IcyVeil: An Output-Private DVRF Based on Icy-DVRF

Ahmet Ramazan Ağırtaş[1][0000-0002-4574-0067], Arda Buğra Özer[2][0000-0002-6505-7038], Zülfükar Saygı[3][0000-0002-7575-3272], and Oğuz Yayla[2][0000-0001-8945-2780]

[1] AYWARE, Ankara, Türkiye
a.r.agirtas@gmail.com
[2] Institute of Applied Mathematics, Middle East Technical University, Ankara, Türkiye
{arda.ozer, oguz}@metu.edu.tr
[3] Department of Mathematics, TOBB University of Economics and Technology, Ankara, Türkiye
zsaygi@etu.edu.tr

Abstract. Unbiased and unpredictable randomness is a backbone of Web3 security, yet current Distributed Verifiable Random Function designs often entail a trade-off between performance and privacy. While established protocols like GLOW-DVRF achieve constant-size proofs, they rely on computationally expensive bilinear pairings that impose significant gas overhead in on-chain environments. Existing output-privacy frameworks, such as FlexiRand, are currently limited by the same expensive pairing-based operations that have high on-chain verification costs. In this paper, we present IcyVeil, an output-private DVRF that conjoins the architecture of FlexiRand with the pairing-free efficiency of Icy-DVRF. By integrating a blinding/unblinding mechanism directly into a FROST-inspired preprocessing scheme, IcyVeil enables users to mask inputs with a private nonce while the distributed committee generates partial evaluations and NIZK proofs over the veiled values. This approach eliminates the high cost of bilinear pairings during verification and maintains constant-size proofs. By adopting the pairing-free architecture of Icy-DVRF, IcyVeil inherits a 43% reduction in on-chain gas costs compared to conventional pairing-based protocols, establishing a scalable and cost-effective primitive for latency-sensitive applications such as decentralized gaming and asynchronous reward distribution.

Keywords: Cryptography · Verifiable Random Function · Distributed Verifiable Random Function · Blockchain · Blinding

1 Introduction

Generating a provably unbiased and unpredictable source of randomness remains a foundational challenge that underpins the security of diverse computational frameworks. Cryptographic key generation, digital signing protocols,

online gaming and gambling platforms, consensus protocols on blockchains, and decentralized finance (DeFi) logic all critically rely on true randomness. In systems involving multiple, mutually distrusting participants, ensuring that this randomness is not merely unpredictable but also publicly verifiable and demonstrably unbiased is absolutely crucial to system integrity. This necessity led to the widespread adoption of Verifiable Random Functions (VRFs), introduced by Micali, Rabin, and Vadhan [12], which operate as a public-key analog to cryptographic hash functions, equipped with non-interactive proofs of correctness.

In a standard, non-distributed VRF setting, a single entity utilizes a secret key to evaluate an input, producing both a pseudo-random output and a cryptographic proof. Anyone with the corresponding public key can subsequently verify this proof, mathematically guaranteeing that the randomness was generated legitimately and without manipulation. Prominent oracle services such as Chainlink VRF [3] and consensus mechanisms heavily utilize these single-party VRFs to feed off-chain randomness into smart contracts. However, routing all randomness generation through a monolithic entity introduces severe operational vulnerabilities. If the key-holder is compromised, experiences a Byzantine fault, or colludes with a malicious actor, the foundational guarantee of unpredictability is completely shattered. A leaked or exploited secret key allows an adversary to fully compute and predict future *supposedly random* outcomes, enabling devastating front-running attacks and exploits across any application that relies on that oracle.

To address this inherent fragility, Distributed Verifiable Random Functions (DVRFs) have emerged as a vital cryptographic primitive. By sharing the secret key among multiple participants using threshold cryptographic techniques, DVRFs ensure that no single party can predict, bias, or withhold the output. Earlier works by Dodis [5], Kuchta and Manulis [11] proposed DVRF setups. Later, a non-interactive construction of Hanke et al. [16] (Dfinity-DVRF) were proposed. Galindo et al. [6] introduced two notable constructions, DDH-DVRF, which is based on the Decisional Diffie-Hellman assumption, and GLOW-DVRF, a BLS pairing-based scheme, and provided the first systematic security analysis for fully distributed non-interactive DVRFs. GLOW-DVRF later became the essence of pairing-based DVRFs. Noting the high cost of pairing-based verifications, Ağırtaş et al. [1] proposed a DVRF with a compact proof (DVRFwCP) that requires two additional communication rounds during proof generation, but achieves a verification that is much cheaper compared to its pairing-based counterpart. Later, they introduced Icy-DVRF [2], which reduces the additional communication rounds by using the preprocessing technique of FROST [10], keeping the verification cost minimal.

