

Unveiling the Cost: Gas-Efficient Output-Private Distributed VRFs on Ethereum

Ahmet Ramazan Ağirtas^{1,2}[0000-0002-4574-0067]^{*} and
Arda Buğra Özer²[0000-0002-6505-7038]

¹ AYWARE, Ankara, Türkiye
a.r.agirtas@gmail.com

² Institute of Applied Mathematics, Middle East Technical University, Ankara,
Türkiye
abozer@gmail.com

Abstract. Unbiased and publicly verifiable randomness is a fundamental component of Web3 security, and output-private distributed verifiable random functions (DVRFs) extend this guarantee by keeping the generated randomness hidden from chain observers until the requester chooses to reveal it. Two constructions currently realize output privacy under different algebraic regimes: FlexiRand, whose on-chain verification rests on bilinear pairings, and IcyVeil, a pairing-free design whose verification reduces to a constant number of group operations. While it is folklore that pairing-free verification should be cheaper on the Ethereum Virtual Machine (EVM), the actual on-chain cost of output-private randomness has never been measured for either scheme. In this paper, we present the first EVM implementations of both constructions. Both verifiers are served by a single coordinator contract realizing the common blind-evaluate-preverify lifecycle, so that every measured difference is attributable solely to the verification arithmetic. The isolated verification is 57% cheaper in execution gas for the pairing-free scheme (51,561 against 118,541 gas); over the full request lifecycle, whose remaining stages are shared, the advantage is about 11%. This on-chain saving comes with a cost off chain: the pairing-free scheme’s distributed evaluation requires roughly 2.5 to 8.2 times more aggregate committee compute across committee configurations (n, t) , a gap absorbed comfortably by Ethereum’s block time and one that narrows further once the committee executes concurrently. We report both axes, separating one-time distributed key generation (DKG) cost from IcyVeil’s consumable preprocessing, and reconcile the on-chain and off-chain ratios, which point in opposite directions. All contracts, harnesses, and raw measurements are publicly available.

Keywords: Distributed VRF · Output privacy · Gas cost · Pairing-free cryptography · Smart contracts · EVM.

^{*} Corresponding author.

1 Introduction

Many decentralized systems depend on randomness that no participant can predict or bias: consensus protocols, on-chain gaming and gambling platforms, non-fungible token (NFT) trait assignment, and decentralized finance logic. Verifiable Random Functions (VRFs) [20] answer the verifiability half of this requirement, and their distributed variants (DVRFs) [2, 3, 10, 13, 15] remove the single point of failure by sharing the secret key among a committee, so that no individual party can predict, bias, or withhold the output; we refer to [12] for a systematization of the area. Randomness services such as Chainlink VRF [9], drand [19], and the Supra VRF service [23] already serve this demand in production.

These deployed services, however, all share a limitation that concerns the *visibility* of the generated output rather than its correctness or unpredictability. In a conventional (D)VRF service the output is published on chain the moment the protocol completes. This creates two problems. A requester cannot obtain randomness in advance, since pending outputs are public and open to front-running; and a requester cannot safely reuse an output, for instance as a pseudorandom number generator (PRG) seed, once it has been revealed. *Output privacy* resolves this: the requester blinds the formatted VRF input before the committee evaluates it, the chain verifies the still-veiled result, and only the requester can unveil the final output. This notion was introduced by the FlexiRand framework [17], whose distributed instantiation builds on the pairing-based GLOW-DVRF [15]. More recently, Ağırtaş et al. proposed IcyVeil [4], which transplants the same blinding/unblinding architecture onto the pairing-free Icy-DVRF [3], a FROST-based [18] construction with constant-size proofs; the resulting scheme replaces the pairing product in on-chain verification with a handful of group exponentiations.

The practical magnitude of this pairing-free advantage on the EVM, however, has not been quantified, and the pricing of EVM precompiles suggests it may be smaller than raw operation counts imply. Conventional wisdom dictates that pairing-free verification should be cheaper on the EVM, and the IcyVeil authors estimate a substantial reduction in verification cost [4]. Yet the EVM prices cryptographic operations through precompiles, and the EIP-2537 precompiles [24] price a pairing product far below its relative cost in general-purpose computation, so that compute-time ratios measured off-chain do not translate into gas ratios. Calldata adds a second effect: a pairing-based verifier carries $\mathbb{G}_2$ elements on chain, twice the size of their $\mathbb{G}_1$ counterparts, and gas is charged per byte [6]. To the best of our knowledge, no measured on-chain comparison of output-private DVRFs exists in the literature; existing figures are estimates rather than measurements.

We implement both schemes against a single coordinator contract that realizes the common request lifecycle, which consists of input formatting with replay defense, verification of the requester’s blinding proof, fulfillment with pre-verification of the veiled output, and storage with callback. In this way, the two fulfillment paths differ *only* in their verification arithmetic, so that every

measured gas difference is attributable to the choice between pairing-based and pairing-free checking. Our contributions are as follows:

1. We give the first Solidity implementation of a pairing-free output-private DVRF verifier, realizing IcyVeil’s constant-cost pre-verification, alongside a FlexiRand-style pairing-based baseline over the EIP-2537 BLS12-381 pre-compiles.
2. We report gas measurements for every on-chain stage of the request lifecycle of both schemes, decomposed into precompile, arithmetic, and calldata costs, so that the comparison can be recomputed under future repricings.
3. We report off-chain evaluation cost, measured as aggregate committee compute, for producing the veiled output across committee configurations (n, t) , separating one-time DKG cost from IcyVeil’s consumable FROST-style pre-processing, which the pairing-based baseline does not require.
4. We reconcile the on-chain gas ratio against the off-chain compute-time ratio, which point in opposite directions, and explain why precompile pricing separates them.

Together, the measurements place the pairing-free advantage at 57% on the isolated verification call and at 11% over the complete request lifecycle. We report the two scopes together throughout, since the first is the sharper measure of the arithmetic that distinguishes the schemes while the second is the figure a deployment actually pays.

The on-chain contracts and the off-chain benchmark harness, together with all raw measurements, are publicly available.³

The outline of the paper is as follows. In Section 2 we give the necessary background, including the shared request lifecycle, the verification equations that each scheme places on chain, and the EVM cost model. In Section 3 we give an overview of our implementation and its validation. In Section 4 we describe the benchmark methodology, including the measured timing spans and the parameter grid. In Section 5 we present the results, and in Section 6 we discuss their implications and conclude.

