# Lattice-based Ring Verifiable Random Functions

Jie Xu[1], Muhammed F. Esgin[1], and Ron Steinfeld[1]

Faculty of Information Technology, Monash University, Australia
name.surname@monash.edu

**Abstract.** Verifiable Random Functions (VRFs) provide publicly verifiable pseudorandomness uniquely determined by a secret key and an input. While widely used in decentralized protocols, standard VRF verification reveals the signer's identity, exposing them to targeted adversarial disruption once their eligibility is known.

We study *Ring VRFs* (RVRFs), which allow a member of a public key set (a ring) to publish a VRF value along with a proof of correct generation while hiding the signer's index within the set. We formalize an algorithmic RVRF interface that binds the ring into the evaluated input to prevent cross-ring reuse and ring grinding (i.e., the malicious selection of a specific ring configuration to manipulate the pseudorandom outcome). Diverging from existing UC-based treatments, we propose a comprehensive suite of game-based security notions tailored to verifiable randomness under anonymity: correctness, anonymity, pseudorandomness, and a novel corruption-aware uniqueness notion called $T$-uniqueness. Our main technical result is a modular compiler that transforms any provable VRF into an RVRF by proving a one-out-of-many statement for the induced ring relation. We instantiate the OR layer via an optimized Fiat–Shamir OR (FS-OR) composition in the random oracle model, where the prover utilizes prover-side simulation for all non-witness branches and completes the witness branch only after a global consistency constraint is fixed. Focusing on post-quantum resilience, we provide concrete instantiations of our RVRF framework based on two state-of-the-art lattice VRFs: the long-term lattice VRF LaV by Esgin et al. (Crypto'23) and the few-time lattice VRF LB-VRF by Esgin et al. (FC'19). We provide a detailed analysis of concrete parameters across various ring sizes for both constructions and perform a comprehensive side-by-side comparison of their communication costs and security trade-offs. Our instantiations are modular, with their security reducing cleanly to (i) the base VRF's correctness, pseudorandomness, and per-key uniqueness, and (ii) standard FS-OR properties (simulatability and extractability).

**Keywords:** verifiable random functions · ring primitives · one-out-of-many proofs · post-quantum cryptography · lattices

## 1 Introduction

Verifiable Random Functions (VRFs), introduced by Micali, Rabin, and Vadhan [35], provide *publicly verifiable pseudorandomness* that is deterministically tied to a secret key and an input. Given a public key pk and an input $x$, an

evaluator holding the corresponding secret key $\mathsf{sk}$ outputs a value $v$ together with a proof that $v$ is the unique VRF output prescribed for $(\mathsf{pk}, x)$, while any PPT adversary without $\mathsf{sk}$ cannot predict $v$ on fresh inputs. Beyond their original formulation, VRFs have been studied under a variety of constructions and limitations—including number-theoretic instantiations and lower bounds on what can be achieved from certain primitives [13, 21, 27, 29]. In practice, VRFs have become a standard building block for distributed protocols, where uniqueness/determinism and public verifiability are used to make eligibility or randomness *auditable* and *non-equivocable*.

A central driver for VRF deployment is *public leader election* and *committee selection* in open networks. Prominent examples include Algorand-style cryptographic sortition [9, 26], proof-of-stake leader election in the Ouroboros line [11, 30], and randomness beacons / distributed randomness protocols [28, 41]. Related systems motivate verifiable randomness together with privacy constraints, e.g., Privacy Pass deployments and their recent cryptographic refinements [12, 19]. Across these settings, the ability to *prove* correctness of a pseudorandom value is essential, but so is the ability to withstand adaptive, targeted corruption.

Standard VRF verification is tied to a specific public key, and therefore reveals *which* participant produced an output. In open and adversarial networks, this linkage can be the primary vulnerability: once eligibility or contribution becomes known, an adaptive adversary may mount targeted denial-of-service, censorship, coercion, or bribery before the protocol completes [11, 26]. This motivates an anonymous analogue of VRFs where a party can publish verifiable randomness while hiding which member of an authorized set evaluated.

We study ring verifiable random functions (ring VRFs, RVRFs), first introduced in the ePrint preprint [6]. A ring is a public list of keys $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$. A ring member holding $\mathsf{sk}_i$ for some $\mathsf{pk}_i \in R$ evaluates on input $x$ and outputs a value $v$ and a proof $\pi$. Verification should certify that there exists $i \in [N]$ such that $v$ is a correct VRF value under $\mathsf{pk}_i$, while $\pi$ hides which index was used. This formulation is consistent with recent work exploring anonymous/ring-style VRF semantics in different models and application contexts [6, 27, 42].

**Ring grinding and ring binding.** Without binding the ring into the input, an evaluator could compute VRF outputs for the same message $x$ under multiple rings (or multiple encodings/orderings of the same ring) and selectively publish the one yielding a favorable value, a form of *ring grinding*. By defining the evaluated input as $\mu := \mathsf{Bind}(x, R) := x \| H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$, where $\mathsf{Canon}$ fixes a canonical representation of the ring $R$, $\mathsf{Enc}$ encodes it unambiguously, and $H_R$ is a (domain-separated) random oracle, we prevent such cross-ring reuse and ring grinding.

The RVRF interface is designed to be a drop-in replacement for VRFs in applications where *verifiable randomness* must coexist with *anti-targeting* requirements. *Correctness* ensures an honest ring member can always produce an accepting output. *Anonymity* formalizes anti-targeting: verification reveals that *some* authorized party evaluated, but not which one. *Pseudorandomness* captures unpredictability of $v$ on fresh inputs and, in the VRF setting, implies a natural fresh-input unforgeability guarantee (Remark 1). Finally, uniqueness must

remain meaningful under anonymity. A natural (but overly strong) requirement would be *ring-wide uniqueness*, i.e., that for any fixed $(R, x)$ there exists a single accepting value $v$. However, this contradicts correctness when keys are generated independently. For the "member-derived" semantics we target—accepted outputs certify $v = \mathsf{Eval}_{\mathsf{VRF}}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{sk}_i, \mu)$ for some (hidden) $i \in R$—different honest members may legitimately obtain different accepting outputs on the same $(R, x)$. Imposing a single ring-wide value would therefore either change the semantics (making $v$ independent of the member) or exclude natural VRF+OR constructions. We therefore do not require ring-wide uniqueness and instead adopt $T$-uniqueness, which captures corruption-limited equivocation: even if the adversary corrupts up to $T$ members, it cannot produce more than $T$ distinct accepting values for the same $(R, x)$. This notion aligns with system-level guarantees where each corrupted key corresponds to at most one "identity/pseudonym/eligibility token" per input, and honest members remain anonymous within the ring. It is natural that a party holding $T$ of the $N$ public keys in a ring can generate $T$ different outputs (using a different secret key in each case).

Consider an anonymous access-control or identity layer where users register public keys, the system publishes an authenticated digest of the registered set, and each service instance fixes a public input $x$ (e.g., a URL, domain, or epoch identifier). A registered user can run $\mathsf{RingEval}(\mathsf{pp}, \mathsf{sk}_i, R, x)$ and obtain a value $v$ that acts as a per-service pseudonym, together with a proof $\pi$ that $v$ is legitimate for *some* registered key. Here anonymity protects privacy; pseudorandomness avoids biased or linkable identifiers; and $T$-uniqueness provides a clean anti-evasion guarantee under bounded compromise: with at most $T$ corruptions, an adversary cannot generate more than $T$ distinct valid pseudonyms for the same service input. This avoids external revocation mechanisms such as time-bounded credentials or privacy revocation by a central authority or an anonymous committee.

### 1.1 Comparison with Related Primitives

Table 1 summarizes the relationship between RVRFs and nearby anonymity primitives, emphasizing VRF-style pseudorandom outputs, ring binding, and the relevant uniqueness notion.

**Relation to ring signatures and unique ring signatures.** Ring signatures and group signatures provide anonymous authentication [4, 8, 38, 39], but they do not enforce the *VRF semantics* that are essential in randomness-driven protocols. In particular, a ring signature authenticates an arbitrary message of the evaluator's choice; it does not certify that an attached value is the *unique, deterministic* output of a prescribed algorithm on a fixed input. This distinction is exactly what prevents "try-many" biasing in leader election and lottery-style applications. Unique ring signatures (URS)[23, 24, 31] add a per-input unique tag and thus appear closer to ring VRFs in identity-style use cases, but URS tags are not designed to be *pseudorandom* (nor to remain pseudorandom under adversarially generated keys), and URS does not directly provide the "verifiable randomness" interface needed when the value itself drives selection thresholds.

Table 1: High-level comparison.

| Primitive | Anonymity | V-PRF | Ring Binding | Uniqueness | Model | PQ |
|---|---|---|---|---|---|---|
| Ring sig. [38, 39]/sig.[4, 8] | ✓ | × | × | n/a | varies | varies |
| URS / link tags[23, 24, 31] | ✓ | × | × | link | varies | varies |
| Anon. VRF[25, 42] | ✓ | ✓ | NA | per-key | varies | NA |
| UC Ring VRF[6] | ✓ | ✓ | varies | varies | UC | NA |
| This work (RVRF) | ✓ | ✓ | ✓ | $T$-unq | Game | ✓ |

**Notes.** "V-PRF" means publicly verifiable pseudorandom output; "PQ" indicates whether a post-quantum instantiation is provided or explicitly addressed. "NA" means not addressed; "n/a" means not applicable; "varies" means scheme-dependent across the literature."URS" means unique ring signatures.

By contrast, a ring VRF output is a publicly verifiable *pseudorandom* value tied to the ring-bound input, while still supporting anonymity. Compared with an anonymous VRF[25, 42], which mainly hides the original public key at verification time, an RVRF directly proves that the output comes from some member of a given ring without revealing which one, and is therefore better suited to applications centered on anonymous set membership, anonymous authentication, and nullifier-style identifiers.

**Relation to existing (preprint) definitions.** The ePrint preprint [6] introduces ring VRFs in the UC framework and studies additional features such as "ZK continuations." Our focus and formalization differ in ways that matter for the use cases above. First, we adopt an explicit algorithmic RVRF interface with a ring-bound input $\mu$ and a suite of game-based notions in the ROM. This makes the intended deployment semantics, verifiable pseudorandom outputs under ring anonymity, explicit at the level of concrete experiments, and it emphasizes the *input-tied* nature of VRF outputs (the value is computationally indistinguishable from random, yet still determined by the evaluated input). Second, we parameterize uniqueness under anonymity by a corruption bound via $T$-uniqueness, which captures corruption-limited equivocation in open networks: an adversary controlling up to $T$ member keys can induce at most $T$ distinct accepting values for the same $(R, x)$. This $T$-parameterized notion aligns with deployment assumptions about bounded compromise and supports clean modular reductions to the underlying VRF and the one-out-of-many proof layer. [1]

### 1.2 Our approach and technical overview

Our approach is modular. We start from a *provable VRF*

$$\Pi_{\mathsf{vrf}} = (\mathsf{Setup}_{\mathsf{vrf}}, \mathsf{KeyGen}_{\mathsf{vrf}}, \mathsf{Eval}_{\mathsf{vrf}}, \mathsf{Verify}_{\mathsf{vrf}}),$$

[1] Relative to [6], our formulation makes explicit (i) the dependence of equivocation on the number of controlled keys (via $T$-uniqueness), and (ii) the requirement that outputs remain input-tied pseudorandom values suitable for randomness-driven protocols ( for more details see Appendix A).

where evaluation on $\mu$ produces the VRF output and auxiliary witness material: $(v, \omega_i) \leftarrow \mathsf{Eval}_{\mathsf{vrf}}(\mathsf{pp}_{\mathsf{vrf}}, \mathsf{sk}_i, \mu)$. The evaluator then proves, via a one-out-of-many (OR) proof, that $v$ is consistent with *some* public key in the ring, without revealing the index. We introduce our modular construction in Fig. 2.

For our concrete instantiation, the OR layer is realized by a Fiat–Shamir OR composition in the random oracle model. Conceptually, this follows the classical "proofs of partial knowledge" paradigm for OR composition of $\Sigma$-type protocols [10] together with Fiat–Shamir compilation and its ROM analysis toolkit [3,7,20,36]. Algorithmically, the prover simulates all non-witness branches by choosing random challenge seeds, and completes the witness branch only after a global Fiat–Shamir constraint is fixed; this yields a globally consistent transcript while hiding the witness index.

**Post-quantum instantiation from lattices.** Classical VRFs based on discrete-log/pairing techniques inherit Shor-type quantum vulnerabilities [40]. This motivates post-quantum VRFs based on lattice assumptions such as SIS/LWE and their module variants (e.g., [1, 37]). Lattice proof systems often have a linear-algebraic structure well suited to OR composition, and many practical constructions employ Fiat–Shamir-with-aborts/rejection sampling techniques [32,33]. We instantiate our framework with the long-term lattice VRF LaV [17]. Its provability relies on a rounding layer certified via the LANES$^+$ methodology, which builds on the LANES framework—a combination of techniques from [2, 16, 34]. We also contrast this instantiation with a few-time lattice VRF [15]. For broader post-quantum context and concrete parameter baselines, we relate to standard lattice primitives used in practice and standardization efforts [5, 14, 22].