There is a practical limitation to the utility of smart-contract-based VRF services, especially in gaming services: the requester cannot submit the request in advance to have it ready when the online game begins, because the VRF output is immediately made public on the blockchain upon protocol completion. This means the request must be synchronized with VRF generation, introducing latency and halting gameplay. Also, since the VRF output is public, the requester

is prohibited from reusing this VRF value in the future: they cannot use a pseudorandom number generator with the VRF output as the seed to generate multiple successive random values for later use. This forces gaming platforms to design their games so that players cannot exploit publicly available VRF inputs, ensuring that latency and cost overheads remain affordable. To overcome this critical limitation, the concept of Output-Private VRFs (Pri-VRF) and Output-Private DVRFs (Pri-DVRFs) were introduced by the FlexiRand [8] framework, which are used in the Supra VRF service [14]. FlexiRand ensures that while the pseudo-randomness and public verifiability properties of VRFs are maintained, only the requester can observe the final output. It achieves this through a novel cryptographic blinding and unblinding mechanism: the user blinds their input before sending it to the network, and the smart contract verifies a blinded version of the final output. This allows randomness to be generated asynchronously and reused securely without exposing it to the public. However, the distributed instantiations of this privacy primitive are based on GLOW-DVRF, which rely on computationally expensive bilinear pairings, limiting their practical efficiency in gas-constrained blockchain environments.

In this work, we propose IcyVeil, an *output-private* distributed verifiable random function that fuses the output privacy of FlexiRand with the verification efficiency of Icy-DVRF. IcyVeil integrates blinding and unblinding operations directly into the pairing-free architecture of Icy-DVRF. Under our protocol, the user blinds their initial input with a private nonce, and the distributed VRF committee computes partial evaluations and Non-Interactive Zero-Knowledge (NIZK) proofs of equality of discrete logarithms over the blinded values. The aggregated *veiled* proof is verified efficiently without pairings and is subsequently *unveiled* exclusively by the requester. This partnership allows IcyVeil to deliver unpredictable, output-private randomness with constant-size proofs. Although Icy-DVRF, and hence IcyVeil, incurs an additional communication round among the DVRF clan compared to Flexirand, the veiling/unveiling mechanism eliminates the latency handicap of this interaction, resulting in a highly scalable and cost-effective DVRF primitive for decentralized applications.

2 Use Cases and Framework

Our main motivation in this work is to provide *latency-free* and *scalable* DVRF access to a diverse range of applications, with a primary focus on Web3 gaming services. As noted in Kate et al. [8], DeFi Kingdoms [9] uses Supra dVRF [14] for hero summoning and pet hatching. In addition, Illuvium [7] uses Chainlink VRF service [3] for NFT attribute generation and raffles. A variety of prominent Web3 titles, such as Gods Unchained, Alien Worlds, and Sorare, as well as expansive ecosystems like The Sandbox and Axie Infinity, rely on core random mechanics that would gain significant credibility through public verifiability.

We use a smart contract framework similar to Flexirand [8] for IcyVeil, which comprises three entities: the **User**, the **Smart Contract**, and the **VRF Clan**.

1. **User** initiates the process, sending an initial input α to **Smart Contract**.

