Icy-DVRF: A Distributed Verifiable Random Function based on FROST signatures

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, 06420 Ankara, Türkiye
a.r.agirtas@gmail.com
[2] Institute of Applied Mathematics, Middle East Technical University, 06800 Ankara, Türkiye
abozer@gmail.com, oguz@metu.edu.tr
[3] Department of Mathematics, TOBB University of Economics and Technology, 06560 Ankara, Türkiye
zsaygi@etu.edu.tr

Published in IEEE Access, vol. 14, 2026. DOI: 10.1109/ACCESS.2026.3712805

Abstract. Unbiased and unpredictable randomness is a cornerstone of Web3 security, underpinning everything from consensus protocols to DeFi logic. Although Distributed Verifiable Random Functions (DVRFs) eliminate central points of failure, current designs often have to compromise performance. Most existing protocols are hindered by one of three limitations: proofs that scale linearly with the number of participants, high computational cost of bilinear pairings, or latency introduced by mandatory interactive steps during generation. In this work, we present Icy-DVRF, a protocol that improves DVRFwCP by employing a preprocessing scheme similar to FROST to reduce the number of interaction rounds among participants and lowering the additional communication cost from $O(n^2t)$ to $O(t)$ while maintaining constant-size proofs. The downside of our construction is that, relative to DDH-DVRF and GLOW-DVRF, this approach incurs an additional off-chain communication round due to the threshold structure of our non-interactive zero-knowledge proof. This architecture ensures that verification costs remain low, regardless of the set of participants. While theoretical estimates suggest verification costs of approximately one quarter of those of standard designs, our empirical benchmarks on the Sepolia testnet, utilizing the EIP-2537: Precompile for BLS12-381 curve operations, confirm that Icy-DVRF requires only 88,803 gas for full execution. This represents a significant 43.02% reduction in total gas consumption compared to existing pairing-based constructions, saving 67,035 gas per on-chain verification. Off-chain, eliminating DVRFwCP's Augmented Secure-DKG round yields a per-node speedup ranging from a factor of 1.46 at $(n, t) = (5, 3)$ to a factor of 4.43 at $(n, t) = (50, 34)$.

⋆ Corresponding author

Keywords: Blockchain · cryptography · decentralization · distributed verifiable random function · verifiable random function.

1 Introduction

The generation of unpredictable and statistically uniform random values is a fundamental challenge across numerous computational domains. Applications critically relying on true randomness range from cryptographic key generation and digital signing protocols to the operational integrity of online gaming and gambling platforms. In recent years, the advent and exponential growth of blockchain technologies and the broader Web3 ecosystem, particularly within decentralized finance (DeFi), have significantly increased the demand for trustworthy sources of randomness. Within these emerging paradigms, systems often involve multiple, mutually distrusting participants interacting via complex protocols. Consequently, ensuring that any randomness used is not merely unpredictable but also publicly verifiable and demonstrably unbiased towards any specific party shifts from being merely important to absolutely crucial for system integrity and fairness. This need is particularly noticeable in the consensus mechanisms of many blockchain platforms, where processes such as block producer selection frequently require a collective, tamper-proof method for sampling random values to prevent manipulation and ensure equitable participation. A common approach to mitigating reliance on a single, potentially corruptible trusted entity is to adopt cryptographic processes that enable the distributed generation of randomness, whose origin and properties can be publicly verified.

Addressing the need for verifiable randomness generation led to the introduction of Verifiable Random Functions (VRFs) by Micali, Rabin, and Vadhan [36]. Conceptually, a VRF operates as a public-key analogue to a keyed cryptographic hash function or a pseudorandom function equipped with a non-interactive proof of correctness. Given a secret key sk, on input α, a VRF computes a pseudorandom output β accompanied by a proof π. This proof π allows any third party possessing the corresponding public key pk to verify that β is indeed the unique and correct output associated with input α under the key pk, without revealing any information about the secret key sk. The realization that VRFs can be linked to unique digital signatures led to the early construction of VRFs based on RSA signatures. Subsequent research has yielded more efficient VRF schemes, notably the construction by Dodis and Yampolskiy [20], which uses bilinear pairings and collision-resistant hash functions. Further advancements, driven by the pursuit of practical efficiency, have led to constructions based on elliptic-curve cryptography, as introduced by Papadopoulos et al. [38] and later standardized at the IETF by Goldberg et al. [29]. Recognizing the impending threat posed by quantum computing, the research focus has also expanded to include post-quantum secure VRFs, with notable contributions including constructions based on symmetric primitives by Buser et al. [11] and lattice-based, few-time verifiable schemes explored by Esgin et al. [23]. This consistent evolution underscores

the cryptographic community's recognition of VRFs as a vital primitive and the ongoing effort to enhance their efficiency and security guarantees.

The practical deployment of VRFs within decentralized systems reveals a critical vulnerability when implemented centrally. Widespread adoption of VRFs is evident across the ecosystem, with platforms such as Algorand [28], Cardano [18], Polkadot [31], and Mina [9] using VRFs for consensus integrity, and services such as Chainlink [10] using VRFs for application-layer randomness. Additionally, newer high-throughput architectures, such as Sui [37] and Aptos [3], have integrated VRF primitives directly into their core developer tooling. Although popular, these architectures rely on a centralized or quasi-centralized model in which specific nodes hold the VRF secret key. This centralization fundamentally contradicts the fundamental principles of decentralization. It reintroduces a single point of failure: if such a node is compromised, the attacker obtains the secret key. Consequently, all future VRF outputs generated using this key become entirely predictable, nullifying the intended benefits of randomness for security, fairness, or unpredictability in applications such as leader election in block proposals, secure lotteries, or gaming outcomes. This inherent fragility necessitates the development and deployment of intrinsically decentralized VRF services, eliminating the reliance on any single trusted entity for key management and computation.

To address this critical gap, the concept of Distributed Verifiable Random Functions (DVRFs) has emerged. A DVRF protocol enables a group of participants to collectively compute a single VRF output (β) and its corresponding proof (π) for a given input (α). Crucially, this computation occurs without any single party possessing or reconstructing the secret key; instead, the key is typically shared among the participants using threshold cryptographic techniques. As long as a predetermined minimum number (threshold) of participants follow the protocol honestly, the resulting output β remains pseudorandom, unpredictable, and publicly verifiable using the group's single public key. This approach ensures that the system remains secure even if a limited number of participants are compromised. Therefore, DVRFs are essential for maintaining fairness, transparency, and security in multi-party decentralized applications, where unbiased, unpredictable randomness is paramount.

Parallel to ECVRF adoption, pairing-based threshold BLS signatures have emerged as the standard for distributed randomness beacons and high-availability oracle networks. This approach is exemplified by the Drand beacon [34] used by Filecoin and the League of Entropy, as well as by the consensus mechanisms of the Internet Computer [13] and by newer oracle solutions such as Supra [44]. Although Ethereum's Beacon Chain primarily utilizes a commit-reveal scheme, it similarly integrates BLS signatures to verify the validator entropy. The design and implementation of efficient and robust DVRF protocols to support such systems pose significant cryptographic and engineering challenges, particularly with respect to the communication overhead and computational complexity.

Early theoretical work by Dodis [19] laid the groundwork for these primitives but required a trusted dealer for setup. Kuchta and Manulis [33] later

proposed a generic DVRF construction leveraging unique aggregate signatures. Driven by the practical demands of networks such as the Internet Computer, non-interactive constructions have gained prominence, most notably the pairing-based DVRF introduced by Camenisch et al. [13] (Dfinity-DVRF). Galindo et al. [26] subsequently provided the first systematic security analysis for fully distributed non-interactive DVRFs, introducing two notable constructions: DDH-DVRF, based on the Decisional Diffie-Hellman assumption, and GLOW-DVRF, another pairing-based scheme.

Despite significant progress, these constructions exhibit notable drawbacks. DDH-DVRF suffers from proofs whose size scales linearly with the threshold parameter t, directly impacting the verification efficiency. On the other hand, pairing-based constructions like GLOW-DVRF and Dfinity-DVRF achieve constant-size proofs but rely on computationally expensive bilinear pairing operations during verification. Subsequent research has aimed to improve previous constructions in the following fashion: Kate et al. [30] introduced FlexiRand to enhance output privacy; Patel et al. [39] proposed RB-F-DVRF to improve setup and reconstruction in large-scale settings; and Liu and Manulis [35] integrated strategies from SNARKs and secret sharing. To eliminate expensive pairings, Ağırtaş et al. [1] proposed a DVRF with a compact proof (DVRFwCP); however, this design requires two additional communication rounds during proof generation.

