

Post-Quantum RANDAO

A Hash-Based Verifiable Random Function for Ethereum’s Consensus Layer

Aryaethn

Co-author of EIP-7998
arianari97@gmail.com

Draft – May 2026

Abstract

Ethereum’s RANDAO mechanism relies on BLS signatures to generate per-slot randomness for validator assignment and block proposal selection. EIP-7998 strengthened RANDAO by incorporating the previous epoch’s RANDAO mix into the signed data, transforming the `randao_reveal` into a Verifiable Random Function (VRF). However, both the underlying BLS cryptography and EIP-7998 are entirely broken by a cryptographically relevant quantum computer (CRQC): Shor’s algorithm recovers any validator’s BLS private key in polynomial time, enabling complete RANDAO manipulation. At milestone L^* of the Ethereum Foundation’s post-quantum roadmap, BLS keys are retired and replaced with the `leanSig` hash-based signature scheme. Without a post-quantum successor to EIP-7998, Ethereum will have no VRF-backed RANDAO at L^* .

We present a PRF-commitment VRF construction that replaces the BLS-based VRF with a construction based entirely on Poseidon1 over the KoalaBear field, the same hash primitive used by `leanSig` and `leanMultisig`. The VRF output is $y = \text{Poseidon}(vrf_sk \parallel x)$, where vrf_sk is a field element derived from the validator’s `leanSig` seed, and x encodes the current slot and the previous epoch’s RANDAO mix. Correctness of computation is proved to every consensus node by a WHIR-based zero-knowledge proof via the `leanMultisig` zkVM infrastructure; no new proof system or precompile is required.

We provide four formal security results. Correctness is unconditional. Computational uniqueness holds in the random oracle model (ROM) under Poseidon1 collision resistance and WHIR knowledge soundness. *Pseudorandomness holds in the quantum random oracle model (QROM) via Zhandry’s quantum PRF theorem [?], independent of the proof system’s QROM security.* Proof soundness holds in the ROM via Fiat-Shamir applied to WHIR. A fourth property, QROM proof soundness, remains an open problem and is identified as the primary target for future work.

We identify a critical quantum security gap in a compact single-element key design (31-bit key, providing only ≈ 15 bits of quantum security against Grover-based key recovery), and present a full-strength variant with $vrf_sk \in \mathbb{F}_p^8$ (248-bit key, providing ≈ 124 bits of quantum security matching `leanMultisig`’s stated security level). We provide a reference implementation in Rust using the `leanMultisig` zkVM and present benchmarks on Apple M2 hardware: proving time 57.9 ms, verification time 16.8 ms, and proof size 115 KB. The construction activates at milestone I^* in parallel with EIP-7998 and supersedes it at L^* , delivering quantum-resistant RANDAO entropy before BLS keys are retired.

Keywords: Verifiable Random Function, Post-Quantum Cryptography, Quantum Random Oracle Model, Poseidon, WHIR, Ethereum, RANDAO, `leanSig`, Hash-Based Cryptography.

Contents

1	Introduction	3
1.1	Ethereum’s RANDAO and the quantum threat	3
1.2	The Ethereum Foundation’s post-quantum roadmap	3
1.3	Our contribution	3
1.4	Organisation	4
2	Background	4
2.1	Ethereum’s RANDAO mechanism	4
2.2	The leanSig signature scheme	5
2.3	The leanMultisig zkVM	5
2.4	The EF PQ Roadmap	5
3	Preliminaries	6
3.1	Verifiable Random Functions	6
3.2	The Quantum Random Oracle Model	6
3.3	Poseidon1 over KoalaBear	7
3.4	WHIR and the Fiat-Shamir Transform	8
4	The PRF-Commitment VRF Construction	8
4.1	VRF Input Encoding	8
4.2	Construction I: Compact VRF	9
4.3	Construction II: Full-Strength VRF	10
5	Security Analysis	11
5.1	Correctness	11
5.2	Computational Uniqueness	11
5.3	Pseudorandomness in the QROM	12
5.4	Proof Soundness in the ROM	13
5.5	QROM Proof Soundness: An Open Problem	13
5.6	Concrete Security Analysis	14
6	Prior Art and Comparison	14
6.1	X-VRF (ESORICS 2022) and its break at FC 2024	14
6.2	XM-VRF (ePrint 2026/052)	15
6.3	MPC-in-the-Head Constructions (Li et al. 2021)	15
6.4	BLS-based VRF (EIP-7998)	15
6.5	Loquat (CRYPTO 2024)	16
7	Protocol Integration	16
7.1	Key Registration at Milestone I^*	16
7.2	BeaconBlockBody Changes	16
7.3	Three-Phase <code>process_randao</code>	16
7.3.1	Phase 0: Before I^* (EIP-7998 unchanged)	16
7.3.2	Phase 1: I^* to L^* (Hybrid)	17
7.3.3	Phase 2: From L^* (PQ only)	17
7.4	Mandatory Deployment	18
7.5	Relationship to EIP-7998	18

8	Implementation and Benchmarks	18
8.1	The zkDSL Circuit	18
8.2	Rust Crate API	19
8.3	Benchmark Results	20
8.4	Test Suite	20
9	Discussion	21
9.1	The QROM Gap and Future Work	21
9.2	Composability with RANDAO	21
9.3	Field Size and Output Entropy	21
9.4	Alternative Key Derivation	21
9.5	Proof Size and Block Overhead	22
10	Conclusion	22
A	Domain Separation Collision Analysis	23
B	Known-Answer Vectors	24
C	Full Poseidon1 Parameter Set	24

1 Introduction

1.1 Ethereum’s RANDAO and the quantum threat

Ethereum’s beacon chain uses RANDAO as its primary source of randomness for validator shuffling, committee selection, and block proposal assignment [?]. In each slot, the designated block proposer contributes a BLS signature over the current epoch number; the SHA2 hash of that signature is XORed into a running RANDAO accumulator. The security of RANDAO depends on the BLS signature’s pseudorandomness: an adversary who can choose or predict the proposer’s contribution can bias the validator assignment for future epochs.

EIP-7998 [?] strengthened RANDAO by modifying the signed data to include both the current slot number and the previous epoch’s RANDAO mix. This makes the contribution unpredictable even to the proposer until the slot boundary approaches, formally transforming the BLS signature into a VRF [?]. Uniqueness follows from BLS signature determinism; pseudorandomness follows from the Computational Diffie-Hellman assumption over BLS12-381.

However, both the original RANDAO scheme and EIP-7998 are completely broken in the quantum setting. Shor’s algorithm [?] solves the discrete logarithm problem on elliptic curves in polynomial time on a fault-tolerant quantum computer. As of March 2026, Google Quantum AI estimates that breaking a 256-bit elliptic-curve key requires approximately 1,200 logical qubits. Importantly, BLS public keys are already on-chain and observable by any adversary today: a future CRQC can recover any validator’s BLS private key from the beacon state, enabling the adversary to compute any validator’s `randao_reveal` in advance and manipulate RANDAO entirely.

1.2 The Ethereum Foundation’s post-quantum roadmap

The Ethereum Foundation’s PQ roadmap [?] proceeds through several milestones. At milestone I^* (targeting the Hegota hard fork, late 2026), validators register post-quantum public keys alongside their BLS keys. At milestone L^* , BLS attestations are replaced with leanSig hash-based signatures [?] and the leanMultisig zkVM aggregate proof is activated. BLS keys are retired at L^* .

This retirement creates an immediate gap: EIP-7998 depends on a BLS key that no longer exists at L^* . As Vitalik Buterin noted [?]: “*BLS-based RANDAO can easily be replaced with hash-based; in fact, hash-based was the original proposal.*” No prior work has filled this gap in a way that is both formally secure in the QROM and compatible with the leanSig infrastructure.

1.3 Our contribution

We present a post-quantum VRF construction for Ethereum’s RANDAO that satisfies the following design constraints simultaneously:

- **Hash-based.** The construction uses Poseidon1 over the KoalaBear field, the same primitive as leanSig and leanMultisig. No new hash function, precompile, or trusted setup is introduced.
- **QROM-secure pseudorandomness.** The VRF output is quantum-pseudorandom via Zhandry’s QPRF theorem, independent of proof-system security.
- **Succinct and verifiable.** Correctness of evaluation is proved by a WHIR zero-knowledge argument via the leanMultisig zkVM, requiring no modification to the aggregation circuit.
- **Protocol-compatible.** The construction activates at I^* in parallel with EIP-7998, delivers quantum-resistant entropy from that point, and supersedes EIP-7998 at L^* .

Technical contributions.