### 1.3 Our Contributions

In this work, we introduce a modular and compatible with post-quantum approaches for Ring Verifiable Random Functions (RVRFs). Our main contributions are summarized as follows:

- **Formal Framework and Security Definitions for RVRFs.** We present an algorithmic RVRF interface with ring binding and a suite of game-based security notions (correctness, anonymity, pseudorandomness, and $T$-uniqueness) designed for verifiable randomness under anonymity.
- **Modular Construction via Prover-Side Simulation.** We give a modular compiler from any provable VRF and any OR proof satisfying appropriate properties (namely, completeness, simulatability, extractability) into an RVRF, and instantiate the OR layer via an optimized FS-OR in the ROM.
- **Post-Quantum Lattice-Based Instantiation.** We instantiate the framework with LaV[17](long-term) and LB-VRF[15] (few-time), and provide concrete cost tables and deployment guidance.

### 1.4 Organization

Section 2 defines RVRFs and states the security experiments. Section 3 presents the modular construction, the FS-OR layer (emphasizing prover-side simulation

of fake branches), and the LaV instantiation. Section 4 gives the security proofs, first establishing OR-layer lemmas and then composing them into RVRF theorems. Section 5 discusses parameter settings. The appendix provides a few-time lattice instantiation (LB-VRF) and proof details.

## 2 Preliminaries

**Notation.** We write $\lambda$ for the security parameter. Unless stated otherwise, all algorithms are probabilistic polynomial-time (PPT). A function $\mathsf{negl}(\lambda)$ is *negligible* if it vanishes faster than any inverse polynomial. We write $x \leftarrow S$ for sampling $x$ from a set or distribution $S$. A ring is represented as a (multi-)set or list of public keys. To avoid ambiguity (ordering, duplicates, and encoding variants), we fix: (i) a deterministic canonicalization procedure $\mathsf{Canon}(\cdot)$ and (ii) an injective serialization $\mathsf{Enc}(\cdot)$. Concretely, $\mathsf{Canon}(R)$ rejects malformed public-key encodings, sorts the remaining byte-strings under a public total order (e.g., lexicographic order on the serialized keys), and *rejects duplicates*.[2] The encoding $\mathsf{Enc}(\mathsf{Canon}(R))$ uses length-delimited framing so that no two different canonical rings map to the same string. All hash calls are domain-separated with fixed tags. In particular, we define ring binding as $\mu = \mathsf{Bind}(x, R) := x \,\|\, H_R\big(\mathsf{Enc}(\mathsf{Canon}(R))\big)$, and we use disjoint tags for $H_R$, $H$, and $H_{\mathsf{FS}}$. We work in the random oracle model (ROM), treating $H_{\mathsf{FS}}, H$, and $H_R$ either as independent random oracles or as domain-separated calls to a single oracle; all statements below hold under either interpretation. Unless stated otherwise, all public keys in a ring are assumed *valid*: each $pk$ is either generated by $\mathsf{KeyGen}_{\mathsf{VRF}}$ or is a well-formed encoding accepted by the underlying VRF's public procedures (malformed encodings are rejected). Equivalently, rings may be viewed as drawn from a registration directory enforcing proof-of-possession. All experiments reject adversary-chosen rings containing invalid public keys.

### 2.1 OR / one-out-of-many proof systems (interface)

We will use an OR (or one-out-of-many) proof system for the relation $R_{\mathsf{ring}}$.

**Definition 1 (OR proof system (syntax)).** *An OR proof system for $R_{\mathsf{ring}}$ is a pair of PPT algorithms $\Pi_{\mathsf{OR}} = (\mathsf{Prove}_{\mathsf{OR}}, \mathsf{Verify}_{\mathsf{OR}})$ such that:*

1. $\mathsf{Prove}_{\mathsf{OR}}((\mathsf{pp}_{\mathsf{vrf}}, R, \mu, v); i, w) \to \pi$;
2. $\mathsf{Verify}_{\mathsf{OR}}((\mathsf{pp}_{\mathsf{vrf}}, R, \mu, v), \pi) \to \{0, 1\}$.

**Completeness.** If $\big((\mathsf{pp}_{\mathsf{vrf}}, R, \mu, v), (i, w)\big) \in R_{\mathsf{ring}}$ then

$$\mathsf{Verify}_{\mathsf{OR}}((\mathsf{pp}_{\mathsf{vrf}}, R, \mu, v), \mathsf{Prove}_{\mathsf{OR}}((\mathsf{pp}_{\mathsf{vrf}}, R, \mu, v); i, w)) = 1$$

except with negligible probability.

[2] One may alternatively remove duplicates; the rule must be fixed publicly.

**Extractability and simulatability (informal).** For our compiler theorem, we will require that $\Pi_{\mathsf{OR}}$ is (i) *extractable* (knowledge sound) with an extractor $\mathcal{E}$, and (ii) *simulatable* with a simulator $\mathsf{Sim}$, as formalized in the security notions for the resulting ring VRF in Subsection 2.2. (Concrete instantiations in Subsection 3.3 will realize $\Pi_{\mathsf{OR}}$ via FS-OR in the ROM.)

### 2.2 Ring VRFs

**Definition 2 (Ring VRF).** *A ring VRF (RVRF) $\Pi_{\mathsf{RVRF}}$ is a tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RingEval}, \mathsf{RingVer})$. *The* $\mathsf{Setup}$ *and* $\mathsf{KeyGen}$ *algorithms generate public parameters* $\mathsf{pp}$ *and a public/secret key pair* $(\mathsf{pk}, \mathsf{sk})$, *respectively. For a ring $R$ (a finite list of public keys) and an input $x \in \{0,1\}^*$, evaluation is performed on the ring-bound input $\mu := x \,\|\, H_R\big(\mathsf{Enc}(\mathsf{Canon}(R))\big)$. The procedures* $\mathsf{Canon}(\cdot)$ *and* $\mathsf{Enc}(\cdot)$ *are fixed globally as specified in the Notation, and $H_R$ is modeled as a random oracle (or as a domain-separated call to a single random oracle). The algorithms are defined as follows:*

1. $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ *outputs public parameters (including the public parameters of the underlying VRF and OR layer, and the description of the random-oracle interfaces used).*
2. $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$ *outputs a key pair.*
3. $\mathsf{RingEval}(\mathsf{pp}, \mathsf{sk}, R, x) \to (v, \pi)$ *takes as input public parameters* $\mathsf{pp}$*, a secret key* $\mathsf{sk}$ *corresponding to some public key in $R$, a ring $R$, and an input $x$. It outputs a value $v \in \mathcal{V}$ and a proof $\pi$.*
4. $\mathsf{RingVer}(\mathsf{pp}, R, x, v, \pi) \to \{0,1\}$ *takes as input* $\mathsf{pp}$*, a ring $R$, an input $x$, and a candidate pair $(v, \pi)$, and outputs $1$ if it accepts and $0$ otherwise.*

**Definition 3 (Correctness).** *A ring VRF with* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RingEval}, \mathsf{RingVer})$ *is correct if for all $\lambda \in \mathbb{N}$, for all rings $R$, and for all $x \in \{0,1\}^*$, for all* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *and all keypairs* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *with* $\mathsf{pk} \in R$*, it holds that*

$$\Pr[\mathsf{RingVer}(\mathsf{pp}, R, x, v, \pi) = 1 \mid (v, \pi) \leftarrow \mathsf{RingEval}(\mathsf{pp}, \mathsf{sk}, R, x)] \geq 1 - \mathsf{negl}(\lambda).$$

**Definition 4 (Anonymity).** *$\Pi_{\mathsf{RVRF}}$ is* anonymous *if for all PPT $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{anon}}_{\Pi_{\mathsf{RVRF}}}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{RVRF}}}(\mathcal{A}) = 1\right] - \tfrac{1}{2}\right| \leq \mathsf{negl}(\lambda),$$

*where the experiment* $\mathsf{Exp}^{\mathsf{anon}}_{\Pi_{\mathsf{RVRF}}}$ *is defined in Figure 1.*

We define pseudorandomness using a simulator $\mathsf{Sim}$ that on input a public instance $(\mathsf{pp}, R, x, v)$ outputs a proof $\pi$ intended to be indistinguishable from an honestly generated proof for the same $(R, x, v)$, without using any secret key.

**Definition 5 ($k$-time pseudorandomness).** *$\Pi_{\mathsf{RVRF}}$ is $k$-*time pseudorandom *if there exists a PPT simulator* $\mathsf{Sim}$ *such that for all PPT $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{pr},k}_{\Pi_{\mathsf{RVRF}},\mathsf{Sim}}(\mathcal{A}) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{pr},k}_{\Pi_{\mathsf{RVRF}},\mathsf{Sim}}(\mathcal{A}) = 1\right] - \tfrac{1}{2}\right| \leq \mathsf{negl}(\lambda),$$

*where the experiment* $\mathsf{Exp}^{\mathsf{pr},k}_{\Pi_{\mathsf{RVRF}},\mathsf{Sim}}$ *is defined in Figure 1.*

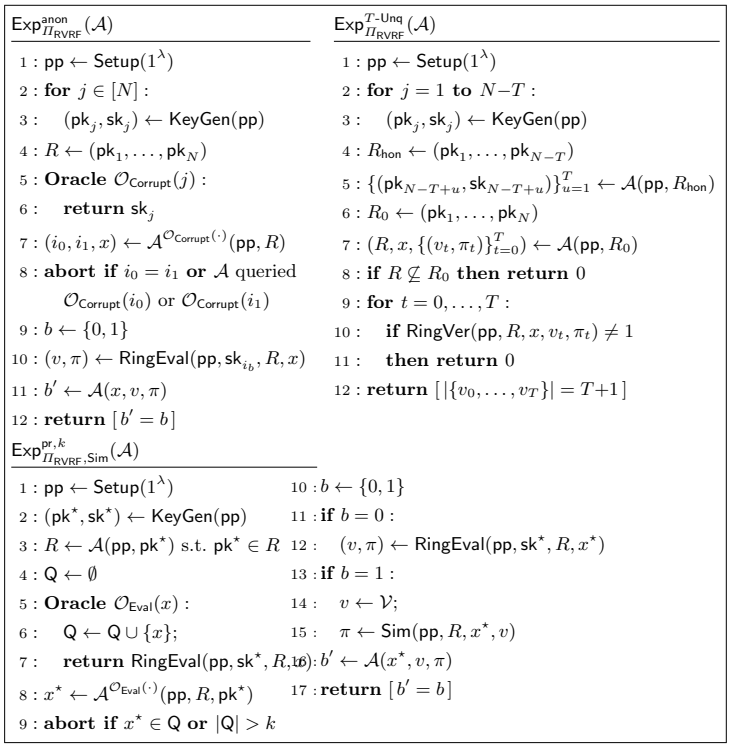


Fig. 1: RVRF experiments.

**Long-term case.** The long-term regime is the same experiment as in Definition 5, with the query bound replaced by $|\mathsf{Q}| \leq \mathsf{poly}(\lambda)$ (and still requiring $x^\star \notin \mathsf{Q}$).

*Remark 1 (Pseudorandomness ⇒ fresh-input unforgeability).* Assuming correctness and per-input uniqueness for the (ring-bound) VRF value, $k$-time (resp. long-term) pseudorandomness immediately implies existential unforgeability on fresh inputs: any adversary that outputs a new accepting $(v^\star, \pi^\star)$ for an unqueried $x^\star$ can be turned into a distinguisher in the pseudorandomness game by checking whether $v^\star$ matches the challenge value; the full reduction is given in Appendix B.

**Definition 6 ($T$-uniqueness).** *Let $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RingEval}, \mathsf{RingVer})$ be an RVRF scheme. We say that $\Pi$ satisfies $T$-*uniqueness *if for every PPT adversary $\mathcal{A}$, $\Pr\left[\mathsf{Exp}^{T\text{-}\mathsf{Unq}}_{\Pi}(\mathcal{A}) = 1\right] \leq \mathsf{negl}(\lambda)$, where the experiment $\mathsf{Exp}^{T\text{-}\mathsf{Unq}}_{\Pi}(\mathcal{A})$ is defined in Figure 1.*

*Remark 2 (Why ring-wide uniqueness is not required).* A ring-wide uniqueness requirement (a single accepting value for each fixed $(R, x)$) is not the notion we target. Our RVRF semantics are *member-derived*: an accepting output certifies $v = \mathsf{Eval}_{\mathsf{VRF}}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{sk}_i, \mu)$ for some (hidden) $i \in R$, so different honest members

may legitimately yield different valid values on the same $(R, x)$. We therefore adopt $T$-uniqueness, which bounds equivocation as a function of the number of keys controlled by the adversary.

