\square$

RPrefresh Refresh Gadget from [11]. The RPrefresh gadget introduced in [11] is parameterized by an integer γ that specifies the number of iterations. At each iteration, two distinct indices $i, j \in [n]$ are sampled uniformly at random and the atomic function $\text{Atomic}_{i,j}$ is applied.

In [11], the gadget is defined as a zero encoding-based refresh: the sequence of γ atomic functions is first applied to a zero encoding and the resulting sharing is then added to the input sharing. Formally,

$$\llbracket x \rrbracket \mapsto \text{Atomic}_{i_{\gamma}, j_{\gamma}} \circ \dots \circ \text{Atomic}_{i_1, j_1}(\llbracket 0 \rrbracket) + \llbracket x \rrbracket,$$

where each pair (i_k, j_k) is sampled uniformly among all pairs of distinct indices of $[n]$. As per Remark 2, we can also consider the direct equivalent of this gadget based on the same sequence of atomic functions directly applied to the input sharing.

We now formalize the cardinal RPC envelope of the gadget **RPreFresh** in its direct version. Recall that the pair of indices is sampled uniformly among all pairs of distinct indices of $[n]$. We start by describing the cardinal RPC envelope of the atomic function described with pair of indices sampled uniformly, we note this atomic function **RandomAtomic**. Consequently, we can define **RPreFresh** as sequence of γ **RandomAtomic** gadgets.

Lemma 11 (Cardinal RPC of Direct RandomAtomic). *Let $n \in \mathbb{N}$ and $p \in [0, 1]$. The direct version of **RandomAtomic** is $(p, \mathcal{E})$ -cardinal RPC, where for every $t^{in} \in [0, n]$ we have*

$$\begin{aligned}\mathcal{E}_{t^{in}-2}(t^{in}) &= \frac{(n-t^{in}+2)(n-t^{in}+1)p^2}{n(n-1)} \\ \mathcal{E}_{t^{in}-1}(t^{in}) &= \frac{(n-t^{in}+1)(n-t^{in})2p(1-p)}{n(n-1)} + \frac{2(t^{in}-1)(n-t^{in}+1)p(1-(1-p)^4)}{n(n-1)} \\ \mathcal{E}_{t^{in}}(t^{in}) &= \frac{(n-t^{in})(n-t^{in}-1)(1-p)^2}{n(n-1)} + \frac{2t^{in}(n-t^{in})(1-p-(1-p)^5+p(1-p)^4)}{n(n-1)} + \frac{t^{in}(t^{in}-1)}{n(n-1)} \\ \mathcal{E}_{t^{in}+1}(t^{in}) &= \frac{2(t^{in}+1)(n-t^{in}-1)(1-p)^5}{n(n-1)}\end{aligned}$$

for every valid output index.

Proof. Recall by Lemma 7 that the general RPC envelope of the atomic refresh function is known, we can derive the general RPC envelope of **RandomAtomic**. We start by describing, for a random pair of distinct indices (i, j) sampled uniformly, the events :

- Event A : $J \cap \{i, j\} = \emptyset$ occurs with probability $\binom{n-|J|}{2} \frac{2}{n(n-1)}$.
- Event B : $J \cap \{i, j\} = \{i\}$ occurs with probability $|J|(n-|J|) \frac{1}{n(n-1)}$.
- Event C : $J \cap \{i, j\} = \{j\}$ occurs with probability $|J|(n-|J|) \frac{1}{n(n-1)}$.
- Event D : $J \cap \{i, j\} = \{i, j\}$ occurs with probability $\binom{|J|}{2} \frac{2}{n(n-1)}$.

By multiplying these events by the probability of the general RPC envelope of Lemma 7 and by summing over all the possible couple of indices (i, j) , we get the desired cardinal RPC envelope. $\square$

We get the envelope of **RPreFresh** in its direct version by composing **RandomAtomic** γ times.

5.3 Extension of IronMask

In order to validate both the composition through atomic refresh functions and our analytical results, we rely on and extend the functionality of the **IronMask** tool, introduced in [9]. The corresponding extensions are available at <https://github.com/CryptoExperts/IronMask>. **IronMask** is a verification tool that performs exact analyses of several masking security notions for a circuit given as input. In particular, the original work focuses on computing the *Set of Input Shares* (SIS) function. The authors show that this function is sufficient to verify various probing and random probing security notions. We recall hereafter the definition of the SIS function in Definition 13

Definition 13 (SIS from [9]). *Let G be an $(n$ -share, ℓ -to-1) gadget mapping ℓ input shares $(\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \in (\mathbb{K}^n)^\ell$ to an output sharing $\llbracket y \rrbracket \in \mathbb{K}^n$. Let $\mathcal{W}$ be a set of probes on G and $J \subseteq [n]$ a set of output indices. The function SIS_G maps $\mathcal{W}$ and J to the smallest sets of input indices $I_1, \dots, I_\ell$ such that $(\mathcal{W}, \llbracket y \rrbracket|_J)$ can be perfectly simulated from $\llbracket x_1 \rrbracket|_{I_1}, \dots, \llbracket x_\ell \rrbracket|_{I_\ell}$.*

For the simple case $\ell = 1$ (e.g., refresh gadgets), knowing the SIS function for every wire tuple $\mathcal{W}$ and output set J directly yields a general RPC envelope. We initialize all entries $\mathcal{E}_J(I)$ to 0, and for every pair $(J, \mathcal{W})$, compute $I_{\min} = SIS_G(\mathcal{W}, J)$ using **IronMask** and update

$$\mathcal{E}_J(I_{\min}) \leftarrow \mathcal{E}_J(I_{\min}) + p^{|\mathcal{W}|}(1-p)^{|G|-|\mathcal{W}|}.$$

Summing over all output sets and probe tuples gives the final envelope $\mathcal{E}$.

Furthermore, we can also compute a cardinal RPC envelope of the refresh gadget, following the same overall approach. We still enumerate all probe sets and all sets of output share indices. However, instead of directly adding the probability of the corresponding leakage event to the exact set $I_{\min}$, we group together all sets $I_{\min}$ having the same cardinality. Then, for each output share set J of a given cardinality, we keep the maximum probability among all corresponding events. The pseudo code is given in Algorithm 5 in Appendix B.

The exact verification of the SIS function performed by **IronMask** guarantees that the computed general RPC and cardinal RPC envelopes for our 1-to-1 refresh gadgets are the tightest possible ones.

Once these tight envelopes are obtained, we use them as a reference to validate our analytical formulas. For each refresh, we compute the corresponding cardinal/general RPC envelope, compare it with the one obtained using **IronMask**. We then verify that the relative error of each entry is bounded by $c \cdot \phi$, where ϕ denotes the machine precision. In our experiments with $n = 8$ shares, we set $c = 100$. The maximum relative error observed between the envelopes obtained from the analytical formulas and those computed by **IronMask** is 3.33×10^{-15} , confirming that the two envelopes are numerically indistinguishable.

6 Application

We now apply our results to derive complete masked implementations of AES and evaluate their random probing security under different refresh gadgets in the cardinal RPC framework. We further apply the best trade-offs identified in our analysis to Raccoon [17,24], whose circuits and analysis are provided in Appendix E.

6.1 Description of the Circuit Compiler

The compiler of [10] uses a fixed refresh gadget **RRefresh**. Here, we generalize it to a family of compilers parameterized by a refresh gadget **Ref**, each choice of **Ref** defining one compiler instance. Each instance includes the following gadgets:

- *Addition, addition by a constant, multiplication by a constant, and squaring:* The gadgets Add^{Ref} , cAdd^{Ref} , $\text{cMult}^{\text{Ref}}$, and Sq^{Ref} are described in Algorithms 6, 8, 7, and 9, respectively. The cardinal RPC envelopes of these gadgets are derived in [11] from the cardinal RPC envelope of the refresh gadget **Ref**.
- *Multiplication:* The multiplication gadget Mult^{Ref} , described in Algorithm 12, follows the framework of [10]. It first performs an expansion phase using $\text{UnifMatMult}^{\text{Ref}}$ (Algorithm 10) which computes all cross-products $x_i \cdot y_j$ and produces an n^2 -sharing. This sharing is then compressed into an n -sharing using $\text{TreeAdd}^{\text{Ref}}$ (Algorithm 11). Given the cardinal RPC envelope of **Ref**, that of Mult^{Ref} follows directly from [10] by replacing **RRefresh** with **Ref** in the corresponding composition analysis.

The circuit compilation consists in replacing each gate in the original circuit by an n -share gadget with corresponding functionality, and each wire by a set of n wires carrying a n -linear sharing of the original wire. In particular, each of its random gates is replaced by n random gates, which duly produce a n -linear sharing of a random value. Moreover, when a secret needs to be copied, our compilers duplicate the corresponding shares and insert the appropriate permutations, as specified in [10]. Since these compilers support all the operations involved in AES, they can be used to compile AES circuits into their masked counterparts.

6.2 Description of the AES Circuit

We describe the circuits corresponding to the main building blocks of AES.

- *SubBytes:* Each S-box consists of two parts. The first one is an exponentiation step mapping a byte $b \in \mathbb{F}_{256}$ to b^{254} , as illustrated in Figure 3, where the small empty gates represent permutations. The second is the affine transformation $\text{Affine} : \mathbb{F}_{256} \rightarrow \mathbb{F}_{256}$, whose masked circuit is shown in Figure 4. Combining these two parts yields the cardinal RPC envelope of a single masked S-box. The *SubBytes* operation then consists of 16 such S-boxes operating in parallel, one for each byte of the AES state.

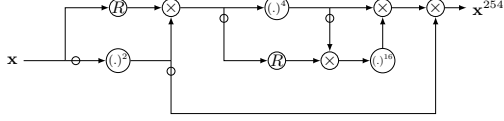


Fig. 3: Exponentiation circuit from [10].

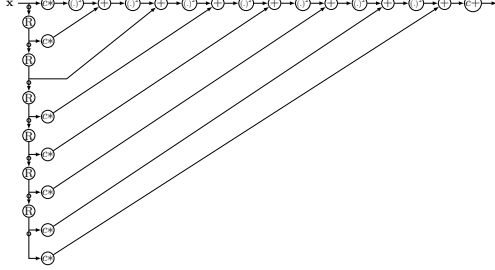


Fig. 4: Circuit for **Affine** from [10].