In this study, we introduce Icy-DVRF, a distributed verifiable random function that redesigns DVRFwCP by integrating the preprocessing and binding mechanisms of the FROST signature scheme, yielding compact proofs that do not rely on computationally expensive bilinear pairings. Our construction successfully reduces one round of intermediate interaction previously required by the DVRFwCP scheme, while also reducing the overhead communication cost of this interaction from $O(n^2t)$ to $O(t)$. Although our approach still requires an additional round of interaction amongst the t provers to derive a shared challenge for the NIZK proof, thereby increasing the cost of combined proof generation, this tradeoff yields substantial benefits: it ensures that the proof size remains constant and significantly reduces the computational burden of the final verification. In the context of blockchain applications, this design is particularly advantageous since the distributed key generation and the requisite interactions can be performed off-chain, effectively mitigating the impact of the sizes of partial proofs and generation latency on the system's on-chain performance. Our empirical benchmarks on the Sepolia testnet, utilizing the EIP-2537: Precompile for BLS12-381 curve operations, show that Icy-DVRF requires only 88,803 gas for full execution, while GLOW-DVRF requires 155,838. Hence, Icy-DVRF achieves a 43.02% reduction in total gas consumption compared to GLOW-DVRF, the building block of all existing pairing-based DVRF constructions, saving 67,035 gas per on-chain verification. Off-chain, eliminating DVRFwCP's Augmented Secure-DKG round yields total per-request speedups of a factor of 1.40 at $(n, t) = (5, 3)$, a factor of 1.43 at $(n, t) = (7, 4)$, and a factor of 7.14 at $(n, t) = (50, 34)$, with corresponding per-node speedups of 1.46, 1.48, and 4.43.

The remainder of this paper is organized as follows: Section 2 establishes the cryptographic foundations, covering verifiable secret sharing, distributed key generation, and essential VRF primitives. Section 3 formally presents the Icy-DVRF protocol. In Section 4, we prove the security of the Icy-DVRF protocol, deferring the lengthier portions of the proofs to the appendices. Section 5 provides a comparative analysis of our construction against established DVRF schemes and evaluates the required mathematical operations, communication costs, and estimated gas costs. Finally, Section 6 discusses real-world viability and deployment considerations, and Section 7 concludes the paper.

2 Preliminaries

We define the underlying hard problems, the formal definition of Distributed Verifiable Random Function (DVRF), and other cryptographic primitives used in our construction. We use the notation $r \xleftarrow{\$} S$ to mean that *the element r is chosen with uniform probability from the set S*. Let $\mathbb{Z}_q$ be the residue ring of prime order q.

2.1 Lagrangian Interpolation

For any polynomial $f(x) \in \mathbb{Z}_q[x]$ of degree at most $\theta-1$ and let $\mathcal{I} = \{p_1, \ldots, p_\theta\}$, where $1 \leq p_i \leq n$ is a reconstruction index set. $f(x)$ can be reconstructed as follows:

$$f(x) = \sum_{p_i \in \mathcal{I}} \lambda_{p_i,\mathcal{I}}(x)\ f(i),$$

where $\lambda_{p_i,\mathcal{I}}(x) = \prod_{p_k \in \mathcal{I} \setminus \{p_i\}} \frac{x-p_k}{p_i-p_k} \in \mathbb{Z}_q[x]$ are the *Lagrange basis polynomials*.

Specifically, one can reconstruct the constant term $f(0)$ using the *Lagrange coefficients* $\lambda_{p_i,\mathcal{I}} = \lambda_{p_i,\mathcal{I}}(0) \in \mathbb{Z}_q^*$ as

$$f(x) = \sum_{p_i \in \mathcal{I}} \lambda_{p_i,\mathcal{I}}\ f(p_i)\ .$$

It is worth noting that the Lagrange coefficients depend merely on the list $\{p_1, \ldots, p_\theta\} \subset \{1, \ldots, n\}$, and not on the values $f(p_1), \ldots, f(p_\theta)$; if n is relatively small, the Lagrange coefficients for every possible reconstruction set may be precomputed for optimization.

Lagrangian Interpolation, as its name suggests, is the process of reconstructing $f(x)$ and/or $f(0)$. We simplify the notation and write $\lambda_j = \lambda_{p_i,\mathcal{I}}$ for these coefficients whenever $\mathcal{I}$ is clear from the context.

2.2 NIZK Proof for Equality of Discrete Logarithms

Suppose $\mathbb{G} = \langle g \rangle$ is the cyclic group of prime order q in multiplicative notation. Then, any element $y \in \mathbb{G}$ can be written as $y = g^k$ for some $k \in \mathbb{Z}_q$; the integer k is called *the discrete logarithm of y with respect to g*, denoted $\log_g y$.

The Discrete Logarithm Equality (DLEQ) proof system by Chaum and Pedersen [15], is a pair of algorithms $(\mathrm{ProveEq}_H, \mathrm{VerifyEq}_H)$ to create a Non-Interactive Zero-Knowledge (NIZK) proof of $k = \log_g x = \log_h y$, as detailed below.

Consider a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ modeled as a random oracle.

- $\mathrm{ProveEq}_H(g, h, x, y; k)$ 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 $s = r - k \cdot c$.
- $\mathrm{VerifyEq}_H(g, h, x, y, \pi)$ 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.

2.3 Underlying Hard Problems

The security of our protocol relies on established hardness assumptions in the discrete logarithm setting. The following definitions establish the computational framework and intractability assumptions central to our protocol.

Definition 1. *Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. We say that the Discrete Logarithm (DL) assumption holds for $\mathbb{G}$ if for all PPT adversaries $\mathcal{A}$, the value*

$$\mathsf{Adv}^{\mathrm{DL}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[g^x = h : g \xleftarrow{\$} \mathbb{G}^*; h \xleftarrow{\$} \mathbb{G}; x \xleftarrow{\$} \mathcal{A}(h)\right]$$

is negligible.

The discrete logarithm problem (DLP) in $\mathbb{G}$ is for given $g, h \in \mathbb{G}$, to find an integer x (if it exists) such that $g^x = h$.

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 probabilistic polynomial-time (PPT) adversary. The Decisional Diffie-Hellman assumption holds if for $\gamma \xleftarrow{\$} \mathbb{Z}_q^*$ the value*

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

is negligible.

Definition 3. *Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. The One-More Discrete Logarithm (OMDL) assumption holds for $\mathbb{G}$ if for all PPT adversaries $\mathcal{A}$, the value*

$$\mathsf{Adv}^{\mathrm{OMDL}}_{\mathbb{G}}(\mathcal{A}) = \Pr\left[\mathcal{A} \text{ wins } \mathsf{G}^{\mathrm{omdl}}_{\mathbb{G}}\right]$$

is negligible, where the game $\mathsf{G}^{\mathrm{omdl}}_{\mathbb{G}}$ is defined as follows. The challenger initializes a challenge counter $c_{\mathrm{id}} \leftarrow 0$ and a discrete logarithm query counter $\ell \leftarrow 0$, and grants $\mathcal{A}$ access to two oracles:

- $\mathsf{Chal}()$*: samples $x \xleftarrow{\$} \mathbb{Z}_q$, increments $c_{\mathrm{id}} \leftarrow c_{\mathrm{id}} + 1$, stores $x_{c_{\mathrm{id}}} \leftarrow x$, and returns g^x.*

– $\mathsf{Dlog}(X)$*: on input* $X \in \mathbb{G}$*, increments* $\ell \leftarrow \ell + 1$ *and returns* $\log_g X$*.*

After interacting with these oracles, $\mathcal{A}$ *outputs* $(y_1, \ldots, y_{c_{\mathrm{id}}})$*, and wins if and only if* $\ell < c_{\mathrm{id}}$ *and* $y_i = x_i$ *for all* $i \in \{1, \ldots, c_{\mathrm{id}}\}$*. That is,* $\mathcal{A}$ *must compute the discrete logarithms of all* c_{id} *challenge points while making strictly fewer than* c_{id} *queries to the discrete logarithm oracle.*

We additionally rely on the Algebraic Group Model (AGM) [25], an idealized model in which every adversary is *algebraic*: whenever it outputs a group element $Z \in \mathbb{G}$, it also outputs a representation $(z_1, \ldots, z_k) \in \mathbb{Z}_q^k$ such that

$$Z = \prod_i g_i^{z_i} \ ,$$

where $g_1, \ldots, g_k$ are the group elements the adversary has received so far.