2 Background

In this section we recall the notions and mechanisms that our measurements rest on: output-private DVRFs, the smart-contract request lifecycle that both schemes share, the verification equations that each scheme places on chain, and the EVM cost model against which they are priced. The account here is informal and self-contained: we recall the formal security definitions, games, and proofs only by reference to the respective protocol papers [4, 17], since they are not needed to interpret the gas and off-chain measurements that follow.

³ On-chain implementation: <https://github.com/abozer/OP-DVRF-Verify>.
Off-chain benchmarks: <https://github.com/abozer/DVRF-Comparison-ark>.

2.1 Notation

Let $\mathbb{G}$ be a cyclic group of prime order q with generator g , and let $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}$, $H_2 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ (challenge derivation), and $H_3 : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ (output derivation) be hash functions modeled as random oracles. Our implementation instantiates $\mathbb{G}$ as the $\mathbb{G}_1$ subgroup of BLS12-381 (Section 3). The committee consists of n nodes holding Shamir shares $x_1, \dots, x_n$ of a long-lived secret key x with threshold t , generated by a DKG protocol; the group public key is $Y = g^x$. We write $S \subseteq \{1, \dots, n\}$ for the set of nodes participating in a given evaluation and $\theta = |S|$ for its size, $t \leq \theta \leq n$. For the pairing-based scheme we additionally use groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order q with an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$; group elements of $\mathbb{G}_2$ are twice the size of those of $\mathbb{G}_1$ in the BLS12-381 instantiation, as discussed in Section 2.5.

2.2 Output-Private DVRFs

A distributed verifiable random function lets a committee jointly evaluate a pseudorandom function on an input α . No coalition of fewer than t nodes can predict or bias the output, and anyone can verify from a public proof that it is the unique correct value for α under the group public key [13, 15]. In a conventional DVRF service the final output β becomes public the moment the evaluation completes.

Output privacy [17] hides the result from chain observers until the requester reveals it. The requester *blinds* the formatted VRF input with a secret masking factor, the committee evaluates over the blinded point, and the chain verifies the still-veiled result against the group public key; only the requester can then *unveil* the final output β . Informally, chain observers learn that a correct evaluation took place, but learn nothing about β itself until it is unveiled; the formal game gives the adversary the entire protocol view and is stated in [4] for the pairing-free setting and in [17] for the pairing-based one. Output privacy is what allows a requester to obtain randomness in advance, or to reuse an output as the seed of a local PRG, without exposing it to front-running.

2.3 The Shared Request Lifecycle

Both schemes operate inside the same smart-contract framework, introduced by FlexiRand [17] and adopted by IcyVeil [4]. A request proceeds as follows.

1. The requester submits an initial input α to the coordinator contract.
2. The contract deterministically formats the VRF input string INP from α together with environment metadata (requester identity, chain identifier, block data, and a per-request nonce); this binding provides replay defense, since the same α submitted twice yields distinct inputs. The point to be evaluated is $h = H_1(\text{INP})$.
3. The requester fetches h , samples a secret masking factor κ , computes the blinded point $\hat{h}$ from h and κ , and submits $\hat{h}$ together with a knowledge-of-exponent non-interactive zero-knowledge (NIZK) proof $\hat{\pi}$ of well-formedness.

4. The contract verifies $\hat{\pi}$ and rejects the request otherwise. This gate ensures that the committee only ever evaluates points whose unblinding the requester can actually perform.
5. The committee nodes fetch $\hat{h}$ from the contract, and each node re-verifies $\hat{\pi}$ before evaluating. Namely, the nodes never see the plain evaluation point h as an input to their partial evaluations; they operate on the veiled point throughout.
6. Each participating node computes its partial evaluation over $\hat{h}$ with a proof of correct partial evaluation; a combiner verifies the partials off chain and aggregates them into the veiled output with an aggregated proof Π .
7. The combiner submits the veiled output to the contract, which *pre-verifies* it against the group public key Y and then stores it and serves the requester's callback. Pre-verification here means that correctness of the evaluation is checked while the output is still veiled.
8. The requester unveils the output off chain using κ , obtaining β , and can later publish a proof of correct unveiling for third-party verification.

Notice that steps 1–2, 4, and 7's storage and callback are scheme-agnostic: the two schemes differ only in the algebra filling steps 3–7, i.e. in how blinding, partial evaluation, aggregation, and the two on-chain checks (the blinding gate of step 4 and the pre-verification of step 7) are instantiated. Our implementation preserves this structure, isolating the schemes' difference to the two checks.

2.4 What the Contract Checks

Two on-chain verifications dominate the per-request cost, the blinding gate and the pre-verification of the veiled output, which we recall for each scheme.

IcyVeil. The first on-chain check is the blinding gate, where the contract verifies the requester's Schnorr-style knowledge-of-exponent proof over $\mathbb{G}$, at a cost of two exponentiations and one hash. The second is the output pre-verification, which receives the compact aggregated proof $\Pi = (\hat{\gamma}, c, \hat{z}) \in \mathbb{G} \times \mathbb{Z}_q \times \mathbb{Z}_q$, where $\hat{\gamma}$ is the veiled output, and checks the aggregated discrete-logarithm equality

$$T = g^{\hat{z}} \cdot Y^{-c}, \quad \bar{T} = \hat{h}^{\hat{z}} \cdot \hat{\gamma}^{-c}, \quad (1)$$

$$c \stackrel{?}{=} H_2(g, \hat{h}, Y, \hat{\gamma}, T, \bar{T}, \alpha). \quad (2)$$

This second check costs four exponentiations, two multiplications, and one hash in $\mathbb{G}$, independently of (n, t, θ) : all per-node proof verification is performed off chain by the combiner, and the chain only ever sees the three-element proof Π . Both the computation and the calldata of the IcyVeil path are therefore constant in the committee size.

FlexiRand. The first on-chain check is the analogous blinding gate, a knowledge-of-exponent check in $\mathbb{G}_1$. The second is the output pre-verification, which receives the veiled output $\hat{\gamma} \in \mathbb{G}_1$ and checks the pairing equation

$$e(\hat{\gamma}, g_2) \stackrel{?}{=} e(\hat{h}, Y_2), \quad (3)$$

where $Y_2 \in \mathbb{G}_2$ is the group public key, matching the PreVer algorithm of the FlexiRand construction [17].⁴ This is a two-pairing product, evaluated on the EVM through the EIP-2537 pairing-check precompile [24], with the public key carried in $\mathbb{G}_2$. Like IcyVeil’s check, it is constant in the committee size; the comparison between the two schemes is therefore a comparison of constants, namely a pairing product against a handful of exponentiations, and of the calldata each requires.