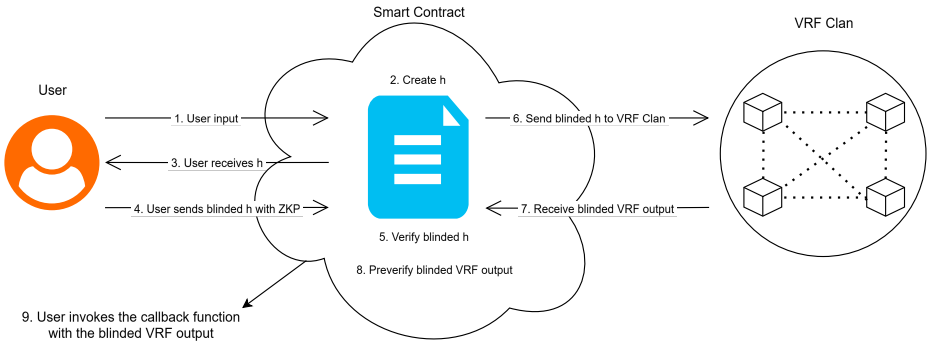


Fig. 1. Message flow for IcyVeil via Smart Contract

2. **Smart Contract** processes this input by combining it with metadata like block hashes and nonces to create a formatted VRF input, referred to as h.
3. **User** fetches h from **Smart Contract**.
4. **User** *blinds* h by applying a secret random masking factor κ. They then send this blinded h, named $\hat{h}$, back to the **Smart Contract** along with a NIZK proof of knowledge of κ. This proof allows the service to verify that the blinding was done correctly.
5. **Smart Contract** verifies the zero-knowledge proof of correct blinding. If verification fails, **Smart Contract** halts the process.
6. **VRF Clan**, which is a committee of nodes, fetches $\hat{h}$ from **Smart Contract**.
7. **VRF Clan** performs the following.
   (a) Each node in **VRF Clan** verifies the blinding ZKP.
   (b) If valid, **VRF Clan** proceeds to generate Icy-DVRF output on input of $\hat{h}$.
   (c) Upon success, **VRF Clan** submits the blinded output $\hat{\gamma}$ to **Smart Contract**.
8. **Smart Contract** performs a "pre-Verification" of the blinded output. Because the output remains blinded, the contract can verify that it was computed correctly using the **VRF Clan**'s group public key without knowing the random value. If this pre-verification evaluates to false, **Smart Contract** halts the process.
9. **Smart Contract** stores the blinded DVRF output $\hat{\gamma}$ along with its proof components, i.e., $\mathit{\Pi} = (\hat{\gamma}, c, \hat{z})$.

At the end of this procedure, the **User** can use their secret masking factor to *unblind* the value they received from the **Smart Contract**, finally obtaining the actual DVRF output for their private use. During the unblinding process, the **User** generates an NIZK proof of discrete logarithm equality, proving the integrity of the blinding and unblinding operations. When **User** wants to use the unblinded DVRF output, they have the option to publicize this NIZK proof for verification. While the user could alternatively reveal the secret masking

factor to verify the computation, the NIZK proof is the preferred method for maintaining privacy.

It is worth mentioning that, in this framework, an entity cannot forge ownership of the initial input via replay attacks, since the smart contract carefully constructs h from a context string that reflects not only the owner's identity but also other environment-related nonces.

3 Preliminaries

Suppose $\mathbb{G} = \langle g \rangle$ is the cyclic group of prime order q in multiplicative notation. Then, any element $x \in \mathbb{G}$ can be written as $x = g^k$ for some $k \in \mathbb{Z}_q$; the integer k is called *the discrete logarithm of x with respect to g*, denoted $\log_g x$. Consider a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ modeled as a random oracle.

3.1 NIZK Proof for Knowledge of Exponent

We present the non-interactive zero-knowledge (NIZK) proof for the knowledge of the exponent in the statement $x = g^k$, introduced by Schnorr [13].

- $\text{ProveExp}_H(g, x; k)$ chooses $r \xleftarrow{\$} \mathbb{Z}_q$, computes $u \leftarrow g^r$ and sets $c \leftarrow H(g, x, u)$. The output is a non-interactive proof $\pi = (c, s)$, where $s = r - k \cdot c$.
- $\text{VerifyExp}_H(g, x, \pi)$ parses $\pi = (c, s)$, computes $\bar{u} \leftarrow g^s x^c$, and outputs 1 if $c \leftarrow H(g, x, \bar{u})$; otherwise, outputs 0.

3.2 NIZK Proof for Equality of Discrete Logarithms

The Discrete Logarithm Equality (DLEQ) proof system by Chaum and Pedersen [4], is a pair of algorithms $(\text{ProveEq}_H, \text{VerifyEq}_H)$ to create a Non-Interactive Zero-Knowledge (NIZK) proof of $k = \log_g x = \log_h y$, as detailed below.

- $\text{ProveEq}_H(g, h, x, y; k)$ chooses $r \xleftarrow{\$} \mathbb{Z}_q$, computes $u \leftarrow g^r$, $v \leftarrow h^r$ and sets $c \leftarrow H(g, h, x, y, u, v)$. The output is a non-interactive proof $\pi = (c, s)$, where $s = r - k \cdot c$.
- $\text{VerifyEq}_H(g, h, x, y, \pi)$ parses $\pi = (c, s)$, computes $\bar{u} \leftarrow g^s x^c$, $\bar{v} \leftarrow h^s y^c$, and outputs 1 if $c = H(g, h, x, y, \bar{u}, \bar{v})$; otherwise, outputs 0.

3.3 Security Assumptions

The security guarantees of our protocol depend on well-established discrete logarithm hardness assumptions. We outline the underlying computational framework and relevant intractability assumptions in the following definitions.

Definition 1 *Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. Let $X \leftarrow \left(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}\right)$ where $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q^*$. Let $\mathcal{A}$ be a probabilistic polynomial-time (PPT) adversary. The* Decisional Diffie-Hellman *(DDH) assumption holds if for $\gamma \xleftarrow{\$} \mathbb{Z}_q^*$ the value*

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{DDH}}(\lambda) = \left|\Pr\left[\mathcal{A}\left(X, g^{\alpha\beta}\right) = 1\right] - \Pr\left[\mathcal{A}\left(X, g^{\gamma}\right) = 1\right]\right|$$

is negligible in λ.