Hereafter, assume $\mathbb{G} = \langle g \rangle$ is a group of prime order q in which the DDH-assumption holds.

2.4 Verifiable Secret Sharing

A (t, n)-threshold scheme is a cryptographic protocol designed for sharing a secret s among a group of n participants so that while any t participants working together can reconstruct the secret, any subset containing fewer than t participants gains no usable information about the secret.

Shamir's Secret Sharing [43] allows a secret $s \in \mathbb{G}$ to be divided among n parties using a degree $t - 1$ polynomial, such that any t or more parties can reconstruct the secret using Lagrange interpolation. However, this scheme is vulnerable if the dealer or the dishonest parties provide incorrect shares. Feldman's Verifiable Secret Sharing (Feldman-VSS) [24] extends Shamir's scheme, allowing parties to verify the consistency of shares received from the dealer and identify erroneous shares during reconstruction. Pedersen's Verifiable Secret Sharing (Pedersen-VSS) [41] further enhances this procedure by integrating a second generator $\bar{g}$ to the cyclic group $\mathbb{G}$, thereby providing information-theoretic security for the secret.

2.5 Distributed Key Generation

Building on Feldman-VSS, Pedersen proposed the first distributed key generation (DKG) protocol [40], which involves each party running a parallel instance of Feldman-VSS as a dealer. Gennaro et al. [27] later demonstrated that Pedersen's DKG could enable an attacker to influence the distribution of the generated key pair, resulting in a non-uniform distribution. To address this, they proposed a new DKG protocol, known as the Secure-DKG protocol, which is now widely used in distributed systems, including the three DVRF protocols in Section 2.8. Similar to Pedersen-VSS, the Secure-DKG protocol incorporates a generator of $\bar{g} \in \mathbb{G}$ where computing $\log_g(\bar{g})$ is infeasible.

2.6 Flexible Round-Optimized Schnorr Threshold Signatures

The Flexible Round-Optimized Schnorr Threshold Signatures (FROST) protocol, developed by Komlo and Goldberg [32], represents a significant advancement in threshold cryptography. It effectively tackles the major obstacles that previously limited Schnorr-based threshold schemes by offering improved round efficiency (one round with preprocessing or two rounds), which is crucial for performance. FROST introduced a novel *nonce binding* technique that ensures provable security against concurrent execution attacks, such as those by Drijvers et al. [21] and the attack via ROS Solver by Benhamouda et al. [7], which are significant vulnerabilities in earlier protocols.

FROST uses a Signature Aggregator (SA) for efficiency, which can be one of the signers, collecting, verifying, and combining the partial signatures provided by the participating signers during the final signing stage. The SA, which can be any participant or external party with information of the public key shares, is responsible for collecting signatures, reporting misbehaving participants, and publishing the final group signature. While trusted for these tasks, a malicious SA cannot compromise the private key or cause improper messages to be signed. It is limited to performing denial-of-service attacks or falsely reporting misbehavior. The FROST protocol remains secure against adaptive attacks, even if the SA is malicious, because its view is not more privileged than that of a standard adversary.

FROST consciously trades robustness for efficiency. Unlike protocols that guarantee a valid signature output even if up to $t-1$ participants misbehave (often requiring additional communication rounds or complexity), FROST adopts an *abort and identify* model. If any participant submits an invalid partial signature during the signing process, the SA detects this during the mandatory verification step. The protocol instance is then aborted for that session, and the misbehaving participant is identified. Honest participants could subsequently retry the signing operation, potentially excluding the identified faulty party. This is a pragmatic choice suitable for practical deployments where malicious behavior or failures are considered exceptional events, allowing for optimized performance in the common, optimistic case where all participants follow the protocol correctly.

2.7 Adversarial and Network Model

We consider a distributed system with n participants, where a threshold t of honest parties is required for successful protocol execution. Our security analysis assumes the following.

- **Network Synchrony:** The protocol operates in a partially synchronous setting. While the core cryptographic primitives (NIZK proofs) are non-interactive, the coordination between the provers and the Combiner assumes that messages are delivered within a maximum delay to ensure liveness.

- **Communication Channels:** Participants are connected via authenticated secure channels to prevent impersonation and man-in-the-middle attacks. We also assume a reliable broadcast medium for the Preprocess and PartialProofGen phases.
- **Adversary:** We assume a PPT adversary that can statically corrupt up to $t-1$ nodes. The adversary has full control over the network but cannot break the underlying hard problems, such as the Discrete Logarithm Problem or Decisional Diffie-Hellman problem.

2.8 Distributed Verifiable Random Functions

The foundation of our study is based on various distributed verifiable random functions. We discuss and briefly explain them to understand the difficulties and enhancements of the proposed novel DVRF algorithm.

All the DVRF protocols mentioned below share a similar setup where $(\mathbb{G}, g, q)$ is a multiplicative group of prime order q where the DDH assumption holds. In addition, let all hash functions mentioned implicitly be either from $\mathbb{Z}_q$ to binary strings or vice versa, chosen appropriately.

DDH-DVRF Proposed by Galindo et al. [26] in 2020, the DDH-DVRF algorithm, secured by the Decisional Diffie-Hellman (DDH) assumption, constructs a distributed verifiable random function without relying on bilinear pairings. Its security stems from the computational difficulty of distinguishing specific cryptographic tuples, enabling the generation of strong pseudorandomness through collaborative efforts. This makes it suitable for environments where pairing-based cryptography is less desirable.

The algorithm operates in four phases: DistKG establishes a shared secret and public keys among multiple nodes using verifiable secret sharing, as described in [27]. In PartialEval, each node computes a partial random value and generates a Non-Interactive Zero-Knowledge proof of correctness. The Combine phase combines the partial evaluations. It reconstructs the final verifiable random value from a threshold of valid partial evaluations using Lagrange interpolation, and the combined proof is the concatenation of the individual proofs into a list. Finally, the Verify process uses the global public key and participant verification keys to verify the validity of the partial and combined final random values and the combined proof, ensuring public verifiability.

The drawback of this construction is that the combined proof size grows linearly with the threshold number of participants t, and the verification algorithm must verify each public partial evaluation separately, which makes verification slow and costly, and places a burden on the public verifiers.

GLOW-DVRF The GLOW-DVRF algorithm is a pairing-based DVRF, similar to DDH-DVRF, which leverages bilinear pairings and BLS signatures [8] to achieve a distributed verifiable random function. Bilinear pairings are mathematical functions that take two elements as input, one from a cryptographic

group $\mathbb{G}_1$ and another from a group $\mathbb{G}_2$, and map them to an element in a third group $\mathbb{G}_T$, while satisfying specific bilinear properties. These properties enable the verification of intricate relationships between elements across these groups. The security of GLOW-DVRF relies on additional assumptions, as elaborated in the original publication.

The GLOW-DVRF algorithm begins with DistKG, a Distributed Key Generation phase, similar to DDH-DVRF, where participating nodes establish a shared secret key using individual verification keys in $\mathbb{G}_1$ and the global public key in $\mathbb{G}_2$. Following this, PartialEval is very similar to that of DDH-DVRF. The partial evaluations are then combined in the Combine phase, requiring a threshold of valid shares to reconstruct an intermediate value, which is then hashed to obtain the final pseudorandom value and compact proof. The Verify process used in GLOW-DVRF relies on checking a specific bilinear pairing equation involving the proof, the input message, and the global public key.

A key distinction between GLOW-DVRF and DDH-DVRF lies in the proof size, where, unlike DDH-DVRF, GLOW-DVRF achieves a significant advantage by offering constant-size proofs. This makes GLOW-DVRF particularly beneficial in applications where bandwidth and storage efficiency are critical, such as in blockchain technology. However, the involvement of computationally intensive bilinear pairings leads to increased gas fees on blockchains and introduces complex implementation challenges.

DVRFwCP In 2024, Ağırtaş et al. [1] proposed a DVRF scheme with compact proofs combining the techniques of DDH-DVRF and a threshold NIZK system. DVRFwCP achieves compact proofs, similar to those of GLOW-DVRF, but features a verification system that does not rely on expensive bilinear pairings.

This scheme also uses the Secure-DKG protocol [27] for the distributed key generation phase (DistKeyGen), followed by PartialEval to produce individual VRF outputs and EvalCombine to produce the interim combined output. To enable a t-out-of-n threshold NIZK proof, a joint challenge must be created in PartialProofGen, requiring an additional interaction using an augmented Secure-DKG protocol. Using Lagrange interpolation, ProofCombine merges the partial proofs into a concise proof, and the interim VRF output is hashed to create the final pseudorandom value. The Verify algorithm merely needs to check a single NIZK proof.