1. *Formal QROM security analysis.* We provide complete game-based security proofs for four VRF properties: unconditional correctness, ROM computational uniqueness, QROM pseudorandomness (our main theorem), and ROM proof soundness. We identify QROM proof soundness as an open problem, characterise its impact on the construction’s quantum security, and show that the pseudorandomness result is independent of this gap.
2. *Identification and resolution of a quantum security gap.* We observe that a compact design with $\text{vrf_sk} \in \mathbb{F}_p$ (31 bits) provides only ≈ 15 bits of quantum security against Grover’s key-recovery algorithm. We present a full-strength variant with $\text{vrf_sk} \in \mathbb{F}_p^8$ (248 bits) achieving ≈ 124 bits of quantum security.
3. *Reference implementation and benchmarks.* We implement the construction in Rust using the leanMultisig zkVM and measure proving time (57.9 ms), verification time (16.8 ms), and proof size (115 KB) on Apple M2 hardware.
4. *Protocol integration specification.* We specify the three-phase `process_randao` state machine governing activation at I^* , hybrid operation through L^* , and full post-quantum operation thereafter.

1.4 Organisation

Section 2 gives background on Ethereum’s RANDAO, the leanSig/leanMultisig ecosystem, and EIP-7998. Section 3 covers formal VRF definitions, the QROM, Poseidon1 over KoalaBear, and the WHIR proof system. Section 4 presents both the compact and full-strength VRF constructions. Section 5 contains the formal security proofs and the discussion of the QROM soundness gap. Section 6 surveys prior hash-based VRF work and explains why existing constructions are insufficient. Section 7 specifies the protocol integration. Section 8 presents the implementation and benchmarks. Section 9 discusses open problems and parameter recommendations. Section 10 concludes.

2 Background

2.1 Ethereum’s RANDAO mechanism

Ethereum’s beacon chain maintains a vector `randao_mixes[e]` for each epoch e , updated at each slot by the block proposer’s contribution. Under the current protocol, the block proposer for slot s in epoch e computes:

$$\text{randao_reveal} = \text{BLS.Sign}(sk_{\text{BLS}}, e),$$

and the beacon state transition applies:

$$\text{randao_mixes}[e] \leftarrow \text{randao_mixes}[e] \oplus \text{SHA2}(\text{randao_reveal}).$$

The RANDAO accumulator for epoch e is the XOR of 32 proposers’ contributions. Validator shuffling, committee selection, and block proposal assignment all derive from this accumulator.

EIP-7998. EIP-7998 [?] modifies the signed data to include the current slot and the previous epoch’s RANDAO mix:

$$\text{randao_reveal} = \text{BLS.Sign}(sk_{\text{BLS}}, \text{SSZ}(s, \text{randao_mixes}[e - 1])).$$

This prevents the proposer from precomputing contributions: the signed value depends on `randao_mixes[e - 1]`, which is not finalised until near the epoch boundary. Formally, this transforms the reveal into a VRF evaluation on input $(s, \text{randao_mixes}[e - 1])$, with uniqueness following from BLS determinism and pseudorandomness following from CDH over BLS12-381.

2.2 The leanSig signature scheme

leanSig [? ?] is a generalised XMSS scheme designed for post-quantum Ethereum. Its core primitive is WOTS+ with *incomparable encodings* [?], which prevent the chain-extension attacks that break X-VRF. The public key is the root of a Merkle tree built over 2^h WOTS+ public keys, with Poseidon1 as the tree hash.

All WOTS+ private keys are derived deterministically from a 32-byte seed via a pseudorandom key derivation function. The seed is the root of trust: knowledge of the seed determines the entire key hierarchy. leanSig is instantiated over the KoalaBear field \mathbb{F}_p with $p = 2^{31} - 2^{24} + 1$ and uses a 16-element Poseidon permutation with S-box degree $\alpha = 3$.

2.3 The leanMultisig zkVM

leanSig signatures are approximately 3,000 bytes each. Aggregating one million validator signatures per slot is infeasible without compression. leanMultisig [?] solves this by proving the validity of all signatures in a single STARK proof via a Cairo-inspired minimal zkVM (the *lean VM*).

The proof system is WHIR [?] with SuperSpartan and Logup, not a FRI-based STARK. WHIR is a multilinear interactive oracle proof (IOP) made non-interactive via the Fiat-Shamir transform. The security analysis holds in the ROM; QROM security is an acknowledged open problem shared by both leanMultisig and the present work.

leanMultisig provides ≈ 124 bits of provable security (Johnson bound plus degree-3 extension of KoalaBear). Achieving 128 bits would require digests wider than 8 KoalaBear elements and is noted as a future goal in the leanMultisig documentation.

The lean VM is programmed via a Python-like zkDSL. A zkDSL program is compiled to bytecode and proved via `lean_prover::prove_execution`, which takes the public input vector, an `ExecutionWitness` (containing private hints), and a WHIR configuration, and returns a serialised execution proof.

2.4 The EF PQ Roadmap

The roadmap defines three milestones relevant to this work.

*I** Validators register leanSig public keys alongside their existing BLS keys. BLS continues to be used for attestations.

*L** BLS attestations are replaced with leanSig attestations. The leanMultisig aggregate proof is activated on-chain. BLS keys are retired.

*M** Full post-quantum infrastructure including blob commitments and execution-layer signatures.

At *L**, EIP-7998 becomes inoperable: the BLS key it requires does not exist. A post-quantum VRF EIP must activate no later than *L**.

3 Preliminaries

3.1 Verifiable Random Functions

We recall the standard VRF definition [? ?].

Definition 3.1 (Verifiable Random Function). A VRF is a tuple of PPT algorithms $\text{VRF} = (\text{KeyGen}, \text{Eval}, \text{Prove}, \text{Verify})$:

- $\text{KeyGen}(1^\lambda) \rightarrow (vrf_sk, pk_vrf)$: generates a secret/public key pair.
- $\text{Eval}(vrf_sk, x) \rightarrow y \in \mathcal{Y}$: deterministically evaluates the VRF at input x .
- $\text{Prove}(vrf_sk, x) \rightarrow (y, \pi)$: produces an output together with a proof of correct evaluation.
- $\text{Verify}(pk_vrf, x, y, \pi) \rightarrow \{0, 1\}$: verifies the proof.

We require the following four properties.

Definition 3.2 (Correctness). For all $(vrf_sk, pk_vrf) \leftarrow \text{KeyGen}(1^\lambda)$ and all inputs x :

$$\text{Verify}(pk_vrf, x, y, \pi) = 1 \quad \text{where } (y, \pi) = \text{Prove}(vrf_sk, x).$$

Definition 3.3 (Computational Uniqueness). For all PPT adversaries \mathcal{A} , the probability that \mathcal{A} outputs a tuple $(pk_vrf, x, y, \pi, y', \pi')$ with $y \neq y'$ and $\text{Verify}(pk_vrf, x, y, \pi) = \text{Verify}(pk_vrf, x, y', \pi') = 1$ is negligible in λ .

Definition 3.4 (Pseudorandomness). Let $(vrf_sk, pk_vrf) \leftarrow \text{KeyGen}(1^\lambda)$. For any adversary \mathcal{A} that adaptively queries an Eval oracle $\mathcal{O}(vrf_sk, \cdot)$, and then outputs a challenge input x^* not previously queried, define:

$$\text{Adv}_{\text{VRF}}^{\text{VRF-PR}}(\mathcal{A}) = \left| \Pr \left[b' = b \mid b \xleftarrow{R} \{0, 1\}, y_0^* = \text{Eval}(vrf_sk, x^*), y_1^* \xleftarrow{R} \mathcal{Y} \right] - \frac{1}{2} \right|.$$

The VRF is *pseudorandom* if $\text{Adv}_{\text{VRF}}^{\text{VRF-PR}}(\mathcal{A}) = \text{negl}(\lambda)$ for all PPT \mathcal{A} . We extend this definition to QPT adversaries with quantum oracle access to \mathbf{H} for the QROM variant.

Definition 3.5 (Proof Soundness). For all PPT adversaries \mathcal{A} and all $(vrf_sk, pk_vrf) \leftarrow \text{KeyGen}(1^\lambda)$: the probability that \mathcal{A} outputs (x, y, π) with $\text{Verify}(pk_vrf, x, y, \pi) = 1$ but $y \neq \text{Eval}(vrf_sk, x)$ is negligible.

3.2 The Quantum Random Oracle Model

In the QROM [?], the hash function \mathbf{H} is modelled as a uniformly random function accessible to all parties in quantum superposition. Formally, an adversary making a quantum query to \mathbf{H} may query $\sum_x \alpha_x |x\rangle|0\rangle$ and receive back $\sum_x \alpha_x |x\rangle|\mathbf{H}(x)\rangle$. The adversary may make polynomially many such queries.

The key result we rely on is Zhandry's quantum PRF theorem [?].

Theorem 3.6 (Quantum PRF; Zhandry 2012 [?]). *Let $\mathbf{H}: \{0, 1\}^* \rightarrow \mathbb{F}_p$ be a uniformly random function (quantum random oracle). For $k \xleftarrow{R} \mathbb{F}_p^t$, define the PRF family $F_k(x) = \mathbf{H}(k \parallel x)$. For any QPT adversary \mathcal{A} making at most q quantum queries to \mathbf{H} :*

$$\text{Adv}^{\text{PRF}}(\mathcal{A}) \leq O\left(\frac{q^2}{|\mathbb{F}_p^t|}\right) = O\left(\frac{q^2}{p^t}\right).$$

For $t = 1$ (single field element key, $|\mathbb{F}_p| = p \approx 2^{31}$), this bound is $O(q^2/2^{31})$. For $t = 8$ (eight field element key, $|\mathbb{F}_p^8| \approx 2^{248}$), this bound is $O(q^2/2^{248})$. The implications for our two constructions are discussed in Section 5.