Definition 2 *Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. Let $X \leftarrow \left(\mathbb{G}, q, g, g^{\alpha}, g^{\beta}\right)$ where $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q^*$. Let $\mathcal{A}$ be a (PPT) adversary. The* Computational Diffie-Hellman *assumption holds if the value*

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{CDH}}(\lambda) = \Pr\left[\mathcal{A}(X) = g^{\alpha\beta}\right]$$

is negligible in λ.

Definition 3 *Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. Let $\mathcal{A}$ be a PPT adversary with access to a challenge oracle* CHAL *and a CDH oracle* CDH*. The* CHAL *oracle, on each call i, samples $\alpha_i, \beta_i \xleftarrow{\$} \mathbb{Z}_q^*$ and returns $(g^{\alpha_i}, g^{\beta_i})$. The* CDH *oracle, on input (X, Y), returns g^{xy} where $x = \log_g X$ and $y = \log_g Y$. Let n be the number of calls to* CHAL *and m be the number of calls to* CDH*. The One-More Computational Diffie-Hellman assumption holds if the value $\mathrm{Adv}_{\mathcal{A}}^{\mathrm{OMCDH}}(\lambda)$ defined by*

$$\Pr\left[\mathcal{A}^{\mathsf{CHAL},\mathsf{CDH}}(\mathbb{G}, q, g) = (g^{\alpha_1\beta_1}, \ldots, g^{\alpha_n\beta_n}) \wedge m < n\right]$$

is negligible in λ.

4 IcyVeil

In the interest of presenting a self-contained algorithm, the notation and setup fundamental to the construction are detailed below.

Let $(\mathbb{G}, g, q)$ be a multiplicative group where the DDH assumption holds. Let $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_3 : \{0,1\}^* \to \{0,1\}^{\Lambda}$ be hash functions, where Λ denotes the desired output length. Additionally, assume that we have a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of participants, the *DVRF clan*, who communicate using the secure implementation practices described earlier. Also, assume that we have a user U who requests the blinded DVRF value and is likely not in $\mathcal{P}$. Then, let $\mathcal{V}^{\mathrm{IcyVeil}}$ be defined using the following nine algorithms, where the algorithms Blind and Unblind are called only by U. We envision that the DVRF clan is formed before U submits a request.

DistKeyGen:

Round 1

1. Every participant P_i samples t random values $\left(a_{i,0}, \ldots, a_{i,(t-1))}\right) \xleftarrow{\$} \mathbb{Z}_q$, and uses these values as coefficients to define a degree $t-1$ polynomial

$$f_i(x) = \sum_{j=0}^{t-1} a_{i,j}\, x^j \ .$$

2. Every participant P_i computes a proof of knowledge of their secret $a_{i,0}$ by calculating $\sigma_i = (R_i, \mu_i)$, where $k \xleftarrow{\$} \mathbb{Z}_q$, and $R_i = g^k$, $c_i = H_2\left(i, \Phi, g^{a_{i,0}}, R_i\right)$, $\mu_i = k - a_{i,0} \cdot c_i$, where Φ is a context string to prevent replay attacks.
3. Every participant P_i computes public commitments $\boldsymbol{C}_i = \left\langle \phi_{i,0}, \ldots, \phi_{i,(t-1)} \right\rangle$, where $\phi_{i,j} = g^{a_{i,j}}, 0 \leq j \leq t-1$. In particular, note that $\phi_{i,0} = g^{a_{i,0}}$.
4. Every P_i broadcasts $\boldsymbol{C}_i, \sigma_i$ to all other participants, or to the public.
5. Upon receiving $\boldsymbol{C}_\ell, \sigma_\ell$ from participants $1 \leq \ell \leq n, \ell \neq i$, each participant P_i verifies $\sigma_\ell = (R_\ell, \mu_\ell)$, aborting on failure, by checking $R_\ell \stackrel{?}{=} g^{\mu_\ell} \cdot \phi_{\ell,0}^{c_\ell}$, where $c_\ell = H_2\left(\ell, \Phi, \phi_{\ell,0}, R_\ell\right)$. Upon success, participants delete $\{\sigma_\ell : 1 \leq \ell \leq n\}$.

Round 2

1. Each P_i securely sends to each other participant P_ℓ a secret share $(\ell, f_i(\ell))$, deleting f_i and each share afterward except for $(i, f_i(i))$, which they keep for themselves.
2. Each P_i verifies their shares by calculating

$$g^{f_\ell(i)} \stackrel{?}{=} \prod_{k=0}^{t-1} \left(\phi_{\ell,k}\right)^{i^k \bmod q}$$

   aborting if the check fails.
3. Each P_i calculates their long-lived private signing share by computing $s_i = \sum_{\ell=1}^{n} f_\ell(i)$, stores s_i securely, and deletes each $f_\ell(i)$.
4. Each P_i calculates their public verification share $Y_i = g^{s_i}$, and the group's public key $Y = \prod_{j=1}^{n} \phi_{j,0}$. Any participant can compute the public verification share of any other participant by calculating

$$Y_i = \prod_{j=1}^{n} \prod_{k=0}^{t-1} \left(\phi_{j,k}\right)^{i^k \bmod q} .$$

Blind[4]:

On input $h \in \mathbb{G}$, the blinding algorithm proceeds as follows.