DVRFwCP offers a notable design improvement by eschewing bilinear pairings, in contrast to schemes such as GLOW-DVRF. Instead, it relies on the DDH assumption and achieves a constant proof size, distinguishing it from DDH-DVRF. A trade-off of this construction is the requirement of an additional interactive phase, consisting of two communication rounds, with $\mathcal{O}(n^2)$ complexity, as it necessitates running an augmented version of Secure-DKG prior to generating the final proof, unlike the aforementioned DVRF schemes.

3 Icy-DVRF

We propose Icy-DVRF, a DVRF system similar to DVRFwCP [1], that leverages the preprocessing and nonce-binding features of FROST [32] to reduce the number of interactions required in DVRFwCP. Before presenting our proposed scheme, we formally define the Icy-Distributed Verifiable Random Function as follows.

Definition 4. *(Icy-Distributed Verifiable Random Function) A t-out-of-n Icy-DVRF $\mathcal{V}$ consists of six algorithms, i.e., $\mathcal{V} =$ (**DistKeyGen**, **Preprocess**, **PartialEval**, **PartialProofGen**, **Combine**, **Verify**), described below.*

- ***DistKeyGen*** $\left(1^{\lambda}, t, n\right) \rightarrow \left(\left\{(s_i, Y_i)\right\}_{i\in[n]}, Y\right)$ *takes a security parameter λ, the threshold parameter t and the number of participating nodes n as inputs, and outputs a list of partial secret-public key pairs $\left\{(s_i, Y_i)\right\}_{i\in[n]}$ and a group public key Y.*
- ***Preprocess*** $(\Omega) \rightarrow L_i = \left\{(D_{i,j}, E_{i,j})\right\}_{j=1}^{\Omega}$ *takes the number of commitment shares to be generated Ω, outputs a list L_i of commitment pairs of the form $(D_{i,j}, E_{i,j})$ for $1 \leq j \leq \Omega$.*
- ***PartialEval*** $(\alpha, j, B, s_i, Y_i) \rightarrow \left(\gamma_i, \pi_i, \bar{D}_i, \bar{E}_i\right)$ *takes a plaintext α, a single-use index j, the ordered prover list B, a partial secret and verification keys (s_i, Y_i) as inputs, and outputs a partial evaluation γ_i, a partial evaluation NIZK proof π_i, and auxiliary values $(\bar{D}_i, \bar{E}_i)$.*
- ***PartialProofGen*** $\left(\alpha, B, S, s_i, Y_i, Y, \left(D_i, E_i, \bar{D}_i, \bar{E}_i\right)_{i\in S}\right) \rightarrow (z_i, c)$ *takes a plaintext α, the ordered prover list B, the prover set S, a partial secret/public key pair (s_i, Y_i), public key Y, a commitment list of all users $(D_i, E_i)_{i\in S}$ and auxiliary values $(\bar{D}_i, \bar{E}_i)_{i\in S}$ as inputs and outputs a partial NIZK proof (z_i, c).*
- ***Combine*** $\left(\alpha, (z_i, c)_{i\in S}, Y, \left(D_i, E_i, \bar{D}_i, \bar{E}_i\right)_{i\in S}\right)$
  $\rightarrow (\beta, (\gamma, c, z))$ *takes a plaintext α, a list of partial NIZK proofs $(z_i, c)_{i\in S}$, the public key Y, a commitment list of all users $(D_i, E_i)_{i\in S}$ and auxiliary values $(\bar{D}_i, \bar{E}_i)_{i\in S}$, outputs either a DVRF value β with a combined NIZK proof (γ, c, z) or aborts.*
- ***Verify*** $(\alpha, Y, \beta, (\gamma, c, z)) \rightarrow 0/1$ *takes a plaintext α, a public key Y, a DVRF value β, a NIZK proof (γ, c, z) as inputs, it outputs $0/1$.*

A DVRF should at least satisfy the following security properties.

- *Correctness*: This requires that, a correctly generated VRF value β always passes verification.
- *Consistency:* This means that, for a given input α, successful execution of the protocol should produce the same VRF output β, regardless of the collection of correctly formed shares used.
- *Public verifiability*: Anyone should be able to verify the proof for the correctness of the pseudorandom value β, removing the need to trust the parties involved in the VRF value or proof generation.

- *Uniqueness*: For a given input α, the VRF output value β should be deterministic. This is particularly vital in decentralized applications, as it is crucial for selecting block proposers and validators, preventing forks, and ensuring consensus.
- *Pseudorandomness*: This ensures that the output VRF value β is unpredictable before its creation and that its distribution is statistically indistinguishable from random. This property also implies that no one can bias the output in their favor.

Implementations must ensure that participants are identifiable within the group so that misbehavior can be reported, communicate via a secure, authenticated channel, and validate messages before use.

3.1 Domain Separation of Hash Functions

To ensure cryptographic hygiene and prevent the reuse of one random-oracle output in the role of another, all random oracles in Icy-DVRF are strictly domain-separated with unique context tags. Throughout the paper the shorthands H_1, H_2, H_3 are used for typographical clarity; each is instantiated through a single cryptographically secure function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^*$ (mapped into the relevant range) under an explicit Domain Separation Tag (DST). Crucially, H_2 is queried in three structurally distinct roles, each of which receives its own tag so that their query spaces are disjoint:

- $\mathbf{H_1}(\alpha)$ maps the input α to a group element in $\mathbb{G}_1$:
$$H_1(\alpha) = \mathcal{H}(\texttt{"ICY-MAP2G1"} \parallel \alpha).$$
- $\mathbf{H_2^{pok}}(\mathbf{i}, \mathbf{\Phi}, \phi_{\mathbf{i},\mathbf{0}}, \mathbf{R_i})$ is the Schnorr proof-of-knowledge challenge $c_i \in \mathbb{Z}_q$ in DistKeyGen:
$$c_i = \mathcal{H}(\texttt{"ICY-POK-CHL"} \parallel i \parallel \Phi \parallel \phi_{i,0} \parallel R_i).$$
- $\mathbf{H_2^{bind}}(\ell, \alpha, \mathbf{B})$ is the per-prover FROST-style binding value $\rho_\ell \in \mathbb{Z}_q$:
$$\rho_\ell = \mathcal{H}(\texttt{"ICY-FROST-BND"} \parallel \ell \parallel \alpha \parallel B).$$
- $\mathbf{H_2^{chl}}(\mathbf{g}, \mathbf{h}, \mathbf{Y}, \gamma, \mathbf{R}, \bar{\mathbf{R}}, \alpha)$ is the aggregate T-DLEQ proof challenge $c \in \mathbb{Z}_q$:
$$c = \mathcal{H}(\texttt{"ICY-DLEQ-CHL"} \parallel g \parallel h \parallel Y \parallel \gamma \parallel R \parallel \bar{R} \parallel \alpha).$$
- $\mathbf{H_3}(\gamma)$ is the final pseudorandom output $\beta \in \{0,1\}^\Lambda$:
$$\beta = \mathcal{H}(\texttt{"ICY-OUTPUT"} \parallel \gamma).$$

The three H_2 roles share the notation H_2 but never the domain: the leading DST makes the binding, challenge, and proof-of-knowledge query spaces pairwise disjoint, so no binding value can ever coincide with a proof challenge. This is the separation we rely on in Appendix B, where the knowledge-soundness reduction programs the binding entries $H_2(\ell, \alpha, B)$ and the challenge entry $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$ independently and counts them as distinct H_2 entries. The interactive variant of T-DLEQ Appendix A additionally uses H_ρ, H_π, and H_4, each carrying its own DST under the same convention.

3.2 Icy-DVRF Algorithm

Our construction initially employs the FROST distributed key generation algorithm, rather than the Secure-DKG [27] used in DVRFwCP. We replace the Augmented Secure-DKG protocol used in DVRFwCP with a nonce-generation technique that incorporates elements of FROST's binding technique.

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 who communicate using the secure implementation practices described earlier. Let $\mathcal{V}^{\text{Icy-DVRF}}$ be defined using the following six algorithms.

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 $\phi_{i,j} = g^{a_{i,j}}, 0 \le j \le t-1$. 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 $1 \le \ell \le n, \ell \ne i$, 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 $\{\sigma_\ell : 1 \le \ell \le n\}$.