Post-Unveiling Verification. For completeness we note the final, post-unveiling public checks, which we also measure. In FlexiRand, after the requester unveils $\pi = \hat{\gamma}^{\kappa^{-1}}$ and derives $\beta = H_3(\pi)$, anyone can verify the triple via a second two-pairing check $e(h, Y_2) \stackrel{?}{=} e(\pi, g_2)$, where h is the formatted point of Section 2.3, together with the hash equality $H_3(\pi) \stackrel{?}{=} \beta$ [17].⁵ In IcyVeil, the requester publishes a discrete-logarithm-equality NIZK binding the veiled and unveiled values, and full public verification amounts to the pre-verification above plus one such NIZK check and a hash equality [4]; both schemes thus support verification of the entire request transcript by third parties.

2.5 EVM Cost Model

Every Ethereum transaction pays gas in three parts: a fixed intrinsic cost, a calldata cost of 16 gas per non-zero byte and 4 gas per zero byte [6], and the execution cost of the invoked code. Cryptographic operations over BLS12-381 are priced through the EIP-2537 precompiles [24], which expose, among others, $\mathbb{G}_1$ and $\mathbb{G}_2$ point addition and scalar multiplication and a pairing-check operation priced with a base cost plus a per-pair increment. The three operations that our two verifiers use are priced as in Table 1.

Table 1. Measured EIP-2537 precompile gas on the local Prague network.

Operation	Gas
$\mathbb{G}_1$ addition	734
$\mathbb{G}_1$ scalar multiplication	12,341
Pairing check ($k = 2$ pairs)	103,347

`staticcall`-measured, including minimal call overhead; these are the costs that drive the on-chain comparison of Section 5.2.

The EIP-2537 Choice. The choice of the EIP-2537 precompiles rather than the older `alt_bn128` precompiles of EIP-197 [21], which predate them on mainnet, is dictated by the security level at which the two constructions are specified.

⁴ Note that FlexiRand’s public key lives in $\mathbb{G}_2$, whereas IcyVeil’s key Y lies in $\mathbb{G} = \mathbb{G}_1$.

⁵ We restate FlexiRand’s verification equations in the notation of Section 2.1.

Advances in the tower number field sieve have reduced the effective security of the BN254 curve underlying `alt_bn128` to roughly 100 bits [7], which is what motivated the standardization of BLS12-381 and, with it, EIP-2537. Therefore, instantiating either scheme on `alt_bn128` would not realize the parameters that its security analysis assumes. Benchmarking both algorithms on the same curve is also what enables the comparison of verification arithmetic rather than differences of the curves. Notice that this choice is the conservative one for our conclusion. Under the EIP-1108 pricing of the `alt_bn128` precompiles [22], namely 150 gas per $\mathbb{G}_1$ addition, 6,000 per scalar multiplication, and $45,000 + 34,000k$ for a k -pair check, the same two verification equations would cost 24,300 against 113,000 gas, a factor of 4.65 rather than the factor of 2.03 we measure on BLS12-381. Thus, the pairing-free advantage we report on BLS12-381 is not an artifact of the curve; in fact, the curve we selected is the one least favorable to it.

Two properties of this cost model matter for the comparison in Section 5. First, the pairing-check precompile carries a large base cost that is paid even for a two-pairing product, whereas exponentiations in a pairing-free group are priced individually and modestly. Second, calldata is priced per byte, so a verifier whose inputs include $\mathbb{G}_2$ elements pays roughly twice per element compared to one operating on $\mathbb{G}_1$ or on a pairing-free curve; on rollups, where calldata dominates total cost, this second effect is amplified. Both effects act on the *constants* of the two $O(1)$ verifiers, so their combined effect on gas cannot be predicted from asymptotic cost or from off-chain compute ratios; we therefore measure it directly.

3 Implementation Overview

We give a brief architectural overview here; the contracts, the harnesses, and the engineering decisions behind them are documented in the public repository referenced in Section 1.

On-Chain Components. Our on-chain implementation consists of two Solidity contracts. The *verifier* exposes both fulfillment checks of Section 2.4 behind a common interface: the pairing-free path implementing Eqs. (1) and (2) with $\mathbb{G}_1$ arithmetic, and the pairing-based path implementing Eq. (3) through the EIP-2537 pairing-check precompile [24]; both operate over BLS12-381, so the two paths are priced on the same curve. The *coordinator* implements the FlexiRand request lifecycle [17] of Section 2.3: a per-request state machine moves through input formatting, the blinding gate, fulfillment, and callback, with one fulfillment entry point per scheme. The VRF input INP is assembled on chain from the same seven fields as in the FlexiRand framework (user input, block hash, per-request nonce, chain identifier, requester address, callback selector, and a scheme flag), and the contract commits to each served INP as replay defense. Notice that this state is kept per request, with no shared counter or global lock that would serialize independent requests against one another; we return to what this means for concurrent demand in Section 6. Note also that the request-creation,

blinding-gate, and storage/callback code is shared verbatim between the two schemes; only the fulfillment functions differ. Partial evaluations, per-partial proof checks, and aggregation all remain off chain at the combiner; the chain only ever receives the aggregated veiled output with its proof.

Off-Chain Components. The off-chain side is a Python harness implementing all four schemes of the 2×2 matrix of Section 4, namely Icy-DVRF, GLOW-DVRF, and their output-private counterparts, behind a common scheme interface (blind, verify-blinding, evaluate, pre-verify, unblind), over a single shared library stack, so that the measured ratios reflect the schemes themselves rather than differences between libraries. The pairing-free column reuses one FROST-style DKG and preprocessing module; the pairing-based column reuses one BLS partial-evaluation module with Chaum–Pedersen partial proofs, which serves GLOW-DVRF directly and FlexiRand when fed the veiled point. Timing instrumentation attributes cost to the spans defined in Section 4.