1. Choose a random secret value $\kappa \xleftarrow{\$} \mathbb{Z}_p$.
2. Compute $\hat{h} := h^\kappa$.
3. Compute a NIZK proof $\hat{\pi} \leftarrow \text{ProveExp}_{H_2}\left(h, \hat{h}; \kappa\right)$.
4. Output $(\hat{h}, \hat{\pi})$.

Preprocess:

Each participant P_i for $i \in \{1, \ldots, n\}$ performs the steps below. Let j be a counter for a specific nonce/commitment share pair, and Ω be the number of pairs generated at a time, such that Ω is the number of tries the combiner performs before performing another preprocessing step.

1. Create an empty list L_i. Then, for $1 \leq j \leq \Omega$, perform the following:

[4]This algorithm is executed exclusively by the user, who maintains κ confidential.

   a) Sample single-use nonces $(d_{i,j}, e_{i,j}) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.
   b) Derive commitment shares $(D_{i,j}, E_{i,j}) = \left(g^{d_{i,j}}, g^{e_{i,j}}\right)$.
   c) Append $(D_{i,j}, E_{i,j})$ to L_i. Store these values for later use.
2. Publish (i, L_i) to a predetermined location, as specified by the implementation.

PartialEval:

Let C be the combiner (who might be chosen amongst the participants). Let S be the set of θ participants selected for the generation of the DVRF value, where $t \leq \theta \leq n$, and let Y be the group public key set during the DistKeyGen phase.

1. C sends $\left(\left(\alpha, h, \hat{h}, \hat{\pi}\right), j, B\right)$ to $P_i,\ i \in S$, where $\left(\alpha, h, \hat{h}, \hat{\pi}\right)$, B is the ordered list of parties in S, and j is the index of the selected single-use commitments from the Preprocess stage.
2. Each P_i performs the following.
   (a) Checks if B is valid. If this check fails, abort.
   (b) Checks if $\text{VerifyExp}_{H_2}\left(h, \hat{h}; \hat{\pi}\right) = 1$. If this check fails, abort.
   (c) Computes partial evaluation $\hat{\gamma}_i = \hat{h}^{s_i} \in \mathbb{G}$.
   (d) Computes partial evaluation proof $\hat{\pi}_i \leftarrow \text{ProveEq}_{H_2}\left(g, \hat{h}, Y_i, \hat{\gamma}_i; s_i\right)$.
   (e) Computes $\bar{D}_i = \hat{h}^{d_{i,j}}$ and $\bar{E}_i = \hat{h}^{e_{i,j}}$.
   (f) Outputs $(\hat{\gamma}_i, \hat{\pi}_i, \bar{D}_i, \bar{E}_i)$.

PartialProofGen:

Now that each participant P_i published $\left(\hat{\gamma}_i, \hat{\pi}_i, D_i, E_i, \bar{D}_i, \bar{E}_i\right)$, each P_i produces their partial DVRF proof, using the following procedure.

1. Compute the set of binding values $\rho_\ell = H_2(\ell, \alpha, B)$ for all $\ell \in S$.
2. Compute $k_i = d_i + \rho_i \cdot e_i$.
3. Compute the (public) group commitments $R = \prod_{\ell \in S} D_\ell \cdot (E_\ell)^{\rho_\ell} = g^k$ and $\bar{R} = \prod_{\ell \in S} \bar{D}_\ell \cdot \left(\bar{E}_\ell\right)^{\rho_\ell} = \hat{h}^k$, where $k = \sum_{\ell \in S} k_\ell$, unknown to individual parties.
4. Compute $\hat{\gamma} = \prod_{\ell \in S} {\hat{\gamma}_\ell}^{\lambda_\ell}$, where λ_ℓ is the ℓ^{th} Lagrange coefficient in S.
5. Compute the group challenge $c = H_2\left(g, \hat{h}, Y, \hat{\gamma}, R, \bar{R}, \alpha\right)$.
6. Computes their response using their long-lived partial secret key s_i by computing the partial proof $\hat{z}_i = k_i + \lambda_i \cdot s_i \cdot c = d_i + \rho_i \cdot e_i + \lambda_i \cdot s_i \cdot c$, which is in additive form, where λ_i is the i^{th} Lagrange coefficient in S.
7. Securely deletes d_i, e_i from their local storage, and broadcasts $(\hat{z}_i, c)$.