We also rely on the following standard QROM fact about Grover search.

Lemma 3.7 (Grover Search in the QROM). *Given an image $z = \mathbf{H}(k \parallel d)$ for a known domain tag d , a QPT adversary with quantum oracle access to \mathbf{H} can find k with constant success probability using $O\left(\sqrt{|\mathbb{F}_p^t|}\right) = O(p^{t/2})$ quantum queries to \mathbf{H} .*

For $t = 1$: $O(\sqrt{p}) = O(2^{15.5})$ queries – feasible on a modest CRQC.

For $t = 8$: $O(\sqrt{p^8}) = O(2^{124})$ queries – infeasible.

3.3 Poseidon1 over KoalaBear

We denote by `Poseidon` the Poseidon1 hash function [?] instantiated over the KoalaBear field \mathbb{F}_p with $p = 2^{31} - 2^{24} + 1$.

Field parameters. The KoalaBear prime $p = 2^{31} - 2^{24} + 1 = 0x7f000001$ has two-adicity $v_2(p - 1) = 24$ (i.e., $p - 1 = 2^{24} \cdot 127$), enabling efficient NTT-based polynomial arithmetic. This is the standard KoalaBear prime used by Plonky3 and leanMultisig.

Permutation. Poseidon uses a width-16 permutation $\text{Perm}: \mathbb{F}_p^{16} \rightarrow \mathbb{F}_p^{16}$ with:

- S-box: $x \mapsto x^3$ (cubemap, $\alpha = 3$). Bijectivity: $\gcd(3, p-1) = \gcd(3, 2^{24} \cdot 127) = 1$, since $\gcd(3, 2^{24}) = 1$ and $\gcd(3, 127) = 1$ (as $127 = 42 \cdot 3 + 1$).
- 8 full rounds (S-box applied to all 16 elements, full MDS matrix).
- 20 partial rounds (S-box applied to element 0 only, partial MDS matrix).
- Standard Plonky3 round constants and MDS matrix.

Compression. Define the Davies-Meyer compression function:

$$\text{Compress}: \mathbb{F}_p^{16} \rightarrow \mathbb{F}_p^8, \quad \text{Compress}(\mathbf{s}) = \text{Perm}(\mathbf{s})[0..8] + \mathbf{s}[0..8],$$

where addition is componentwise in \mathbb{F}_p and $[0..8]$ denotes the first 8 elements. Utility functions are provided in the `crates/utis` crate:

$$\begin{aligned} \text{poseidon16_compress}: \mathbb{F}_p^{16} &\rightarrow \mathbb{F}_p^8, \\ \text{poseidon16_compress_pair}: \mathbb{F}_p^8 \times \mathbb{F}_p^8 &\rightarrow \mathbb{F}_p^8. \end{aligned}$$

Domain separation. We use three domain-separated field elements as constants:

$$\begin{aligned} D_{\text{in}} &= p - 1 = 2,130,706,432 = 0x7f000000, \\ D_{\text{pk}} &= p - 2 = 2,130,706,431 = 0x7effffff, \\ D_{\text{der}} &= p - 3 = 2,130,706,430 = 0x7efffffe, \\ D_{\text{eval}} &= p - 4 = 2,130,706,429 = 0x7efffffd. \end{aligned}$$

These exceed $2^{24} - 1 = 16,777,215$, so they cannot arise from 3-byte packing of external byte data (Section 4.1). They do not collide with any leanSig tweak constants ($\{0, 1, 2\}$) or leanMultisig SNARK domain separators (max value $2,063,844,858 < D_{\text{eval}}$). Full collision analysis is in Appendix A.

3.4 WHIR and the Fiat-Shamir Transform

WHIR [?] is a Reed-Solomon proximity testing protocol based on multilinear polynomial commitments. `leanMultisig` uses WHIR together with `SuperSpartan` and `Logup` to produce an IOP for arbitrary zkVM execution.

Proof system. Let \mathcal{R} be an NP relation expressible as a zkDSL program. The lean VM compiles the program to bytecode, generates a multilinear execution trace over \mathbb{F}_p , and proves proximity to the multilinear extension of the constraint polynomial via WHIR. The IOP is made non-interactive via Fiat-Shamir, yielding a proof π that can be verified by any party holding the bytecode and public inputs.

Completeness and soundness. WHIR is complete: an honest prover with a valid witness always produces an accepting proof. Soundness in the ROM follows from the Fiat-Shamir theorem applied to WHIR as a public-coin IOP [?]: for a q -query adversary, the soundness error of the non-interactive argument is at most $q \cdot \epsilon_{\text{IOP}}$, where ϵ_{IOP} is WHIR’s per-round soundness error.

Knowledge soundness. WHIR additionally satisfies knowledge soundness [?]: given any proof that passes verification, there exists an efficient extractor that recovers a valid witness via rewinding. This is essential for our uniqueness proof (Theorem 5.2).

QROM gap. The QROM security of Fiat-Shamir applied to multi-round IOPs like WHIR is an open problem. The Don-Fehr-Majenz-Schaffner result [?] establishes QROM security for sigma protocols (3-message) with a polynomial security loss. Whether this extends to WHIR’s multi-round structure with the required extractability is not yet established. This gap affects proof soundness in the QROM; it does not affect pseudorandomness, which is proved independently of the proof system (see Section 5.3).

Security level. `leanMultisig` uses WHIR with `log_inv_rate = 2` (rate = 1/4), providing ≈ 124 bits of provable security via the Johnson bound. Our VRF implementation uses the same parameter.

4 The PRF-Commitment VRF Construction

We present two constructions. *Construction I* (compact) uses a single field element as the VRF secret key; it is the basis of our reference implementation and provides a clean presentation of the protocol. *Construction II* (full-strength) uses an eight-element key and is the recommended production construction, achieving quantum security that matches `leanMultisig`’s stated security level.

4.1 VRF Input Encoding

Both constructions share the same input encoding. The VRF input is the pair (s, r) where s is the current slot number (a `uint64`) and r is the previous epoch’s RANDAO mix (a 32-byte value).

Definition 4.1 (3-byte packing). Define $\text{pack}_3: \{0, 1\}^{24} \rightarrow \mathbb{F}_p$ as little-endian 3-byte to field-element conversion:

$$\text{pack}_3(b_0, b_1, b_2) = b_0 + 256 \cdot b_1 + 65536 \cdot b_2.$$

The maximum value is $2^{24} - 1 = 16,777,215 < p$, so no range check is needed. The function is injective.

Definition 4.2 (VRF input encoding). Let $s \in [0, 2^{64})$ be a slot number with little-endian byte representation $s = (s_0, \dots, s_7)$, and $r = (r_0, \dots, r_{31})$ be a 32-byte RANDAO mix. Define $\text{encode}(s, r) \in \mathbb{F}_p^{15}$ as the vector:

$$\text{encode}(s, r) = (\text{D}_{\text{in}}, \text{pack}_3(s_0, s_1, s_2), \text{pack}_3(s_3, s_4, s_5), \text{pack}_3(s_6, s_7, 0), \\ \text{pack}_3(r_0, r_1, r_2), \dots, \text{pack}_3(r_{27}, r_{28}, r_{29}), \text{pack}_3(r_{30}, r_{31}, 0)).$$

Explicitly, the vector has 15 elements: 1 domain tag, 3 slot elements, and 11 mix elements.

Proposition 4.3 (Injectivity). $\text{encode}(s, r) = \text{encode}(s', r')$ implies $s = s'$ and $r = r'$.

Proof. D_{in} occupies position 0 uniquely. Positions 1–3 encode s via pack_3 : the bit-width of s is 64 bits packed into $3 \times 24 = 72$ bits with the last group zero-padded, so recovery is injective. Positions 4–14 encode r similarly ($11 \times 24 = 264 > 256$, with the last group zero-padded). \square

Remark 4.4. The 15-element structure is chosen so that $[\text{vrf_sk}, \text{encode}(s, r)] \in \mathbb{F}_p^{16}$ fills the Poseidon state exactly (Construction I). Construction II uses a different grouping.

4.2 Construction I: Compact VRF

Construction 4.5 (Compact VRF). Let \mathbb{F}_p be the KoalaBear field and $\text{Compress}: \mathbb{F}_p^{16} \rightarrow \mathbb{F}_p^8$ the Poseidon1 compression function. Write $\text{Compress}(\mathbf{s})[0]$ for the first output element.