Validation. To validate the proof algebra, we reimplemented every verification equation of Section 2.4 in a separate library and checked the contract’s expected inputs, including byte-exact EIP-2537 point encodings, against it; all 17 assertions pass. End-to-end tests then exercise both fulfillment paths through the complete request lifecycle on a local devnet, including negative tests in which tampered proofs, replayed inputs, and malformed blindings must be rejected at the correct stage.

Toolchain. Contracts compile under `solc` 0.8.30 (optimizer enabled, `viaIR`, EVM target `prague`) and are measured on a local Hardhat devnet from transaction receipts. The off-chain harness and its measurement configuration are pinned in the repository for reproducibility.

4 Benchmark Methodology

4.1 On-Chain Measurements

All gas figures are read from transaction receipts on a local Hardhat devnet running the toolchain of Section 3; no figure in this paper is an estimate. For every stage of the request lifecycle, namely request creation, blinded-input submission (including the blinding gate), and fulfillment (pre-verification, storage, and callback dispatch), we report two numbers: the *execution gas* of the stage, and the *full-transaction gas*, i.e., execution plus the intrinsic transaction cost and the calldata cost of the stage’s inputs [6]. We report both figures since the two can diverge sharply: the schemes carry different-sized elements on chain, not only different verification arithmetic (Section 2.5), so their execution-gas gap and their calldata-gas gap need not agree. As our primary focus is on L1 deployments where execution cost is the dominant factor, we prioritize the execution gas metric. Stages shared verbatim between the schemes (Section 3) are measured once and reported once; IcyVeil exposes a single fulfillment path, so it occupies one row.

4.2 Off-Chain Measurements

Off-chain evaluation costs are measured for the 2×2 scheme matrix, which consists of the base DVRFs Icy-DVRF [3] and GLOW-DVRF [15] and their output-private counterparts IcyVeil [4] and FlexiRand [17]. All four run on the single library stack of Section 3, single-threaded, on one disclosed machine (Apple M4, 34.4 GB RAM, macOS; Python 3.13, `py-arkworks-bls12381` 0.5.0). Each per-request cell is measured over 500 iterations, and each setup cell (Table 4) over 100, after 10 untimed warmup iterations, with cyclic garbage collection disabled during timing; we report the median over the full sample, and compute the mean and standard deviation over the sample with the slowest 5% trimmed. Following the measurement approach of the FlexiRand evaluation [17], we time each step and report three spans, defined identically across schemes:

- **Blinding span** (private schemes only): the requester-local cost of sampling the masking factor, computing the veiled point, and producing the knowledge-of-exponent proof.
- **Evaluation span** (primary): from the moment the evaluation point is available to the committee until the (veiled) output and its aggregated proof are ready to submit. This span comprises the partial evaluations, the combiner’s per-partial proof checks, and the aggregation, all executed sequentially in a single process; the span is therefore the *aggregate committee compute* required to produce a veiled output, and not the wall-clock latency of a geographically distributed committee. For the private schemes, the partial evaluations include each node’s re-verification of the blinding proof. The clock starts after the requester’s blinding, so a base scheme and its private counterpart have directly comparable evaluation spans: the within-family gap prices output privacy, while the across-family gap prices the pairing.
- **Full span**: the sum of the two, i.e., input known to output ready.

The Evaluation Span Model. We report aggregate compute, the total work of producing a veiled output when every step runs sequentially in one process. It excludes two aspects of a real deployment. First, the θ partial evaluations are independent; a real committee runs them concurrently and pays the largest per-node cost rather than the sum, whereas our harness runs them in sequence. Only the combiner’s per-partial checks and the aggregation are sequential at one party. Second, it omits network cost entirely, namely the round trips that send the veiled point to the nodes and return the partial evaluations to the combiner. In Section 5.3 we bound the effect of both on the reported ratios, showing they are absorbed by costs the two families pay equally.

Setup costs are reported separately from per-request costs, in three columns: the one-time DKG, the preprocessing batch, and its amortized per-request share, the latter two applying only to the pairing-free family, since Icy-DVRF and IcyVeil generate single-use FROST-style nonce pairs [18] in batches of Ω and consume one pair per participating node per request. Steady-state per-request figures for the pairing-free column therefore include the amortized preprocessing

cost (batch cost divided by Ω); we also report the raw evaluation span without it, since the amortized and marginal costs answer different questions.

4.3 Parameter Grid and Repetitions

All four schemes are measured on the shared grid of (n, t) pairs $(5, 3)$, $(7, 4)$, $(10, 7)$, $(16, 11)$, $(25, 17)$, and $(50, 34)$, where t is the minimum quorum: sharing is of degree $t - 1$, any t shares reconstruct, and every run uses the minimum quorum $\theta = t$. Under the measurement protocol of Section 4.2, observed standard deviations were below 1.2% of the mean in all evaluation-span cells (Table 3) and below 1.6% in the setup cells (Table 4).

The grid stops at $n = 50$ to reflect deployed committee sizes, not a constraint of the harness. In real-world deployments to date, threshold randomness committees are small. For example, Internet Computer subnets operate a few dozen participants [10], and drand is comparable [19]. At $n = 50$ the grid already reaches beyond them. Evaluating on a larger grid would add little beyond what the current one already shows, for two reasons. The on-chain verification cost is constant in (n, t) for both schemes (Section 2.4), so committee size does not enter the main claim. Off chain, in turn, the span scales close to linearly in θ over the grid’s tenfold range, so behavior past $n = 50$ follows from the fit rather than being qualitatively new. Off-chain scaling is thus supporting evidence rather than the main claim. The harness configuration, raw measurement logs, and the scripts producing every table and figure of Section 5 are included in the repository.

5 Results

5.1 On-Chain Gas

Recall from Section 4.1 that we report two gas figures per stage. The *execution gas* isolates the cost of running the verification code and is the sharper measure of the arithmetic difference between the schemes; the *full-transaction gas* adds the fixed intrinsic cost and the calldata cost of the inputs, and is what a requester actually pays. The two give slightly different ratios: the intrinsic and calldata components are almost identical across the schemes, so they dilute the full-transaction ratio toward one. We report both throughout, and quote the execution figure as the primary comparison, since it reflects the verification arithmetic that distinguishes the schemes.

Table 2 reports per-stage gas for the full request lifecycle. The request-creation and blinded-input stages are shared between the two schemes and reported once; the fulfillment and isolated pre-verification rows are where the schemes differ.