Combine:

Upon receiving $(\hat{z}_\ell, c)$ from P_ℓ for $\ell \in S$, the combiner C performs the following actions.

1. Computes the following.[5]
   (a) $\rho_\ell = H_2(\ell, \alpha, B)$ for $\ell \in S$.
   (b) $R_\ell = D_\ell \cdot (E_\ell)^{\rho_\ell}$ for $\ell \in S$.
   (c) $\bar{R}_\ell = \bar{D}_\ell \cdot \left(\bar{E}_\ell\right)^{\rho_\ell}$ for $\ell \in S$.
   (d) The group commitment $R = \prod_{\ell \in S} R_\ell$.
   (e) The group commitment $\bar{R} = \prod_{\ell \in S} \bar{R}_\ell$.
   (f) $\hat{\gamma} = \prod_{\ell \in \mathcal{S}} \hat{\gamma_\ell}^{\lambda_\ell}$, where λ_ℓ is the ℓ^{th} Lagrange coefficient in S.
   (g) The group challenge $c = H_2\left(g, \hat{h}, Y, \hat{\gamma}, R, \bar{R}, \alpha\right)$.
2. Verifies the following.
   (a) Verifies that each c received from P_ℓ matches the c value the combiner computed in Step 1(g) above.
   (b) Verifies each partial evaluation proof by checking
   $$\text{VerifyEq}_{H_2}\left(g, \hat{h}, Y_\ell, \hat{\gamma_\ell}, \hat{\pi}_\ell\right) \stackrel{?}{=} 1$$
   for each $(\hat{\gamma_\ell}, \hat{\pi}_\ell), \ell \in S$, where Y_ℓ is the partial public key of P_ℓ.
   (c) Verifies the validity of the response of $P_\ell, \ell \in S$, by checking
   $$g^{\hat{z_\ell}} \stackrel{?}{=} R_\ell \cdot Y_\ell^{c \cdot \lambda_\ell}$$
   $$\hat{h}^{\hat{z_\ell}} \stackrel{?}{=} \bar{R}_\ell \cdot \hat{\gamma}_\ell^{c \cdot \lambda_\ell}$$
   for each $\hat{z_\ell}, \ell \in S$.

   If any of these checks fail for a participant, identify and report the misbehaving participant. At the end of this step, if at least t out of the θ members of S are not qualified, abort the protocol. Otherwise, continue.
3. Compute the group response $\hat{z} = \sum_{\ell \in S} \hat{z_\ell}$.
4. Output the core DVRF value $\hat{\gamma}$ and the blinded proof as $\mathit{\Pi} = (\hat{\gamma}, c, \hat{z})$.

PreVerify[6]: Upon receiving $\mathit{\Pi}$, the Smart Contract executes this algorithm and the pre-verification algorithm to take the input $\mathit{\Pi} = (\hat{\gamma}, c, \hat{z})$ and verify that $\hat{\gamma}$ and $\hat{z}$ were created properly for the input $\hat{h}$ by performing the following steps.

1. Compute $T = g^{\hat{z}} \cdot Y^{-c}$.
2. Compute $\bar{T} = \hat{h}^{\hat{z}} \cdot \gamma^{-c}$.
3. Check if $c \stackrel{?}{=} H_2\left(g, \hat{h}, Y, \hat{\gamma}, T, \bar{T}, \alpha\right)$. If the check is satisfied, output 1; otherwise, output 0.

Unblind[7]: The unblinding algorithm takes the blinded proof $\mathit{\Pi} = (\hat{\gamma}, c, \hat{z})$ and performs the following.

1. Check whether $\text{PreVerify}(\hat{\gamma}, c, \hat{z}) = 1$. If not, abort.[8]

[5] This step is optional if C is a participant.

[6] Anyone can execute this algorithm.

[7] This algorithm is executed exclusively by the user.

[8] This step is optional, as the operation is executed directly within the smart contract.

2. Use κ to unblind $\hat{\gamma}$, i.e., $\gamma = (\hat{\gamma})^{\kappa^{-1}}$.
3. Generate $\pi_{\text{eq}} = \text{ProveEq}_H(h, \gamma, \hat{h}, \hat{\gamma}; \kappa)$, proving that $\log_h \hat{h} = \kappa = \log_\gamma \hat{\gamma}$.
4. Compute the final DVRF output $\beta = H_3(\gamma)$.
5. Output $(\beta, \gamma, \pi_{\text{eq}})$.

Verify[6]:

Given the group public key Y, the blinded proof $\Pi = (\hat{\gamma}, c, \hat{z})$, and the Unblind output $(\beta, \gamma, \pi_{\text{eq}})$. Also, all parties have access to $\hat{h}, \hat{\gamma}$. Perform verification as follows.