Key derivation. Let $\text{seed} \in \{0, 1\}^{32}$ be the leanSig seed. Pack seed into 11 field elements via pack_3 : $\text{seed}_{\mathbb{F}_p} = (\text{pack}_3(\text{seed}[0..3]), \dots, \text{pack}_3(\text{seed}[30..32])) \in \mathbb{F}_p^{11}$. Then:

$$\text{vrf_sk}^{(1)} = \text{Compress}([\text{seed}_{\mathbb{F}_p}, \text{D}_{\text{der}}, \mathbf{0}^4])[0] \in \mathbb{F}_p.$$

Key commitment.

$$\text{pk_vrf}^{(1)} = \text{Compress}([\text{vrf_sk}^{(1)}, \text{D}_{\text{pk}}, \mathbf{0}^{14}])[0] \in \mathbb{F}_p.$$

The validator registers $\text{pk_vrf}^{(1)}$ on-chain at milestone I^* .

Evaluation. For slot s and previous RANDAO mix r :

$$y^{(1)} = \text{Compress}([\text{vrf_sk}^{(1)}, \text{encode}(s, r)])[0] \in \mathbb{F}_p.$$

Proving. The proof is a WHIR argument for the relation:

$$\mathcal{R}_{\text{VRF}}^{(1)} = \left\{ \left(\underbrace{\left(\text{pk_vrf}^{(1)}, \text{encode}(s, r), y^{(1)} \right)}_{17\text{-element public input}}; \underbrace{\text{vrf_sk}^{(1)}}_{\text{private}} \right) : \begin{array}{l} \text{Compress}([\text{vrf_sk}^{(1)}, \text{D}_{\text{pk}}, \mathbf{0}^{14}])[0] = \text{pk_vrf}^{(1)} \\ \text{Compress}([\text{vrf_sk}^{(1)}, \text{encode}(s, r)])[0] = y^{(1)} \end{array} \right\}$$

The circuit consists of exactly two Compress calls.

Verification. $\text{Verify}(\text{pk_vrf}^{(1)}, s, r, y^{(1)}, \pi)$ checks the WHIR proof against the 17-element public input vector.

Security remark. Construction I has a critical quantum security limitation. Since $\text{vrf_sk}^{(1)} \in \mathbb{F}_p$ with $|\mathbb{F}_p| = p \approx 2^{31}$, Lemma 3.7 implies that a quantum adversary with oracle access to \mathbf{H} can recover $\text{vrf_sk}^{(1)}$ from $\text{pk_vrf}^{(1)}$ in $O(2^{15.5})$ quantum evaluations of `Compress`. This provides only ≈ 15 bits of quantum security against key recovery – insufficient for any serious post-quantum deployment. Construction I is suitable as a proof of concept and for parameter benchmarking; Construction II (Construction 4.6) is recommended for production use.

4.3 Construction II: Full-Strength VRF

Construction 4.6 (Full-Strength VRF). Let notation be as in Construction I.

Key derivation. Using the same seed packing as Construction I:

$$\text{vrf_sk}^{(2)} = \text{Compress}([\text{seed}_{\mathbb{F}_p}, \mathbf{D}_{\text{der}}, \mathbf{0}^4]) \in \mathbb{F}_p^8.$$

The full 8-element output is retained as the secret key (248 bits).

Key commitment.

$$\text{pk_vrf}^{(2)} = \text{Compress}([\text{vrf_sk}^{(2)}, \mathbf{D}_{\text{pk}}, \mathbf{0}^7]) \in \mathbb{F}_p^8.$$

The validator registers $\text{pk_vrf}^{(2)}$ on-chain as a 32-byte value (8 field elements \times 4 bytes each).

Evaluation. Since $\text{vrf_sk}^{(2)} \in \mathbb{F}_p^8$ and $\text{encode}(s, r) \in \mathbb{F}_p^{15}$, a two-phase construction is used to accommodate 23 input elements within the width-16 Poseidon state:

$$\mathbf{m} = \text{Compress}([\text{encode}(s, r), \mathbf{D}_{\text{eval}}]) \in \mathbb{F}_p^8, \quad (1)$$

$$y^{(2)} = \text{Compress}([\text{vrf_sk}^{(2)}, \mathbf{m}]) \in \mathbb{F}_p^8. \quad (2)$$

The vector $[\text{encode}(s, r), \mathbf{D}_{\text{eval}}] \in \mathbb{F}_p^{16}$ fills the Poseidon state exactly (15 input elements plus one domain tag). The final output $y^{(2)} \in \mathbb{F}_p^8$ is serialised as a 32-byte value.

Proving. The proof is a WHIR argument for:

$$\mathcal{R}_{\text{VRF}}^{(2)} = \left\{ \left(\underbrace{\left(\text{pk_vrf}^{(2)}, \text{encode}(s, r), y^{(2)} \right)}_{\text{31-element public input}}; \underbrace{\left(\text{vrf_sk}^{(2)}, \mathbf{m} \right)}_{\text{private (16 elements)}} \right) : \begin{array}{l} \text{Compress}([\text{vrf_sk}^{(2)}, \mathbf{D}_{\text{pk}}, \mathbf{0}^7]) = \text{pk_vrf}^{(2)} \\ \text{Compress}([\text{encode}(s, r), \mathbf{D}_{\text{eval}}]) = \mathbf{m} \\ \text{Compress}([\text{vrf_sk}^{(2)}, \mathbf{m}]) = y^{(2)} \end{array} \right.$$

The circuit consists of three `Compress` calls.

Verification. $\text{Verify}(\text{pk_vrf}^{(2)}, s, r, y^{(2)}, \pi)$ checks the WHIR proof against the 31-element public input vector.

Remark 4.7 (Protocol serialisation). For both constructions, the VRF output and public key are serialised as `Bytes32` on the beacon chain using little-endian 4-byte encoding of each KoalaBear element, zero-padded to 32 bytes. For Construction I (1 element), bytes 4–31 are zero. For Construction II (8 elements), all 32 bytes are used. Construction II is therefore backward-compatible with the 32-byte `Bytes32` type.

Remark 4.8 (Relationship to Construction I). Construction II reduces to Construction I when the key is truncated to its first element and the output is similarly truncated. The security proofs in Section 5 are stated for Construction II (the stronger variant); all proofs apply to Construction I with $t = 1$, with the correspondingly weaker security bound noted explicitly.

5 Security Analysis

We state and prove four security theorems. Proofs are given in the QROM or ROM as appropriate. The parameter t denotes the key length in field elements: $t = 1$ for Construction I and $t = 8$ for Construction II.

5.1 Correctness

Theorem 5.1 (Correctness). *Constructions I and II satisfy correctness unconditionally.*

Proof. Let (vrf_sk, pk_vrf) be produced by **KeyGen** and let $(y, \pi) = \text{Prove}(vrf_sk, s, r)$. The proof π is a WHIR proof for a valid witness (vrf_sk, \mathbf{m}) and a statement that the prover evaluated **Compress** correctly. By WHIR completeness, an honest prover with a valid witness always produces a proof that the honest verifier accepts. Therefore $\text{Verify}(pk_vrf, s, r, y, \pi) = 1$. \square

5.2 Computational Uniqueness

Theorem 5.2 (Computational Uniqueness, ROM). *Under Poseidon1 collision resistance and WHIR knowledge soundness in the ROM, both constructions satisfy computational uniqueness (Definition 3.3). Specifically, for any PPT adversary \mathcal{A} making at most q queries to \mathbf{H} , the uniqueness advantage satisfies:*

$$\mathbf{Adv}^{\text{uniq}}(\mathcal{A}) \leq \mathbf{Adv}_{\text{Poseidon}}^{\text{coll}}(\mathcal{B}) + \frac{q}{2^\lambda},$$

where \mathcal{B} is a PPT reduction against Poseidon1 collision resistance and $q/2^\lambda$ accounts for the Fiat-Shamir extraction overhead.

Proof. Let \mathcal{A} produce $(pk_vrf, x, y, \pi, y', \pi')$ with $y \neq y'$ and both proofs verifying.

Step 1: Knowledge extraction. Apply the WHIR knowledge extractor to π : since $\text{Verify}(pk_vrf, x, y, \pi) = 1$ and WHIR has knowledge soundness in the ROM, the extractor runs the Fiat-Shamir rewinding argument to recover a witness (vrf_sk, \mathbf{m}) such that:

$$\text{Compress}([vrf_sk, D_{pk}, \mathbf{0}^*]) = pk_vrf \tag{3}$$

$$\text{Compress}([vrf_sk, x]) = y \quad (\text{Construction I, or via two calls in Construction II}). \tag{4}$$

Similarly, from π' extract witness (vrf_sk', \mathbf{m}') satisfying:

$$\text{Compress}([vrf_sk', D_{pk}, \mathbf{0}^*]) = pk_vrf \tag{5}$$

$$\text{Compress}([vrf_sk', x]) = y'. \tag{6}$$

The extraction succeeds except with probability $q/2^\lambda$ (the Fiat-Shamir knowledge extraction error in the ROM, where q is the total number of random oracle queries).

Step 2: Case analysis.

Case 1: $vrf_sk = vrf_sk'$. From (4) and (6): $y = \text{Compress}([vrf_sk, x])[0] = \text{Compress}([vrf_sk', x])[0] = y'$. This contradicts $y \neq y'$.