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

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

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:
   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 (α, j, B) to P_i, $i \in S$, where B is the ordered list of parties in S and j is the index of the single-use commitments from the Preprocess stage to be used.
2. Each P_i checks if B is valid and if α is the message they want to use. If the checks fail, abort.
3. Each P_i performs the following.
   (a) Compute $h \leftarrow H_1(\alpha) \in \mathbb{G} \setminus \{1\}$.
   (b) Compute partial evaluation $\gamma_i = h^{s_i} \in \mathbb{G}$.
   (c) Compute partial evaluation proof $\pi_i \leftarrow \mathrm{ProveEq}_{H_2}(g, h, Y_i, \gamma_i; s_i)$.
   (d) Compute $\bar{D}_i = h^{d_{i,j}}$ and $\bar{E}_i = h^{e_{i,j}}$.
   (e) Outputs $(\gamma_i, \pi_i, \bar{D}_i, \bar{E}_i)$.

PartialProofGen:

Now that each participant P_i published $\left(\gamma_i, \pi_i, D_i, E_i, \bar{D}_i, \bar{E}_i\right)$, 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
   $\bar{R} = \prod_{\ell \in S} \bar{D}_\ell \cdot \left(\bar{E}_\ell\right)^{\rho_\ell} = h^k$, where $k = \sum_{\ell \in S} k_\ell$, unknown to individual parties.
4. Compute $\gamma = \prod_{\ell \in S} \gamma_\ell^{\lambda_\ell}$, where λ_ℓ is the ℓth Lagrange coefficient in S.
5. Compute the group challenge $c = H_2\left(g, h, Y, \gamma, R, \bar{R}, \alpha\right)$.
6. Compute the response using their long-lived partial secret key s_i by computing the partial proof $z_i = k_i + \lambda_i \cdot s_i \cdot c = d_i + \rho_i \cdot e_i + \lambda_i \cdot s_i \cdot c$, which is in additive form, where λ_i is the ith Lagrange coefficient in S.
7. Securely delete d_i, e_i from their local storage, and broadcast (z_i, c).

Combine:

Upon receiving (z_ℓ, c) from P_ℓ for $\ell \in S$, the combiner C performs the following actions.

1. Computes the following.[4]
   (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) $\gamma = \prod_{\ell \in \mathcal{S}} \gamma_\ell^{\lambda_\ell}$, where λ_ℓ is the ℓth Lagrange coefficient in S.
   (g) The group challenge $c = H_2\left(g, h, Y, \gamma, R, \bar{R}, \alpha\right)$.
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

$$\text{VerifyEq}_{H_2}\left(g, h, Y_\ell, \gamma_\ell, \pi_\ell\right) \stackrel{?}{=} 1$$

   for each $(\gamma_\ell, \pi_\ell), \ell \in S$, 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

$$g^{z_\ell} \stackrel{?}{=} R_\ell \cdot Y_\ell^{c \cdot \lambda_\ell}$$
$$h^{z_\ell} \stackrel{?}{=} \bar{R}_\ell \cdot \gamma_\ell^{c \cdot \lambda_\ell}$$

   for each $z_\ell, \ell \in S$.

   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 $z = \sum_{\ell \in S} z_\ell$.
4. Output the VRF value $\beta \leftarrow H_3\left(\gamma\right)$ and the proof as $\pi = (\gamma, c, z)$.

Verify:

Given the group public key Y, the proof $\pi = (\gamma, c, z)$, the VRF value β and the message α, perform verification as follows:

[4] This step can be skipped entirely if C is one of the participants.

1. Compute $T = g^z \cdot Y^{-c}$.
2. Compute $h = H_1(\alpha)$ and $\bar{T} = h^z \cdot \gamma^{-c}$.
3. Check if $c \stackrel{?}{=} H_2(g, h, Y, \gamma, T, \bar{T}, \alpha)$ and $\beta \stackrel{?}{=} H_3(\gamma)$. If both are satisfied, output 1; otherwise, output 0.

We remark that the distributed management of preprocessed commitment lists L_i requires strict synchronization to prevent protocol failure.

- **Counter-Based Synchronization:** To prevent desynchronization, the Combiner $\mathcal{C}$ includes a unique single-use index j in the initial request. Provers are required to use the commitment pair $(D_{i,j}, E_{i,j})$ corresponding to this index. If a node has already consumed index j or finds it out of sequence, it issues a synchronization error, prompting a state update.
- **Prevention of State Exhaustion:** A malicious actor might attempt to trigger the DVRF repeatedly to exhaust the pool of Ω preprocessed nonces. To counter this, Icy-DVRF implementations should employ a request-validation layer. Nonces are only consumed for requests that pass pre-verification (e.g., payment of a fee). Furthermore, the Preprocess phase is designed to be highly efficient, allowing nodes to replenish the pool L_i in the background before exhaustion occurs.

4 Security

4.1 Correctness

During the DistKeyGen phase, let $F_1(x) = \sum_{j=1}^{n} f_j(x)$ be the polynomial defining the secret sharing of the private signing key s, where $s_i = F_1(i)$ and the secret key is $s = F_1(0)$. Similarly, during the PartialEval phase, let S be the set of θ participants, with $t \leq \theta \leq n$ selected for proof generation, each using a pair of nonces (d_i, e_i) to define a degree $\theta - 1$ polynomial $F_2(x)$, interpolating the points $\left(i, \frac{1}{\lambda_{i,S}} \left(d_i + e_i \cdot H_2(\ell, \alpha, B)\right)\right)$, so that $F_2(0) = \sum_{i \in S} d_i + e_i \cdot \rho_i = \sum_{i \in S} k_i = k$.

Now let $F_3(x) = F_2(x) + c \cdot F_1(x)$, where $c = H_2\left(g, h, Y, \gamma, R, \bar{R}, \alpha\right)$. Then, $z_i = d_i + \rho_i \cdot e_i + \lambda_{i,S} \cdot s_i \cdot c = \lambda_{i,S} \left(F_2(i) + c \cdot F_1(i)\right) = \lambda_{i,S} F_3(i)$, hence $z = \sum_{i \in S} z_i$ is merely the Lagrange interpolation of $F_3(0) = k + c \cdot s$.

Since, $R = \prod_{i \in S} D_i \cdot (E_i)^{\rho_i} = g^{\sum_{i \in S} d_i + \rho_i \cdot e_i} = g^k$ and $\bar{R} = \prod_{i \in S} \bar{D}_i \cdot \left(\bar{E}_i\right)^{\rho_i} = h^{\sum_{i \in S} d_i + \rho_i \cdot e_i} = h^k$ and $c = H_2\left(g, h, Y, \gamma, R, \bar{R}, \alpha\right)$, $\pi = (\gamma, c, z)$ is a correctly formed VRF proof with VRF value $\beta = H_3(\gamma)$, and Verify evaluates to 1, proving the correctness property of the Icy-DVRF protocol.

4.2 Consistency

In the creation of the final VRF output $\beta = H_3(\gamma)$, once the DistKeyGen phase is complete, the secret share s_i of each participant $P_i \in \mathcal{P}$ is set, and

once the messages α and the set S are fixed in PartialEval, $h = H_1(\alpha)$ is determined, hence each $\gamma_i = h^{s_i}$. Because the (intermediate) DVRF value γ is the (t, n)-threshold combination of $\{\gamma_i\}_{i \in S}$, it is consistently generated.

4.3 Uniqueness and Pseudorandomness

To prove that Icy-DVRF satisfies the uniqueness and pseudorandomness properties, we turn our attention to demonstrating that the Threshold Discrete Logarithm Equality (T-DLEQ) proof system employed in the Icy-DVRF protocol satisfies the properties of a NIZK proof. First, we establish that the interactive variant of the system is zero-knowledge.

Theorem 1. *The Interactive Threshold Discrete Logarithm Equality (T-DLEQ-Int) proof system in Icy-DVRF is complete, honest-verifier zero-knowledge, and knowledge-sound under the discrete logarithm assumption in the random oracle model. The concrete extraction loss is quantified in Theorem 3.*

Proof. Completeness immediately follows from the Correctness and Consistency properties. To establish Honest Verifier Zero-Knowledge, let g, h belong to a cyclic group $\mathbb{G}$ of prime order q, and suppose a prover P wants to prove to a verifier V that they know $s \in \mathbb{Z}_q$ so that $s = \log_g(Y) = \log_h(\gamma)$ where $Y, \gamma \in \mathbb{G}$.