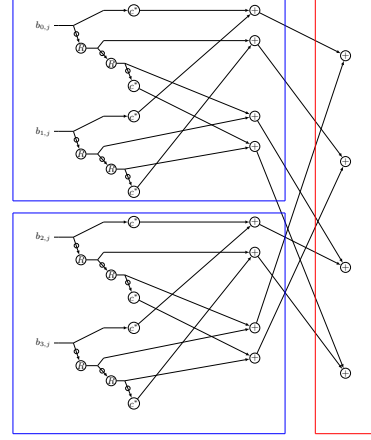


Fig. 5: Circuit for the computation of one column of **MixColumns** from [10].

- *ShiftRows*: This operation only permutes the state rows and thus affects the circuit wiring without introducing any gadget.
- *MixColumns*: This operation is linear over the state columns and is depicted in Figure 5. Due to the complexity of computing the cardinal RPC envelope of the full block, we decompose it into three sub-circuits. The two blue-framed sub-circuits contain all operations except the final four additions, which form the red-framed sub-circuit. The AES circuit contains 4 such gadgets in parallel.
- *AddRoundKey*: This operation adds two secret values and is implemented with 16 addition gadgets in parallel.

We briefly recall how the random probing security of the masked AES is derived, following [10]. The strategy is as follows. First, we compute the cardinal RPC envelope for each block, up to the largest order that can be handled within a reasonable computation time. Then, for each block, we derive a threshold RPC bound from its cardinal RPC envelope, by setting t the threshold parameter to optimize the security results. Assume that the circuit is decomposed into k blocks, yielding threshold RPC advantages $\varepsilon_1, \dots, \varepsilon_k$. By composition, the overall random probing advantage of the masked AES is bounded by

$$k \cdot \max(\varepsilon_1, \dots, \varepsilon_k).$$

Following this strategy, for AES-128 encryption, we obtain the following decomposition:

- 16 blocks of *AddRoundKey* $\rightarrow$ *SubBytes* with known cardinal RPC security envelopes $\mathcal{E}^1$.
- $8 \times 16 = 128$ blocks of *MixColumns* $\rightarrow$ *AddRoundKey* $\rightarrow$ *SubBytes* with known cardinal RPC security envelopes $\mathcal{E}^2$.
- 16 blocks of *MixColumns* $\rightarrow$ *AddRoundKey* $\rightarrow$ *SubBytes* $\rightarrow$ *AddRoundKey* with known cardinal RPC security envelopes $\mathcal{E}^3$.
- $9 \times 8 = 72$ blocks for the first part of *MixColumns* with known cardinal RPC security envelopes $\mathcal{E}^4$.

From these cardinal RPC envelopes, we derive the threshold RPC security for each block. We note ε_t^i the t -threshold RPC security derived from the envelopes $\mathcal{E}^i$. Finally, we take $\varepsilon = \min_t \max(\varepsilon_t^1, \varepsilon_t^2, \varepsilon_t^3, \varepsilon_t^4)$. The AES implementation is then $(p, 232 \cdot \varepsilon)$ random probing secure.

6.3 Security Results for AES

For this application on AES, we consider a representative set of refresh gadgets from the literature for our compiler, including the linear refresh of [26], the half refresh [7], the double half refresh inspired from [4], the $\mathcal{O}(n \log n)$ refresh [5], and RPreFresh [11].

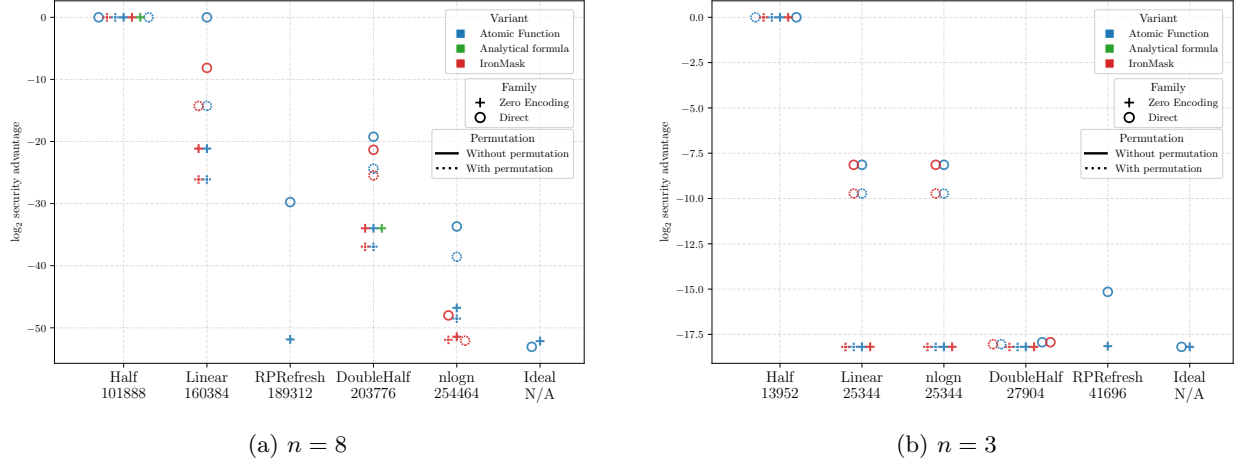


Fig. 6: Random probing security of a masked AES with $p = 2^{-16}$ and n shares using the compiler from [10] with different refresh gadgets

For RPreRefresh, following [10], the number of iterations is chosen so as to preserve (up to a negligible loss) the overall security level of the compiled circuit. Concretely, we first evaluate the circuit security using a very large iteration parameter γ , and then decrease the number of iterations at each refresh call as much as possible while ensuring that the resulting degradation of the global security remains negligible. Moreover, we assign a specific number of iterations to each AES building block. For $\text{Mult}^{\text{RPreRefresh}}$, we additionally specify a sequence of iteration counts, one for each recursive call to RPreRefresh. Hence, an instantiation of our compiler for AES is characterized by $(\gamma_{ark}, \gamma_{mc}, \gamma_{sb}, \gamma_{mult})$, where γ_{ark} , γ_{mc} , and γ_{sb} denote the numbers of iterations used for the AddRoundKey, MixColumns, and SubBytes building blocks, respectively, while $\gamma_{mult} := (\gamma_2, \gamma_3, \dots, \gamma_{\lceil \frac{n}{2} \rceil})$ specifies the iteration counts used in the recursive calls of $\text{Mult}^{\text{RPreRefresh}}$.

For all of these refresh gadgets, we consider both the direct and ZE-based variants, as well as their possible composition with a random output permutation. As explained in [10], gadgets whose general RPC envelopes do not satisfy a certain output-uniformity property compose less efficiently. To address this issue, [10] introduced the notion of *uniformly output (UO) cardinal RPC*, and showed that applying a random permutation to the output shares makes any gadget satisfy this property. Altogether, this leads to a total of 18 different refresh gadgets (RPreRefresh being already UO, we do not consider its composition with a random permutation). We also include the results obtained for the ideal direct and zero encoding-based refresh gadgets. These values constitute formal upper bounds for constructions satisfying Assumption 1. More generally, we use them as best-case reference benchmarks for the corresponding refresh families, based on the intuition that they capture the strongest realistic decoupling between input and output sharings.

We primarily focus on a number of shares $n = 8$ in order to obtain the cardinal RPC envelopes of all refresh gadgets under the different methods, namely composition of atomic functions, analytical formula (when available) and IronMask. The results are displayed in Figure 6a for $p = 2^{-16}$. We then consider $n = 3$, as practical AES implementations typically use two or three shares. We do not focus on $n = 2$, since several refresh constructions coincide at such a small number of shares; for instance, the $\mathcal{O}(n \log n)$, half, and linear refresh gadgets are equivalent when $n = 2$. The results are displayed in Figure 6b for the same leakage rate.

Description of Figure 6. The y -axis represents the random probing security of a masked AES instantiated using the compiler described in Section 6.1. The x -axis includes the different gadgets with the associated randomness complexity (total number of random values required for the full AES). For each gadget, the ZE-based version is depicted by a ‘+’ marker whereas the direct-based version is depicted by a ‘o’ marker.

We consider three approaches for computing the cardinal RPC envelopes of the underlying refresh gadgets:

1. *Using atomic refresh functions.* The results obtained with this approach are represented in blue in Figure 6. For the ZE-based refresh gadgets, we first compute the corresponding general RPP envelope by composing the atomic zero-encoding refresh functions according to the appropriate indices. We then derive the associated cardinal RPC envelope using Lemma 1. For the direct refresh gadgets, we instead compute the general RPC envelope directly from the composition of the atomic direct refresh functions, from which the cardinal RPC envelope is then derived.
2. *Using analytical formulas.* For the ZE-based half and double half refresh gadgets, analytical formulas are available to compute the cardinal RPC envelopes. These results are also included in Figure 6a and are represented in green. However, they are not included in Figure 6b, since the analytical formulas available for both refresh gadgets only apply when the number of shares n is even.
3. *Using IronMask.* Cardinal RPC envelopes can also be computed using the extension of **IronMask** introduced in Section 5.3. The results obtained with this approach are represented in red in Figure 6. Importantly, the cardinal RPC envelope of **RPreFresh** cannot be computed with **IronMask**, since random indices are not supported by the tool.

We consider additional variants of the refresh gadgets where random permutations are applied to the outputs of the refresh gadgets, leading to UO cardinal RPC envelopes as introduced in [10]. The results obtained for these permuted gadgets are represented by dotted markers in Figure 6 (**RPreFresh** is not included since it is UO by design). To derive these UO cardinal RPC envelopes with the different approaches, we first compute the general RPC envelopes either from composition of the atomic refresh functions and Theorem 1 (blue markers) or by using **IronMask** (red markers). We then define a new general RPC envelope $\mathcal{E}'$ as follows:

$$\mathcal{E}'_J(I) = \sum_{J': |J'|=|J|} \frac{\mathcal{E}_{J'}(I)}{\binom{n}{|J|}}$$

where I is a set of input indices and J is a set of output indices. This construction artificially models the effect of applying an additional permutation to the outputs of the refresh gadget, which in turn allows us to derive a UO cardinal RPC envelope.