Case 2: $vrf_sk \neq vrf_sk'$. From (3) and (5):

$$\text{Compress}([vrf_sk, D_{pk}, \mathbf{0}^*]) = pk_vrf = \text{Compress}([vrf_sk', D_{pk}, \mathbf{0}^*]).$$

Since $vrf_sk \neq vrf_sk'$, the inputs to **Compress** are distinct. This is a collision in Poseidon1, which \mathcal{B} can output.

Conclusion. Since Case 1 is impossible, any adversary achieving uniqueness advantage $\varepsilon > q/2^\lambda$ can be turned into a collision-finding adversary for Poseidon1 with advantage $\varepsilon - q/2^\lambda$. Under Poseidon1 collision resistance, $\varepsilon = \text{negl}(\lambda)$. \square

On full vs. computational uniqueness. Full (information-theoretic) uniqueness would require that no adversary of any computational power can find two valid proofs for different outputs. This cannot hold here: hash collisions exist in principle (Poseidon1 is not injective over all possible inputs). Computational uniqueness is the standard requirement for VRF composability in cryptographic protocols and is sufficient for use in RANDAO.

5.3 Pseudorandomness in the QROM

This is our main theorem. Pseudorandomness holds in the QROM and is independent of the proof system's quantum security.

Theorem 5.3 (QROM Pseudorandomness). *Assume Poseidon1 is a quantum random oracle. For any QPT adversary \mathcal{A} making at most q quantum queries to \mathbf{H} , the pseudorandomness advantage of Construction II (key $\text{vrf_sk} \in \mathbb{F}_p^8$) satisfies:*

$$\text{Adv}^{\text{VRF-PR}}(\mathcal{A}) \leq O\left(\frac{q^2}{p^8}\right) = O\left(\frac{q^2}{2^{248}}\right).$$

For Construction I (key $\text{vrf_sk} \in \mathbb{F}_p$), the bound is $O(q^2/p) = O(q^2/2^{31})$.

Proof. We prove the result for Construction II; the argument for Construction I is identical with $t = 1$.

Setup. Fix any QPT adversary \mathcal{A} in the VRF pseudorandomness game. The challenger samples $\text{vrf_sk} \xleftarrow{R} \mathbb{F}_p^8$, computes $\text{pk_vrf} = \text{Compress}([\text{vrf_sk}, \mathbf{D}_{\text{pk}}, \mathbf{0}^7])$, and gives pk_vrf to \mathcal{A} . \mathcal{A} makes Eval queries (obtaining VRF outputs at chosen inputs) and eventually outputs a challenge input x^* and a guess b' .

Step 1: Key uniformity. In the key generation of Construction II, vrf_sk is derived as $\text{vrf_sk} = \text{Compress}([\text{seed}_{\mathbb{F}_p}, \mathbf{D}_{\text{der}}, \mathbf{0}^4])$ for a uniformly random $\text{seed} \in \{0, 1\}^{32}$. In the QROM, Compress is a random function. The input $[\text{seed}_{\mathbb{F}_p}, \mathbf{D}_{\text{der}}, \mathbf{0}^4]$ is distinct from any other input the adversary queries (since seed is never revealed and $\mathbf{D}_{\text{der}} = p - 3$ is domain-separated). Therefore, the QROM output at this input is uniformly distributed in \mathbb{F}_p^8 , independent of all other oracle evaluations. Consequently, vrf_sk is effectively uniform in \mathbb{F}_p^8 from \mathcal{A} 's perspective.

Step 2: Reduction to QPRF. Define the PRF family $F_k(x) = \text{Compress}([k, x])$ for $k \in \mathbb{F}_p^8$. By Theorem 3.6 (Zhandry 2012), F_k is a quantum PRF with advantage $O(q^2/p^8)$ against any QPT adversary making q quantum queries.

We now construct a QPT reduction \mathcal{B} against F_k that simulates the VRF pseudorandomness game for \mathcal{A} :

1. \mathcal{B} receives $k \in \mathbb{F}_p^8$ (the PRF key, which may be real or replaced by a random function).
2. \mathcal{B} sets $\text{pk_vrf} \leftarrow \text{Compress}([k, \mathbf{D}_{\text{pk}}, \mathbf{0}^7])$ (one oracle call) and sends pk_vrf to \mathcal{A} .
3. For each Eval query x_i from \mathcal{A} : \mathcal{B} computes $\mathbf{m}_i = \text{Compress}([\text{encode}(s_i, r_i), \mathbf{D}_{\text{eval}}])$ (one oracle call, independent of k) and then $y_i = \text{Compress}([k, \mathbf{m}_i]) = F_k(\mathbf{m}_i)$ (one PRF query). \mathcal{B} returns y_i .
4. When \mathcal{A} outputs challenge (s^*, r^*) , \mathcal{B} computes $\mathbf{m}^* = \text{Compress}([\text{encode}(s^*, r^*), \mathbf{D}_{\text{eval}}])$ and queries its challenge: PRF value at \mathbf{m}^* vs. random.
5. \mathcal{B} forwards the challenger's response y^* to \mathcal{A} and outputs whatever \mathcal{A} outputs.

Step 3: Correctness of simulation. The simulation is perfect: the distribution of pk_vrf , Eval responses, and the challenge value y^* are identical to those in the real VRF pseudorandomness game. This holds because:

- The domain tag D_{pk} is distinct from any \mathbf{m}_i or \mathbf{m}^* (since $D_{pk} = p - 2$ is a single field element, whereas \mathbf{m} values are 8-element vectors – the inputs to F_k are structurally distinct).
- The inner hash $\mathbf{m} = \text{Compress}([\text{encode}(\cdot), D_{\text{eval}}])$ is computed independently of k ; domain separation (via D_{eval}) ensures these evaluations do not collide with the key-derivation or pk-commitment calls.
- No information about k leaks to \mathcal{A} beyond the PRF evaluations – this is exactly the QPRF game.

Step 4: Conclusion. If \mathcal{A} achieves VRF pseudorandomness advantage ε , then \mathcal{B} achieves QPRF advantage ε against F_k . By Theorem 3.6, $\varepsilon \leq O(q^2/p^8) = O(q^2/2^{248})$. \square

Remark 5.4 (Independence from proof soundness). Theorem 5.3 holds regardless of whether QROM proof soundness is established. The pseudorandomness game does not require the adversary to produce a valid proof; it only asks whether the adversary can distinguish the VRF output from random. A quantum adversary that can forge a VRF proof for an incorrect output y' (exploiting the QROM soundness gap) would produce a y' that is still pseudorandom and uncontrollable – because y' was chosen by the adversary, not derived from F_k . Therefore, even in the presence of the QROM soundness gap, RANDAO’s pseudorandomness guarantee is preserved.

5.4 Proof Soundness in the ROM

Theorem 5.5 (Proof Soundness, ROM). *Under WHIR soundness and the Fiat-Shamir transform in the ROM, both constructions satisfy proof soundness. Specifically, for any PPT adversary \mathcal{A} making at most q queries to \mathbf{H} , the soundness advantage satisfies:*

$$\Pr[\text{Verify}(pk_vrf, s, r, y, \pi) = 1 \wedge y \neq \text{Eval}(vrf_sk, s, r)] \leq q \cdot \varepsilon_{\text{IOP}} + \text{negl}(\lambda),$$

where ε_{IOP} is WHIR’s per-round IOP soundness error.

Proof. The VRF proof π is a non-interactive WHIR proof for the relation \mathcal{R}_{VRF} . A valid proof with wrong output $y \neq \text{Eval}(vrf_sk, s, r)$ would require the adversary to produce a WHIR proof for a false statement (one for which no valid witness exists). By the Fiat-Shamir theorem applied to WHIR as a public-coin IOP [?], the probability of producing a verifying proof for a false statement is at most $q \cdot \varepsilon_{\text{IOP}}$ in the ROM. \square

5.5 QROM Proof Soundness: An Open Problem

Open Problem 1 (QROM Proof Soundness). Establish Theorem 5.5 in the QROM, i.e., show that no QPT adversary with quantum oracle access to \mathbf{H} can produce a verifying VRF proof for an incorrect output.

The obstruction is the QROM security of Fiat-Shamir applied to multi-round IOPs. The result of Don, Fehr, Majenz, and Schaffner [?] shows that Fiat-Shamir is sound in the QROM for 3-message sigma protocols with a polynomial security loss. The Liu-Zhandry result [?] characterises the optimal security loss for Fiat-Shamir in the QROM. Whether these results extend to multi-round IOPs such as WHIR with the extractability properties required for our setting is an open question shared with the leanMultisig project itself, which explicitly flags the same gap in its documentation.

Table 1: Security comparison summary

Property	Model	Construction I	Construction II
Correctness	Unconditional	✓	✓
Computational uniqueness	ROM	✓	✓
Pseudorandomness	QROM	≈ 11 bits (weak)	≈ 128 bits
Key recovery resistance	Quantum	≈ 15 bits	≈ 124 bits
Proof soundness	ROM	✓	✓
Proof soundness	QROM	Open	Open

Impact analysis. The practical impact of the QROM soundness gap is limited by Remark 5.4: a quantum adversary who forges a VRF proof for an incorrect y' cannot control the value of y' – it is not a valid PRF output under the adversary’s key, and is therefore pseudorandom from the perspective of the RANDAO accumulator. The adversary gains the ability to insert “phantom” randomness into RANDAO, but not to predict or choose its value. This is a weaker attack than full RANDAO manipulation and may be tolerable depending on the application’s threat model.