Here, from the verifier's perspective ($\text{View}_V(\cdot)$), the Simulator's (`Sim`) output must be indistinguishable from the distribution over transcripts. We have $\{\text{View}_V(P(Y, \gamma; s)) \longleftrightarrow V(Y, \gamma)\} = \{(g^k, h^k, c, k + c \cdot s \ : \ k \xleftarrow{\$} \mathbb{Z}_q, c \leftarrow V(\cdot)\}$
$= \{(R, \bar{R}, c, z) \ : \ c, z \xleftarrow{\$} \mathbb{Z}_q, g^z = R \cdot Y^c \wedge h^z = \bar{R} \cdot \gamma^c\}$.

In other words, on input (Y, γ), `Sim` computes $c_{\text{sim}}, z_{\text{sim}} \xleftarrow{\$} \mathbb{Z}_q$, where $R_{\text{sim}} = g^{z_{\text{sim}}} \cdot Y^{-c_{\text{sim}}}$, $\bar{R}_{\text{sim}} = h^{z_{\text{sim}}} \cdot \gamma^{-c_{\text{sim}}}$, and outputs $(R_{\text{sim}}, \bar{R}_{\text{sim}}, z_{\text{sim}})$. Note that this always yields an admissible conversation as required.

In a real-world conversation, c and z are independent and uniformly distributed over $\mathbb{Z}_q$. Given $c, z \in \mathbb{Z}_q$, the values R_{sim} and $\bar{R}_{\text{sim}}$ are uniquely determined by the equation pair $g^z = R_{\text{sim}} \cdot Y^c, h^z = \bar{R}_{\text{sim}} \cdot \gamma^c$. It is now clear that this gives the same output distribution for `Sim`. The proof of the knowledge-soundness property of T-DLEQ-Int is a direct consequence of Theorem 3 in Appendix A.3. □

We now turn to the non-interactive counterpart of T-DLEQ-Int, which is the variant originally presented in Section 3.2. Its security guarantees, which rest on the one-more discrete logarithm assumption rather than the plain discrete logarithm assumption, are established by the following theorem.

Theorem 2. *The Threshold Discrete Logarithm Equality (T-DLEQ) proof system is a non-interactive zero-knowledge argument of knowledge: it is complete, zero-knowledge, and knowledge-sound under the one-more discrete logarithm assumption, with the distributed key generation analyzed under the Schnorr knowledge-of-exponent assumption. The concrete extraction loss is quantified in Theorem 4.*

Proof. Completeness follows from the Correctness property of Section 4: for an honestly generated proof (γ, c, z) with $z = k + c \cdot s$, $R = g^k$, $Y = g^s$, and $\gamma = h^s$, the verifier recomputes $T = g^z Y^{-c} = g^k = R$ and $\bar{T} = h^z \gamma^{-c} = h^k = \bar{R}$, so the challenge check $c = H_2(g, h, Y, \gamma, T, \bar{T}, \alpha)$ succeeds and **Verify** outputs 1.

To show that the system is zero-knowledge, note that for each statement (g, h, Y, γ) with $h = H_1(\alpha)$, the simulator samples $\bar{c}, \bar{z} \xleftarrow{\$} \mathbb{Z}_q$, sets $R = g^{\bar{z}} Y^{-\bar{c}}$ and $\bar{R} = h^{\bar{z}} \gamma^{-\bar{c}}$, and programs $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha) \leftarrow \bar{c}$. By Lemma 2 of Appendix B, each simulated proof is distributed identically to an honest one whenever the programmed point was previously undefined. The probability that any of the at most Q programmed points collides with a prior H_2 entry is at most Q^2/q, as in the proof of Lemma 7 of Appendix B. Hence, the simulated transcripts are statistically indistinguishable from real ones, and T-DLEQ is zero-knowledge in the programmable random oracle model.

Knowledge-soundness requires the full reduction, and is proved in Theorem 4 of Appendix B. We show that any PPT adversary producing a verifying, non-trivial T-DLEQ proof can be turned into a solver for the one-more discrete logarithm (OMDL) problem, with the distributed key generation handled by the Schnorr knowledge-of-exponent assumption (implied by DL in the AGM). Concretely,

$$\mathsf{Adv}^{\text{t-dleq}}(A) \leq \mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}}) + 2\max\{\epsilon_{\mathsf{quo}}, \epsilon_{\mathsf{sub}}\} + \frac{Q^2}{q},$$

where $\mathcal{B}_{\mathsf{koe}}$ is the Schnorr-KoE adversary, $\epsilon_{\mathsf{quo}}, \epsilon_{\mathsf{sub}}$ are the shorthand notations for the two branch bounds of Theorem 4. The reduction is modular using the intermediate Bi-DLEQ assumption (Definition 5, Lemma 4), which confines all rewinding to a single forking step.

Thus, having demonstrated completeness, zero-knowledge, and knowledge-soundness, we conclude that T-DLEQ is a valid NIZK proof system. □

Having established that the T-DLEQ proof system within Icy-DVRF satisfies the definition of a Non-Interactive Zero-Knowledge proof, we now address the remaining security properties. Structurally, Icy-DVRF diverges from DDH-DVRF [26] only in its use of a threshold mechanism to be able to aggregate the individual DLEQ proofs into a single, compact group element. Crucially, the proofs for uniqueness and strong pseudorandomness established for DDH-DVRF rely solely on the NIZK property of the DLEQ proof element(s). Consequently, because Icy-DVRF fulfills the requisite NIZK conditions, it preserves the uniqueness and strong pseudorandomness properties inherent to the DDH-DVRF scheme.

4.4 Analysis of Security Tightness

The security reduction of Icy-DVRF Interactive of Appendix A relies on the Generalized Forking Lemma, which inherently implies a non-tight reduction. Specifically, the advantage ϵ' of solving the Discrete Logarithm Problem is bounded by:

$$\epsilon' \leq \frac{\epsilon^2}{2n_h + \Omega \cdot n_p} ,$$

where n_h represents the number of hash queries, Ω denotes the preprocess batch size, and $\Omega \cdot n_p$ accounts for the total number of preprocessed commitment pairs.

The preprocessing phase introduces specific constraints on simulation complexity, as the simulator must program the random oracle for every pre-published commitment $D_{i,j}$ and $E_{i,j}$. This requirement increases the denominator in the advantage equation, leading to a quadratic loss that slightly loosens the security bound relative to standard, non-preprocessed DLEQ proofs.

To mitigate this effect and preserve a 128-bit security level, we recommend utilizing elliptic curves with at least a 256-bit prime order q. Under these parameters, the security margin remains robust for decentralized applications even when accounting for up to 2^{60} oracle queries, ensuring the protocol's practical viability despite the increased complexity of the reduction.

In the variant of Icy-DVRF described in Section 3, the reduction is also non-tight: it rewinds once through a forking step, incurring a quadratic loss. Inverting the dominant maximand of Theorem 4, if OMDL is $(\tau', \epsilon'_{\text{omdl}})$-hard and Schnorr-KoE is $(\tau', \epsilon'_{\text{koe}})$-hard in $\mathbb{G}$, then the non-interactive T-DLEQ proof system over $\mathbb{G}$ with n provers, threshold t, and preprocess batch size Ω is knowledge-sound with forgery advantage ϵ bounded by

$$\epsilon \;\leq\; \epsilon'_{\text{koe}} \;+\; 2\sqrt{Q\Big(\epsilon'_{\text{omdl}} + \frac{3(n+1)^2Q^2}{q}\Big)} \;+\; \frac{Q^2}{q},$$

where $Q = n_h + n_{\mathsf{pr}} + 1$ counts the random-oracle (n_h) and proving (n_{pr}) queries. Equivalently, solving OMDL succeeds with advantage at least

$$\epsilon'_{\text{omdl}} \;\gtrsim\; \frac{\epsilon^2}{4Q} - \frac{3(n+1)^2Q^2}{q},$$

the same quadratic ϵ^2/Q shape as in the bound of the interactive variant, now also featuring the FROST1 internal-binding constant $(n+1)^2$ (Remark 1) in place of the interactive denominator's preprocessing term.

The preprocess batch size Ω does not appear in the later bounds, since, unlike the interactive analysis where each of the Ω pairs per round is simulated and forked over, the non-interactive reduction charges only the pairs an adversary *consumes*, which are counted directly by n_{pr}.

To preserve a 128-bit security level against up to 2^{60} oracle queries, we again recommend a curve with at least a 256-bit prime order q, under which the quadratic loss and the factor of $(n+1)^2$ leave a comfortable margin for the committee sizes considered in Section 5.