For the sake of readability, the randomness complexity reported in Figure 6 only accounts for the random values required to refresh the shares, and does not include the randomness required to draw the random indices in **RPreFresh** or the random permutations of the UO variants. As argued in [10], this second type of randomness is of different nature. In particular, it can be leaked to the adversary which makes it potentially easier to generate. Moreover, the numbers of iterations used for the **RPreFresh** gadgets in Figure 6a are set to $\gamma_{ark} = 51$, $\gamma_{mc} = 23$, $\gamma_{sb} = 8$, and $\gamma_{mult} = (0, 0, 2)$. For Figure 6b, they are set to $\gamma_{ark} = 8$, $\gamma_{mc} = 7$, $\gamma_{sb} = 3$, and $\gamma_{mult} = (0)$.

Interpretation of Figure 6. We can make several observations from these results:

- All variants of the half refresh gadget lead to the worst possible bound, namely 1. This behavior stems from the intrinsically poor cardinal RPC envelopes of this gadget. Indeed, in the cardinal setting, the simulator only knows the number of output shares to simulate, but not their precise indices. Therefore, one must consider the worst-case configuration, namely when the probed output shares correspond to pairs of shares refreshed together. In that case, simulating $|J|$ output shares requires knowledge of at least $|J|$ input shares (and possibly more in presence of leakage), meaning that the refresh operation essentially provides no gain from the cardinal RPC perspective. This ultimately leads to trivial security bounds
- For all gadgets, ZE-based variants provide better security bounds than their direct counterparts. This can probably be explained by the fact that, in the zero encoding approach, the input shares are less manipulated than in the direct approach, as already discussed in Section 3.1. These results therefore confirm that ZE-based refresh gadgets can provide stronger compositional security guarantees than direct-based refresh gadgets.
- As expected, among the variants without additional output permutation, the bounds obtained with **IronMask** are the tightest. This is consistent with the discussion from Section 5.3, where **IronMask** is

shown to provide the tightest possible cardinal RPC envelopes. In contrast, our compositional framework only propagates the sets of shares that must be simulated, but does not preserve finer information such as which sums of shares are required. Since this information can be exploited by *IronMask* to obtain tighter simulations, its loss through composition naturally explains the observed gap in tightness.

- Introducing a permutation on the output shares can improve the resulting security bounds, for both families, further highlighting the interest of UO cardinal RPC property introduced in [10].
- Specifically for Figure 6a, in their zero-encoding versions, both *RPRrefresh* and the $\mathcal{O}(n \log n)$ refresh gadget achieve security bounds that are very close to those of the ideal zero-encoding refresh gadget. This suggests that these two refresh gadgets are natural candidates for approaching the best-case benchmark considered in this work. Moreover, the former appears to require significantly less randomness than the latter (about 25% less for this setting with $n = 8$ and $p = 2^{-16}$), which further highlights the interest of *RPRrefresh* for practical implementations (still assuming that the random indices can be generated at a moderate cost).

In conclusion of our applications, we can observe that the choice of the refresh gadget has a significant impact on the resulting security bounds. In particular, the half refresh gadget appears to be a poor choice in the context of the cardinal RPC framework. On the other hand, both the $\mathcal{O}(n \log n)$ refresh gadget and *RPRrefresh* appear to be strong candidates for approaching the ideal benchmarks for a number of shares up to $n = 8$. For larger numbers of shares, the $\mathcal{O}(n \log n)$ refresh gadget appears to be less suitable due to the computational cost of deriving tight cardinal RPC envelopes, while the *RPRrefresh* provides strong security guarantees, remaining close to the corresponding ideal benchmark up to $n = 16$ shares. Moreover, *RPRrefresh* requires significantly less randomness than the $\mathcal{O}(n \log n)$ refresh gadget (about 25% less for AES with $n = 8$ and $p = 2^{-16}$) when excluding the randomness required to draw the random indices (which is of different nature and might be generated at a moderate cost). For contexts where the generation of random indices is costly and for a small enough number of shares, the $\mathcal{O}(n \log n)$ refresh gadget might still offer a better trade-off, as illustrated in Figure 6b. Finally, we also observed that applying a random permutation to the outputs of the refresh gadgets can further improve the resulting security bounds, which further highlights the interest of UO cardinal RPC property introduced in [10]. It would be interesting to further investigate the different trade-offs on concrete implementation scenarios, taking into account the cost of the two types of randomness (random values and random indices/permutations).

Acknowledgments. This work was co-funded by the European Union (ERC, AMaskZONE, 101077506). 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 ACM STOC. pp. 715–724. ACM Press (Jun 2011). <https://doi.org/10.1145/1993636.1993731>
2. Ananth, P., Ishai, Y., Sahai, A.: Private circuits: A modular approach. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 427–455. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_15
3. Andrychowicz, M., Dziembowski, S., Faust, S.: Circuit compilers with $O(1/\log(n))$ leakage rate. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 586–615. Springer, Berlin, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_21
4. Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F.X., Strub, P.Y.: Parallel implementations of masking schemes and the bounded moment leakage model. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 535–566. Springer, Cham (Apr / May 2017). https://doi.org/10.1007/978-3-319-56620-7_19
5. Battistello, A., Coron, J.S., Prouff, E., Zeitoun, R.: Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 23–39. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53140-2_2

6. Beierle, C., Feldtkeller, J., Guinet, A., Güneysu, T., Leander, G., Richter-Brockmann, J., Sasdrich, P.: INDIANA - verifying (random) probing security through indistinguishability analysis. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part VIII. LNCS, vol. 15608, pp. 33–63. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91101-9_2
7. Belaïd, S., Cassiers, G.: PERSEUS - probabilistic evaluation of random probing security using efficient sampling. In: Daemen, J., Thomé, E. (eds.) Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10–14, 2026, Proceedings, Part VII. pp. 506–536. Lecture Notes in Computer Science, Springer (2026). https://doi.org/10.1007/978-3-032-25336-1_18
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.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 339–368. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_12
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 (May 2022). <https://doi.org/10.1109/SP46214.2022.9833600>
10. Belaïd, S., Normand, V., Rivain, M.: Masked circuit compiler in the cardinal random probing composability framework. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part II. LNCS, vol. 16246, pp. 375–406. Springer, Singapore (Dec 2025). https://doi.org/10.1007/978-981-95-5096-8_12
11. Belaïd, S., Rivain, M., Rossi, M.: New techniques for random probing security and application to raccoon signature scheme. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part VIII. LNCS, vol. 15608, pp. 94–123. Springer, Cham (May 2025). https://doi.org/10.1007/978-3-031-91101-9_4
12. 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.) EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 313–343. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-77886-6_11
13. Belaïd, S., Rivain, M., Taleb, A.R., Vergnaud, D.: Dynamic random probing expansion with quasi linear asymptotic complexity. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 157–188. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92075-3_6
14. Berti, F., Faust, S., Orlt, M.: Provable secure parallel gadgets. IACR TCHES **2023**(4), 420–459 (2023). <https://doi.org/10.46586/tches.v2023.i4.420-459>
15. Cassiers, G., Faust, S., Orlt, M., Standaert, F.X.: Towards tight random probing security. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 185–214. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84252-9_7
16. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M.J. (ed.) CRYPTO’99. LNCS, vol. 1666, pp. 398–412. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_26
17. 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), available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures>
18. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Berlin, Heidelberg (May 2014). https://doi.org/10.1007/978-3-642-55220-5_24
19. Goubin, L., Patarin, J.: DES and differential power analysis (the “duplication” method). In: Kog, Çetin Kaya., Paar, C. (eds.) CHES’99. LNCS, vol. 1717, pp. 158–172. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48059-5_15
20. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27
21. Jahandideh, V., Mennink, B., Batina, L.: An algebraic approach for evaluating random probing security with application to AES. IACR TCHES **2024**(4), 657–689 (2024). <https://doi.org/10.46586/tches.v2024.i4.657-689>
22. Mathieu-Mahias, A.: Sécurisation of Implementations of Cryptographic Algorithms in the Context of Embedded Systems. Thèse de doctorat, Université Paris-Saclay, Cryptography and Security [cs.CR] (2021), nNT: 2021UP-ASG095, tel-03537322
23. Meunier, Q.L., Taleb, A.R.: Verifmsi: Practical verification of hardware and software masking schemes implementations. In: di Vimercati, S.D.C., Samarati, P. (eds.) Proceedings of the 20th International Conference on Security and Cryptography, SECRIPT 2023, Rome, Italy, July 10–12, 2023. pp. 520–527. SCITEPRESS (2023). <https://doi.org/10.5220/0012138600003555>, <https://doi.org/10.5220/0012138600003555>

24. del Pino, R., Prest, T., Rossi, M., Saarinen, M.J.O.: High-order masking of lattice signatures in quasilinear time. In: 2023 IEEE Symposium on Security and Privacy. pp. 1168–1185. IEEE Computer Society Press (May 2023). <https://doi.org/10.1109/SP46215.2023.10179342>
25. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Berlin, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_9
26. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Mangard, S., Standaert, F.X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Berlin, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-15031-9_28

A Additional Security Notions

For completeness, we recall here the notions of threshold random probing composability and cardinal random probing composability for randomized-wiring arithmetic circuits, following [10] and the notion of cardinal random probing partitioning for zero encoding primitives [11].

A.1 Threshold Random Probing Composability

The threshold RPC notion characterizes security through a single threshold t : any leakage involving internal probes and up to t output shares must be simulatable using at most t input shares.

Definition 14 (Threshold Random Probing Composability). *Let $n, \ell, m, t \in \mathbb{N}$ and let $p \in [0, 1]$. An n -share randomized-wiring gadget*

$$G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$$

is (p, t, ε) -threshold random probing composable ((p, t, ε) -threshold RPC) if there exists a two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ and every collection $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ such that $|J_1| \leq t, \dots, |J_m| \leq t$, the outputs of Experiment 1 satisfy

1. $\mathbb{P}(|I_1| \leq t, \dots, |I_\ell| \leq t) \leq \varepsilon,$
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\text{AssignWires}(G^\rho, R, \mathcal{W}, \llbracket \mathbf{x} \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(G^\rho, p)$ and $\llbracket \mathbf{y} \rrbracket \leftarrow G^\rho(\llbracket \mathbf{x} \rrbracket)$.

A.2 Cardinal Random Probing Composability

The cardinal RPC notion refines threshold RPC by specifying the simulation failure probability as a function of the number of output shares to be simulated and the number of input shares used by the simulator.