The advantage of the pairing-free path depends on the scope of the comparison, and it is worth stating all three. The verification arithmetic itself, isolated as the pre-verification call, costs 51,561 against 118,541 execution gas: the pairing-free scheme is 57% cheaper (44% in full-transaction gas). Within the complete

Table 2. Per-stage gas of the request lifecycle, measured from transaction receipts on a local Prague (EIP-2537) network.

Stage	Execution gas	Full-Transaction gas
Request creation (shared)	193,484	215,488
Blinded input + gate (shared)	184,318	209,062
Pre-verification, FlexiRand	118,541	147,101
Pre-verification, IcyVeil	51,561	82,017
Fulfillment, FlexiRand	166,635	189,783
Fulfillment, IcyVeil	105,073	129,805
Total per request, FlexiRand	544,437	614,333
Total per request, IcyVeil	482,875	554,355

(a) Median over 500 repetitions. (b) The isolated pre-verification is the arithmetic comparison; fulfillment additionally performs storage and callback dispatch. (c) The total additionally includes the two shared stages.

fulfillment transaction, which also performs storage and callback dispatch, the gap is a factor of 1.59 in execution gas (1.46 full-transaction). Across the entire request lifecycle, whose request-creation and blinded-input stages are shared and together dominate the cost, it narrows to 11% (10% full-transaction). The last of the three is the figure a deployment pays per served request, and we pair it with the isolated 57% wherever the latter appears. The descent across these three scopes is itself an instance of the point we return to in Section 5.4: a ratio means little without its denominator.

5.2 Why the Pairing-Free Path Wins

The gap is fully explained by precompile arithmetic, which we now decompose. Under the measured EIP-2537 costs of Table 1, the FlexiRand pre-verification of Eq. (3) is one $k=2$ pairing check, hence 103,347 gas. The IcyVeil pre-verification of Eqs. (1) and (2) is four $\mathbb{G}_1$ scalar multiplications and two point additions, hence $4 \cdot 12,341 + 2 \cdot 734 = 50,832$ gas plus one hash, a factor of 2.03 at the precompile level. In the multiplicative notation of Section 2.4, these are the four exponentiations and two multiplications of the pre-verification. This precompile-level ratio matches the 57% isolated-pre-verification saving of Table 2 once the fixed per-call overhead common to both is added.

On calldata, the pairing-free scheme is the heavier of the two: IcyVeil’s aggregated proof $H = (\hat{\gamma}, c, \hat{z})$ is 192 bytes to FlexiRand’s 128. The pairing-free scheme wins overall despite carrying more calldata; the saving comes from replacing the pairing, not from smaller inputs. The pairing-based scheme’s $\mathbb{G}_2$ group key enters storage once at deployment, not per-request calldata. This arithmetic advantage is diluted, not reinforced, by the surrounding per-request costs: input formatting, state-machine transitions, storage, and callback are identical by construction (Section 3) and shrink the precompile-level factor of 2.03 to the 11%

Table 3. Off-chain evaluation span (aggregate committee compute, ms) across the (n, t) grid. Each cell gives the median $\tilde{x}$, the mean $\bar{x}$, and the standard deviation σ .

(n, t)	Base DVRFs		Output-private	
	GLOW-DVRF	Icy-DVRF	FlexiRand	IcyVeil
(5, 3)	$\tilde{x} = 2.50$	$\tilde{x} = 7.19$	$\tilde{x} = 3.17$	$\tilde{x} = 7.86$
	$\bar{x} = 2.49$	$\bar{x} = 7.18$	$\bar{x} = 3.16$	$\bar{x} = 7.85$
	$\sigma = 0.03$	$\sigma = 0.05$	$\sigma = 0.03$	$\sigma = 0.05$
(7, 4)	$\tilde{x} = 3.49$	$\tilde{x} = 10.69$	$\tilde{x} = 4.37$	$\tilde{x} = 11.55$
	$\bar{x} = 3.49$	$\bar{x} = 10.68$	$\bar{x} = 4.37$	$\bar{x} = 11.55$
	$\sigma = 0.03$	$\sigma = 0.06$	$\sigma = 0.03$	$\sigma = 0.06$
(10, 7)	$\tilde{x} = 6.14$	$\tilde{x} = 22.79$	$\tilde{x} = 7.61$	$\tilde{x} = 24.31$
	$\bar{x} = 6.14$	$\bar{x} = 22.81$	$\bar{x} = 7.61$	$\bar{x} = 24.31$
	$\sigma = 0.03$	$\sigma = 0.09$	$\sigma = 0.04$	$\sigma = 0.08$
(16, 11)	$\tilde{x} = 9.78$	$\tilde{x} = 44.63$	$\tilde{x} = 12.03$	$\tilde{x} = 46.97$
	$\bar{x} = 9.78$	$\bar{x} = 44.65$	$\bar{x} = 12.03$	$\bar{x} = 46.97$
	$\sigma = 0.05$	$\sigma = 0.15$	$\sigma = 0.06$	$\sigma = 0.22$
(25, 17)	$\tilde{x} = 15.26$	$\tilde{x} = 90.47$	$\tilde{x} = 18.64$	$\tilde{x} = 94.01$
	$\bar{x} = 15.26$	$\bar{x} = 90.47$	$\bar{x} = 18.65$	$\bar{x} = 94.18$
	$\sigma = 0.07$	$\sigma = 0.32$	$\sigma = 0.11$	$\sigma = 1.10$
(50, 34)	$\tilde{x} = 30.96$	$\tilde{x} = 300.08$	$\tilde{x} = 37.54$	$\tilde{x} = 306.86$
	$\bar{x} = 30.95$	$\bar{x} = 300.10$	$\bar{x} = 37.53$	$\bar{x} = 306.94$
	$\sigma = 0.09$	$\sigma = 0.75$	$\sigma = 0.17$	$\sigma = 0.93$

Over 500 repetitions on an Apple M4, single-threaded; $\bar{x}$ and σ are computed over the slowest-5%-trimmed sample. Standard deviations are below 1.2% of the mean in every cell.

whole-lifecycle figure of Table 2. Notice that this decomposition also reveals a sensitivity: since the gap is governed by precompile pricing, a future repricing of the pairing check would shift the ratio on its own, with neither protocol changing.