# 3 Modular RVRF Construction and its Post-Quantum Realization from LaV

We present a modular construction that combines a provable VRF with a one-out-of-many (OR) proof system. The construction fixes (i) a provable VRF $\Pi_{\mathsf{VRF}}$, (ii) the ring-binding rule $\mu := \mathsf{Bind}(x, R)$, and (iii) an OR proof system for the induced ring relation.

## 3.1 The induced ring relation

We model a provable VRF as $\Pi_{\mathsf{VRF}} = (\mathsf{Setup}_{\mathsf{VRF}}, \mathsf{KeyGen}_{\mathsf{VRF}}, \mathsf{Eval}_{\mathsf{VRF}}, \mathsf{Verify}_{\mathsf{VRF}})$. Define the base evaluation relation

$$\big((\mathsf{pp}_{\mathsf{VRF}}, \mathsf{pk}, \mu, v), \omega\big) \in R_{\mathsf{VRF}} \iff \mathsf{Verify}_{\mathsf{VRF}}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{pk}, \mu, v, \omega) = 1,$$

where we treat the VRF proof (and any auxiliary public data carried by the proof) as the witness $\omega$.

For a ring $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ define the induced ring relation

$$\Big((\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v), (i, \omega_i)\Big) \in R_{\mathsf{ring}} \iff \mathsf{pk}_i \in R \wedge \big((\mathsf{pp}_{\mathsf{VRF}}, \mathsf{pk}_i, \mu, v), \omega_i\big) \in R_{\mathsf{VRF}}.$$

Thus $(\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v)$ is in the language iff there exists a ring index whose key validates $v$ as a correct VRF value on $\mu$.

## 3.2 Generic compiler

The generic construction is shown in Fig. 2. Given public parameters $\mathsf{pp}$, each user runs $\mathsf{KeyGen}(\mathsf{pp})$ to obtain a keypair $(\mathsf{pk}, \mathsf{sk})$ and publishes $\mathsf{pk}$. To evaluate on input $x$ with respect to a ring $R$, a evaluator with $\mathsf{pk}_i \in R$ computes $\mu := \mathsf{Bind}(x, R)$, evaluates the base VRF to obtain $(v, \omega_i)$, and then proves in zero knowledge that $(\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v)$ satisfies the ring relation by producing an OR proof. Verification checks only the OR proof on the public statement.

## 3.3 FS-OR as a concrete OR layer (simulated branches)

We instantiate the OR proof system $\Pi_{\mathsf{OR}}$ by a Fiat–Shamir OR composition (FS-OR) in the ROM. At a high level, FS-OR turns an OR statement over $N$ branches into a single non-interactive proof by letting the prover *freely choose* the challenges of all *simulated* branches, and then *derive* the remaining (real) branch challenge from a global Fiat–Shamir consistency constraint. Concretely, for each branch $j \in [N]$ the prover samples a seed $\rho_j \leftarrow \{0,1\}^{\ell}$ and derives a

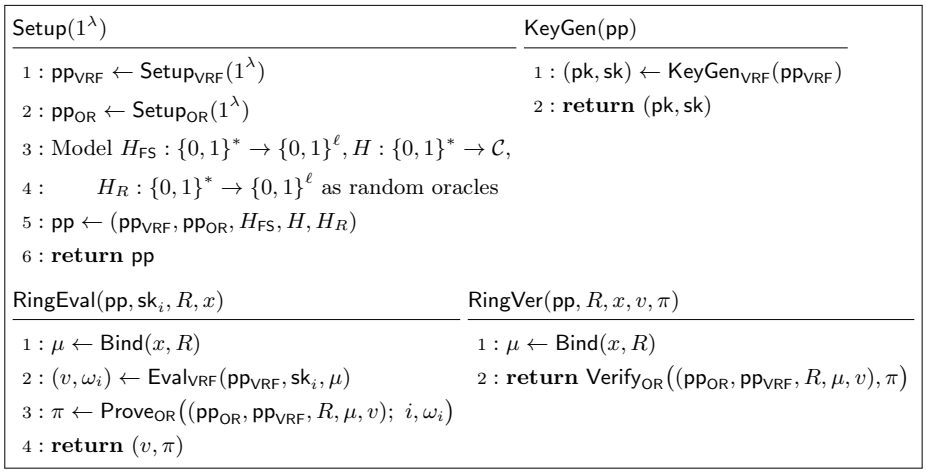


Fig. 2: Generic compiler from a provable VRF and an OR proof system to an RVRF. We include $\mathsf{pp}_{\mathsf{OR}}$ in the public statement of the OR proof system, since the OR layer may depend on its own public parameters (e.g., CRS or soundness parameters).

branch challenge $c_j := H(\rho_j) \in \mathcal{C}$. For all simulated branches $j \neq i$, the prover uses a chosen-challenge branch simulator to generate an accepting transcript $\sigma_j$ under the prescribed $c_j$. For the real branch $i$, the prover *does not* fix $c_i$ upfront. Instead, it first reconstructs a per-branch *pre-challenge commitment string* $\mathsf{com}_j$ from each transcript via a deterministic map $\mathsf{Rec}(\cdot)$ (so that the verifier can recompute it as well), and defines a single global Fiat–Shamir seed $\rho^\star := H_{\mathrm{FS}}\big(\mathsf{pp}, R, \mu, v, (\mathsf{com}_1, \ldots, \mathsf{com}_N)\big) \in \{0,1\}^\ell$. It then sets the real-branch seed by XOR gluing $\rho_i := \rho^\star \oplus \bigoplus_{j \neq i} \rho_j$, hence $c_i := H(\rho_i)$. Finally, the prover completes branch $i$ using the witness material produced by the base VRF evaluation, producing a transcript $\sigma_i$ whose embedded challenge equals $c_i$. The proof is the list $\pi_{\mathsf{OR}} = ((\rho_j, \sigma_j))_{j \in [N]}$.

Verification checks (i) that every branch transcript is accepting under the derived challenge $c_j = H(\rho_j)$, (ii) that the reconstructed commitments $\mathsf{com}_j = \mathsf{Rec}(\cdot, \sigma_j)$ match the hash input, and (iii) that the XOR constraint $\rho^\star = \bigoplus_{j=1}^N \rho_j$ holds for the recomputed $\rho^\star = H_{\mathrm{FS}}(\cdot)$. Importantly, the "simulate-then-complete" step is part of the *honest prover's* algorithm: the prover uses simulation for all $j \neq i$ to *construct* a globally consistent transcript that hides the witness index, rather than only as a reduction-level artifact.

**Interfaces required by the compiler.** The generic RVRF compiler uses the OR layer only through three interfaces: (i) completeness of honest proofs; (ii) simulatability in the ROM (for false statements); and (iii) extractability (knowledge soundness) in the ROM via forking, assuming special soundness of the branch protocol. These correspond exactly to Lemmas 1–3. Formal proofs are given in Appendix E.

### 3.4 Instantiation with LaV

We instantiate the base provable VRF $\Pi_{\mathsf{VRF}}$ with the long-term lattice VRF LaV [18]. Rather than duplicating the full LaV evaluation algorithm and its internal proof stack (rounding proof via LANES$^+$, including Fiat–Shamir-with-aborts and rejection sampling), we present a *blocked* view that exposes exactly the interaction points required by the FS-OR layer.

A LaV evaluation transcript has the form $\sigma = (t, \pi)$, where $t$ is public and $\pi = (\pi_L, (c, z, f))$ embeds an explicit challenge value $c \in \mathcal{C}$. In our ring construction, every branch challenge is derived as $c_j := H(\rho_j)$ from a seed $\rho_j \in \{0,1\}^\ell$. Accordingly, branch verification enforces *challenge consistency* by checking that the embedded challenge in $\sigma_j$ equals $c_j$. To plug LaV into FS-OR, we use the following interfaces:

- a *commit/response* split of LaV evaluation: $(v, \mathsf{st}, \mathsf{com}^{\mathsf{pre}}) \leftarrow \mathsf{LaV.Com}(\mathsf{pp}_{\mathsf{LaV}}, \mathsf{sk}, \mu)$, $\sigma \leftarrow \mathsf{LaV.Resp}(\mathsf{st}; c)$, where $\mathsf{com}^{\mathsf{pre}}$ captures the pre-challenge data fixed by LaV before the challenge is determined (and LaV.Resp may restart/abort as in the original scheme);
- a chosen-challenge simulator $\mathsf{Sim}_{\mathsf{LaV}}$ that outputs an accepting transcript $\sigma$ with embedded challenge equal to a prescribed $c$;
- a deterministic reconstruction map $\mathsf{Rec}_{\mathsf{LaV}}$ that maps a branch transcript to the canonical encoding of the *pre-challenge commitment string* hashed by the global oracle $H_{\mathrm{FS}}$.

Figure 3 presents the resulting blocked instantiation of FS-OR over LaV. Intuitively, the prover first commits to the real branch using LaV.Com (thus fixing $\mathsf{com}_i^{\mathsf{pre}}$), then fills all non-witness branches by chosen-challenge simulation, and only afterwards derives the real branch challenge from the global XOR-glue constraint and completes the real branch by calling LaV.Resp.

**About $\mathsf{Rec}_{\mathsf{LaV}}$.** The reconstruction map $\mathsf{Rec}_{\mathsf{LaV}}(\mathsf{pp}_{\mathsf{LaV}}, \mathsf{pk}, \mu, v, \sigma)$ deterministically outputs the canonical encoding of the *pre-challenge commitment string* associated with the branch transcript $\sigma$, i.e., the public data that is fixed before the challenge is determined and is therefore eligible to be hashed by the global oracle $H_{\mathrm{FS}}$ in FS-OR. The only requirement for our compiler and proofs is that $\mathsf{Rec}_{\mathsf{LaV}}$ is uniquely determined by public inputs and $\sigma$ (so the verifier recomputes the same value) and that the resulting string matches the commitment material used to define the FS-OR hash point [3].

## 4 Security Proofs

### 4.1 OR layer lemmas (FS-OR)

We use the FS-OR layer as in Section 3 (seed-derived challenges and XOR-glue). The three properties below match the interfaces required by the RVRF games. Due to space constraints and to maintain the focus on our protocol's design, the formal security proofs for Lemmas 1–3 are deferred to Appendix E.

[3] An inlined expansion that makes the exact $H_{\mathrm{FS}}$ input and $\mathsf{Rec}_{\mathsf{LaV}}$ explicit is given in Appendix C.

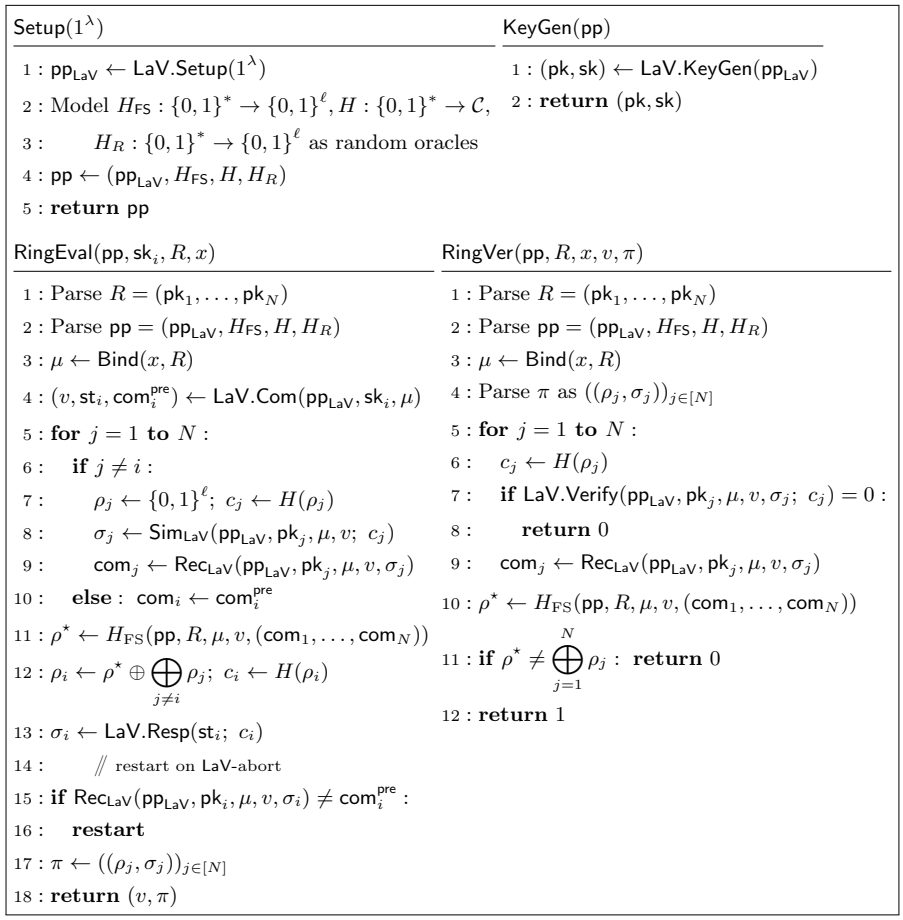


Fig. 3: Blocked view of the lattice RVRF from LaV with FS-OR glue. All LaV-internal steps (rounding proof via LANES$^+$, including rejection sampling/aborts) are encapsulated by LaV.Com/LaV.Resp/LaV.Verify. The OR proof system (and its public parameters $\mathsf{pp}_{\mathsf{OR}}$) is instantiated separately at the wrapper level and is not expanded here. The OR layer uses (i) chosen-challenge simulation $\mathsf{Sim}_{\mathsf{LaV}}$ for all $j \neq i$, and (ii) deterministic commitment reconstruction $\mathsf{Rec}_{\mathsf{LaV}}$ to define the global FS query and enable verifier recomputation.