Definition 15 (Cardinal Random Probing Composability). *Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^c$ be a collection of probability envelopes indexed by $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$ and defined over $[0, n]^\ell$. An n -share randomized-wiring gadget*

$$G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$$

is $(p, \mathcal{E}^c)$ -cardinal random probing composable ($(p, \mathcal{E}^c)$ -cardinal RPC) for some $p \in [0, 1]$ if there exists a two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ and every collection $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$, the outputs of Experiment 1 satisfy:

1. $(|I_1|, \dots, |I_\ell|) \lesssim \mathcal{E}_{t_1^{out}, \dots, t_m^{out}}^c$, where $t_1^{out} = |J_1|, \dots, t_m^{out} = |J_m|$, and
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\text{AssignWires}(G^\rho, R, \mathcal{W}, \llbracket \mathbf{x} \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(G^\rho, p)$ and $\llbracket \mathbf{y} \rrbracket \leftarrow G^\rho(\llbracket \mathbf{x} \rrbracket)$.

Experiment 3: Cardinal Random Probing Partition

$\mathcal{W} \leftarrow \text{LeakingWires}(\text{ZeroEnc}, p)$	Drawing the leaking wires.
$\rho \leftarrow \$$	Drawing a random tape.
$\llbracket z \rrbracket \leftarrow \text{ZeroEnc}(\rho)$	Computing $\llbracket z \rrbracket$ a sharing of 0 from ρ .
$\text{leak} \leftarrow \text{AssignWires}(\text{ZeroEnc}, \rho, \mathcal{W}, \emptyset)$	Assigning the leaked values.
$\llbracket w \rrbracket \leftarrow \text{Sim}(\mathcal{W}, \text{leak})$	Leaked values are given to the simulator. It outputs $\llbracket w \rrbracket$, a sharing of 0 corresponding to the leaked part of $\llbracket z \rrbracket$.
$(P, \sigma) := \phi(\mathcal{W})$	Computing an integer partition and a permutation corresponding to the leaking wires.
$\llbracket u \rrbracket \leftarrow \sigma(\text{CPZE}(P))$	Drawing an encoding of 0 corresponding to the unrevealed randomness to compensate the randomness that was revealed in the leakage.
$\llbracket z' \rrbracket \leftarrow \llbracket w \rrbracket + \llbracket u \rrbracket$	Summing the revealed randomness together with the unrevealed randomness.
return $(P, \llbracket z \rrbracket, \llbracket z' \rrbracket)$	

A.3 Cardinal Random Probing Partitioning

Definition 16 (Cardinal Partition ZeroEncoding from [11]). For an integer partition $P = (n_1, \dots, n_\ell) \in \mathcal{P}(n)$, we define $\text{CardinalPartitionZeroEnc}(P)$ (CPZE) as a particular type of distribution producing a sharing $\llbracket z \rrbracket = \llbracket z^1 \rrbracket \parallel \dots \parallel \llbracket z^\ell \rrbracket$ where $\llbracket z^i \rrbracket$ is a fresh n_i -sharing of 0 for every $i \in [1, \ell]$. In other words, a sample

$$\left(\underbrace{z_1, \dots, z_{n_1}}_{\sum z_i = 0}, \underbrace{z_{n_1+1}, \dots, z_{n_1+n_2}}_{\sum z_i = 0}, \dots, \underbrace{z_{n-n_\ell+1}, \dots, z_n}_{\sum z_i = 0} \right) \leftarrow \text{CPZE}(n_1, \dots, n_\ell)$$

is uniformly random over $\mathbb{K}^n$ subject to the above zero-sum constraints.

Definition 17 (Cardinal RPP from [11]). Let $n \in \mathbb{N}$ and $\mathcal{P}(n)$ the set of partitions of n . Let $\mathcal{E}$ be a distribution envelope over $\mathcal{P}(n)$. An n -share zero-encoding gadget $\text{ZeroEnc} : \emptyset \rightarrow \mathbb{F}^n$ is $(p, \mathcal{E})$ -cardinal random probing partitioned (cRPP) for some $p \in [0, 1]$ if there exists a deterministic algorithm ϕ and a PPT simulator Sim such that the outputs of Experiment 3 satisfy :

1. $P \lesssim \mathcal{E}$
2. $\llbracket z \rrbracket$ and $\llbracket z' \rrbracket$ are identically distributed.

B Further Algorithms

Algorithm 1 ZeroEnc^{Double half}

Output : a n sharing $\llbracket z \rrbracket = (z_1, \dots, z_n)$ of 0.

```

 $\llbracket z \rrbracket \leftarrow (0, \dots, 0)$ 
for  $\ell \in [0, \frac{n}{2} - 1]$  do
     $r \xleftarrow{\$} \mathbb{F}$ 
     $z_{2\ell+1} \leftarrow z_{2\ell+1} + r$ 
     $z_{2\ell+2} \leftarrow z_{2\ell+2} - r$ 
end for
for  $\ell \in [0, \frac{n}{2} - 2]$  do
     $r \xleftarrow{\$} \mathbb{F}$ 
     $z_{2\ell+2} \leftarrow z_{2\ell+2} + r$ 
     $z_{2\ell+3} \leftarrow z_{2\ell+3} - r$ 
end for
     $r \xleftarrow{\$} \mathbb{F}$ 
     $z_1 \leftarrow z_1 + r$ 
     $z_n \leftarrow z_n - r$ 
return  $\llbracket z \rrbracket$ 

```

$\left. \begin{array}{l} \text{ZeroEnc}^{\text{half}} \\ \text{ZeroEnc}^{\text{half}} \text{ shifted by one index} \end{array} \right\}$

Algorithm 2 Double Half Refresh gadget

Input : a n sharing $\llbracket x \rrbracket = (x_1, \dots, x_n)$.

Output : a n sharing $\llbracket y \rrbracket = (y_1, \dots, y_n)$ such that $\sum_{i=1}^n x_i = \sum_{i=1}^n y_i$

```

 $\llbracket z \rrbracket \leftarrow \text{ZeroEnc}^{\text{Double half}}$ 
for  $i \in [1, n]$  do
     $y_i \leftarrow x_i + z_i$ 
end for
return  $\llbracket y \rrbracket$ 

```

Algorithm 3 ZeroEnc $^{\mathcal{O}(n \log n)}$

Require: A size of sharing n .

Ensure: A random n -sharing $\llbracket z \rrbracket$ of 0.

```
if  $n = 2$  then
   $r \xleftarrow{\$} \mathbb{F}$ 
  return  $(r, -r)$ 
end if
if  $n = 3$  then
   $(r_1, r_2) \xleftarrow{\$} \mathbb{F} \times \mathbb{F}$ 
  return  $(r_1, r_2, -r_1 - r_2)$ 
end if
 $(z_1, \dots, z_{\lfloor \frac{n}{2} \rfloor}) \leftarrow \mathcal{R}_{\lfloor \frac{n}{2} \rfloor}$ 
 $(z_{\lceil \frac{n}{2} \rceil}, \dots, z_n) \leftarrow \mathcal{R}_{\lceil \frac{n}{2} \rceil}$ 
for  $i = 1$  to  $\lfloor \frac{n}{2} \rfloor$  do
   $r_i \xleftarrow{\$} \mathbb{F}$ 
   $z_i \leftarrow z_i + r_i$ 
   $z_{\lfloor \frac{n}{2} \rfloor + i} \leftarrow z_{\lfloor \frac{n}{2} \rfloor + i} - r_i$ 
end for
return  $(z_1, \dots, z_n)$ 
```

Algorithm 4 $\mathcal{O}(n \log n)$

Require: $\llbracket x \rrbracket$ a n -sharing.

Ensure: A refreshed version $\llbracket y \rrbracket$ of x .

```
 $(z_1, \dots, z_n) \leftarrow \mathcal{R}_n$ 
for  $i = 1$  to  $n$  do
   $y_i \leftarrow x_i + z_i$ 
end for
return  $(y_1, \dots, y_n)$ 
```

Algorithm 5 Cardinal RPC Envelope Computation with IronMask

Input: A gadget G and a leakage rate $p \in [0, 1]$.

Output: A cardinal RPC envelope $\mathcal{E}$ of G .

```

1: Initialize  $\mathcal{E}_{t^{\text{out}}}(t^{\text{in}}) \leftarrow 0$  for all  $t^{\text{in}}, t^{\text{out}} \in [n]$ 
2: for each  $t^{\text{out}} \in [n]$  do
3:   for each set of output share indices  $J$  such that  $|J| = t^{\text{out}}$  do
4:     Initialize  $\mathcal{E}_J(t^{\text{in}}) \leftarrow 0$  for all  $t^{\text{in}} \in [n]$ 
5:     for each tuple of probes  $\mathcal{W}$  in  $G$  do
6:        $I_{\min} \leftarrow \text{SIS}_G(\mathcal{W}, J)$ 
7:        $\mathcal{E}_J(|I_{\min}|) \leftarrow \mathcal{E}_J(|I_{\min}|) + p^{|\mathcal{W}|}(1-p)^{|G|-|\mathcal{W}|}$ 
8:     end for
9:     for each  $t^{\text{in}} \in [n]$  do
10:       $\mathcal{E}_{t^{\text{out}}}(t^{\text{in}}) \leftarrow \max(\mathcal{E}_{t^{\text{out}}}(t^{\text{in}}), \mathcal{E}_J(t^{\text{in}}))$ 
11:    end for
12:  end for
13: end for
14: return  $\mathcal{E}$ 

```

▷ Computed using IronMask

Algorithm 6 Add^{Ref}

Input : $\llbracket x \rrbracket, \llbracket y \rrbracket$ two n -sharing.

Input : Ref a refresh gadget.

Output : $\llbracket z \rrbracket$ a n -sharing of $x + y$.

```

 $\llbracket x' \rrbracket \leftarrow \text{Ref}(\llbracket x \rrbracket)$ 
 $\llbracket y' \rrbracket \leftarrow \text{Ref}(\llbracket y \rrbracket)$ 
for  $i = 1$  to  $n$  do
   $z_i \leftarrow x'_i + y'_i$ 
end for
return  $\llbracket z \rrbracket$ 

```

Algorithm 7 cMult^{Ref}

Input : $\llbracket x \rrbracket$ a n -sharing, c a constant.

Input : Ref a refresh gadget.

Output : $\llbracket y \rrbracket$ a n -sharing of cx .

```

 $\llbracket x' \rrbracket \leftarrow \text{Ref}(\llbracket x \rrbracket)$ 
for  $i = 1$  to  $n$  do
   $y_i \leftarrow c \cdot x'_i$ 
end for
return  $\llbracket y \rrbracket$ 

```

Algorithm 8 cAdd^{Ref}

Input : A n -sharing $\llbracket x \rrbracket = (x_1, \dots, x_n)$, a constant $c \in \mathbb{K}$

Input : Ref a refresh gadget.