The sensitivity is bounded, since the pairing check would have to be repriced to less than half its present 103,347 gas before the ordering reverses. A repricing in the other direction, or a chain whose pairing precompile costs more than Ethereum’s, widens the gap instead.

5.3 Off-Chain Evaluation Cost

On chain, the pairing-free scheme is the cheaper of the two; off chain, the ordering reverses. Table 3 reports the evaluation span (the primary span of Section 4.2) across the (n, t) grid for the 2×2 matrix, all four schemes on the single `py-arkworks-bls12381` stack.

Three observations follow from the data. First, all four schemes scale roughly linearly in θ , dominated by the combiner’s per-partial proof checks. Second,

output privacy is inexpensive within each family: IcyVeil adds only 9% over Icy-DVRF at (5, 3), and FlexiRand adds 27% over GLOW-DVRF. Third, and in the opposite direction to the on-chain result, the across-family gap favors the pairing-based schemes and grows with committee size: within the output-private pair that our comparison targets, IcyVeil is a factor of 2.5 more expensive than FlexiRand at (5, 3) and 8.2 at (50, 34); in the base pair, Icy-DVRF is 2.9 and 9.7 more expensive than GLOW-DVRF.

The FROST-style construction pays for its cheap on-chain verification with per-node discrete-logarithm-equality proofs and nonce handling that cost more off chain than a BLS exponentiation. We return to this trade, and what it means for deployment, in Section 6.

From Aggregate Compute to Wall-Clock Latency. The figures of Table 3 are aggregate committee compute (Section 4.2), not the latency a deployed committee would observe. Under concurrent execution, wall-clock latency is approximately $\max_{i \in S} \text{eval}_i + \theta \cdot \text{check} + \text{agg} + \text{net}$: one partial evaluation and its proof run in parallel across nodes, while the θ per-partial checks and the aggregation run sequentially at the combiner. Parallelism removes the node-side term but not the linear scaling itself, since the combiner’s per-partial checks dominate the span and stay sequential at one party; the across-family ordering therefore survives concurrent execution, though the gap narrows. Standard batch-verification techniques for discrete-logarithm-equality proofs [8] target precisely this dominant term and are the natural next optimization for the pairing-free family; we have not implemented them and claim no benefit here.

The network term is likely to dominate the other two. A wide-area round trip between committee members takes tens to low hundreds of milliseconds; at realistic committee sizes this exceeds either family’s entire compute span, leaving the across-family compute gap a second-order effect on the latency a requester sees. Notice that both families pay these off-chain costs equally, so the difference cancels, whereas the on-chain difference is paid on every verification, by every party that checks the output. The result sharpens the paper’s conclusion rather than weakening it. A second asymmetry points the same way: since Icy-DVRF and IcyVeil consume nonce pairs committed during preprocessing, those commitments are already published when a request arrives, so the pairing-free family adds no communication round at request time. Its compute disadvantage brings no round-complexity disadvantage.

To complete the picture, we report the setup and requester-side costs in Table 4. Distributed key generation is a one-time cost and, using the same protocol in both families, is essentially independent of the scheme; we therefore report it once. Preprocessing is a recurring cost specific to the pairing-free family: Icy-DVRF and IcyVeil generate a batch of $\Omega = 510$ single-use nonce pairs, one consumed per participating node per request, so the per-request share is the batch cost divided by Ω . The requester-side blinding step, by contrast, is a flat sub-millisecond cost independent of committee size (median 0.36–0.39 ms for both private schemes across the grid), and we do not tabulate it separately.

Table 4. One-time DKG and preprocessing costs (ms). Each cell gives the median $\tilde{x}$, the mean $\bar{x}$, and the standard deviation σ .

(n, t)	DKG	Preproc. batch	Preproc. / request
(5, 3)	$\tilde{x} = 6.00$	$\tilde{x} = 530.3$	$\tilde{x} = 1.04$
	$\bar{x} = 5.99$	$\bar{x} = 532.0$	$\bar{x} = 1.04$
	$\sigma = 0.09$	$\sigma = 6.2$	$\sigma = 0.012$
(7, 4)	$\tilde{x} = 11.08$	$\tilde{x} = 743.5$	$\tilde{x} = 1.46$
	$\bar{x} = 11.13$	$\bar{x} = 745.1$	$\bar{x} = 1.46$
	$\sigma = 0.17$	$\sigma = 5.7$	$\sigma = 0.011$
(10, 7)	$\tilde{x} = 25.01$	$\tilde{x} = 1060.5$	$\tilde{x} = 2.08$
	$\bar{x} = 25.01$	$\bar{x} = 1062.2$	$\bar{x} = 2.08$
	$\sigma = 0.10$	$\sigma = 9.2$	$\sigma = 0.018$
(16, 11)	$\tilde{x} = 79.85$	$\tilde{x} = 1691.7$	$\tilde{x} = 3.32$
	$\bar{x} = 79.82$	$\bar{x} = 1692.8$	$\bar{x} = 3.32$
	$\sigma = 0.19$	$\sigma = 4.4$	$\sigma = 0.009$
(25, 17)	$\tilde{x} = 330.29$	$\tilde{x} = 2640.6$	$\tilde{x} = 5.18$
	$\bar{x} = 331.08$	$\bar{x} = 2642.5$	$\bar{x} = 5.18$
	$\sigma = 3.62$	$\sigma = 6.7$	$\sigma = 0.013$
(50, 34)	$\tilde{x} = 4871.21$	$\tilde{x} = 5279.0$	$\tilde{x} = 10.35$
	$\bar{x} = 4874.48$	$\bar{x} = 5288.5$	$\bar{x} = 10.37$
	$\sigma = 9.33$	$\sigma = 43.9$	$\sigma = 0.086$

Over 100 repetitions on an Apple M4, single-threaded; $\bar{x}$ and σ are computed over the slowest-5%-trimmed sample. Standard deviations are below 1.6% of the mean in every cell. The per-request share is the batch cost divided by $\Omega = 510$.

5.4 Reconciling Gas and Compute Ratios

The same pair of schemes is described by ratios that not only differ in magnitude but point in opposite directions, and it is worth placing them side by side. Off chain, in raw computation, the pairing-free evaluation is a factor of 2.9 *more expensive* at (5, 3) (Icy-DVRF against GLOW-DVRF, Table 3, i.e., the base pair; the corresponding figure for the output-private pair is 2.5). On chain, the same schemes reverse: the isolated pairing-free verification is 2.3 times *cheaper* in execution gas (Table 2), because the EIP-2537 pairing check costs 103,347 gas, far above the pairing-free arithmetic yet far below the relative cost of a pairing in general-purpose computation. The pairing-free scheme loses on compute time and wins on EIP-2537 gas, so which one is cheaper depends on the unit of comparison. The on-chain saving is the one paid on every verification.