**Lemma 1 (FS-OR completeness).** *If there exists an index $i$ and witness $\omega_i$ satisfying the OR statement, then the proof produced by the honest FS-OR prover verifies except with probability* $\mathsf{negl}(\lambda)$.

**Lemma 2 (FS-OR simulatability (ROM)).** *Assume the branch protocol admits a simulator for chosen challenges. Then there exists a PPT simulator* $\mathsf{Sim}_{\mathsf{OR}}$ *that, on input a public statement $S = (\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v)$, outputs a proof*

$\tilde{\pi}_{\mathsf{OR}}$ *such that* $\mathsf{Verify}_{\mathsf{OR}}(S, \tilde{\pi}_{\mathsf{OR}}) = 1$ *except with negligible probability, even if* $S$ *is false.*

*Proof sketch.* The simulator samples accepting branch transcripts and reconstructs the commitment vector $(\mathsf{com}_1, \ldots, \mathsf{com}_N)$. It then programs $H_{\mathsf{FS}}$ at the unique recomputable query point so that the XOR constraint holds; the only failure event is a prior adversary query to that exact point, which is negligible since it contains fresh simulated commitments.

**Lemma 3 (FS-OR extractability (ROM)).** *Assume the branch protocol has special soundness: from two accepting transcripts for the same branch with the same reconstructed commitment but different challenges one can extract a witness for that branch statement. Then FS-OR is knowledge-sound in the ROM: there exists a PPT extractor* $\mathcal{E}_{\mathsf{OR}}$ *that, given an accepting FS-OR proof for* $S = (\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v)$*, outputs an index* $i \in [N]$ *and a witness* $\omega_i$ *such that* $\big((\mathsf{pp}_{\mathsf{VRF}}, \mathsf{pk}_i, \mu, v), \omega_i\big) \in R_{\mathsf{VRF}}$*, except with negligible probability.*

### 4.2 Generic compiler theorem

**Theorem 1 (Modular compilation to RVRFs).** *Let* $\Pi_{\mathsf{RVRF}}$ *be the construction of Fig. 2, obtained by combining a provable VRF*

$$\Pi_{\mathsf{VRF}} = (\mathsf{Setup}_{\mathsf{VRF}}, \mathsf{KeyGen}_{\mathsf{VRF}}, \mathsf{Eval}_{\mathsf{VRF}}, \mathsf{Verify}_{\mathsf{VRF}})$$

*with an OR layer* $\Pi_{\mathsf{OR}}$*. Suppose:*

1. $\Pi_{\mathsf{VRF}}$ *is correct;*
2. $\Pi_{\mathsf{VRF}}$ *is pseudorandom in regime* $G$ *(either* $k$*-time or long-term)* for adversarially chosen public keys*: the challenger-generated challenge key* $(pk^\star, sk^\star)$ *may be embedded into an adversary-chosen ring, and the adversary may additionally hold secret keys for other (possibly adversarially generated) ring members, subject to the usual restriction that it does not query the evaluation oracle at the challenge input.*
3. $\Pi_{\mathsf{VRF}}$ *is per-*$\mathsf{pk}$ *computationally unique (for the ring-bound input* $\mu$*);*
4. *the OR layer is complete (Lemma 1), simulatable (Lemma 2), and extractable (Lemma 3) in the sense required by the RVRF games.*

*Then* $\Pi_{\mathsf{RVRF}}$ *satisfies correctness, anonymity, pseudorandomness in regime* $\mathcal{G}$*, and* $T$*-uniqueness (Definition 6).*

*Proof.* Fix $\lambda$ and let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. For any ring $R$ and input $x$, write $\mu := \mathsf{Bind}(x, R)$.

**Correctness.** Let $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ and let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ with $\mathsf{pk}_i \in R$. The evaluator computes $(v, \omega_i) \leftarrow \mathsf{Eval}_{\mathsf{VRF}}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{sk}_i, \mu)$ and then runs $\pi \leftarrow \mathsf{Prove}_{\mathsf{OR}}((\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v); i, \omega_i)$. By correctness of $\Pi_{\mathsf{VRF}}$, the OR statement is true for witness $(i, \omega_i)$. By Lemma 1, $\mathsf{Verify}_{\mathsf{OR}}((\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v), \pi) = 1$ except with negligible probability, hence $\mathsf{RingVer}(\mathsf{pp}, R, x, v, \pi) = 1$ except with negligible probability.

**$k$-time pseudorandomness.** Assume $\Pi_{\mathsf{VRF}}$ is $k$-time pseudorandom. Let $\mathcal{A}$ be any PPT adversary in $\mathsf{Exp}^{\mathsf{pr},k}_{\Pi_{\mathsf{RVRF}},\mathsf{Sim}}$. We build a PPT reduction $\mathcal{B}$ against $k$-time pseudorandomness of $\Pi_{\mathsf{VRF}}$.

$\mathcal{B}$ forwards $\mathsf{pp}_{\mathsf{VRF}}$ from its base challenger, samples $\mathsf{pp}_{\mathsf{OR}}$ honestly, and sets $\mathsf{pp} := (\mathsf{pp}_{\mathsf{VRF}}, \mathsf{pp}_{\mathsf{OR}})$. It runs $\mathcal{A}$. When $\mathcal{A}$ queries its RVRF evaluation oracle on input $x$ (under the ring $R$ fixed in the PR game), $\mathcal{B}$ forms $\mu = \mathsf{Bind}(R, x)$ and queries its own base evaluation oracle on $\mu$ to obtain $(v, \omega)$; it then returns $(v, \pi)$ where $\pi \leftarrow \mathsf{Prove}_{\mathsf{OR}}((\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v); i, \omega)$ for the corresponding index $i$. This preserves the $k$-time restriction since $\mathcal{B}$ makes exactly one base query per RVRF query.

At the challenge step, $\mathcal{A}$ outputs a fresh $x^\star$ (as enforced by $\mathsf{Exp}^{\mathsf{pr},k}_{\Pi_{\mathsf{RVRF}},\mathsf{Sim}}$). $\mathcal{B}$ forms $\mu^\star = x^\star \| H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$ and obtains from its base challenger a value $v^\star$ that is either the true VRF output on $\mu^\star$ or uniform (depending on the base bit). It then computes $\pi^\star \leftarrow \mathsf{Sim}_{\mathsf{OR}}(\mathsf{pp}_{\mathsf{VRF}}, R, \mu^\star, v^\star)$ and returns $(v^\star, \pi^\star)$ to $\mathcal{A}$. By Lemma 2, $\pi^\star$ is indistinguishable from an honestly generated OR proof on the same statement. Therefore any non-negligible advantage of $\mathcal{A}$ transfers to $\mathcal{B}$, contradicting base $k$-time pseudorandomness.

**Long-term pseudorandomness.** The argument is identical, except that $\mathcal{A}$ may issue polynomially many $\mathcal{O}_{\mathsf{Eval}}(\cdot)$ queries and $\mathcal{B}$ forwards each to its long-term base oracle on the corresponding ring-bound input.

**Anonymity.** Let $\mathcal{A}$ be any PPT adversary in the anonymity experiment. Fix the ring $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$, input $x$, and distinct indices $i_0 \neq i_1$ output by $\mathcal{A}$, and write $\mu = \mathsf{Bind}(R, x)$. Hybrid $\mathsf{H}_0$ is the real experiment: the challenger computes $(v, \omega) \leftarrow \mathsf{Eval}_{\mathsf{VRF}}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{sk}_{i_b}, \mu)$ and $\pi \leftarrow \mathsf{Prove}_{\mathsf{OR}}((\mathsf{pp}_{\mathsf{VRF}}, R, \mu, v); i_b, \omega)$. In $\mathsf{H}_1$, the FS-OR proof is simulated and thus independent of the witness index, so any dependence on $b$ is only through the base-VRF value $v = \mathsf{Eval}_{\mathsf{VRF}}(\mathsf{pp}_{\mathsf{VRF}}, sk_{i_b}, \mu)$ on the ring-bound input $\mu = \mathsf{Bind}(R, x)$. We perform a two-step hybrid argument: first replace $v$ under $pk_{i_0}$ by uniform, then replace uniform by $v$ under $pk_{i_1}$. Each step reduces to pseudorandomness of $\Pi_{\mathsf{VRF}}$ for the corresponding public key in the same regime $G$, and the total distinguishing advantage is bounded by the sum of the two base advantages (via a union bound).

**$T$-uniqueness.** Suppose $\mathcal{A}$ wins $\mathsf{Exp}^{T\text{-}\mathsf{Unq}}_{\Pi_{\mathsf{RVRF}}}$ with non-negligible probability, outputting $R \subseteq R_0$, an input $x$, and $T{+}1$ verifying pairs $\{(v_t, \pi_t)\}_{t=0}^{T}$ with all $v_t$ distinct. For each $t$, apply OR extractability (Lemma 3) to obtain an index $i_t$ and witness $\omega_t$ such that $\mathsf{Verify}_{\mathsf{VRF}}(\mathsf{pp}_{\mathsf{VRF}}, \mathsf{pk}_{i_t}, \mu, v_t, \omega_t) = 1$. By per-$\mathsf{pk}$ uniqueness of $\Pi_{\mathsf{VRF}}$, distinct $v_t$ imply distinct extracted indices $i_t$, hence we obtain $T{+}1$ distinct indices. Since $\mathcal{A}$ can corrupt at most $T$ indices in $\mathsf{Exp}^{T\text{-}\mathrm{Unq}}$, at least one extracted index $i^\dagger$ corresponds to an uncorrupted member. For this $i^\dagger$, the extractor outputs a witness $\omega^\dagger$ such that $\mathsf{Verify}_{\mathsf{VRF}}(pp_{\mathsf{VRF}}, pk_{i^\dagger}, \mu, v^\dagger, \omega^\dagger) = 1$. Because $\mathcal{A}$ does not have access to $sk_{i^\dagger}$ and never queried an evaluation oracle for that key on the challenge input, the pair $(v^\dagger, \omega^\dagger)$ constitutes a *fresh-input forgery* for the base provable VRF at input $\mu = \mathsf{Bind}(R, x)$. By Remark 1 (formalized in Appendix B), fresh-input unforgeability follows from pseudorandomness (under correctness and per-input uniqueness), yielding a contradiction. Hence $\Pi_{\mathsf{RVRF}}$ satisfies $T$-uniqueness.

### 4.3 Corollary for the LaV instantiation

**Corollary 1.** *Assume* LaV *satisfies correctness, per-*pk *pseudorandomness in its long-term regime, and per-*pk *uniqueness. Assume further that the LANES$^+$-based rounding layer underlying* LaV *provides: (i) a simulator for chosen challenges and (ii) special soundness for branch transcripts, so that Lemmas 2–3 apply. Then the instantiation of Section 5 is a long-term lattice RVRF satisfying the security notions of Subsection 2.2.*

*Proof.* Directly from Theorem 1 with $\Pi_{\mathsf{VRF}} = \mathsf{LaV}$ and $\Pi_{\mathsf{OR}}$ realized by FS-OR over the LaV branch protocol, using the deterministic reconstruction map $\mathsf{Rec}_{\mathsf{LaV}}$ and the chosen-challenge simulator $\mathsf{Sim}_{\mathsf{LaV}}$.

## 5 Parameter Setting

### 5.1 Concrete parameter selection (RVRF[LaV])

**Security assumptions and whether we can reuse LaV parameters.** Our instantiation RVRF[LaV] wraps the long-term lattice VRF LaV [18] with the FS-OR ring layer. Crucially, the wrapper does *not* alter the underlying lattice hardness assumptions:

- **Pseudorandomness.** Inherited from LaV's long-term pseudorandomness, which is based on an MLWE/MLWR-type assumption in the long-term regime (as formalized in [18]).
- **(Per-key) uniqueness and proof soundness.** Inherited from LaV's correctness/uniqueness and the soundness of its rounding-proof stack (LANES$^+$), which is ultimately reduced to an MSIS/MSIS$^\infty$-type hardness assumption in [18].