Output : A n -sharing $\llbracket z \rrbracket$ where $z = x + c$.

```

 $\llbracket y \rrbracket \leftarrow \text{Ref}(\llbracket x \rrbracket)$ 
 $z_1 \leftarrow y_1 + c$ 
return  $(z_1, y_2, \dots, y_n)$ 

```

Algorithm 9 Sq^{Ref}

Input : A n -sharing $\llbracket x \rrbracket$ of x

Input : Ref a refresh gadget.

Output : A n -sharing $\llbracket z \rrbracket$ of $z = x^2$

```

 $\llbracket y \rrbracket \leftarrow \text{Ref}(\llbracket x \rrbracket)$ 
for  $i \in [1, n]$  do
   $z_i \leftarrow y_i^2$ 
end for
return  $\llbracket z \rrbracket$ 

```

Algorithm 10 $\text{UnifMatMult}^{\text{Ref}}$

Input: n_x -sharing $\llbracket x \rrbracket$ and n_y -sharing $\llbracket y \rrbracket$

Input : Ref a refresh gadget.

Output: $(n_x \cdot n_y)$ -sharing $\mathbf{M} = (M_{i,j})_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}}$ such that $\sum_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}} M_{i,j} = x \cdot y$

```

if  $n_x = 1$  and  $n_y = 1$  then
   $\mathbf{M} \leftarrow (x_1 \cdot y_1)$ 
else if  $n_x = 1$  and  $n_y = 2$  then
   $\mathbf{M} \leftarrow (x_1 \cdot y_1 \ x_1 \cdot y_2)$ 
else if  $n_x = 2$  and  $n_y = 1$  then
   $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 \\ x_2 \cdot y_1 \end{pmatrix}$ 
else if  $n_x = 2$  and  $n_y = 2$  then
   $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 & x_1 \cdot y_2 \\ x_2 \cdot y_1 & x_2 \cdot y_2 \end{pmatrix}$ 
else
   $\mathbf{x}^{(1)} \leftarrow (x_1, \dots, x_{\lfloor \frac{n_x}{2} \rfloor})$ ,  $\mathbf{x}^{(2)} \leftarrow (x_{\lceil \frac{n_x}{2} \rceil}, \dots, x_{n_x})$ 
   $\mathbf{y}^{(1)} \leftarrow (y_1, \dots, y_{\lfloor \frac{n_y}{2} \rfloor})$ ,  $\mathbf{y}^{(2)} \leftarrow (y_{\lceil \frac{n_y}{2} \rceil}, \dots, y_{n_y})$ 

   $\sigma_1, \sigma_2, \sigma_3, \sigma_4 \xleftarrow{\$} \mathcal{S}_*$ 
   $\mathbf{x}_1^{(1)} \leftarrow \sigma_1(\mathbf{x}^{(1)})$ ,  $\mathbf{x}_1^{(2)} \leftarrow \sigma_2(\mathbf{x}^{(2)})$ 
   $\mathbf{y}_1^{(1)} \leftarrow \sigma_3(\mathbf{y}^{(1)})$ ,  $\mathbf{y}_1^{(2)} \leftarrow \sigma_4(\mathbf{y}^{(2)})$ 
   $\mathbf{x}_1^{(1)} \leftarrow \text{Ref}(\mathbf{x}_1^{(1)})$ ,  $\mathbf{x}_1^{(2)} \leftarrow \text{Ref}(\mathbf{x}_1^{(2)})$ 
   $\mathbf{y}_1^{(1)} \leftarrow \text{Ref}(\mathbf{y}_1^{(1)})$ ,  $\mathbf{y}_1^{(2)} \leftarrow \text{Ref}(\mathbf{y}_1^{(2)})$ 
   $\mathbf{M}^{(1,1)} \leftarrow \text{UnifMatMult}^{\text{Ref}}(\mathbf{x}_1^{(1)}, \mathbf{y}_1^{(1)})$ 
   $\mathbf{x}_2^{(1)} \leftarrow \text{Ref}(\mathbf{x}_1^{(1)})$ ,  $\mathbf{y}_2^{(1)} \leftarrow \text{Ref}(\mathbf{y}_1^{(1)})$ 
   $\mathbf{M}^{(1,2)} \leftarrow \text{UnifMatMult}^{\text{Ref}}(\mathbf{x}_2^{(1)}, \mathbf{y}_1^{(2)})$ 
   $\mathbf{M}^{(2,1)} \leftarrow \text{UnifMatMult}^{\text{Ref}}(\mathbf{x}_1^{(2)}, \mathbf{y}_2^{(1)})$ 
   $\mathbf{x}_2^{(2)} \leftarrow \text{Ref}(\mathbf{x}^{(2)})$ ,  $\mathbf{y}_2^{(2)} \leftarrow \text{Ref}(\mathbf{y}^{(2)})$ 
   $\mathbf{M}^{(2,2)} \leftarrow \text{UnifMatMult}^{\text{Ref}}(\mathbf{x}_2^{(2)}, \mathbf{y}_2^{(2)})$ 
   $\mathbf{M} = \begin{pmatrix} \mathbf{M}^{(1,1)} & \mathbf{M}^{(1,2)} \\ \mathbf{M}^{(2,1)} & \mathbf{M}^{(2,2)} \end{pmatrix}$ 
end if
return  $\mathbf{M}$ 

```

Algorithm 11 $\text{TreeAdd}^{\text{Ref}}$

Input : $(M_{i,j})_{\substack{1 \leq i \leq k \\ 1 \leq j \leq n}}$ k n -sharings.

Input : Ref a refresh gadget.

Output : An n -sharing of $\sum_{i=1}^k \sum_{j=1}^n M_{i,j}$.

```

if  $k = 1$  then return  $(M_{1,j})_{1 \leq j \leq n}$ 
if  $k = 2$  then return  $\text{Add}^{\text{Ref}}((M_{1,j})_{1 \leq j \leq n}, (M_{2,j})_{1 \leq j \leq n})$ 
 $\llbracket z_1 \rrbracket \leftarrow \text{TreeAdd}^{\text{Ref}}((M_{i,j})_{\substack{1 \leq i \leq \lfloor \frac{k}{2} \rfloor \\ 1 \leq j \leq n}})$ ;  $\llbracket z_2 \rrbracket \leftarrow \text{TreeAdd}^{\text{Ref}}((M_{i,j})_{\substack{\lceil \frac{k}{2} \rceil \leq i \leq k \\ 1 \leq j \leq n}})$ 
return  $\text{Add}^{\text{Ref}}(\llbracket z_1 \rrbracket, \llbracket z_2 \rrbracket)$ 

```

Algorithm 12 Mult^{Ref}

Input: two n -sharings $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$

Input : Ref a refresh gadget.

Output: n -sharing $\llbracket z \rrbracket$ such that $z = x \cdot y$.

```

 $(M_{i,j})_{1 \leq i,j \leq n} \leftarrow \text{UnifMatMult}^{\text{Ref}}(\llbracket x \rrbracket, \llbracket y \rrbracket)$ 
 $\llbracket z \rrbracket \leftarrow \text{TreeAdd}^{\text{Ref}}((M_{i,j})_{1 \leq i,j \leq n})$ 
return  $\llbracket z \rrbracket$ 

```

C Convergence of Ideal Refresh Gadgets under Composition

We now highlight a fundamental difference between direct refresh gadgets and zero encoding-based refresh gadgets in terms of random probing security, that we aim to illustrate with our ideal versions:

- In the case of direct refresh, each execution manipulates the input sharing. Consequently, composing the gadget k times creates (at least) k independent opportunities for an adversary to observe all shares and recover the secret. As k increases, these opportunities accumulate, and the probability of recovering the secret increases accordingly, eventually converging towards 1.
- In contrast, zero encoding-based refresh separates randomness generation from the manipulation of the input sharing. Composing k zero encoding primitives only combines independent zero-sharings before they are added to the input shares. The original input sharing is injected only once at the end. As a result, the adversary does not obtain multiple independent opportunities to recover the secret.

We illustrate this structural difference by analyzing the idealized versions of both constructions under repeated composition and evaluating the resulting threshold RPC property (for the optimal threshold t). For the ideal direct refresh, we compute the general RPC envelope of its k -fold composition from [15]. For the ideal zero encoding-based refresh, we first combine k zero-encoding primitives in a binary tree using Lemma 2, derive the corresponding gRPP envelope, and then obtain the induced general RPC envelope from Theorem 1. In both cases, we extract the best threshold RPC advantage (for the optimal t) from the final general RPC envelope. Figure 7 reports the resulting security levels for a leakage rate $p = 2^{-5}$, for several number of shares n and according to the number k of composed ideal direct refresh gadgets or ideal zero encoding primitives combined prior to injection.

The convergence towards 1 as k increases is expected for the composition of ideal direct refresh gadgets. Recall from Equation 1 in Lemma 3 that, when all input shares are needed in the simulation (*i.e.*, $t_{in} = n$), the corresponding failure probability is

$$p^n + (1 - p^n) \cdot p^{2(n-t^{\text{out}})}.$$

Hence, for a fixed value of t^{out} , this probability is at least p^n , and is in fact of order

$$\max(p^n, p^{2(n-t^{\text{out}})}).$$

After k compositions, these opportunities accumulate, so the overall advantage grows at least as fast as $k \cdot p^n$. Consequently, one expects convergence towards 1 no later than when

$$k \approx \frac{1}{p^n}.$$

For example, in Figure 7 with $p = 2^{-5}$ and $n = 3$, we expect convergence to 1 for values of k on the order of $2^{15} = 32768$. This matches the behavior observed in the figure: the yellow solid curve corresponding to $k = 30000$ already reaches a value close to 2^{-1} and then increases rapidly towards 1. This also explains why the convergence becomes significantly slower as the number of shares n increases.

In contrast, we observe that the number of compositions of the ideal zero encoding-based refresh (the dotted curves in Figure 7) has no significant impact on the final threshold RPC advantage. This is again intuitive: the input sharing is manipulated only once, after all zero encoding blocks have been composed. Hence, unlike the ideal direct refresh, the adversary only gets a single opportunity to learn information about the input shares, namely during the final addition with the zero encoding.