6 Discussion and Conclusion

A Trade-Off Between On-Chain and Off-Chain Costs. The measurements describe a trade rather than a uniform win. On chain, where it counts, the pairing-

free scheme is cheaper to verify: 57% on the isolated verification and 11% over the full request lifecycle, in execution gas. The size of this saving is set by precompile pricing rather than by the protocols themselves. EIP-2537 prices a two-pairing check at 103,347 gas, well below the relative cost of a pairing in general-purpose computation, so what is a factor of 2.9 (base pair) aggregate-compute advantage for the pairing-based scheme off chain becomes a factor of 2.03 gas advantage at the precompile level, and 2.3 on the isolated verification call, for the pairing-free scheme on chain. Pairings on chain are no longer as expensive as they once were; the pairing-free scheme is simply cheaper still to verify.

Sensitivity to Pricing and Deployment Context. The cost of that on-chain saving is paid off chain, where the pairing-free evaluation is 2.5 to 8.2 times more expensive in aggregate committee compute (Section 5.3). In practice this is not a bottleneck. Even at the largest committee we measured, $n = 50$, the pairing-free evaluation completes in approximately 307 milliseconds; at deployment-realistic sizes it is tens of milliseconds. Ethereum produces a block every 12 seconds [14], and even with the active proposal to halve this to 6 seconds [1], a single slot would still fit many veiled outputs in sequence, roughly 700 at (5, 3) and around 20 even at (50, 34). These counts divide the slot by the sequential compute the harness measures (Section 4.2); a real committee runs its partial evaluations concurrently, so each finishes in roughly one node’s time rather than the whole committee’s, which increases the number that fit per slot. Conversely, network delays, not taken into account here, lower it again.

The off-chain cost is thus absorbed by block latency, whereas the on-chain saving is paid on every verification, by every party that checks the output. IcyVeil additionally carries the per-request cost of nonce management, its single-use preprocessing batches adding an amortized 1.04 to 10.35 milliseconds per request across the grid (at $\Omega = 510$); this also sits far inside a slot. Rather than one scheme dominating, the two place their costs in different currencies: FlexiRand spends gas to save committee time, while IcyVeil spends committee time it can spare to save gas.

Throughput Under Concurrent Demand. The measurements above price a single request, whereas a deployed service faces many at once. On chain, the coordinator keeps its state per request (Section 3), with no shared counter or global lock that would serialize independent requests against one another, so concurrency is bounded by block capacity rather than by the contract. At the 60 million block gas limit of EIP-7935 [16], a block devoted entirely to randomness can support roughly 108 IcyVeil request lifecycles against 97 FlexiRand ones, at the 554,355 and 614,333 full-transaction gas per request of Table 2. The 11% lifecycle saving thus becomes 11% more served requests per unit of block space, the form in which it matters to an operator. At L1 rates the binding constraint is block space, not the off-chain committee. For calibration, deployed randomness services such as Chainlink VRF [11] serve request volumes well below this per-block ceiling, so at present demand the coordinator is not the limiting factor for either scheme.

IcyVeil carries one scalability cost that FlexiRand does not. Preprocessing supplies single-use nonce pairs, one per node per request, so each node’s batch of $\Omega = 510$ lasts 510 requests. Under sustained demand this replenishment must be pipelined against request service, since a committee that exhausts its batch stalls until a new one is generated. The batch itself is inexpensive, 0.53 to 5.28 seconds across the grid (Table 4), so the requirement is one of scheduling rather than throughput, but one FlexiRand does not share. A second, slower effect is state growth: the coordinator commits to each served input for replay defense, so per-request storage accumulates over a deployment’s lifetime. Bounding the retained commitments to a recent window, or moving them to logs under an on-chain digest, removes the growth at the cost of a shorter replay window. We keep the simple design here because it is shared verbatim between the two schemes and so cancels from the comparison.

Limitations. Our measurements are single-chain (a local devnet under one pricing regime), single-machine, and single-library off chain; they exclude network latency, consensus, and the operational costs of running committee nodes. The off-chain harness is implemented in Python (`py-arkworks-bls12381`), and its absolute times are not strictly comparable to highly optimized, natively compiled implementations. Re-implementing the off-chain components in a systems language such as Rust (e.g., via `arkworks-rs`) or Go would likely narrow the observed off-chain performance gap between the pairing-free and pairing-based schemes. Since our claims primarily concern relative ratios, we report the published optimized-stack figures of [17] alongside ours for calibration. Larger committee sizes and multi-chain gas measurements are left to future work.

This paper reports measurements of two implementations rather than an evaluation of their security. The security definitions, games, and proofs for both constructions are those of [4, 17], which we neither restate nor revisit. What we verify is an implementation claim, that the deployed equations match the ones those papers specify, through the byte-exact reimplementations and negative tests of Section 3. Adversarial behavior beyond what those checks reject is outside the scope of our measurements. Our committee runs at the minimum quorum $\theta = t$ with an honest combiner, so a participant that aborts or equivocates mid-evaluation halts the request rather than being routed around, and we measure no recovery path. Robustness in this sense is orthogonal to the on-chain comparison, since it changes which partial evaluations reach the combiner and not what the contract verifies, though a deployment facing an adversarial network would need it. The robust asynchronous variant (R)Icy-DVRF [5] realizes exactly this, at the cost of two additional exponentiations per partial evaluation and no change to on-chain verification, so our gas comparison carries over to it unchanged.