1. $h \leftarrow H(\alpha)$
2. Check whether $\beta = H(\gamma)$.
3. Check whether $\text{PreVerify}(\hat{\gamma}, c, \hat{z}) = 1$. If not, abort.[8]
4. Check if $\text{VerifyEq}_H(h, \gamma, \hat{h}, \hat{\gamma}, \pi_{\text{eq}})$, guaranteeing $\log_h(\hat{h}) = \log_\gamma(\hat{\gamma})$.

5 Security of Output-Privacy

The security of IcyVeil is based on three hardness assumptions outlined in Section 3.3, each capturing a distinct aspect of the protocol. First, the **Decisional Diffie–Hellman (DDH)** assumption (Definition 1) upholds the pseudorandomness of the base Icy-DVRF construction: it ensures that the blinded partial evaluations and the group challenge computed during PartialProofGen are computationally indistinguishable from random group elements, hence no adversary can predict or bias the DVRF output.

Second, **output-privacy**, i.e., the guarantee that an on-chain observer cannot recover the unblinded output $\gamma = h^s$ from the published pair $(\hat{h}, \hat{\gamma}) = (h^\kappa, h^{\kappa s})$, reduces to the **Computational Diffie–Hellman (CDH)** assumption (Definition 2). Intuitively, recovering h^s from h^κ and $h^{\kappa s}$ without knowing the user's secret blinding scalar κ is precisely the CDH problem in $\mathbb{G}$.

Third, **unforgeability**, i.e., the guarantee that an adversary who observes k valid blinded output pairs cannot produce a $(k+1)$-th valid pair for a fresh input, requires a stronger interactive assumption. If the adversary is purely external and observes only the smart contract state, this reduces to the **One-More CDH (OM-CDH)** assumption (Definition 3), since the clan submits only the final aggregated blinded output $\hat{\gamma} = \hat{h}^s$ on-chain and the threshold structure remains hidden. However, if the adversary additionally corrupts up to $f \leq t - 1$ clan members and observes their partial blinded outputs $\hat{h}^{s_i}$, the reduction requires the stronger **Threshold One-More CDH (T-OM-CDH)** assumption. In this setting, the adversary gains access to evaluations of the underlying Shamir polynomial at f points and must still be unable to reconstruct a new full evaluation $\hat{h}^s$ beyond those it has legitimately obtained.

6 Comparison

In this work, we introduce IcyVeil as a privacy-preserving framework seamlessly integrated into the pairing-free architecture of Icy-DVRF, much like FlexiRand

provides an output-privacy layer for the pairing-based GLOW-DVRF. Our protocols are instantiated on the BLS12-381 curve, a Type-3 pairing-friendly curve standardized in EIP-2537 [15]. Under the Symmetric External Diffie-Hellman (SXDH) assumption, the DDH problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. We further assume the hardness of the CDH and OM-CDH problems in $\mathbb{G}_1$ to guarantee protocol integrity. To optimize for gas efficiency, we map the underlying group $\mathbb{G}$ to the subgroup $\mathbb{G}_1$, since element representations are smaller and group operations are significantly cheaper than in $\mathbb{G}_2$.

As shown in Table 1, based on the findings established by Ağırtaş et al. [2], while GLOW-DVRF achieves constant-size proofs, its reliance on computationally expensive bilinear pairing operations drives its theoretical verification cost to roughly 201,280 gas [1]. By relying entirely on more efficient multiscalar multiplications and hashes, Icy-DVRF reduces the estimated baseline verification cost to approximately 51,792 gas, representing nearly a fourfold theoretical efficiency improvement.

Table 1. Theoretical gas cost estimations of verification costs on BLS12-381.

Gas cost	Verification cost	Calldata cost	Total cost
GLOW-DVRF	≈ 200000	1280	$\approx$ **201280**
Icy-DVRF	≈ 50000	1792	$\approx$ **51792**

The core divergence between Icy-DVRF and IcyVeil lies in how users interact with the VRF service smart contract. While Icy-DVRF evaluates a public input directly, IcyVeil requires the user to mask the formatted input h with a secret random scalar κ to produce a blinded value $\hat{h}$. To prevent malicious inputs, the user must also compute a Schnorr-based Non-Interactive Zero-Knowledge (NIZK) proof ($\hat{\pi}$) demonstrating knowledge of κ. The VRF Clan subsequently evaluates this blinded input and returns a blinded proof that only the user can locally unveil. Verifying this proof via VerifyExp requires the smart contract to compute $\overline{u} \leftarrow g^s \hat{h}^c$, which translates to a multiscalar multiplication in the $\mathbb{G}$ group, along with a hash evaluation.