D Proofs

D.1 Proof of Lemma 1

Proof. Starting from a partition $P = (I_1, \dots, I_\ell)$ at the output of **ZeroEnc**. We recall that, for all $i \in [1, \ell]$, the shares in the set I_i form a perfect zero encoding given the probed wires at the output of **ZeroEnc**.

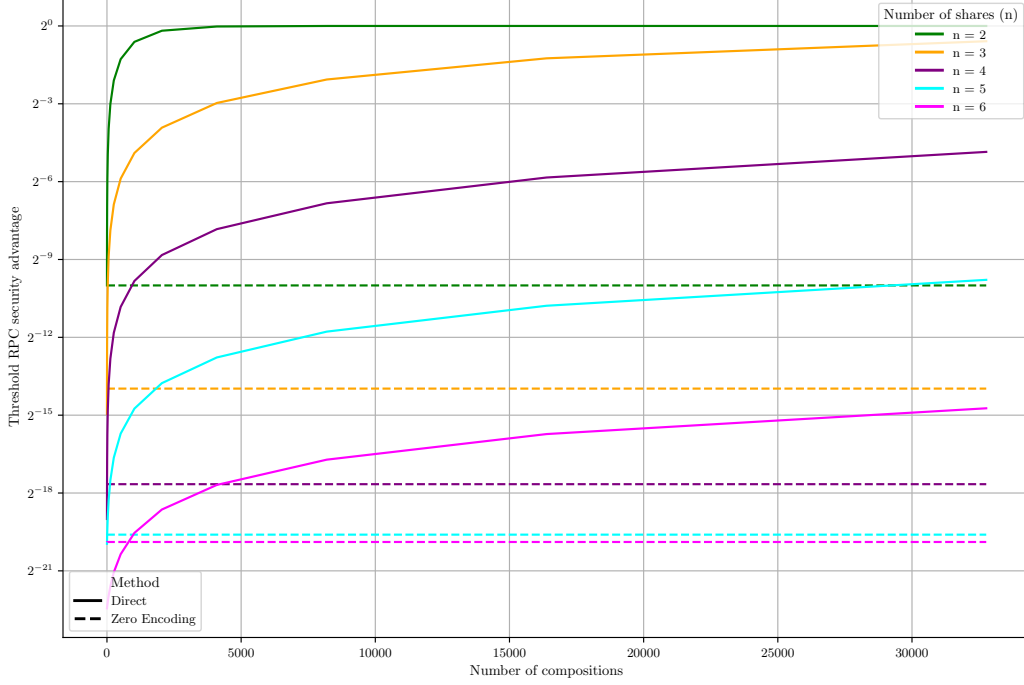


Fig. 7: Threshold RPC security of k composed ideal refresh gadgets for $p = 2^{-5}$

Consider now that we make an additional manipulation on the output shares and the output shares in $J = \{j_1, \dots, j_k\}$ leaks their values. This event occurs with probability $p^k(1-p)^{n-k}$, since exactly k shares among the n output shares leak. In this situation, each share $j \in J$ that belongs to some block I_i of the partition P is extracted from I_i , since its value is now known. This creates a new singleton block $\{j\}$ in the partition. If a single share j leaks, the partition becomes

$$P' = (I_1, \dots, I_i \setminus \{j\}, \dots, I_\ell, \{j\}).$$

Applying this operation to all shares in J yields the partition

$$P' = (I_1 \setminus J, \dots, I_\ell \setminus J, \{j_1\}, \dots, \{j_k\}).$$

The probability of obtaining this partition is the probability of starting from P (i.e., $\mathcal{E}(P)$) multiplied by the probability of the corresponding leakage event (i.e., $p^k(1-p)^{n-k}$). Finally, since the same partition P' may arise from different initial partitions $P \in \Pi_n$ or from different leakage sets J , we sum over all partitions $P \in \Pi_n$ and all leakage sets $J \subseteq [n]$. $\square$

D.2 Proof of Theorem 1

Proof. The proof follows the same strategy as Theorem 1 of [11], adapted to the general partition setting. For $I, J \subseteq [n]$, in the general RPC experiment, the simulator is given exactly the input shares indexed by I and must perfectly simulate (i) the internal leakage of the refresh gadget and (ii) the output shares indexed by J . Given the random probing leakage, we must compute the corresponding probability of this scenario.

As in [11], we decompose I into two parts, $I' \subseteq I$ and $I \setminus I'$, corresponding respectively to the input shares necessary to simulate the zero encoding leakage and the outputs, and the input shares necessary to simulate the remaining leaking input shares:

- By the gRPP property on ZeroEnc_+ , the output of the zero-encoding block induces a random partition $P \in \Pi_n$ drawn according to $\mathcal{E}^{\text{ZeroEnc}_+}$. Each block I_j corresponds to an independent zero-sharing. Whenever a block I_j is entirely contained in J , since these shares form a zero-sharing, the corresponding shares in J are no longer protected by fresh randomness and must therefore be simulated using the corresponding input shares. Let

$$I' = \bigcup_{S \in P, S \subseteq J} S$$

This set I' is precisely the set of input shares that are structurally necessary to simulate the leakage of the zero encoding block together with the output shares in J . Summing over the probabilities of all partitions P for which this union equals a given I' yields the probability $q(I', J)$ that we need the input shares in I' to simulate the leakage of the zero encoding block plus the output shares in J .

- We now have to compute the probability that we need the input shares in $I \setminus I'$ to simulate the leaking input shares that are not in I' . By the independent random probing assumption, the probability that the input shares in $I \setminus I'$ are leaking, and the input shares in $[n] \setminus I$ are not is given by

$$p^{|I \setminus I'|} (1 - p)^{n - |I|}.$$

Combining the probabilities of the two leakage sources gives the claimed expression for $\mathcal{E}_J(I)$. $\square$

D.3 Proof of Lemma 3

Proof.

Case 1: $t^{\text{in}} \neq n$ and $t^{\text{out}} \neq n$. When $t^{\text{in}} \neq n$, the leakage does not reveal the full input sharing and therefore does not determine the secret. Moreover, in our ideal model we deliberately abstract away the internal computations that map the input sharing to the output sharing. As a consequence, we cannot soundly account for leakages on the intermediate wires feeding the output gates: without specifying the operations linking inputs to outputs, such leakages cannot be related to additional information beyond what is already leaked on the input shares. In particular, for $t^{\text{in}} \neq n$, we only retain output-side leakage patterns that do not allow the entire sharing to be reconstructed. Indeed, whenever the output-side leakage is sufficient to recover the secret, the simulator requires all n input shares, and the corresponding probability must therefore be excluded from the present case. Hence, for $t^{\text{in}} \neq n$ and $t^{\text{out}} \neq n$, the relevant event for the $(t^{\text{in}}, t^{\text{out}})$ -envelope reduces to the leakage of exactly t^{in} input shares:

$$\mathcal{E}_{t^{\text{out}}}(t^{\text{in}}) = \binom{n}{t^{\text{in}}} p^{t^{\text{in}}} (1 - p)^{n - t^{\text{in}}} (1 - p^{2(n - t^{\text{out}})}).$$

Case 2: $t^{\text{out}} = n$ and $t^{\text{in}} \neq n$. If all n output shares leak, then the secret is directly revealed by recombination. Simulating this leakage requires knowledge of the full secret, hence all input shares. Therefore, for $t^{\text{in}} \neq n$, we have

$$\mathcal{E}_n(t^{\text{in}}) = 0.$$

Case 3: $t^{\text{in}} = n$. Assume that all n input shares are required for the simulation. The simulator succeeds in two disjoint situations.

1. First, all input-share wires leak with probability p^n . In this case, the simulator reconstructs the full sharing and can simulate any output leakage pattern.
2. Second, even if not all input shares leak (which happens with probability $1 - p^n$), the simulator may still recover the full sharing from the leakage occurring on the wires feeding the non-leaking outputs. For each of the $n - t^{\text{out}}$ outputs that do not leak, the corresponding two-input gate has two incoming wires, they leak with probability $p^{2(n-t^{\text{out}})}$.

Hence,

$$\mathcal{E}_{t^{\text{out}}}(n) = p^n + (1 - p^n) p^{2(n-t^{\text{out}})}.$$

□

D.4 Proof of Lemma 4

Proof. We reason from the wire structure of the ideal zero encoding-based refresh gadget depicted in Figure 1b. In the random probing model, each of the n input-share wires leaks independently with probability p . Each output share is obtained by adding the corresponding input share to the output of one of the last gates of the zero encoding block. In the ideal setting, we explicitly account for the two input wires of these final gates, whereas the internal computations of the zero encoding block are abstracted away.

We compute the probability that the simulator needs t^{in} input shares in order to simulate the random probing leakage together with t^{out} output shares. To do so, we apply the law of total probability depending on whether the outputs of the zero encoding block corresponding to the $n - t^{\text{out}}$ non-observed output shares leak or not. Each of these wires leaks with probability p' , hence all of them leak with probability $p'^{n-t^{\text{out}}}$.

Case 1: all $n - t^{\text{out}}$ wires leak. In this case, the simulator necessarily learns the t^{out} indices corresponding to the observed output shares. It must additionally recover $t^{\text{in}} - t^{\text{out}}$ indices from the leakage of the remaining $n - t^{\text{out}}$ input wires. This occurs with probability

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

if $t^{\text{in}} \geq t^{\text{out}}$ and with probability 0 otherwise.

Case 2: not all $n - t^{\text{out}}$ wires leak. This occurs with probability $1 - p'^{n-t^{\text{out}}}$. Some of the t^{out} output shares may then be recovered through leakages from the zero encoding block with the corresponding indices. Suppose that i of them are recovered in this way. This happens with probability

$$\binom{t^{\text{out}}}{i} \cdot p'^i \cdot (1 - p')^{t^{\text{out}} - i}.$$

The simulator must then recover the remaining $t^{\text{in}} - i$ indices from the leakage of the $n - i$ input wires, which happens with probability

$$\binom{n - i}{t^{\text{in}} - i} \cdot p^{t^{\text{in}} - i} \cdot (1 - p)^{n - t^{\text{in}}}.$$

Summing over all possible values of i yields the first term of the envelope expression, while the first case contributes the second term. Combining the two cases gives the bound stated in the lemma. □

D.5 Proof of Lemma 7

Proof. Let $i, j \in [n]$ with $i \neq j$. The atomic function $\text{Atomic}_{i,j}$ involves four wires: one carrying the input share z_i , one carrying the input share z_j , and three wires carrying the fresh random value r . In the random probing model, each wire leaks independently with probability p .