4.5 Resilience against Denial-of-Service (DoS) Attacks

Unlike fully robust threshold schemes that guarantee output even in the presence of faults, Icy-DVRF utilizes an "abort-and-identify" model to prioritize on-chain

efficiency. Although this introduces a potential Denial-of-Service (DoS) vulnerability, the protocol is designed with two primary defense layers:

- **Cryptographic Identifiability:** Each participant P_i provides a partial evaluation γ_i and a corresponding NIZK proof π_i. During the Combine phase, the Combiner verifies these proofs individually. If any share is invalid, the specific misbehaving participant is cryptographically identified and excluded from the session.
- **Economic Disincentives (Slashing):** In practical decentralized deployments, such as blockchain-based randomness beacons, this identifiability allows for the integration of a slashing mechanism. By requiring participants to stake collateral, any identified fault or malicious abort can result in an automated financial penalty. This economic pressure ensures that the cost of a sustained DoS attack far outweighs any potential benefit for the adversary, effectively neutralizing the robustness trade-off in production environments.

4.6 Resilience against Malicious Combiners

Although the Combiner $\mathcal{C}$ is responsible for collecting and combining partial proofs, its potential for malicious behavior is restricted to service disruption. To mitigate such Denial-of-Service (DoS) risks:

- **Aggregator Redundancy:** The protocol does not rely on a combiner; multiple participants can simultaneously act as aggregators. If one $\mathcal{C}$ fails to publish the final VRF output β and proof π, the provers can re-send their shares to an alternative combiner or broadcast them to a public bulletin board.
- **Public Verifiability of Misbehavior:** Because all partial proofs (z_i, c) and commitments are publicly verifiable, a combiner (or any other member) cannot falsely claim that a participant is malicious without providing a cryptographic proof of failure. Any such false report can be detected by honest participants, resulting in the combiner's (or other members') immediate disqualification.
- **Statelessness:** The combiner $\mathcal{C}$ does not hold any long-lived secret state. This allows the committee to rotate the combiner role frequently or to select it randomly for each session, preventing a single point of failure from persistently stalling the proof network.

4.7 Consensus on Node Exclusion

While Icy-DVRF identifies misbehaving nodes via cryptographic proofs, the final decision to exclude a node from the committee is delegated to the underlying *Consensus Layer* of the hosting network. To prevent network splits during this process, Icy-DVRF provides *publicly verifiable evidence of failure*. Specifically, if a participant P_i submits an invalid partial proof π_i or an incorrect response z_i, the Combiner or any honest observer can broadcast this invalid share alongside the prover's public key Y_i.

Because the verification of these shares is deterministic and based on globally known public parameters, all honest nodes will independently reach the same conclusion regarding the node's fault. This *objective fault attribution* ensures that the exclusion process does not rely on subjective leader-based reports, thereby mitigating the risk of forks or network splits during committee reconfigurations.

4.8 Robustness and Operational Fault-Tolerance Model

To maintain liveness under adversarial or unstable network conditions, Icy-DVRF operates under an explicit *abort-and-identify* recovery paradigm rather than deploying heavy threshold reconstruction cycles. We formalize the handling of operational anomalies as follows:

1. **Non-Response Management:** If a chosen participant P_i fails to submit its partial evaluation share (γ_i, π_i) within a synchronized network delay Δ, the Combiner drops the current session. Because Icy-DVRF requires exactly t active shares to reconstruct the final proof or random value without structural degradation, the missing entity cannot be bypassed seamlessly in the middle of a phase.
2. **Retry Logic and Session Initialization:** Upon detecting a timeout or an invalid share verification via PartialVerify, the Combiner broadcasts an authenticated `ABORT` signal carrying a monotonic epoch counter increment. The honest nodes clear the state registers allocated for the aborted session, reset their pointer states, and await a new execution request.
3. **Combiner Withholding Protection:** A malicious Combiner might deliberately withhold valid shares or isolate subsets of nodes to stall the beacon or cause network splits. To mitigate this risk, any participant can act as a secondary Combiner by tracking the network entries. If the primary Combiner does not publish the aggregated output Ω_τ within the designated time window, backup nodes broadcast their independently collected valid share pools, triggering a deterministic aggregation event.
4. **Committee Exclusion Strategy:** Misbehaving or unresponsive nodes are definitively isolated from future participation. Upon an abort event caused by a cryptographic failure (e.g., an invalid NIZK proof), the explicit proof of failure is submitted to the base consensus layer. The governance state machine then updates the active participant bitmask, excluding the faulty key from the next active committee matrix without requiring a full DKG key refresh.

5 Comparison

In this section, we compare our proposed Icy-DVRF construction with DVR-FwCP [1] and, later, with GLOW-DVRF [26] in terms of operational complexity, estimated gas costs, and communication bandwidth. We instantiate all protocols using the BLS12-381 curve, a Type-3 pairing-friendly curve widely adopted in

blockchain ecosystems and standardized for Ethereum in EIP-2537 [45]. This curve satisfies the Symmetric External Diffie-Hellman (SXDH) assumption, implying that the DDH-problem (Definition 2) is hard in both source groups $\mathbb{G}_1$ and $\mathbb{G}_2$. 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 $\mathbb{G}_2$.

5.1 Mathematical Comparison

By incorporating FROST-style preprocessing, Icy-DVRF outperforms DVRFwCP in both proof generation complexity and communication cost. This advantage necessitates a one-time post-DistKeyGen computation, a requirement not present in DVRFwCP.

In the PartialEval phase, DVRFwCP is slightly more computationally efficient, requiring 3 exponentiations in $\mathbb{G}$ compared to 5 for Icy-DVRF. However, for a (t, n)-threshold DVRF proof generation, in the PartialProofGen phase, Icy-DVRF demonstrates superior scalability by requiring only $3t$ exponentiations per party, whereas DVRFwCP is computationally heavier with $3nt + 4n + 3t$ exponentiations due to the overhead of its nonce generation mechanism. Finally, the trade-off shifts back in favor of DVRFwCP during the Combine phase, where it requires $6t$ exponentiations and $2t$ multiplications, notably less than Icy-DVRF, which demands $9t$ exponentiations, $8t-3$ multiplications, and additional hashing operations. Overall, for verifiable randomness and proof generation, Icy-DVRF is much lower in computational complexity than DVRFwCP. The analysis is detailed in Table 1.

Both schemes achieve highly efficient constant-time verification suitable for blockchain environments by avoiding expensive bilinear pairings during verification. Assuming the EIP-2537 precompile standards for the BLS12-381 curve, where an exponentiation in $\mathbb{G}_1$ costs approximately 12000 gas and multiplication costs 500 gas, the verification cost for both schemes is identical at approximately 49000 gas. Furthermore, both schemes produce constant-size proofs consisting of one group element and two field elements ($1\mathbb{G} + 2\mathbb{Z}_q$). With call data costing 16 gas per byte [2], the data availability cost is approximately 1792 gas for both protocols.

A critical distinction arises in the communication overhead during the evaluation and proof generation phases, specifically regarding the broadcast of group elements versus scalar elements. During the PartialEval phase, DVRFwCP is more bandwidth-efficient, requiring participants to broadcast a partial evaluation and an NIZK proof for a total of $1\mathbb{G} + 2\mathbb{Z}_q$ elements. In contrast, Icy-DVRF requires participants to broadcast the partial evaluation along with two additional commitments $(\bar{D}_i, \bar{E}_i)$ to facilitate the subsequent single-round proof generation, resulting in a higher initial payload of $3\mathbb{G} + 2\mathbb{Z}_q$ elements. However, the bandwidth trade-off reverses significantly during the PartialProofGen phase. To generate the shared challenge in DVRFwCP, participants must collectively run Augmented Secure-DKG, which requires broadcasting polynomial commitments proportional to the threshold t. Specifically, this entails transmitting $2t$

group elements, along with the proof share, for a total of $2t\mathbb{G} + 2\mathbb{Z}_q$ per participant. Conversely, Icy-DVRF participants compute the shared challenge locally using the pre-broadcast commitments, so the only broadcast required is the additive proof share and the hash, totaling merely 2 elements of $\mathbb{Z}_q$. Consequently, while Icy-DVRF incurs a slightly higher constant cost during PartialEval, it is significantly more scalable for large distributed committees: the total communication cost of DVRFwCP scales linearly with the threshold ($(2t+1)\mathbb{G}$), whereas Icy-DVRF maintains a constant communication overhead of $3\mathbb{G}$ elements per participant, regardless of the committee size. The results are summarized in Table 2.