The FS-OR layer introduces only *standard ROM* requirements (for $H_{\mathrm{FS}}$ and the hash-to-challenge map $H$) and a *polynomial* reduction loss that depends on the number of oracle queries and (mildly) on the ring size $N$ (e.g., via a union bound over branches and a forking argument on $H_{\mathrm{FS}}$). Therefore, for all practically relevant ring sizes (e.g., $N$ polynomial in $\lambda$), it is appropriate to reuse LaV's concrete parameters without changing the underlying lattice dimensions.

One subtlety is that in the ring layer we *re-derive* each branch challenge as $c_j := H(\rho_j)$ for fresh seeds $\rho_j \in \{0,1\}^\ell$ and enforce *challenge-consistency* inside each LaV transcript. Since $H$ is modeled as a random oracle (or a domain-separated call to one), the induced distribution of $c_j$ matches the distribution required by LaV's Fiat–Shamir analysis, and the same "challenge difference invertibility" condition used by LaV also suffices for FS-OR extraction. Finally, to keep the FS-OR soundness error negligible, we set $\ell = \lambda = 128$ bits; the (worst-case) soundness error contributed by guessing the FS glue is at most $\mathsf{poly}(\lambda, N) \cdot 2^{-\ell}$, which remains negligible for polynomial-size rings.

**Concrete parameters and resulting sizes.** LaV is instantiated as in [18] (Sec. 6.5), the reported sizes are $|\mathsf{pk}| \approx 5.81\,\mathrm{KB}$, $|v| \approx 124\,\mathrm{B}$, and $|\sigma_{\mathsf{LaV}}| \approx$

10.27 KB, where $\sigma_{\mathsf{LaV}}$ denotes the full proof object output by LaV (including the public "first message" $t$ and the Fiat–Shamir proof $\pi$). In RVRF[LaV] (Fig. 3 / Fig. 5), the ring proof has the form $\pi = \big((\rho_j, \sigma_{\mathsf{LaV},j})\big)_{j\in[N]}$, where each seed $\rho_j \in \{0,1\}^\ell$ induces the branch challenge $c_j = H(\rho_j)$ and $\sigma_{\mathsf{LaV},j}$ is a LaV-format transcript for branch $j$ (real for $j = i$, simulated for $j \neq i$). Hence, excluding transmission of $R$, the communication cost is $|\pi| = N \cdot \big(|\rho| + |\sigma_{\mathsf{LaV}}|\big)$ and $|v| + |\pi| = |v| + N \cdot \big(|\rho| + |\sigma_{\mathsf{LaV}}|\big)$. We take $\ell = \lambda = 128$ so $|\rho| = 16$ bytes. Proving requires one honest LaV evaluation/proof and $(N-1)$ simulated branch transcripts; verification performs $N$ calls to LaV.Verify plus the global hash/glue checks. Detailed concrete cost tables across ring sizes are provided in Appendix F.

**Dominant costs and scaling.** The dominant verifier cost is $N$ *invocations of the base VRF verifier* (e.g., LaV.Verify or LB-VRF.Verify); hashing and XOR glue are negligible in comparison. The prover performs one honest base evaluation and $(N-1)$ simulations for the remaining branches. Consequently, under the FS-OR baseline, both proof size and verification time scale linearly in the ring size $N$.

### 5.2 Concrete parameter selection (RVRF[LB-VRF])

We instantiate the base provable VRF with the few-time lattice VRF LB-VRF [15] and adopt its concrete parameter set (same security target as in the base paper). The FS-OR wrapper introduces the same additional ROM assumptions as in the LaV instantiation; the underlying lattice assumptions remain those of LB-VRF (MLWE/MLWR-type pseudorandomness in the few-time regime and MSIS-type soundness/uniqueness). Since LB-VRF provides $k$-time pseudorandomness and is intended for a small number of evaluations per key pair (e.g. $k \leq 5$), RVRF[LB-VRF] should be viewed as a short-lived option relying on key rotation (and ring updates) to remain within its security regime; large rings further amplify the system-level cost because each evaluation already incurs $N$ base verifications and linear communication in $N$. From LB-VRF [15] we take the concrete sizes: $|\mathsf{pk}| \approx 3.32\,\text{KB}, |v| \approx 84\,\text{B}$, $|\sigma_{\mathsf{LB\text{-}VRF}}| \approx 4.94\,\text{KB}$, and again use $|\rho| = \lambda = 128$ bits (so $|\rho| = 16$ bytes). In RVRF[LB-VRF] the proof is $\pi = \big((\rho_j, \sigma_{\mathsf{LB\text{-}VRF},j})\big)_{j\in[N]}$, hence $|\pi| = N \cdot \big(|\rho| + |\sigma_{\mathsf{LB\text{-}VRF}}|\big)$ and $|v| + |\pi| = |v| + N \cdot \big(|\rho| + |\sigma_{\mathsf{LB\text{-}VRF}}|\big)$. Detailed concrete cost tables across ring sizes are provided in Appendix F.

### 5.3 Comparison: RVRF[LB-VRF] vs. RVRF[LaV]

Table 2 summarizes the two instantiations under the same linear FS-OR ring layer: both scale linearly in $N$ and differ mainly in the base VRF regime and transcript size.

### 5.4 Deployment guidance and practicality notes

Our FS-OR baseline yields proof size and verification time that scale linearly with the ring size $N$. This is practical in several common deployment models: (i)

Table 2: Side-by-side comparison under the linear FS-OR ring layer with $|\rho| = 128$ bits. Proof sizes exclude transmitting the ring; add $N \cdot |\mathsf{pk}|$ if $R$ is not already public.

| Instantiation | PR regime | $\lvert\mathsf{pk}\rvert$ | $\lvert v\rvert$ | $\lvert\sigma_{\mathsf{base}}\rvert$ | $\lvert\pi\rvert$ ($N$=32) | $\lvert\pi\rvert$ ($N$=256) |
|---|---|---|---|---|---|---|
| RVRF[LB-VRF] | $k$-time | 3.32 KB | 84 B | 4.94 KB | $\approx$ 158.59 KB | $\approx$ 1.24 MB |
| RVRF[LaV] | long-term | 5.81 KB | 124 B | 10.27 KB | $\approx$ 329.13 KB | $\approx$ 2.57 MB |

**Notes.** (1) In both cases, $\pi = \big((\rho_j, \sigma_{\mathsf{base},j})\big)_{j\in[N]}$, hence $|\pi| = N \cdot (|\rho| + |\sigma_{\mathsf{base}}|)$. (2) Verifier work is $N$ base verifications (plus hashing/XOR glue); prover work is one honest base evaluation plus $(N-1)$ simulated branches.

off-chain verification and bandwidth-rich environments, (ii) bounded anonymity sets, and (iii) repeated verification against a fixed ring, where implementers can cache $\mathsf{Enc}(\mathsf{Canon}(R))$ (and any derived ring digest).

**Which instantiation to use: RVRF[LB-VRF] vs. RVRF[LaV].** Both instantiations share the same FS-OR ring wrapper; the decisive difference lies in the *regime* of the underlying lattice VRF.

**RVRF[LB-VRF] (few-time, smaller transcripts).** LB-VRF targets the few-time regime [15], so RVRF[LB-VRF] is most appropriate when each key pair is used for only a small number of evaluations and the system already supports efficient key rotation.

**RVRF[LaV] (long-term, stable keys).** LaV supports long-term evaluation under fixed keys [18], so RVRF[LaV] is the natural choice when keys remain stable and the application requires many evaluations per key, or when frequent ring/key updates are undesirable[4].

## 6 Extensions and Follow-up Work

This work views an RVRF as a modular composition of (i) a provable VRF and (ii) an OR/one-out-of-many proof system for the induced ring relation, so improvements can come from strengthening either component.

**Sublinear variants via one-out-of-many.** A natural direction is to replace the linear FS-OR wrapper with a compressed one-out-of-many (OOM) proof. At a high level, one can commit to the ring representation (e.g., $\mathsf{Enc}(\mathsf{Canon}(R))$), prove membership of the evaluator key succinctly, and replace the $N$-way OR proof by a polylog($N$)-size OOM proof, reducing verification from $N$ base verifications to checking a membership proof plus a polylog($N$) proof. We leave a full instantiation and tight accounting to future work.

**Model refinements: QROM and standard model.** Our security analysis is in the classical ROM, matching the Fiat–Shamir OR layer and typical provable-VRF analyses. Extending to the QROM faces two concrete issues: (1)

[4] Implementation note: Appendix D provides an inlined reference implementation of RVRF[LB-VRF] for explicit parameter accounting.

our extractability uses classical forking at a random-oracle query, which does not directly apply to quantum superposition queries; and (2) the reduction programs the oracle at points determined by reconstructed commitments (e.g., $(\mathsf{com}_1, \ldots, \mathsf{com}_N)$), whose unpredictability must be argued against a quantum adversary. In the standard model, one can replace Fiat–Shamir by an interactive OR proof or by a CRS-based NIZK for the OR relation; our modular design allows swapping in such proof layers as they become available.

## Acknowledgements

This work was partially supported by Australian Research Council Discovery Grants DP250100229 and DP260100245.

## References

1. Ajtai, M.: Generating hard instances of lattice problems (extended abstract). In: 28th ACM STOC. pp. 99–108. ACM Press (May 1996). https://doi.org/10.1145/237814.237838 5
2. Attema, T., Lyubashevsky, V., Seiler, G.: Practical product proofs for lattice commitments. In: CRYPTO (2). pp. 470–499. LNCS, Springer (2020) 5
3. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168596 5
4. Boneh, D., Shacham, H.: Group signatures with verifier-local revocation. In: Atluri, V., Pfitzmann, B., McDaniel, P. (eds.) ACM CCS 2004. pp. 168–177. ACM Press (Oct 2004). https://doi.org/10.1145/1030083.1030106 3, 4
5. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: Algorithm specifications and supporting documentation. NIST PQC submission / specification (2018) 5
6. Burdges, J., Ciobotaru, A., Alper, Ç.K., Stewart, J., Vasilyev, A.: Ethical identity, ring vrfs, and zero-knowledge continuations. IACR Cryptology ePrint Archive, Report 2023/002 (2023) 2, 4, 21
7. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: 30th ACM STOC. pp. 209–218. ACM Press (May 1998). https://doi.org/10.1145/276698.276741 5
8. Chaum, D., Van Heyst, E.: Group signatures. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 257–265. Springer (1991) 3, 4
9. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theoretical Computer Science **777**, 155–183 (2019) 2
10. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 174–187. Springer, Berlin, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48658-5_19 5
11. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 66–98. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_3 2

12. Davidson, A., Goldberg, I., Sullivan, N., Tankersley, G., Valsorda, F.: Privacy pass: Bypassing internet challenges anonymously. PoPETs **2018**(3), 164–180 (Jul 2018). https://doi.org/10.1515/popets-2018-0026 2
13. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Berlin, Heidelberg (Jan 2005). https://doi.org/10.1007/978-3-540-30580-4_28 2
14. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: Algorithm specifications and supporting documentation. NIST PQC submission / specification (2018) 5
15. Esgin, M.F., Kuchta, V., Sakzad, A., Steinfeld, R., Zhang, Z., Sun, S., Chu, S.: Practical post-quantum few-time verifiable random function with applications to algorand. In: Financial Cryptography (FC) (2). Lecture Notes in Computer Science, vol. 12675, pp. 560–578. Springer (2021), (Full version at ia.cr/2020/1222) 5, 16, 17, 23, 25, 27, 28
16. Esgin, M.F., Nguyen, N.K., Seiler, G.: Practical exact proofs from lattices: New techniques to exploit fully-splitting rings. In: ASIACRYPT (2). LNCS, vol. 12492, pp. 259–288. Springer (2020) 5
17. Esgin, M.F., Steinfeld, R., Liu, D., Ruj, S.: Efficient hybrid exact/relaxed lattice proofs and applications to rounding and VRFs. In: CRYPTO (5). LNCS, vol. 14085, pp. 484–517. Springer (2023), (Full version at ia.cr/2022/141) 5
18. Esgin, M.F., Steinfeld, R., Liu, D., Ruj, S.: Efficient hybrid exact/relaxed lattice proofs and applications to rounding and VRFs. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 484–517. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_16 11, 15, 17, 28
19. Faut, D., Hesse, J., Kohl, L., Rupp, A.: Scalable and fine-tuned privacy pass from group verifiable random functions. In: 2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P). pp. 38–59. IEEE (2025) 2
20. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Berlin, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12 5
21. Fiore, D., Schröder, D.: On the impossibility of verifiable random functions from trapdoor permutations. IACR Cryptology ePrint Archive, Report 2012/689 (2012) 2
22. Fouque, P., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T.: FALCON: Fast-fourier lattice-based compact signatures over NTRU. NIST PQC submission / specification (2019) 5
23. Franklin, M., Zhang, H.: A framework for unique ring signatures. Cryptology ePrint Archive (2012) 3, 4
24. Fujisaki, E.: Sub-linear size traceable ring signatures without random oracles. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences **95**(1), 151–166 (2012) 3, 4
25. Ganesh, C., Orlandi, C., Tschudi, D.: Proof-of-stake protocols for privacy-aware blockchains. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 690–719. Springer, Cham (May 2019). https://doi.org/10.1007/978-3-030-17653-2_23 4
26. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th symposium on operating systems principles. pp. 51–68 (2017) 2