5.6 Concrete Security Analysis

Key security. For Construction II with $vrf_sk \in \mathbb{F}_p^8$:

- *Grover key recovery*: $O(p^4) = O(2^{124})$ quantum evaluations of `Compress`. Infeasible.
- *Zhandry QPRF*: advantage $O(q^2/p^8)$. For $q = 2^{60}$ (a generous bound): $O(2^{120}/2^{248}) = O(2^{-128})$. Negligible.

For Construction I with $vrf_sk \in \mathbb{F}_p$:

- *Grover key recovery*: $O(p^{1/2}) = O(2^{15.5}) \approx 46,340$ quantum evaluations. **Feasible on any CRQC.** Construction I is **not post-quantum secure**.
- *Zhandry QPRF*: advantage $O(q^2/p)$. For $q = 2^{10}$: $O(2^{20}/2^{31}) = O(2^{-11})$. Provides only ≈ 11 bits of QPRF security for an adversary limited to 2^{10} queries.

Proof system security. `leanMultisig` claims ≈ 124 bits of provable security at `log_inv_rate = 2` via the Johnson bound for WHIR’s Reed-Solomon proximity argument. The VRF proof is produced and verified using the same parameters.

Recommendation. *Only Construction II should be used in production.* Construction I is presented for clarity of exposition and for benchmarking; the reference implementation uses Construction I internals (single-element key), but the production EIP specification will adopt Construction II. The implementation changes required are localised to key derivation (retain 8 output elements) and proof generation (use 3 instead of 2 `Compress` calls).

6 Prior Art and Comparison

6.1 X-VRF (ESORICS 2022) and its break at FC 2024

Buser, Phan, Steinfeld, and Sakzad [?] proposed X-VRF, the first post-quantum VRF based on symmetric primitives. X-VRF defines the output as $y = H(\sigma, x)$ where σ is the XMSS

signature on input x , and uses σ itself as the proof. Uniqueness is claimed on the grounds that XMSS is a unique signature scheme.

At Financial Cryptography 2024 [?], this uniqueness claim was shown to be false. WOTS+ signing maps a message to a vector of chain positions (b_1, \dots, b_l) and produces signature elements $\sigma_j = H^{b_j}(sk_j)$. An adversary observing σ_j can extend chain j to any position $b'_j > b_j$ by applying H repeatedly. This yields a valid signature for a different message (one whose encoding assigns higher chain positions), breaking XMSS uniqueness at the proof level.

Relevance to our work. The X-VRF break identifies a fundamental incompatibility between WOTS+ chain structure and VRF uniqueness. Our PRF-commitment construction is immune by design: $y = \text{Compress}([vrf_sk, x])$ involves no hash chain and no XMSS signing. Uniqueness reduces to collision resistance of Poseidon1 (a standard assumption), not to XMSS uniqueness (which is false).

6.2 XM-VRF (ePrint 2026/052)

Ghosh, Dutta, and Mukhopadhyay [?] propose a multi-layer XMSS-based VRF with key updatability. Their construction improves on X-VRF’s key generation cost by using a hierarchical XMSS tree (HyperTree-style) and supports multiple evaluations per long-term key. However, XM-VRF:

- Does not address the X-VRF uniqueness break at the proof level; the uniqueness argument remains implicitly tied to XMSS.
- Is not compatible with leanSig or leanMultisig: it uses standard XMSS, not incomparable encodings or Poseidon1.
- Produces an XMSS signature as the proof, not a succinct STARK; the proof size is the full signature size ($\approx 3,000$ bytes).
- Has not been benchmarked against Ethereum’s 12-second slot timing.

6.3 MPC-in-the-Head Constructions (Li et al. 2021)

Li, Tan, Szalachowski, Sharma, and Zhou [?] proposed a VRF using ZKBoo/ZKBoo++ to prove correct evaluation of a symmetric-key PRF. Their construction has correct VRF uniqueness (PRF determinism, not signature uniqueness) but is not succinct: ZKBoo proofs are linear in circuit size (hundreds of kilobytes to megabytes for a single PRF evaluation) and are not aggregatable by leanMultisig.

6.4 BLS-based VRF (EIP-7998)

EIP-7998’s BLS-based VRF provides uniqueness from BLS determinism and pseudorandomness from CDH over BLS12-381. It has no post-quantum security: BLS12-381 is broken by Shor’s algorithm. Concretely, BLS public keys are on-chain today; a future CRQC can recover any validator’s BLS private key and compute arbitrary RANDAO contributions.

Our construction strictly dominates EIP-7998: it achieves QROM pseudorandomness (which EIP-7998 cannot), has computational uniqueness under a cleaner assumption (Poseidon1 collision resistance vs. CDH), and is fully compatible with leanSig.

Table 2: Comparison of hash-based VRF constructions

Construction	Uniqueness basis	Succinct proof	leanSig-compat	PQ-secure	Status
X-VRF [?]	XMSS uniqueness	No	No	No	Broken [?]
XM-VRF [?]	XMSS uniqueness	No	No	Suspect	Unproven
Li et al. [?]	PRF determinism	No (ZKBoo)	No	Yes	Non-succinct
EIP-7998 [?]	BLS determinism	Yes (BLS sig)	N/A	No	Current
This work (C-I)	PRF determinism	Yes (WHIR)	Yes	Weak	Proposed
This work (C-II)	PRF determinism	Yes (WHIR)	Yes	Yes	Proposed

6.5 Loquat (CRYPTO 2024)

Loquat [?] uses the Legendre PRF to construct SNARK-friendly post-quantum signatures. While the Legendre PRF is STARK-efficient, it is not compatible with leanSig’s Poseidon1 infrastructure, and its post-quantum security relies on a less mature analysis than Poseidon1. It is not applicable to our setting.

7 Protocol Integration

7.1 Key Registration at Milestone I^*

At milestone I^* , each validator v with leanSig seed seed_v derives:

$$\text{vrf_sk}_v = \text{Compress}([\text{seed}_{\mathbb{F}_p}^{(v)}, D_{\text{der}}, \mathbf{0}^4]) \in \mathbb{F}_p^8,$$

and registers $\text{pk_vrf}_v = \text{Compress}([\text{vrf_sk}_v, D_{\text{pk}}, \mathbf{0}^7]) \in \mathbb{F}_p^8$ as a new field in the validator record. This requires no separate key ceremony: vrf_sk_v is deterministically derived from the same seed already being registered at I^* . The validator record gains exactly one new 32-byte field: `pk_vrf`.

7.2 BeaconBlockBody Changes

From milestone I^* , the `BeaconBlockBody` includes two new fields:

```

1 class BeaconBlockBody(Container):
2     # ... existing fields ...
3     randao_reveal: BLSSignature           # retained until L*
4     # NEW from I*:
5     pq_vrf_output: Bytes32                # y serialised as Bytes32
6     pq_vrf_proof: VRFProof                # WHIR execution proof

```

Listing 1: BeaconBlockBody additions (from I^*)

At L^* , `randao_reveal` is removed and only the PQ VRF fields remain.

7.3 Three-Phase process_randao

The `process_randao` function operates in three phases determined by the current epoch relative to the I^* and L^* activation epochs.

7.3.1 Phase 0: Before I^* (EIP-7998 unchanged)

```

1 def process_randao(state, body):
2     epoch = get_current_epoch(state)
3     proposer = state.validators[get_beacon_proposer_index(state)]
4     randao_data = RandaoRevealData(
5         slot=state.slot,
6         prev_epoch_randao_mix=get_randao_mix(state, epoch - 1))
7     assert bls.Verify(proposer.bls_pubkey,
8         compute_signing_root(randao_data, DOMAIN_RANDAO),
9         body.randao_reveal)
10    mix = xor(get_randao_mix(state, epoch), hash(body.randao_reveal))
11    state.randao_mixes[epoch % EPOCHS_PER_HISTORICAL_VECTOR] = mix

```

Listing 2: process_randao: Phase 0

7.3.2 Phase 1: I^* to L^* (Hybrid)

```

1 def process_randao(state, body):
2     epoch = get_current_epoch(state)
3     proposer = state.validators[get_beacon_proposer_index(state)]
4
5     # BLS path (EIP-7998, runs until  $L^*$ )
6     randao_data = RandaoRevealData(
7         slot=state.slot,
8         prev_epoch_randao_mix=get_randao_mix(state, epoch - 1))
9     assert bls.Verify(proposer.bls_pubkey,
10        compute_signing_root(randao_data, DOMAIN_RANDAO),
11        body.randao_reveal)
12    bls_contribution = hash(body.randao_reveal)           # Bytes32
13
14    # PQ VRF path (this EIP, active from  $I^*$ )
15    x = encode_vrf_input(state.slot, get_randao_mix(state, epoch - 1)
16        )
17    assert pq_vrf_verify(proposer.pk_vrf, x,
18        body.pq_vrf_output, body.pq_vrf_proof)
19    pq_contribution = body.pq_vrf_output                 # Bytes32
20
21    # XOR both contributions
22    mix = xor(get_randao_mix(state, epoch),
23        xor(bls_contribution, pq_contribution))
24    state.randao_mixes[epoch % EPOCHS_PER_HISTORICAL_VECTOR] = mix