Conclusion. We presented the first EVM implementations of both an output-private pairing-free DVRF and its pairing-based counterpart, sharing one coordinator and differing only in verification arithmetic, and we replaced the estimates of prior work with deployed, receipt-measured gas figures; the figures, contracts, and harnesses are all public and regenerable from the repository. The measured

picture is a trade with a clear winner in context: IcyVeil is 57% cheaper to verify on chain on the isolated verification call, and 11% cheaper across the complete request lifecycle, at the price of an off-chain evaluation 2.5 to 8.2 times more expensive in aggregate committee compute that nonetheless completes comfortably within an Ethereum slot. The compute and gas ratios point in opposite directions, so whether output privacy is cheaper without pairings depends on which cost is measured and in what setting. Furthermore, the coordinator architecture we implement follows the same request-and-callback gateway pattern as deployed randomness oracles such as Chainlink’s `VRFCoordinator`, which suggests that the integration path for a pairing-free output-private layer is a familiar one; we do not claim interface-level compatibility with any particular deployed service, whose operational requirements we have not evaluated. For an L1-native randomness service [11] where on-chain verification costs vastly outnumber one-time setup and preprocessing runs, this pairing-free path provides a highly economical output-privacy layer without prohibitive gas overhead.

References

1. Adams, B., Feist, D., Silva, M.I., Harris, P.: EIP-7782: Reduce Block Latency. Ethereum Improvement Proposals, no. 7782 (2024), draft. Available: <https://eips.ethereum.org/EIPS/eip-7782>
2. Ağırtaş, A.R., Özer, A.B., Saygi, Z., Yayla, O.: Distributed Verifiable Random Function with Compact Proof. In: Dolev, S., Elhadad, M., Kutylowski, M., Persiano, G. (eds.) *Cyber Security, Cryptology, and Machine Learning*. pp. 119–134. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-76934-4_8
3. Ağırtaş, A.R., Özer, A.B., Saygi, Z., Yayla, O.: Icy-DVRF: A distributed verifiable random function based on FROST signatures. *IEEE Access* **14**, 105882–105909 (2026). <https://doi.org/10.1109/ACCESS.2026.3712805>
4. Ağırtaş, A.R., Özer, A.B., Saygi, Z., Yayla, O.: IcyVeil: An Output-Private DVRF Based on Icy-DVRF. *Cryptology ePrint Archive*, Paper 2026/1138 (2026), <https://eprint.iacr.org/2026/1138>, last accessed 07/07/2026
5. Ağırtaş, A.R., Yayla, O., Yılmaz, M.B.: (R)Icy-DVRF: A robust distributed verifiable random function based on ROAST signatures. *Cryptology ePrint Archive*, Paper 2026/1378 (2026), <https://eprint.iacr.org/2026/1378>
6. Akhunov, A., Ben Sasson, E., Brand, T., Guthmann, L., Levy, A.: EIP-2028: Transaction data gas cost reduction, Ethereum Improvement Proposals, no. 2028 (May 2019), <https://eips.ethereum.org/EIPS/eip-2028>, last accessed 07/07/2026
7. Barbulescu, R., Duquesne, S.: Updating key size estimations for pairings. *Journal of Cryptology* **32**(4), 1298–1336 (2019). <https://doi.org/10.1007/s00145-018-9280-5>
8. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: *Advances in Cryptology — EUROCRYPT’98*. pp. 236–250. Springer (1998). <https://doi.org/10.1007/BFb0054130>
9. Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., et al.: Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks. Whitepaper (2021), <https://research.chain.link/whitepaper-v2.pdf>, last accessed 07/07/2026

10. Camenisch, J., Cerulli, A., Derler, D., Drijvers, M., Dubovitskaya, M., Groth, J., Hanke, T., Neven, G., Pignolet, Y.A., Shoup, V., Tackmann, B., Williams, D.: The Internet Computer for Geeks. Cryptology ePrint Archive, Paper 2022/087 (2022), <https://eprint.iacr.org/2022/087>, last accessed 07/07/2026
11. Chainlink Labs: Chainlink VRF: Verifiable random function. Technical Documentation (2023), <https://docs.chain.link/vrf>, last accessed 07/07/2026
12. Choi, K., Manoj, A., Bonneau, J.: SoK: Distributed Randomness Beacons. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 75–92 (2023). <https://doi.org/10.1109/SP46215.2023.10179419>
13. Dodis, Y.: Efficient Construction of (Distributed) Verifiable Random Functions. In: Public Key Cryptography — PKC 2003. pp. 1–17. Springer (2003). https://doi.org/10.1007/3-540-36288-6_1
14. Ethereum Foundation: Proof-of-Stake (PoS): Gasper and the Beacon Chain. <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/> (2024), slot time: 12 seconds
15. Galindo, D., Liu, J., Ordean, M., Wong, J.M.: Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 88–102 (2021). <https://doi.org/10.1109/EuroSP51992.2021.00017>
16. Gold, S., Jayanthi, P., Wahrstätter, T., Beekhuizen, C., Dietrichs, A., Feist, D., Stokes, A., Rudolph, J., Rebuffo, G., Slivkoff, S., Chodola, K.: EIP-7935: Set default gas limit to 60M. Ethereum Improvement Proposals, no. 7935 (Apr 2025), <https://eips.ethereum.org/EIPS/eip-7935>
17. Kate, A., Mangipudi, E.V., Maradana, S., Mukherjee, P.: FlexiRand: Output Private (Distributed) VRFs and Application to Blockchains. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 1776–1790 (2023). <https://doi.org/10.1145/3576915.3616601>
18. Komlo, C., Goldberg, I.: FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In: Selected Areas in Cryptography: 27th International Conference (SAC 2020). pp. 34–65. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-81652-0_2
19. League of Entropy: Cryptography: An overview of drand’s cryptographic building blocks. drand Documentation, <https://docs.drand.lol/docs/cryptography/>, last accessed 07/07/2026
20. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science. pp. 120–130 (1999). <https://doi.org/10.1109/SFFCS.1999.814584>
21. Reitwießner, C., Buterin, V.: EIP-197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. Ethereum Improvement Proposals, no. 197 (2017), <https://eips.ethereum.org/EIPS/eip-197>
22. Salazar Cardozo, A., Williamson, Z.: EIP-1108: Reduce alt_bn128 precompile gas costs. Ethereum Improvement Proposals, no. 1108 (2018), <https://eips.ethereum.org/EIPS/eip-1108>
23. Supra Research: Supra VRF Service. Technical Whitepaper (2023), <https://supra.com/documents/VRF-Service-Whitepaper.pdf>, last accessed 07/07/2026
24. Vlasov, A., Olson, K., Stokes, A., Sanso, A.: EIP-2537: Precompile for BLS12-381 curve operations, Ethereum Improvement Proposals, no. 2537 (February 2020), <https://eips.ethereum.org/EIPS/eip-2537>, last accessed 07/07/2026