27. Giunta, E., Stewart, A.: Unbiasable verifiable random functions. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part IV. LNCS, vol. 14654, pp. 142–167. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58737-5_6 2
28. Hanke, T., Müller, D., et al.: DFINITY consensus system. arXiv preprint (2018) 2
29. Jager, T.: Verifiable random functions from weaker assumptions. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 121–143. Springer, Berlin, Heidelberg (Mar 2015). https://doi.org/10.1007/978-3-662-46497-7_5 2
30. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 357–388. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63688-7_12 2
31. Liu, J.K., Au, M.H., Susilo, W., Zhou, J.: Linkable ring signature with unconditional anonymity. IEEE Transactions on Knowledge and Data Engineering **26**(1), 157–165 (2013) 3, 4
32. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Berlin, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7_35 5
33. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Berlin, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_43 5
34. Lyubashevsky, V., Nguyen, N.K., Seiler, G.: Practical lattice-based zero-knowledge proofs for integer relations. In: ACM CCS. pp. 1051–1070. ACM (2020) 5
35. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th annual symposium on foundations of computer science (cat. No. 99CB37039). pp. 120–130. IEEE (1999) 1
36. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 387–398. Springer, Berlin, Heidelberg (May 1996). https://doi.org/10.1007/3-540-68339-9_33 5
37. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **51**(6), 899–929 (2005) 5
38. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: International conference on the theory and application of cryptology and information security. pp. 552–565. Springer (2001) 3, 4
39. Shacham, H., Waters, B.: Efficient ring signatures without random oracles. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 166–180. Springer, Berlin, Heidelberg (Apr 2007). https://doi.org/10.1007/978-3-540-71677-8_12 3, 4
40. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: 35th FOCS. pp. 124–134. IEEE Computer Society Press (Nov 1994). https://doi.org/10.1109/SFCS.1994.365700 5
41. Syta, E., Jovanovic, P., Kokoris-Kogias, E., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: 2017 IEEE Symposium on Security and Privacy. pp. 444–460. IEEE Computer Society Press (May 2017). https://doi.org/10.1109/SP.2017.45 2
42. Yao, S., Zhang, D.: An anonymous verifiable random function with applications in blockchain. Wireless Communications and Mobile Computing **2022**(1), 6467866 (2022) 2, 4

# Supplementary Material

## A Two clarifications relative to [6]

We highlight two points that motivate our alternative formulation. **(i) Uniqueness and corruption dependence.** Our $T$-uniqueness notion explicitly parameterizes the equivocation bound by the number of member keys controlled by the adversary: controlling up to $T$ keys yields at most $T$ distinct accepting values for a fixed $(R, x)$. In contrast, the uniqueness requirement in [6] is not parameterized by a corruption bound and is phrased in a way that can allow an adversary to produce as many distinct accepting outputs as the number of ring members it can effectively control (up to $N$ in the worst case). **(ii) Randomness vs. input-tied pseudorandomness.** For randomness-driven protocols, what is needed is *pseudorandomness tied to the input*: the output should be computationally indistinguishable from random *given* the evaluated input, while still being deterministically derived from it. The "randomness" formulation in [6] can be read as aiming for outputs that are "truly random" and not explicitly tied to the evaluated input in the algorithmic interface, which does not match the semantics required by applications such as leader election and lotteries. Our game-based notion makes the input-tied pseudorandomness requirement explicit.

## B Pseudorandomness Implies Fresh-Input Unforgeability

This appendix formalizes and proves the folklore implication used in Remark 1: for a (ring) VRF, once correctness and per-input uniqueness hold, pseudorandomness already rules out existential forgeries on fresh inputs.

### B.1 A fresh-input unforgeability game

We define a standard "fresh-input forgery" experiment for an RVRF scheme $\Pi_{\mathsf{RVRF}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RingEval}, \mathsf{RingVer})$. The oracle $\mathcal{O}_{\mathsf{Eval}}(\cdot)$ returns honest evaluations for a fixed challenge secret key and a fixed ring.

### B.2 From pseudorandomness to unforgeability

**Theorem 2 (Pseudorandomness ⇒ fresh-input unforgeability).** *Let $\Pi_{\mathsf{RVRF}}$ be an RVRF scheme. Assume:*

1. ***Correctness** (Definition 3).*
2. ***Per-input uniqueness for the ring-bound input:** for any fixed $\mathsf{pp}$, ring $R$, and input $x$, letting $\mu = x \| H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$, there exists at most one value $v \in \mathcal{V}$ for which there exists $\pi$ with $\mathsf{RingVer}(\mathsf{pp}, R, x, v, \pi) = 1$ (except negligible probability).*

$\mathsf{Exp}^{\mathsf{EUF}\text{-}fresh,k}_{\Pi_{\mathsf{RVRF}}}(\mathcal{A})$

$1 : \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$

$2 : (\mathsf{pk}^\star, \mathsf{sk}^\star) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$

$3 : R \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}^\star)$ s.t. $\mathsf{pk}^\star \in R$

$4 : \mathsf{Q} \leftarrow \emptyset$

$5 :$ Oracle $\mathcal{O}_{\mathsf{Eval}}(x) :\ \mathsf{Q} \leftarrow \mathsf{Q} \cup \{x\};$ **return** $\mathsf{RingEval}(\mathsf{pp}, \mathsf{sk}^\star, R, x)$

$6 : (x^\star, v^\star, \pi^\star) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Eval}}(\cdot)}(\mathsf{pp}, R, \mathsf{pk}^\star)$

$7 :$ **if** $x^\star \in \mathsf{Q}$ **or** $|\mathsf{Q}| \geq k$ **then return** $0$

$8 :$ **return** $[\,\mathsf{RingVer}(\mathsf{pp}, R, x^\star, v^\star, \pi^\star) = 1\,]$

Fig. 4: Fresh-input unforgeability ($k$-time). The long-term version sets $k = \infty$.

3. ***$k$-time pseudorandomness*** *of $\Pi_{\mathsf{RVRF}}$ in the sense of Definition 5 / experiment $\mathsf{Exp}^{\mathsf{pr},k}_{\Pi_{\mathsf{RVRF}},\mathsf{Sim}}$ in Fig. 1.*

*Then for every PPT adversary $\mathcal{A}$,*

$$\Pr\left[\mathsf{Exp}^{\mathsf{EUF}\text{-}fresh,k}_{\Pi_{\mathsf{RVRF}}}(\mathcal{A}) = 1\right] \leq \mathsf{Adv}^{\mathsf{pr},k}_{\Pi_{\mathsf{RVRF}},\mathsf{Sim}}(\mathcal{D}) \ + \ \mathsf{negl}(\lambda),$$

*for a PPT distinguisher $\mathcal{D}$ constructed from $\mathcal{A}$. In particular, if $\Pi_{\mathsf{RVRF}}$ is $k$-time pseudorandom, then it is fresh-input unforgeable in the above sense. The same implication holds for long-term pseudorandomness by setting $k = \infty$.*

*Proof.* Fix $\mathcal{A}$ and construct a distinguisher $\mathcal{D}$ for the pseudorandomness experiment $\mathsf{Exp}^{\mathsf{pr},k}_{\Pi_{\mathsf{RVRF}},\mathsf{Sim}}$.

$\mathcal{D}$ runs $\mathcal{A}$ and forwards $\mathcal{A}$'s evaluation queries to its own evaluation oracle $\mathcal{O}_{\mathsf{Eval}}(\cdot)$ in $\mathsf{Exp}^{\mathsf{pr},k}$, obtaining honest pairs $(v, \pi)$ and returning them unchanged to $\mathcal{A}$. When $\mathcal{A}$ outputs a candidate forgery $(x^\star, v^\star, \pi^\star)$, $\mathcal{D}$ outputs the same $x^\star$ as its challenge input in $\mathsf{Exp}^{\mathsf{pr},k}$ and receives a challenge pair $(\tilde{v}, \tilde{\pi})$, where:

– if $b = 0$ (real), then $(\tilde{v}, \tilde{\pi}) \leftarrow \mathsf{RingEval}(\mathsf{pp}, \mathsf{sk}^\star, R, x^\star)$;
– if $b = 1$ (random), then $\tilde{v} \leftarrow \mathcal{V}$ and $\tilde{\pi} \leftarrow \mathsf{Sim}(\mathsf{pp}, R, x^\star, \tilde{v})$.

Finally, $\mathcal{D}$ outputs 1 iff (i) $\mathsf{RingVer}(\mathsf{pp}, R, x^\star, v^\star, \pi^\star) = 1$ and (ii) $v^\star = \tilde{v}$.

Now condition on the event that $\mathcal{A}$ respects freshness (i.e., $x^\star$ was not queried before); otherwise both experiments return 0 by definition.

*Case $b = 0$ (real).* If $\mathcal{A}$ outputs an accepting forgery at $(x^\star, v^\star, \pi^\star)$, then by per-input uniqueness (for the ring-bound input induced by $R$ and $x^\star$) we must have $v^\star = \tilde{v}$ except negligible probability. Hence

$$\Pr[\mathcal{D} \text{ outputs } 1 \mid b = 0] \ \geq \ \Pr[\mathsf{Exp}^{\mathsf{EUF}\text{-}fresh,k}_{\Pi_{\mathsf{RVRF}}}(\mathcal{A}) = 1] \ - \ \mathsf{negl}(\lambda).$$

*Case $b = 1$ (random).* Here $\tilde{v}$ is uniform over $\mathcal{V}$ and independent of $\mathcal{A}$'s view (the proof $\tilde{\pi}$ is simulated to always verify). Therefore, even conditioning on

$\mathcal{A}$ outputting some accepting $(v^\star, \pi^\star)$, we have $\Pr[v^\star = \tilde{v}] \leq 1/|\mathcal{V}|$, which is negligible for the usual VRF output spaces. Thus

$$\Pr[\mathcal{D} \text{ outputs } 1 \mid b = 1] \;\leq\; \mathsf{negl}(\lambda).$$

Combining the two cases yields a distinguisher advantage at least the forgery probability up to negligible terms, contradicting $k$-time pseudorandomness. The long-term case is identical with $k = \infty$.

## C Inlined Reference for the LaV+FS-OR Instantiation

This appendix provides an *inlined reference implementation* of our LaV-based ring VRF instantiation. The goal is purely expository and implementational: to expose exactly which quantities are treated as *pre-challenge commitments*, which values are reconstructed by the verifier, and what is hashed by the global Fiat–Shamir oracle $H_{\mathrm{FS}}$ in the FS-OR glue.

The figure below(Fig. 5) should be read as a *line-by-line expansion* of the blocked construction from the main text (Fig. 3). In particular: (i) the underlying VRF value computation and rounding-proof logic are exactly those of LaV; (ii) the only ring-specific modification is the FS-OR wrapper, which derives per-branch challenges as $c_j := \mathcal{H}(\rho_j)$, simulates all branches $j \neq i$, and completes the real branch only after the global glue value $\rho^\star = H_{\mathrm{FS}}(\cdots)$ is fixed; and (iii) the additional function $\mathsf{Rec}_{\mathsf{LaV}}$ is deterministic and simply extracts/reconstructs the canonical *pre-challenge commitment string* from a LaV transcript, so that the verifier can recompute the exact $H_{\mathrm{FS}}$ input.

We follow LaV's original notation and its proof-stack structure: LaV calls its rounding proof, and the rounding proof internally uses LANES$^+$ (which itself relies on LANES). We do *not* redefine these subprotocols; rather, we write out their public inputs/outputs to make explicit: (1) which tuple is used as the branch commitment $\mathsf{com}_j$ fed into $H_{\mathrm{FS}}$, and (2) how the verifier recomputes $\mathsf{com}_j$ from the transmitted transcript via $\mathsf{Rec}_{\mathsf{LaV}}$. Whenever a step is identical to LaV (e.g., sampling, rejection checks, recomputation of $\widehat{\mathbf{w}}_2$), we keep the same order and naming as in the LaV specification to facilitate auditing.

While the blocked view suffices for definitions and proofs, inlining is useful for: (i) avoiding ambiguity about the exact FS-OR hash domain, (ii) enabling byte-accurate proof-size estimates (Section 5), and (iii) easing implementation and interoperability checks against an existing LaV codebase.

## D Few-time Lattice Instantiation from LB-VRF

This appendix records a compact *few-time* lattice instantiation of our single-output RVRF framework obtained by plugging the LB-VRF construction of [15] into the FS-OR ring layer. The LaV-based long-term instantiation is presented in the main body; here we focus on the few-time baseline and the interfaces needed by the FS-OR glue.