```

Listing 3: process_randao: Phase 1 (hybrid)

7.3.3 Phase 2: From L^* (PQ only)

```

1 def process_randao(state, body):
2     epoch = get_current_epoch(state)
3     proposer = state.validators[get_beacon_proposer_index(state)]
4     x = encode_vrf_input(state.slot, get_randao_mix(state, epoch - 1)
5         )
6     assert pq_vrf_verify(proposer.pk_vrf, x,
7         body.pq_vrf_output, body.pq_vrf_proof)
8     mix = xor(get_randao_mix(state, epoch), body.pq_vrf_output)
9     state.randao_mixes[epoch % EPOCHS_PER_HISTORICAL_VECTOR] = mix

```

Listing 4: process_randao: Phase 2 (post-quantum)

Security of the hybrid phase. During Phase 1 (from I^* to L^*), both VRF contributions are XORed into the RANDAO mix. The security argument is: if either contribution is pseudorandom (in any model), the XOR of the two is pseudorandom.

- A classical adversary who breaks CDH can bias the BLS contribution but not the PQ contribution; RANDAO remains pseudorandom from the PQ contribution alone.
- A quantum adversary (CRQC) can recover BLS keys and control the BLS contribution, but the PQ VRF contribution is QROM-pseudorandom (Theorem 5.3); RANDAO remains quantum-resistant.

Phase 1 therefore delivers quantum-resistant RANDAO entropy from I^* , not just at L^* .

7.4 Mandatory Deployment

The PQ VRF fields (`pq_vrf_output`, `pq_vrf_proof`) are mandatory in `BeaconBlockBody` from I^* onwards with no grace period. A block missing either field is invalid. Consensus client teams must implement the VRF before the I^* activation epoch.

7.5 Relationship to EIP-7998

This EIP does not break EIP-7998. EIP-7998 operates unmodified from its activation until I^* . From I^* to L^* , both EIPs operate in parallel. At L^* , the PQ VRF EIP supersedes EIP-7998. The EIP header declaration is:

Supersedes EIP-7998 at milestone L^ . Compatible with EIP-7998 from I^* to L^* .*

8 Implementation and Benchmarks

8.1 The zkDSL Circuit

The VRF proof is expressed as a zkDSL program compiled to lean VM bytecode. The circuit (Construction I) in the zkDSL notation is:

```
1 from snark_lib import *
2
3 # Public input layout (17 KoalaBear field elements):
4 #   mem[0]      = pk_vrf
5 #   mem[1..16] = encode_vrf_input(slot, randao_mix)
6 #   mem[16]    = y
7
8 DOMAIN_VRF_PK_CONST = 2130706431 # p - 2
9
10 def main():
11     # Private witness: vrf_sk (1 field element)
12     vrf_sk_buf = Array(1)
13     hint_witness("vrf_sk", vrf_sk_buf)
14
15     # Check 1: pk_vrf = Compress([vrf_sk, DOMAIN_VRF_PK, 0^14])[0]
16     pk_left  = Array(8)
17     pk_right = Array(8)
```

```

18  pk_left[0] = vrf_sk_buf[0]
19  pk_left[1] = DOMAIN_VRF_PK_CONST
20  for i in unroll(2, 8):
21      pk_left[i] = 0
22  for i in unroll(0, 8):
23      pk_right[i] = 0
24  pk_out = Array(8)
25  poseidon16_compress(pk_left, pk_right, pk_out)
26
27  pub      = 0
28  pub[0] = pk_out[0]          # constrains pk_out[0] ==
    public_input[0]
29
30  # Check 2: y = Compress([vrf_sk, vrf_input[0..15]])[0]
31  y_left = Array(8)
32  y_left[0] = vrf_sk_buf[0]
33  for i in unroll(0, 7):
34      y_left[1 + i] = pub[1 + i] # reads vrf_input[i] from public
    memory
35  y_out = Array(8)
36  poseidon16_compress(y_left, 8, y_out) # right half: public mem
    [8..16]
37
38  pub[16] = y_out[0]          # constrains y_out[0] == public_input
    [16]

```

Listing 5: VRF zkDSL circuit (Construction I)

The circuit calls `poseidon16_compress` twice, with the private witness `vrf_sk` appearing in both calls. The public memory layout enforces that the computed values equal the declared public inputs, creating the constraint system.

8.2 Rust Crate API

The `vrf` crate provides a minimal public API:

```

1 // Key derivation
2 pub fn VrfSecretKey::from_leansig_seed(seed: &[u8; 32]) ->
    VrfSecretKey;
3 pub fn VrfSecretKey::public_key(&self) -> VrfPublicKey;
4
5 // Evaluation
6 pub fn vrf_eval(sk: &VrfSecretKey, slot: u64, randao_mix: &[u8; 32])
7     -> VrfOutput;
8
9 // Proving and verification
10 pub fn vrf_prove(
11     sk: &VrfSecretKey, pk: VrfPublicKey,
12     slot: u64, randao_mix: &[u8; 32])
13     -> Result<(VrfOutput, VrfProof), VrfError>;
14
15 pub fn vrf_verify(
16     pk: VrfPublicKey, slot: u64, randao_mix: &[u8; 32],
17     output: &VrfOutput, proof: &VrfProof)
18     -> Result<(), VrfError>;

```

Listing 6: VRF crate public API

Table 3: Benchmark results (Apple M2, 8 GB RAM)

Operation	Median time	95% CI
<code>vrf_eval</code> (Poseidon only)	1.79 μ s	[1.75, 1.85] μ s
<code>vrf_prove</code> (full WHIR proof)	57.9 ms	[56.1, 59.8] ms
<code>vrf_verify</code>	16.8 ms	[16.76, 16.94] ms
Proof size	115 KB	(118,016 bytes)

Table 4: Comparison with EIP-7998 (BLS baseline)

Metric	BLS (EIP-7998)	PQ VRF (this work)	Ratio
Prove	≈ 1 ms	57.9 ms	58 \times
Verify	≈ 1 ms	16.8 ms	17 \times
Proof size	96 bytes	115 KB	1,200 \times
Post-quantum secure	No	Yes	–

Security note on zeroization. `VrfSecretKey` implements `Zeroize` and `Drop` via volatile byte writes. `vrf_prove` additionally zeroizes the copy of `vrf_sk` placed in the zkVM hints map after proof generation completes. Transient copies in lean VM execution traces and WHIR witness data are not scrubbed – this is an inherent limitation of the current proof system’s memory model. Operators running in high-risk environments (core dump risk, swap-to-disk) should be aware of this residual.

8.3 Benchmark Results

Benchmarks were measured using Criterion on a release build (`target-cpu=native`) with Rust 1.88.0.

Discussion. The proving time of 57.9 ms is negligible against the 12-second Ethereum slot window. The block proposer runs `vrf_prove` once per slot; the marginal cost is dominated by the WHIR trace generation, not the Poseidon evaluation itself (1.79 μ s).

The verification time of 16.8 ms is the per-block cost incurred by every consensus node. For comparison, leanMultisig’s per-slot XMSS aggregate proof verification is reported at sub-second times on comparable hardware.

The proof size of 115 KB is the dominant protocol overhead. This is serialized WHIR polynomial commitment data and is independent of the number of validators. The WHIR rate parameter (`log_inv_rate = 2`) provides ≈ 124 bits of security; increasing the rate to 3 or 4 reduces proof size at the cost of longer proving time.

Extrapolation to Construction II. Construction II requires three `Compress` calls vs. two for Construction I, a 1.5 \times increase in Poseidon constraint count. The dominant cost is WHIR commitment generation, which scales sub-linearly with circuit size. We estimate Construction II proves in ≈ 75 ms and verifies in ≈ 20 ms, with a proof size of ≈ 120 KB. These numbers will be confirmed in the production implementation.

8.4 Test Suite

The reference implementation includes 33 tests: 15 unit tests (key derivation, encoding, evaluation), 8 correctness tests (prove/verify roundtrip), 6 known-answer vector tests, and 4

uniqueness tests. All pass on Rust 1.88.0 (`cargo test -p vrf`). The Clippy linter is clean at `-D warnings` level.

9 Discussion

9.1 The QROM Gap and Future Work

The open problem of QROM proof soundness (Open Problem 1) is the primary theoretical gap in this work. It is not unique to our construction: `leanMultisig` itself acknowledges the same gap for its XMSS aggregation proofs. Resolving it requires either:

1. Extending the Don-Fehr-Majenz-Schaffner [?] result to multi-round IOPs with the specific structure of WHIR, or
2. Providing a direct QROM soundness proof for WHIR with concrete security bounds.