We denote by z_i (resp. z_j) the event that the wire carrying the share z_i (resp. z_j) leaks. Since the value r appears on three wires, the event that r leaks corresponds to the leakage of at least one of these three wires, which occurs with probability $1 - (1 - p)^3$. We denote this event by r .

The envelope is obtained by analyzing which input shares must be known by the simulator in order to reproduce both the random probing leakage and the leakage of the output shares indexed by J . The relevant cases depend on the intersection $J \cap \{i, j\}$. For each configuration of I and J , we characterize the corresponding leakage events, summarized in the following table.

	$J \cap \{i, j\} = \emptyset$	$J \cap \{i, j\} = \{i\}$	$J \cap \{i, j\} = \{j\}$	$J \cap \{i, j\} = \{i, j\}$
$I = J \setminus \{i, j\}$	$\bar{z}_i \wedge \bar{z}_j$	$\bar{r} \wedge \bar{z}_i \wedge \bar{z}_j$	$\bar{r} \wedge \bar{z}_i \wedge \bar{z}_j$	$\perp$
$I = \{i\} \cup J \setminus \{j\}$	$z_i \wedge \bar{z}_j$	$(r \vee z_i) \wedge \bar{z}_j$	$\bar{r} \wedge z_i \wedge \bar{z}_j$	$\perp$
$I = \{j\} \cup J \setminus \{i\}$	$\bar{z}_i \wedge z_j$	$\bar{r} \wedge \bar{z}_i \wedge z_j$	$(r \vee z_j) \wedge \bar{z}_i$	$\perp$
$I = J \cup \{i, j\}$	$z_i \wedge z_j$	$(r \vee z_i) \wedge z_j$	$(r \vee z_j) \wedge z_i$	$\top$

The probabilities appearing in the envelope follow directly from the independence of the leakage events and from the probabilities $\Pr[z_i] = \Pr[z_j] = p$ and $\Pr[r] = 1 - (1 - p)^3$.

Finally, any other set I occurs with probability zero, since the simulator must at least recover the indices corresponding to the output leakage. This yields the envelope stated in the lemma. $\square$

D.6 Proof of Corollary 2

Proof. It suffices to derive a cardinal RPC envelope from the general RPC envelope given in Lemma 9. Let $t^{\text{in}}, t^{\text{out}} \in [n]$. By definition,

$$\mathcal{E}_{t^{\text{out}}}(t^{\text{in}}) = \max_{J \subseteq [n] \mid |J|=t^{\text{out}}} \left(\sum_{I \subseteq [n] \mid |I|=t^{\text{in}}} \mathcal{E}_J(I) \right).$$

Hence, we characterize the sets J of size t^{out} that yield different values of the above expression and maximize over them. By Lemma 9, a set J is characterized by the number of pairs of consecutive indices (with odd starting index) it contains, namely $k_1 = |\mathcal{C}_J|$. This parameter satisfies

$$k_1 \in \left\{ \max(0, t^{\text{out}} - \frac{n}{2}), \dots, \left\lfloor \frac{t^{\text{out}}}{2} \right\rfloor \right\}.$$

Moreover, if $t^{\text{in}} < 2k_1$, then $\mathcal{E}_J(I) = 0$ for all I , so such values of k_1 do not contribute to the maximum. Hence, we restrict to $k_1 \in \mathcal{K}$.

Fix t^{out} and k_1 , and let $k_3 := t^{\text{out}} - 2k_1$ be the number of unpaired indices. Each such index corresponds to a random variable whose leakage requires knowledge of the associated input share. Let ℓ denote the number of such indices for which leakage must be simulated. Then

$$\ell \in \{0, \dots, \min(k_3, t^{\text{in}} - 2k_1)\}.$$

There are $\binom{k_3}{\ell}$ ways to choose these indices, and the corresponding leakage pattern occurs with probability $(1 - q)^\ell q^{k_3 - \ell}$.

Finally, we account for direct input leakage. Since $k_4 := 2k_1 + \ell$ input shares are already fixed, the remaining $t^{\text{in}} - k_4$ leaking shares are chosen among the $n - k_4$ remaining positions, each leaking independently with probability p .

Combining all contributions yields the desired expression. $\square$

D.7 Proof of Lemma 10

Proof. $\text{ZeroEnc}^{\text{Double half}}$ produce a zero encoding $\llbracket z \rrbracket$ where for all $i \in [n]$ odd,

$$z_i = r_{i,i+1} + r_{i-1,i}$$

where the random variable $r_{i,i+1}$ is shared between z_i and z_{i+1} and $r_{i-1,i}$ is shared between z_i and z_{i-1} (when $i = 1$, $i - 1 = n \bmod n$). These are the only variables involved in $\text{ZeroEnc}^{\text{Double half}}$. Hence, the leakage occurs only on the random variables $r_{i,i+1}$ and $r_{i-1,i}$, for all $i \in [n]$ odd, each leaks independently with probability q . We can represent the partition R obtained in an undirected graph $\mathcal{G} = (\mathcal{V}, E)$ where

1. The vertices correspond to the share indices, i.e. $\mathcal{V} = [n]$.
2. The set of possible edges consist of the edges $(i, i + 1)$ for $i \in [n - 1]$, together with $(n, 1)$, i.e., $E \subseteq \{(i, i + 1) | i \in [n - 1]\} \cup \{(n, 1)\}$.
3. The edges $(i, i + 1)$ are added to E if $r_{i,i+1}$ does not leak. These edges are represented on Figure 8.

From the circular structure of the double half zero encoding block, we can observe that $i, j \in [n]$, $i \sim_R j$ if and only if there exists a path in $\mathcal{G}$ from i to j .

Hence the parts of R are the connected components of $\mathcal{G}$ and R is of the form

$$R = \{[i_1 + 1, i_2]_\circ, [i_2 + 1, i_3]_\circ, \dots, [i_k + 1, i_1]_\circ\}$$

where $i_1, \dots, i_k \in [n]$. In this case, there are k missing edges in $\mathcal{G}$, namely:

$$(i_1, i_1 + 1), (i_2, i_2 + 1), \dots, (i_k, i_k + 1)$$

Since each edge is removed independently with probability q , we obtain that for $k \geq 2$, the graph has at least two connected components and

$$\mathcal{E}(R) = (1 - q)^k q^{n-k}$$

On the other hand, when $k \in \{0, 1\}$, $R = \{\{1, \dots, n\}\}$ and we have

$$\mathcal{E}(R) = q^n + nq^{n-1}(1 - q)$$

□

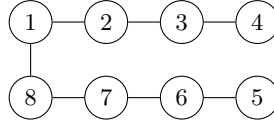


Fig. 8: An example of graph associated to $\text{ZeroEnc}^{\text{Double half}}$ with $n = 8$ and $R = \{\{1, 2, 3, 4, 5, 6, 7, 8\}\}$, where the random $r_{4,5}$ leak.

E Application to Raccoon

In this section, we present an application to Raccoon [17,24], considering $n = 16$ shares. We first describe the family of compilers used to mask Raccoon, which essentially follows the family of compilers introduced in Section 6.1. In particular, the main difference with the construction of [11] lies in the choice of the refresh gadget. We then recall the Raccoon circuit presented in [11], together with the cardinal RPC block partitioning used to derive threshold RPC security, and consequently random probing security. Finally, we compare the security guarantees obtained with the different refresh gadgets and discuss the resulting trade-offs.

E.1 Description of the Circuit Compiler

The family of compilers is the same as that described in Section 6.1, except for the handling of copy operations. Rather than duplicating the value and inserting permutations appropriately, we add a copy gadget Copy^{Ref} into the compiler. This gadget duplicates its input and refreshes each resulting copy using Ref .

E.2 Description of Raccoon-128-16 Sign

We recall that Raccoon operates over the ring $\mathcal{R}_q := \mathbb{Z}_q[x]/(x^{n_R} + 1)$, where $n_R = 512$ for Raccoon-128-16 [11]. Since the gadgets of our compiler are defined over $\mathbb{Z}_q$, we naturally extend them to $\mathcal{R}_q$ by applying the same gadget G independently to each of the n_R coefficients. We denote the resulting coefficient-wise extension by G^{vec} . We now briefly recall the circuits corresponding to the main building blocks of **Sign** for Raccoon-128-16, whose overall structure is depicted in Figure 9.

- $G_{\text{sum}}^{\text{vec}}$: This gadget combines several $G_{\text{add}}^{\text{vec}}$ gadgets arranged in a binary-tree structure in order to produce an n -sharing over $\mathcal{R}_q$ of the sum of the input secrets.
- $G_{\text{cpmult}}^{\text{vec}}$: This gadget multiplies a public polynomial $c_{\text{poly}}^i \in \mathcal{R}_q$ by a shared secret $\llbracket s^i \rrbracket \in \mathcal{R}_q^n$, for $i \in [1, \ell]$. Following the usual polynomial multiplication formulas, it consists of n_R^2 multiplications by a public constant using $\text{cMult}^{\text{Ref}}$, $n_R(n_R - 1)$ additions using Add^{Ref} , and $n_R(n_R - 1)$ copies using Copy^{Ref} .
- $\text{AddNoiseTo}^{\text{vec}}$: This gadget adds fresh randomness to an n -sharing over $\mathcal{R}_q$. It is composed exclusively of Add^{Ref} gadgets.
- $G_{\text{decode}}^{\text{vec}}$: This gadget reconstructs the secret value s from an n -sharing $\llbracket s \rrbracket$. It first applies a refresh gadget Ref so that the decoding does not reveal additional information about the current sharing beyond the reconstructed secret. It then applies the **ShareSum** gadget, which is proven Cardinal-RPC-O in [11].