$\mathsf{Setup}(1^\lambda)$

$\mathsf{pp}_{\mathsf{LaV}} \leftarrow \mathsf{Setup}_{\mathsf{LaV}}(1^\lambda)$
Model random oracles $(H_{\mathrm{FS}}, H, H_R)$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{LaV}}, H_{\mathrm{FS}}, H, H_R)$
**return** $\mathsf{pp}$

$\mathsf{KeyGen}(\mathsf{pp})$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{LaV}}(\mathsf{pp}_{\mathsf{LaV}})$
**return** $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{RingEval}(\mathsf{pp}, \mathsf{sk}_i, R, x)$

Parse $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$
$\mu \leftarrow x \parallel H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$
$\mathbf{b} \leftarrow \mathcal{G}(\mu)$; $(\mathbf{t}, \mathbf{s}) \leftarrow (\mathsf{pk}_i, \mathsf{sk}_i)$
$v \leftarrow \lfloor \langle \mathbf{b}, \mathbf{s} \rangle \rfloor_p, e \leftarrow \langle \mathbf{b}, \mathbf{s} \rangle - \frac{q}{p} \cdot v$
Choose $(\beta, r)$ s.t. $\frac{q}{p} = \beta^r$
$\overrightarrow{b} \leftarrow$ base-$\beta$ digits of coefficients of $e$
$P(e, \overrightarrow{b}) \leftarrow \bigcirc_{k \in [\beta]} (\overrightarrow{b} - \overrightarrow{k})$
$\mathsf{mp}_i \leftarrow \{P\}$, $\mathbf{G} \leftarrow I_d \otimes (1, \beta, \ldots, \beta^{r-1})$
$\mathsf{ulp}_i \leftarrow \big((\mathbf{b}, -1, \frac{q}{p} v), (I_d, \mathbf{G})\big)$
Set $\eta, \eta_m, \phi, \phi_m$ s.t. $\eta \geq \|c\mathbf{s}\| \wedge$
  $\eta_m \geq \|ce\| \ \forall c \in \mathcal{C}$
$u \xleftarrow{\$} \mathbb{D}_{\phi_m, \eta_m, d}$ if $\mathsf{flag}_{\mathsf{rs}} = \mathsf{true}$
  otherwise $u \xleftarrow{\$} R_{q,d}$
$\mathbf{y} \leftarrow \mathcal{D}^{\ell}_{\phi, \eta, d}$
$w_i \leftarrow \mathbf{b}^\top \mathbf{y} - u$
$\mathbf{w}_{2,i} \leftarrow [[\mathbf{A}\mathbf{y}]]_K$
$\overrightarrow{s_i} \leftarrow (\overrightarrow{u}, \overrightarrow{e}, \overrightarrow{b})$
$(t_{L,i}; t'_{L,i}) \leftarrow \mathsf{LANES.Com}_{\mathsf{pp}_L}(\overrightarrow{s_i})$
$\mathsf{com}^{\mathsf{pre}}_i \leftarrow (t_{L,i}, \mathbf{w}_{2,i}, w_i)$
**for** $j = 1$ **to** $N$ :

(cont.)

  **if** $j \neq i$ :
    $\rho_j \leftarrow \{0,1\}^\ell$; $c_j \leftarrow \mathcal{H}(\rho_j)$
    $\sigma_j \leftarrow \mathsf{SimLaV}(\mathsf{pp}_{\mathsf{LaV}}, \mathsf{pk}_j, \mu, v;\ c_j)$
    $\mathsf{com}_j \leftarrow \mathsf{RecLaV}(\mathsf{pp}_{\mathsf{LaV}}, \mathsf{pk}_j, \mu, v, c_j, \sigma_j)$
  **else** $(j = i)$ : $\mathsf{com}_i \leftarrow \mathsf{com}^{\mathsf{pre}}_i$
$\rho^\star \leftarrow H_{\mathrm{FS}}(\mathsf{pp}, R, \mu, v, (\mathsf{com}_1, \ldots, \mathsf{com}_N))$
$\rho_i \leftarrow \rho^\star \oplus \bigoplus_{j \neq i} \rho_j$
$c_i \leftarrow \mathcal{H}(\rho_i)$
$z_i \leftarrow \mathbf{y} + c_i \cdot \mathbf{s}, \quad f_i \leftarrow u + c_i \cdot e$
Restart if $\mathrm{Rej}(z_i, c_i\mathbf{s}, \phi, \eta) = 0 \vee$
  $\mathrm{Rej}(f_i, c_i e, \phi_m, \eta_m) = 0$
$\mathsf{ulp}' \leftarrow \left(\mathbf{L}, \begin{pmatrix} \overrightarrow{f} \\ \mathbf{0} \end{pmatrix}\right)$, where
  $\mathbf{L} := \begin{pmatrix} \mathbf{I}_d \ \mathbf{I} \otimes \mathrm{Rot}(c_i) & \mathbf{0} \\ \mathbf{0} \qquad\qquad \mathbf{I} & -\mathbf{G} \end{pmatrix}$
$\pi_{L,i} \leftarrow \mathsf{LANES.Prove}_{\mathsf{pp}_L}\big((\mathsf{mp}_i, \mathsf{ulp}'_i), (t_{L,i}; t'_{L,i})\big)$
$\pi_i \leftarrow (\pi_{L,i}, (c_i, z_i, f_i))$
$\sigma_i \leftarrow (t_{L,i}, \pi_i)$
**if** $\mathsf{RecLaV}(\mathsf{pp}_{\mathsf{LaV}}, \mathsf{pk}_i, \mu, v, c_i, \sigma_i) \neq \mathsf{com}^{\mathsf{pre}}_i$ :
  Restart
$\pi \leftarrow ((\rho_j, \sigma_j))_{j \in [N]}$
**return** $(v, \pi, R)$

$\mathsf{RingVer}(\mathsf{pp}, R, x, v, \pi)$

Parse $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$
$\mu \leftarrow x \parallel H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$
Parse $\pi$ as $((\rho_j, \sigma_j))_{j \in [N]}$
**for** $j = 1$ **to** $N$ :
  $c_j \leftarrow \mathcal{H}(\rho_j)$
  Parse $\sigma_j = (t_j, (\pi_{L,j}, (c'_j, z_j, f_j)))$
  **if** $c'_j \neq c_j$ : **return** 0

(cont.)

  **if** $\mathrm{LaV.Verify}_{\mathsf{pp}_{\mathsf{LaV}}}(\mathsf{pk}_j, \mu, v, \sigma_j) = 0$ : **return** 0
  $\mathsf{com}_j \leftarrow \mathsf{RecLaV}(\mathsf{pp}_{\mathsf{LaV}}, \mathsf{pk}_j, \mu, v, c_j, \sigma_j)$
$\rho^\star \leftarrow H_{\mathrm{FS}}(\mathsf{pp}, R, \mu, v, (\mathsf{com}_1, \ldots, \mathsf{com}_N))$
**if** $\rho^\star \neq \bigoplus_{j=1}^{N} \rho_j$ : **return** 0
**return** 1

Fig. 5: Lattice-based RVRF from LaV with LANES$^+$ and the FS-OR glue inlined.

**Parameters and branch interface .** Fix symbolic parameters $(q, p, d, n, \ell, k, \beta, \kappa)$ as in [15]. Let $\mathcal{C}$ denote the challenge set used by the underlying lattice $\Sigma$-style proof and let $R_q$ denote the underlying ring/module. All LB-VRF-specific public parameters (including the public matrix $\mathbf{A}$ and norm bounds) are generated by LB-VRF.Setup and are encapsulated in $\mathsf{pp}_{\mathsf{LB}}$.

We model the following as random oracles: $G : \{0,1\}^* \to \bar{R}_p^{n+\ell+k}$, $H_R : \{0,1\}^* \to \{0,1\}^\ell$, $H : \{0,1\}^* \to \mathcal{C}$, $H_{\mathsf{FS}} : \{0,1\}^* \to \{0,1\}^\ell$. For a ring $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ and input $x$, define the ring-bound input $\mu := \mathsf{Bind}(x, R) := x \parallel H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$. Define the (public) derived vector $\mathbf{b} := G(\mu) \in \bar{R}_p^{n+\ell+k}$, so that in the LB-VRF instantiation the VRF output is $v = \langle \mathbf{b}, \mathsf{sk}_i \rangle \in \bar{R}_p$.

**Commitment-reconstructible branch form.** In each FS-OR branch, the prover sends a response object $\sigma$ from which the verifier can reconstruct the pre-challenge commitment deterministically. Concretely, writing $\sigma = \mathbf{z}$ and $c := H(\rho)$, the verifier reconstructs the implicit first-message pair $w^{(1)} := \mathbf{A}\mathbf{z} - c\mathsf{pk}$, $w^{(2)} := \langle \mathbf{b}, \mathbf{z} \rangle - cv$, from $(\mathsf{pk}, \mu, v, c, \mathbf{z})$, where $\mathbf{b} := G(\mu)$ is fixed by the public statement and $\mathbf{A}$ is contained in $\mathsf{pp}_{\mathsf{LB}}$. Thus the transmitted proof material per branch is just $(\rho, \mathbf{z})$, matching the FS-OR "commit–simulate–glue–complete" pattern used in the main text.

**The RVRF[LB-VRF] construction .**
*Simulation/response/reconstruction interfaces.* Let $\mathsf{pp}_{\mathsf{LB}}$ contain the public matrix $\mathbf{A}$ and the norm bounds as in [15]. For the LB-VRF branch family, we take: $\mathsf{LB\text{-}VRF.Com}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}, \mu, v;\ \mathbf{y}) := \big(\mathbf{A}\mathbf{y},\ \langle G(\mu), \mathbf{y} \rangle\big)$, $\mathsf{LB\text{-}VRF.Sim}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}, \mu, v;\ c)$ : $\mathbf{z} \leftarrow S_{\beta-\kappa}^{n+\ell+k}$ (with norm check); **return** $\sigma := \mathbf{z}$, $\mathsf{LB\text{-}VRF.Resp}(\mathsf{sk}, \mathbf{y};\ c)$ : **return** $\sigma := \mathbf{z} := \mathbf{y} + c \cdot \mathsf{sk}$, and $\mathsf{LB\text{-}VRF.Rec}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}, \mu, v, c, \sigma = \mathbf{z}) := \big(\mathbf{A}\mathbf{z} - c\mathsf{pk},\ \langle G(\mu), \mathbf{z} \rangle - cv\big)$. "Response constraints" are the same norm/range checks used by the underlying lattice $\Sigma$-style proof (as in LB-VRF).

**Lemma 4 (Correctness of RVRF[LB-VRF]).** *For all $\lambda$, all $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, all rings $R$, all inputs $x$, and all honestly generated $(\mathsf{pk}_i, \mathsf{sk}_i)$ with $\mathsf{pk}_i \in R$, if $(v, \pi) \leftarrow \mathsf{RingEval}(\mathsf{pp}, \mathsf{sk}_i, R, x)$ then $\mathsf{RingVer}(\mathsf{pp}, R, x, v, \pi) = 1$ except with probability $\mathsf{negl}(\lambda)$.*

*Proof.* Fix $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ and $\mu = \mathsf{Bind}(x, R) = x \parallel H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$. Let $\mathbf{b} := G(\mu)$. In verification, for each branch $j \in [N]$, the verifier computes $c_j = H(\rho_j)$ and reconstructs $\mathsf{com}_j = \mathsf{LB\text{-}VRF.Rec}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}_j, \mu, v, c_j, \sigma_j)$, where $\sigma_j = \mathbf{z}_j$. For simulated branches $j \neq i$, $\sigma_j$ is generated by LB-VRF.Sim and the verifier reconstructs the same $\mathsf{com}_j$ by definition. For the real branch $i$, we have $\mathbf{z}_i = \mathbf{y} + c_i\mathsf{sk}_i$ and $\mathsf{pk}_i = \mathbf{A}\mathsf{sk}_i$, hence

$$\mathbf{A}\mathbf{z}_i - c_i\mathsf{pk}_i = \mathbf{A}(\mathbf{y} + c_i\mathsf{sk}_i) - c_i\mathbf{A}\mathsf{sk}_i = \mathbf{A}\mathbf{y},$$

and similarly, since $v = \langle \mathbf{b}, \mathsf{sk}_i \rangle$,

$$\langle \mathbf{b}, \mathbf{z}_i \rangle - c_i v = \langle \mathbf{b}, \mathbf{y} \rangle + c_i \langle \mathbf{b}, \mathsf{sk}_i \rangle - c_i v = \langle \mathbf{b}, \mathbf{y} \rangle.$$

Thus $\mathsf{LB\text{-}VRF.Rec}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}_i, \mu, v, c_i, \sigma_i) = \mathsf{LB\text{-}VRF.Com}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}_i, \mu, v;\ \mathbf{y})$ whenever the norm constraints pass (otherwise the prover restarts).