We identify this as the primary target for a companion academic paper (targeting Financial Cryptography or IEEE S&P 2027). A positive result would complete the QROM security picture for both our VRF and `leanMultisig`'s aggregate proof.

9.2 Composability with RANDAO

RANDAO mixes per-slot VRF contributions from up to 32 validators per epoch via XOR. The security argument that XOR composition of independent pseudorandom sources yields a pseudorandom output is standard: if any single source is pseudorandom (and the adversary does not control it), the XOR is pseudorandom.

A stronger statement – that QROM pseudorandomness of the per-slot VRF output composes with the full RANDAO accumulation across an epoch – would require a formal composability argument, potentially via the Universal Composability framework [?] or a bespoke reduction. We state this as an additional open problem and note that for practical purposes, the standard XOR argument is widely accepted in the protocol security literature.

9.3 Field Size and Output Entropy

The VRF output $y \in \mathbb{F}_p$ (or \mathbb{F}_p^8) is a KoalaBear field element, not a full 256-bit hash output. For Construction I, $|y| = 31$ bits; for Construction II, $|y| = 248$ bits. The RANDAO mix is 256 bits wide, so Construction I's 31-bit output leaves 225 bits of the mix unchanged from the previous slot's XOR contribution (since y is zero-padded to 32 bytes when XORed in). This reduces the entropy contribution per slot but does not break pseudorandomness: a 31-bit pseudorandom value XORed into a 256-bit accumulator cannot be predicted by an adversary without `vrf_sk`.

For Construction II, the 248-bit output provides near-full-width entropy contribution, with only 8 bits of zero-padding in the 256-bit representation. This is the preferred configuration for production.

9.4 Alternative Key Derivation

Decision 005 (derived VRF key) specifies that `vrf_sk` is derived from the `leanSig` seed via Poseidon1 with domain separation, rather than registered as a separate independent key. This means knowledge of the `leanSig` seed implies knowledge of `vrf_sk`. The security consequence is: if an adversary compromises the `leanSig` seed (e.g., via a cold-storage theft), they also obtain `vrf_sk`. For a separately registered key, the adversary would need to compromise both the signing key and the VRF key independently.

In practice, the seed is the single root of trust for leanSig validators; WOTS+ private keys are already derived from the seed via ShakePRF. Adding a separately registered VRF key would not eliminate this single root of trust – it would only add operational complexity. The derived-key design is therefore the appropriate choice, with the caveat that validators must protect their seed with the same care as all other key material.

9.5 Proof Size and Block Overhead

The 115 KB proof size is the dominant overhead introduced by this EIP. For comparison:

- A BLS signature (`randao_reveal`) is 96 bytes.
- A leanSig signature is $\approx 3,000$ bytes.
- A leanMultisig aggregate proof is $\approx 130\text{--}355$ KiB (depending on WHIR rate), submitted once per slot to cover all attestations.

The VRF proof (115 KB per block) is comparable to the aggregate attestation proof. Both are WHIR proofs over the same KoalaBear field; the VRF proof is smaller because its circuit (2–3 Poseidon calls) is vastly simpler than the XMSS aggregation circuit (hundreds of Poseidon calls).

Optimisation avenues.

- *Recursive aggregation.* The VRF proof could be included in the leanMultisig recursive aggregation proof for the slot, reducing the per-block overhead to zero at the cost of deferring VRF verification to the post-slot aggregation step. This is Architecture B (rejected in Decision 007 due to the optimistic acceptance problem), but could be revisited if the aggregate proof delay is reduced to within the attestation deadline.
- *WHIR rate increase.* Using `log_inv_rate = 3` reduces proof size at the cost of longer proving time. At the proposer’s 12-second slot, additional proving time is acceptable.
- *Multi-slot proof batching.* If the same validator proposes multiple slots (rare but possible during extended epochs), their VRF proofs could be batched into a single WHIR proof.

10 Conclusion

We have presented a post-quantum VRF construction for Ethereum’s RANDAO that is fully compatible with the leanSig/leanMultisig infrastructure targeted by the Ethereum Foundation’s PQ roadmap. The construction replaces BLS-based RANDAO contributions with Poseidon1-based PRF-commitment VRF outputs, provable via the leanMultisig zkVM without introducing any new cryptographic primitive.

Our main result is QROM pseudorandomness via Zhandry’s quantum PRF theorem – a result that holds independently of proof-system quantum security and provides immediate post-quantum protection for RANDAO entropy from milestone I^* onwards. We identified a critical quantum security gap in the compact single-element key design and presented a full-strength eight-element key variant with ≈ 124 -bit quantum security, matching leanMultisig’s stated security level.

The reference implementation in Rust achieves proving times of 57.9 ms and verification times of 16.8 ms on Apple M2 hardware – both well within the 12-second Ethereum slot window. A companion draft EIP specifies the three-phase `process_randao` state machine and the protocol changes required for activation at milestones I^* and L^* .

The primary open problem is QROM proof soundness for Fiat-Shamir applied to WHIR, which would complete the formal QROM security picture. We also identify composability across the RANDAO epoch accumulation as a secondary open problem. Both are natural targets for a follow-up academic contribution.

Acknowledgements

The author thanks Alberto La Rocca (co-author of EIP-7998) for the original VRF design collaboration, and the leanEthereum team (Justin Drake, Dmitry Khovratovich, Mikhail Kudinov, Benedikt Wagner, Will Corcoran) for developing the leanSig and leanMultisig infrastructure that this work extends.

A Domain Separation Collision Analysis

We verify that the VRF domain tags $D_{\text{in}}, D_{\text{pk}}, D_{\text{der}}, D_{\text{eval}}$ do not collide with any constant used in leanSig or leanMultisig.

leanSig tweaks. leanSig uses 8-bit tweak separators $\{0, 1, 2\}$ packed into the low bits of 40–56-bit values. These separators are not standalone field elements; they do not occupy canonical slots that our domain tags would. Our domain tags ($p-1$ through $p-4$) exceed $2^{31} - 5$, which is far from any packed tweak value. No collision.

leanMultisig SNARK domain separators. The leanMultisig SNARK domain separator array is $[130,704,175, 1,303,721,200, \dots, 2,063,844,858]$. The maximum value $2,063,844,858 < p - 4 = D_{\text{eval}}$. No collision.

leanMultisig Logup domain separators. $\text{LOGUP_MEMORY_DOMAINSEP} = 0, \text{LOGUP_PRECOMPILE_DOMAINSEP} = 1, \text{LOGUP_BYTECODE_DOMAINSEP} = 2$. These are $\{0, 1, 2\}$, far from our domain tags. No collision.

3-byte packing values. The maximum value of a 3-byte packed field element is $2^{24} - 1 = 16,777,215$. Our domain tags are at minimum $p - 4 = 2,130,706,429 \gg 2^{24} - 1$. Therefore no encoded byte value can equal any of our domain tags. No collision.

Mutual distinctness. $D_{\text{in}} = p - 1, D_{\text{pk}} = p - 2, D_{\text{der}} = p - 3, D_{\text{eval}} = p - 4$ are four consecutive values below p and are mutually distinct.

Table 5: Domain tag summary

Constant	Value	Hex
D_{in}	2,130,706,432	0x7f000000
D_{pk}	2,130,706,431	0x7effffff
D_{der}	2,130,706,430	0x7effffffe
D_{eval}	2,130,706,429	0x7effffffd

B Known-Answer Vectors

The following known-answer test vectors were generated by the reference implementation on Rust 1.88.0 with KoalaBear prime $p = 2^{31} - 2^{24} + 1$. All values are canonical field-element representations (little-endian 4-byte unsigned integers, zero-padded to 32 bytes for `Bytes32`).

Table 6: KAT vectors (seed \rightarrow `vrf_eval` output for $(s=0, r=0^{32})$)

Seed (32 bytes, hex)	$y = \text{vrf_eval}$ output (canonical u32)
00...00	see <code>crates/vrf/tests/kat.rs</code>
01...01	(pinned in test suite)
ff...ff	(pinned in test suite)
de...de	(pinned in test suite)
ab...ab	(pinned in test suite)

The exact 32-byte output values are pinned in `crates/vrf/tests/kat.rs` via the `kat_all_seeds_pinned` test. They serve as regression tests against any change to the KoalaBear prime, S-box exponent, round constants, or key derivation path.

C Full Poseidon1 Parameter Set

For completeness we record the Poseidon1 parameters confirmed from leanMultisig source at commit 5a5e10dd.

Table 7: Poseidon1 parameters over KoalaBear

Parameter	Value
Field prime p	$2^{31} - 2^{24} + 1 = 2,130,706,433$
Field representation	<code>MontyField31<KoalaBearParameters></code>
Two-adicity $v_2(p - 1)$	24
State width t	16
Full rounds R_f	8 (4 at start, 4 at end)
Partial rounds R_p	20
Total rounds	28
S-box exponent α	3 (cubemap; $\gcd(3, p - 1) = 1$)
Compression output	First 8 elements of Davies-Meyer
Round constants	From Plonky3 <code>poseidon1_koalabear_16</code>
Security level	≈ 124 bits (Johnson bound, degree-3 ext.)