Because IcyVeil, by design, delegates the unblinding phase to the user, the primary additional burden on the smart contract, compared to standard Icy-DVRF, is verifying the user's initial NIZK blinding proof ($\hat{\pi}$). Verifying this proof via VerifyExp requires the smart contract to compute $\overline{u} \leftarrow g^s \hat{h}^c$, translating to a multiscalar multiplication in the $\mathbb{G}$ group alongside a hash evaluation. Assuming the use of EIP-2537 precompiles on the BLS12-381 curve [15], where a $\mathbb{G}_1$ exponentiation costs approximately 12,000 gas, this extra blinding proof verification imposes a minimal theoretical overhead of roughly 12,000 to 15,000 gas per request.

Incorporating empirical benchmarks from the Sepolia testnet, Icy-DVRF operates at 88,803 gas compared to GLOW-DVRF's 155,838 gas, yielding a 43.02% reduction in total gas consumption [2]. This ensures that, even when account-

ing for IcyVeil's privacy-preserving blinding proof, the total on-chain execution remains highly cost-effective by bypassing the expensive pairing operations required for FlexiRand's output verification.

7 Conclusion

Unpredictable and verifiable randomness is an indispensable resource for decentralized applications, particularly within the rapidly expanding Web3 gaming ecosystem. However, the demand for asynchronous generation and output reusability has exposed the limitations of public-by-default randomness oracles. In this work, we introduced IcyVeil to address these challenges, offering a practical, output-private solution that has reduced on-chain gas costs.

By fusing the cryptographic blinding mechanisms of FlexiRand with the pairing-free architecture of Icy-DVRF, IcyVeil achieves output privacy without incurring the prohibitive on-chain verification costs associated with bilinear pairings. While our construction inherits the robustness trade-off of Icy-DVRF, it successfully preserves constant-size proofs and exceptional gas economy.

Ultimately, IcyVeil provides a highly efficient architectural alternative for decentralized networks where both on-chain execution costs and output confidentiality are critical constraints. Parallel research is actively investigating methods to make the underlying Icy-DVRF protocol fully robust without sacrificing its pairing-free efficiency. Future work will also explore deploying this architecture across diverse high-throughput blockchain ecosystems, with the aim of optimizing real-time integration and minimizing latency for production-scale Web3 applications.

References

1. Ağırtaş, A.R., Özer, A.B., Saygı, Z., Yayla, O.: Distributed verifiable random function with compact proof. In: Dolev, S., Elhadad, M., Kutyłowski, M., Persiano, G. (eds.) Cyber Security, Cryptology, and Machine Learning. pp. 119–134. Springer Nature Switzerland, Cham (2025)
2. Ağırtaş, A.R., Özer, A.B., Saygı, Z., Yayla, O.: Icy-DVRF: A distributed verifiable random function based on FROST signatures. Cryptology ePrint Archive, Paper 2026/969 (2026), https://eprint.iacr.org/2026/969, (unpublished)
3. 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
4. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Annual international cryptology conference. pp. 89–105. Springer (1992). https://doi.org/10.1007/3-540-48071-4˙7
5. Dodis, Y.: Efficient construction of (distributed) verifiable random functions. In: Public Key Cryptography—PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6. pp. 1–17. Springer (2002). https://doi.org/10.1007/3-540-36288-6˙1

6. 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
7. Illuvium: Illuvium. Technical Whitepaper (2024), https://docs.illuvium.io/illuvium-whitepaper
8. 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
9. Kingdoms, D.: Defi kingdoms blockchain. Technical Whitepaper (2025), https://docs.defikingdoms.com/how-defi-kingdoms-works/defi-kingdoms-blockchain
10. Komlo, C., Goldberg, I.: Frost: Flexible round-optimized schnorr threshold signatures. In: Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers. p. 34–65. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-81652-0‘¨2, https://doi.org/10.1007/978-3-030-81652-0_2
11. Kuchta, V., Manulis, M.: Unique aggregate signatures with applications to distributed verifiable random functions. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) Cryptology and Network Security. pp. 251–270. Springer International Publishing, Cham (2013). https://doi.org/10.1007/978-3-319-02937-5¨14
12. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039). pp. 120–130 (1999). https://doi.org/10.1109/SFFCS.1999.814584
13. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Advances in Cryptology—CRYPTO'89 Proceedings 9. pp. 239–252. Springer (1990)
14. Team, S.R.: Supra oracles: Bridging real-world data to the blockchain with speed and security. Technical Whitepaper (2023), https://supra.com/whitepapers
15. Vlasov, A., Olson, K., Stokes, A., Sanso, A.: Eip-2537: Precompile for bls12-381 curve operations [draft], ethereum improvement proposals, no. 2537 (February 2020), https://eips.ethereum.org/EIPS/eip-2537, last accessed 15/05/2026
16. Williams, D., et al.: The internet computer for geeks. In: DFINITY Foundation Whitepaper (2022), https://internetcomputer.org/whitepaper.pdf