$\mathsf{Setup}(1^\lambda)$

$\mathsf{pp}_{\mathsf{LB}} \leftarrow \mathsf{LB\text{-}VRF.Setup}(1^\lambda)$
Model random oracles$(H_{\mathsf{FS}}, H, H_R)$
$\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{LB}}, H_{\mathsf{FS}}, H, H_R)$
**return** $\mathsf{pp}$

$\mathsf{KeyGen}(\mathsf{pp})$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{LB\text{-}VRF.KeyGen}(\mathsf{pp}_{\mathsf{LB}})$
**return** $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{RingEval}(\mathsf{pp}, \mathsf{sk}_i, R, x)$

Parse $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$
$\mu \leftarrow \mathsf{Bind}(x, R) := x \parallel H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$
$\mathbf{b} \leftarrow G(\mu) \in \bar{R}_p^{n+\ell+k}$
$v \leftarrow \langle \mathbf{b}, \mathsf{sk}_i \rangle \in \bar{R}_p$
$\mathbf{y} \leftarrow S_\beta^{n+\ell+k}$
$\mathsf{com}_i^{\mathsf{pre}} \leftarrow \mathsf{LB\text{-}VRF.Com}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}_i, \mu, v;\ \mathbf{y})$
**for** $j = 1$ **to** $N$ :
  **if** $j \neq i$ :
    $\rho_j \leftarrow \{0,1\}^\ell;\ c_j \leftarrow H(\rho_j)$
    $\sigma_j \leftarrow \mathsf{LB\text{-}VRF.Sim}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}_j, \mu, v;\ c_j)$
    $\mathsf{com}_j \leftarrow \mathsf{LB\text{-}VRF.Rec}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}_j, \mu, v, c_j, \sigma_j)$
  **else** $(j = i)$ :
    $\mathsf{com}_i \leftarrow \mathsf{com}_i^{\mathsf{pre}}$
$\rho^\star \leftarrow H_{\mathsf{FS}}\Big(\mathsf{pp}, R, \mu, v, \{\mathsf{com}_j\}_{j\in[N]}\Big)$
$\rho_i \leftarrow \rho^\star \oplus \bigoplus_{j\neq i} \rho_j;\ c_i \leftarrow H(\rho_i)$
$\sigma_i \leftarrow \mathsf{LB\text{-}VRF.Resp}(\mathsf{sk}_i, \mathbf{y};\ c_i)$
**if** $\|\sigma_i\|_\infty > \beta - \kappa$: restart
**if** $\mathsf{LB\text{-}VRF.Rec}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}_i, \mu, v, c_i, \sigma_i) \neq \mathsf{com}_i^{\mathsf{pre}}$ :
  restart
$\pi \leftarrow \{(\rho_j, \sigma_j)\}_{j\in[N]}$
**return** $(v, \pi)$

$\mathsf{RingVer}(\mathsf{pp}, R, x, v, \pi)$

Parse $R = (\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$
$\mu \leftarrow \mathsf{Bind}(x, R) := x \parallel H_R(\mathsf{Enc}(\mathsf{Canon}(R)))$
Parse $\pi = \{(\rho_j, \sigma_j)\}_{j\in[N]}$
**for** $j = 1$ **to** $N$ :
  $c_j \leftarrow H(\rho_j)$
  **if** $\sigma_j$ violates response constraints: reject
  $\mathsf{com}_j \leftarrow \mathsf{LB\text{-}VRF.Rec}(\mathsf{pp}_{\mathsf{LB}}, \mathsf{pk}_j, \mu, v, c_j, \sigma_j)$
$\rho^\star \leftarrow H_{\mathsf{FS}}\Big(\mathsf{pp}, R, \mu, v, \{\mathsf{com}_j\}_{j\in[N]}\Big)$
**if** $\rho^\star \neq \bigoplus_{j=1}^{N} \rho_j$ **then reject**
**return** 1

Fig. 6: RVRF[LB-VRF] in the same "commit–simulate–glue–complete" style as our LaV-based RVRF. The only branch-specific public quantity shown explicitly is $\mathbf{b} := G(\mu)$ for the real branch; for simulated branches, any analogous derived values are handled inside LB-VRF.Sim and LB-VRF.Rec.

Finally, the verifier checks the XOR-glue condition:

$$\rho^\star \stackrel{?}{=} \bigoplus_{j=1}^{N} \rho_j, \qquad \text{where } \rho^\star = H_{\mathsf{FS}}\Big(\mathsf{pp}, R, \mu, v, \{\mathsf{com}_j\}_{j\in[N]}\Big).$$

By construction in RingEval, $\rho_i$ is set to satisfy this equality, so the check passes. Therefore verification accepts except with probability $\mathsf{negl}(\lambda)$.

**Corollary 2 (Security of RVRF[LB-VRF] (informal)).** *Assume the standard hardness assumptions required by* LB-VRF *[15] (e.g., the corresponding MLWR/MLWE and MSIS/MSIS$^\infty$ variants under the chosen parameters), and model $(G, H_R, H, H_{\mathsf{FS}})$ as random oracles. Then RVRF[LB-VRF] of Figure 6 satisfies: (i) anonymity, (ii) extractable soundness, and (iii) index-binding uniqueness, and it inherits the $k$-*time *pseudorandomness regime of the underlying* LB-VRF.

*Proof (Proof sketch).* This is an instantiation of the generic compiler theorem proved in the main text: the base component is LB-VRF (few-time pseudorandomness and per-pk uniqueness), and the OR component is FS-OR in the ROM (simulatability and extractability via a standard forking argument).

## E Proofs of FS-OR Properties

### E.1 Proof of Lemma 1 (Completeness)

*Proof.* All simulated branches $j \neq i$ verify by correctness of the chosen-challenge simulator. The real branch verifies by completeness of the underlying branch protocol under challenge $c_i = \mathcal{H}(\rho_i)$. Finally, the XOR check passes by construction of $\rho_i$: the real branch response is tailored to match the XOR sum of all other independently sampled or simulated $\rho_j$ values.

### E.2 Proof of Lemma 2 (Simulatability)

*Proof.* The simulator $\mathsf{Sim}_{\mathsf{OR}}$ operates as follows: It simulates every branch $j \in [N]$ under independently sampled $\rho_j \leftarrow \{0,1\}^\ell$ and $c_j = \mathcal{H}(\rho_j)$, and reconstructs the corresponding commitments $\mathsf{com}_j$. Finally, it programs the Random Oracle $\mathcal{H}_{\mathsf{fs}}$ at the specific verification query point such that $\mathcal{H}_{\mathsf{fs}}(\cdot) = \bigoplus_j \rho_j$. In the Random Oracle Model, the only event leading to simulation failure (abort) is if the adversary pre-queries the exact programmed string before the simulator can define it. This event occurs with negligible probability because the query string contains fresh, independently generated simulated commitments.

### E.3 Proof of Lemma 3 (Extractability)

*Proof.* The extractor $\mathsf{Ext}_{\mathsf{OR}}$ employs a standard forking strategy. It runs the adversary to obtain an initial accepting proof, then forks the adversary at its $\mathcal{H}_{\mathsf{fs}}$

query corresponding to the verifier's recomputable query point. By the Forking Lemma, with non-negligible probability, we obtain two accepting proofs with the same reconstructed commitment vector but different $\mathcal{H}_{\mathsf{fs}}$ outputs (and thus different XOR sums of challenges). This implies that for at least one branch $i$, the derived challenges $c_i = \mathcal{H}(\rho_i)$ must differ while the reconstructed commitment remains identical. Given the special soundness of the underlying branch protocol, the extractor can then use these two transcripts to output the witness $\omega_i$ for branch $i$.

# F Additional Concrete Cost Tables

Table 3: Concrete costs of RVRF[LaV] under the LaV parameter set of [18]. We use $|\sigma_{\mathsf{LaV}}| \approx 10.27\,\mathrm{KB}$, $|v| \approx 124\,\mathrm{B}$, and $|\rho| = 128$ bits (i.e., 16 bytes).

| $N$ | $\lvert\pi\rvert$ | $\lvert v\rvert + \lvert\pi\rvert$ | Prover work | Verifier work |
|---|---|---|---|---|
| 8 | $\approx 82\,\mathrm{KB}$ | $\approx 82\,\mathrm{KB}$ | $1 \times \mathsf{LaV.Eval} + 7 \times \mathsf{Sim}_{\mathsf{LaV}}$ | $8 \times \mathsf{LaV.Verify}$ |
| 16 | $\approx 165\,\mathrm{KB}$ | $\approx 165\,\mathrm{KB}$ | $1 \times \mathsf{LaV.Eval} + 15 \times \mathsf{Sim}_{\mathsf{LaV}}$ | $16 \times \mathsf{LaV.Verify}$ |
| 32 | $\approx 329\,\mathrm{KB}$ | $\approx 329\,\mathrm{KB}$ | $1 \times \mathsf{LaV.Eval} + 31 \times \mathsf{Sim}_{\mathsf{LaV}}$ | $32 \times \mathsf{LaV.Verify}$ |
| 64 | $\approx 658\,\mathrm{KB}$ | $\approx 658\,\mathrm{KB}$ | $1 \times \mathsf{LaV.Eval} + 63 \times \mathsf{Sim}_{\mathsf{LaV}}$ | $64 \times \mathsf{LaV.Verify}$ |
| 128 | $\approx 1.29\,\mathrm{MB}$ | $\approx 1.29\,\mathrm{MB}$ | $1 \times \mathsf{LaV.Eval} + 127 \times \mathsf{Sim}_{\mathsf{LaV}}$ | $128 \times \mathsf{LaV.Verify}$ |
| 256 | $\approx 2.57\,\mathrm{MB}$ | $\approx 2.57\,\mathrm{MB}$ | $1 \times \mathsf{LaV.Eval} + 255 \times \mathsf{Sim}_{\mathsf{LaV}}$ | $256 \times \mathsf{LaV.Verify}$ |

**Notes.** (1) We use $1\,\mathrm{KB} = 1024$ bytes. (2) The table treats the ring $R$ as public input. If $R$ must be transmitted, add $N \cdot |\mathsf{pk}| \approx N \cdot 5.81\,\mathrm{KB}$. (3) Verifier-side hashing includes $N$ calls to $H$, one call to $H_{\mathrm{FS}}$, and an XOR-glue check.

Table 4: Concrete costs of RVRF[LB-VRF] under the LB-VRF parameter set of [15]. We use $|\sigma_{\mathsf{LB\text{-}VRF}}| \approx 4.94\,\mathrm{KB}$, $|v| \approx 84\,\mathrm{B}$, and $|\rho| = 128$ bits. (We use $1\,\mathrm{KB} = 1024$ bytes.)

| $N$ | $\lvert\pi\rvert$ | $\lvert v\rvert + \lvert\pi\rvert$ | Prover work | Verifier work |
|---|---|---|---|---|
| 8 | $\approx 40\,\mathrm{KB}$ | $\approx 40\,\mathrm{KB}$ | $1 \times \mathsf{LB\text{-}VRF.Eval} + 7 \times \mathsf{Sim}_{\mathsf{LB\text{-}VRF}}$ | $8 \times \mathsf{LB\text{-}VRF.Verify}$ |
| 16 | $\approx 79\,\mathrm{KB}$ | $\approx 79\,\mathrm{KB}$ | $1 \times \mathsf{LB\text{-}VRF.Eval} + 15 \times \mathsf{Sim}_{\mathsf{LB\text{-}VRF}}$ | $16 \times \mathsf{LB\text{-}VRF.Verify}$ |
| 32 | $\approx 159\,\mathrm{KB}$ | $\approx 159\,\mathrm{KB}$ | $1 \times \mathsf{LB\text{-}VRF.Eval} + 31 \times \mathsf{Sim}_{\mathsf{LB\text{-}VRF}}$ | $32 \times \mathsf{LB\text{-}VRF.Verify}$ |
| 64 | $\approx 317\,\mathrm{KB}$ | $\approx 317\,\mathrm{KB}$ | $1 \times \mathsf{LB\text{-}VRF.Eval} + 63 \times \mathsf{Sim}_{\mathsf{LB\text{-}VRF}}$ | $64 \times \mathsf{LB\text{-}VRF.Verify}$ |
| 128 | $\approx 634\,\mathrm{KB}$ | $\approx 634\,\mathrm{KB}$ | $1 \times \mathsf{LB\text{-}VRF.Eval} + 127 \times \mathsf{Sim}_{\mathsf{LB\text{-}VRF}}$ | $128 \times \mathsf{LB\text{-}VRF.Verify}$ |
| 256 | $\approx 1.24\,\mathrm{MB}$ | $\approx 1.24\,\mathrm{MB}$ | $1 \times \mathsf{LB\text{-}VRF.Eval} + 255 \times \mathsf{Sim}_{\mathsf{LB\text{-}VRF}}$ | $256 \times \mathsf{LB\text{-}VRF.Verify}$ |

**Notes.** (1) The ring $R$ is treated as public input. If $R$ must be transmitted, add $N \cdot |\mathsf{pk}|$. (2) Multiply the ring-dependent costs by the repetition parameter if the OR layer is repeated (when applicable).