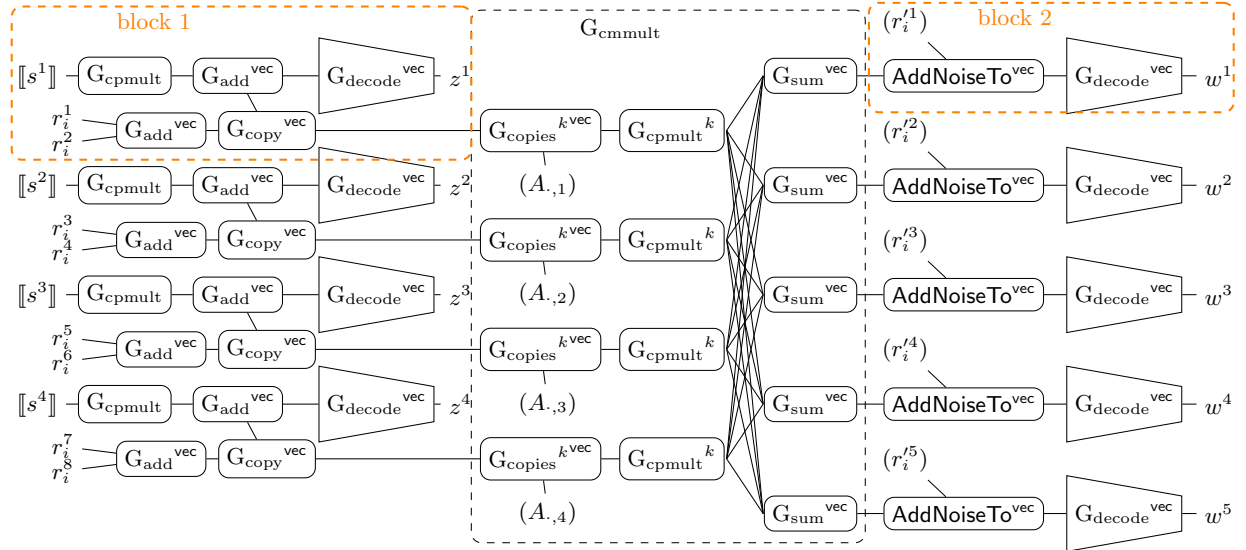


Fig. 9: RP-Sign for Raccoon-128 with $\ell = 4$ and $k = 5$

Using the cardinal-RPC envelopes of the gadgets provided by our compiler, we derive the cardinal-RPC envelopes of Block 1, Block 2, and G_{cmmult} for each of the n_R coefficients. From these envelopes, we derive the corresponding threshold-RPC security bounds, which are then combined to establish the random-probing security of the complete signature circuit. For a more detailed description and analysis of this masked scheme, we refer the reader to [11].

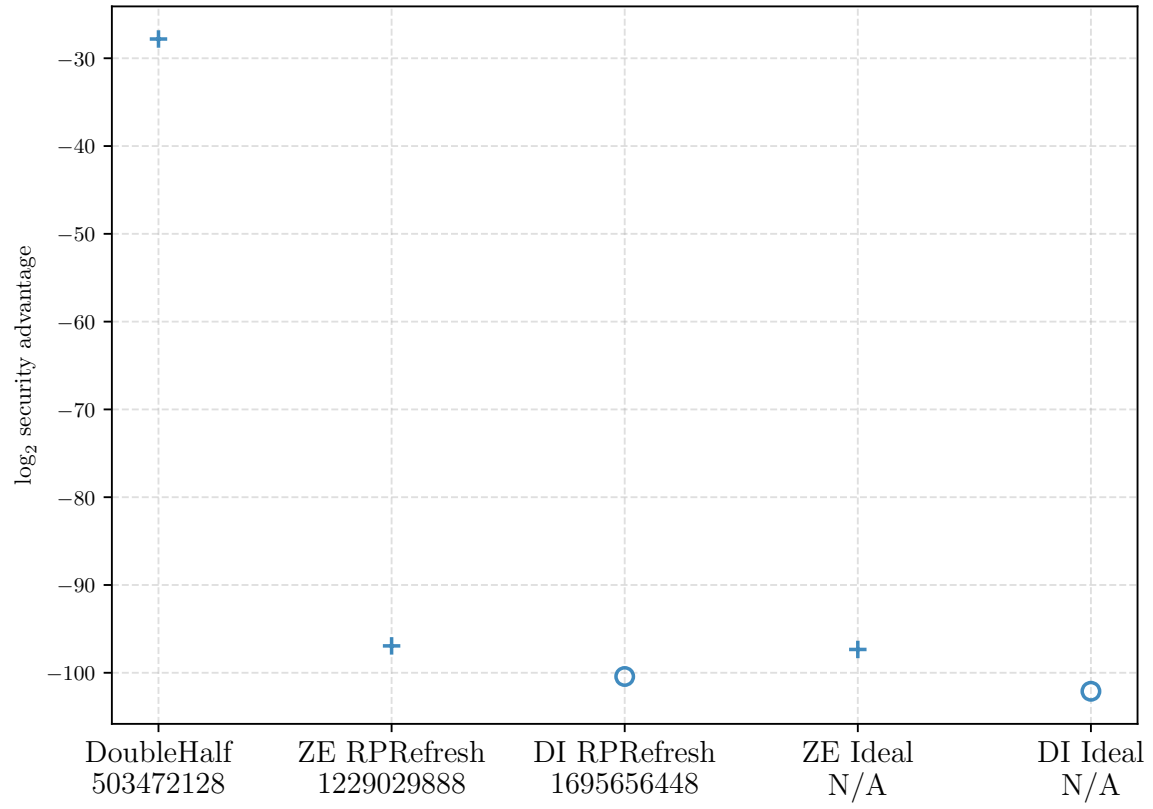
E.3 Security Results of Raccoon-128-16 Sign

We use our family of compilers with $n = 16$ shares, different refresh gadgets and derived the random probing security from Raccoon-128-16-Sign as explained in Section E.1. We focus on the most promising refresh gadgets, as well as on those for which obtaining tight cardinal RPC envelopes remains computationally affordable, namely the double half refresh gadget, the $\mathcal{O}(n \log n)$ refresh gadget and RPRrefresh. As in the previous section, we consider two leakage rates, namely $p = 2^{-16}$ and $p = 2^{-20}$. The results are displayed in Figure 10 (where the different gadgets are ordered on the x -axis according to their randomness complexity).

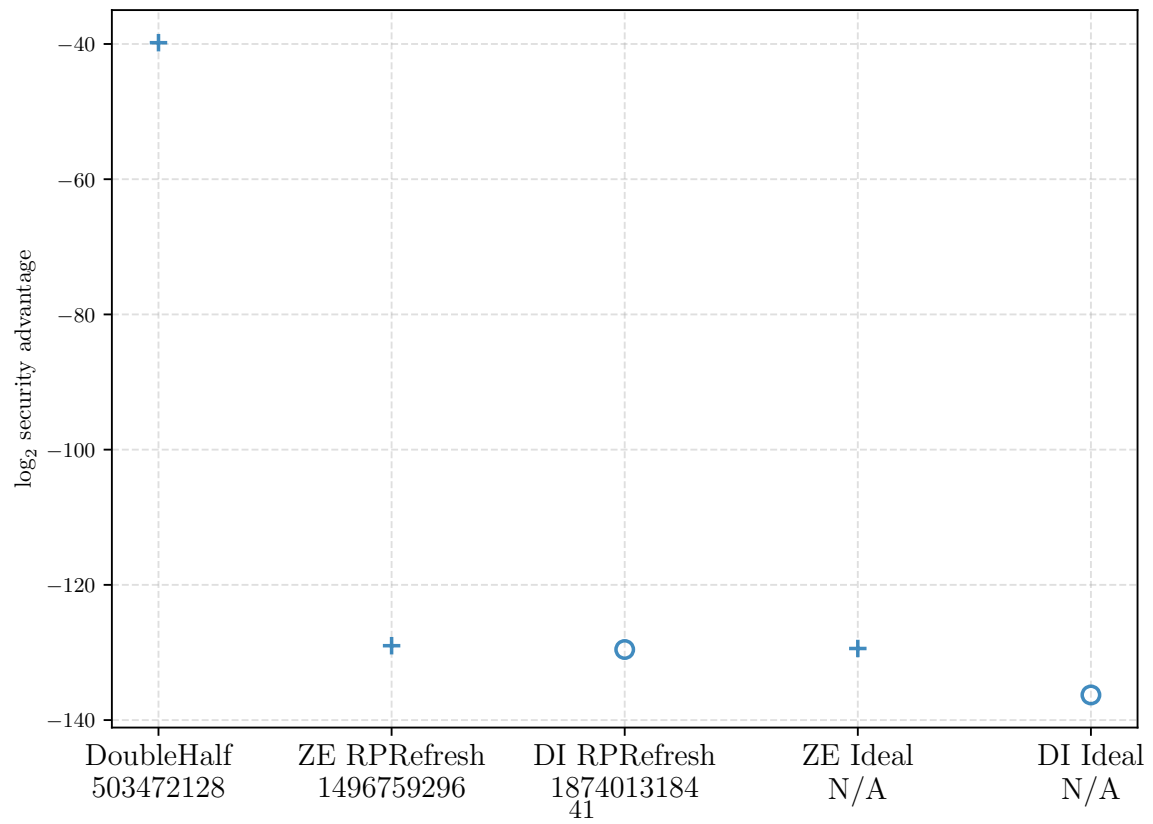
As discussed above, the $\mathcal{O}(n \log n)$ refresh gadget appears to be a strong candidate for achieving near-optimal security bounds. However, for larger numbers of shares, there is currently no efficient way to compute its cardinal RPC envelope tightly. Indeed, the exhaustive enumeration of probe tuples performed by IronMask has complexity exponential in the circuit size and already reaches its practical limit for $n = 16$. Similarly, computing the general RPC or general RPP envelopes obtained through the composition of atomic refresh functions also becomes infeasible at this scale. One may still derive a cardinal RPC envelope for the $\mathcal{O}(n \log n)$ refresh gadget by composing the cardinal RPC/RPP envelopes of the atomic refresh functions. However, as for the half refresh (see the discussion of Figure 6a), this compositional approach is not tight and therefore yields poor security bounds.

In contrast, we derived a tight analytical formula for the **double half** refresh gadget (see Corollary 3). We therefore include its zero encoding variant, whose cardinal RPC envelope is obtained from its cardinal RPP envelope. The resulting bounds nevertheless remain significantly below the ideal ones.

Finally, we observe that the security bounds obtained with RPRrefresh are very close to the optimal bounds achieved by the ideal refresh gadgets for both refresh families. This further suggests that the direct version of RPRrefresh provides slightly stronger security guarantees, at the cost of a higher randomness complexity.



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



(b) $p = 2^{-20}$

Fig. 10: Random probing security of a masked Raccoon with $n = 16$ shares