As demonstrated by the theoretical analysis of Ağırtaş et al. [1], DDH-DVRF incurs a linear verification cost of approximately $70768 \cdot t$ gas, which becomes prohibitively expensive for large committees. Similarly, GLOW-DVRF, while achieving constant-size proofs, incurs a high fixed cost of approximately 201280 gas due to the reliance on pairing checks during verification. In stark contrast, both DVRFwCP and Icy-DVRF operate at a total estimated cost of approximately 51792 gas (verification plus calldata). This represents a roughly 4 times improvement over GLOW-DVRF and eliminates the linear scaling costs associated with DDH-DVRF, making the Icy-DVRF scheme far more suitable for gas-constrained blockchain environments.

5.2 Empirical Results and Benchmark

In general, VRF services, such as Chainlink VRF [14], generate verifiable randomness by bridging on-chain requests to off-chain computation. When a contract requests randomness, an off-chain oracle node generates a pseudo-random value and a cryptographic proof using its private key. The on-chain VRF Coordinator then verifies this proof against the public key of the node, ensuring that the result is tamper-proof and unpredictable before delivering it to the consumer contract.

To validate the theoretical efficiency of our proposal, we implemented the verification logic for Icy-DVRF and GLOW-DVRF in a single Solidity smart-contract on the Sepolia testnet, using the EIP-2537 precompiles [45] for BLS12-381 curve operations. Our empirical study specifically focused on the on-chain verification phase for the following reasons:

1. **Cost asymmetry:** While the preprocessing and proof generation phases occur entirely off-chain, incurring zero gas costs for the decentralized network, the verification phase is a **mandatory on-chain operation** that consumes restricted block space.
2. **Bottleneck identification:** In production environments, the primary scalability bottleneck is the high cost of on-chain execution and data availability, and off-chain computation is measured in mere milliseconds, even when the number of nodes scales up to hundreds.

It should be noted that our Solidity contract code and the benchmarks are not on par with production code in terms of optimization and security,

but merely written for on-chain benchmark and demonstration purposes. The implementation of the smart contract and benchmark functions, along with the Sepolia Testnet address of the deployed contract, can be found at `https://github.com/abozer/DVRF_Verify`.

Our performance benchmarks reveal a substantial efficiency gain in Icy-DVRF compared with existing constructions, a result that warrants analysis across three distinct dimensions.

- **Calldata gas:** The calldata gas consumption for the verification functions was evaluated in accordance with the gas pricing specifications defined in the EIP-2028. As detailed in Table 4, the Icy-DVRF Verify function requires 6,784 gas for the calldata, whereas the GLOW-DVRF Verify counterpart consumes 7,224 gas. This results in a reduction of 440 gas in favor of the Icy-DVRF protocol.
- **Precompile-only costs:** By isolating precompile costs, we can evaluate the intrinsic cryptographic complexity of each protocol independent of implementation-specific overhead or Solidity logic. This comparison, detailed in Table 5, highlights the impact of primitive operations, where the Verify function of Icy-DVRF achieves a significant efficiency advantage at 51,984 gas by avoiding the expensive elliptic curve pairings that drive GLOW-DVRF to 120,908 gas, resulting in a reduction of 68,924 gas in core cryptographic operations in Icy-DVRF Verify.
- **Full gas costs on Sepolia:** Experimental results for the Verify function on the Sepolia testnet demonstrate superior performance for the Icy-DVRF, which maintained a constant execution cost of 88,803 gas. In contrast, GLOW-DVRF exhibited a significantly higher execution cost of 155,838 gas, whereas Icy-DVRF achieved a **reduction of 67,035** gas (**43.02%**) compared to its counterpart.

The discrepancy between the experimental findings and the theoretical estimates in Table 3 is primarily attributable to implementation-level optimizations within the BLS12-381 precompile. Specifically, the observed gas savings in the live environment result from the efficient handling of pairing-based comparisons, which significantly reduces the effective computational overhead relative to standard theoretical pricing models. Notably, while we exclude a direct benchmark against DDH-DVRF, its verification costs scale linearly with the threshold number t of provers, further emphasizing the scalability of our approach.

Table 4. Calldata Comparison for Icy-DVRF and GLOW-DVRF Verify.

Metric	Icy-DVRF	GLOW-DVRF
Selector	4B	4B
Head	192B	128B
Tail Arguments	Y: 160B	pk: 288B
	γ: 160B	x: 64B
	c: 64B	v: 64B
	z: 64B	π: 160B
	α: 64B	
	β: 64B	
Tail (Total)	576B	576B
Total Bytes	**772B**	**708B**
Calldata Gas (EIP-2028)	**6784**	**7224**

(a) The names of the arguments are from each respective algorithm. (b) In EIP-2028, zero-bytes and non-zero-bytes cost 4 and 16 gas, respectively.

Table 5. Comparison of Precompile-only Costs (excluding DVRF_Verify overhead)

Operation	Icy-DVRF	GLOW-DVRF	Difference
Hash-to-$\mathbb{G}_1$ [a]	5600	5600	0
G1MSM [b]	45752	12100	$-33,652$
SHA-256	632	208	-424
Pairing in BLS12-381	0	103000	$+103000$
Total (precompiles)	**51,984**	**120,908**	**+68,924**

(a) In Icy-DVRF, $\mathbb{G} = \mathbb{G}_1$ (b) Multiscalar Multiplication in $\mathbb{G}_1$ (c) Varied SHA-256 costs reflect the different number of inputs and total data size processed by each protocol.

The discrepancy between the experimental findings and the theoretical estimates in Table 3 is primarily attributable to implementation-level optimizations within the BLS12-381 precompile. Specifically, the observed gas savings in the live environment result from the efficient handling of pairing-based comparisons, which significantly reduces the effective computational overhead relative to standard theoretical pricing models. Notably, while we exclude a direct benchmark against DDH-DVRF, its verification costs scale linearly with the threshold number t of provers, further emphasizing the scalability of our approach.

By offloading interaction complexity to an off-chain environment and maintaining a minimal on-chain footprint, Icy-DVRF emerges as a superior architectural alternative for high-throughput blockchains and decentralized oracle networks (DONs). Through its pairing-free verification and constant gas costs,

the protocol provides an architectural foundation capable of scaling verifiable randomness services such as Chainlink VRF [14] without the prohibitive burden of on-chain execution overhead.

Having established the on-chain verification advantage of Icy-DVRF above, we now quantify its off-chain efficiency by benchmarking the full protocol across a range of committee sizes (n, t). All measurements were performed on a MacBook Air M4 (32 GB) using a single-process simulation with BLS12-381 $\mathbb{G}_1$ operations via `py_ecc` [12] and a network model with a 120 ms latency per interaction cycle. The reported figures are the means over 100 iterations for each benchmark. The implementation of the benchmark functions is available at `https://github.com/abozer/DVRF_Comparison`.

In Table 6, we report the total per-request runtime for both protocols and show that Icy-DVRF is (up to 7 times overall; 4.5 times per node) faster at large committee sizes owing to its elimination of the Augmented Secure-DKG round. Table 7 instantiates the communication formulas of Table 2 at concrete sizes, confirming Icy-DVRF's constant per-node broadcast totals versus DVRFwCP's linear growth: Icy-DVRF with $3\mathbb{G}_1 + 4\mathbb{Z}_q$ against DVRFwCP with $(2t{+}1)\mathbb{G}_1 + 4\mathbb{Z}_q$. In Table 8, we present the one-time setup costs, including the amortized cost of Icy-DVRF's offline Preprocess phase, which we present in more detail in Section 5.3. Finally, Table 9 gives a per-phase breakdown when $(n, t) = (10, 7)$ to complement the operation counts of Table 1.

5.3 Practical Scalability and Operational Overheads

Although the theoretical scalability of Icy-DVRF is established through its $O(t)$ communication complexity, we further evaluate the practical bottlenecks related to storage and state management that arise in large-scale deployments:

- **Storage Requirements for Preprocessing:** Storing preprocessed commitment pairs $(D_{i,j}, E_{i,j})$ does not impose a significant burden on provers. On the BLS12-381 curve [22], a compressed $\mathbb{G}_1$ element occupies 48 bytes. Consequently, each commitment pair requires 96 bytes of storage. Even for a substantial batch size of $\Omega = 10,000$, a participant only needs to maintain approximately **0.96 MB of local storage**. This confirms that the preprocessing model is highly sustainable for modern decentralized infrastructure, even for nodes with limited storage capacity.
- **Nonce Management and Synchronization:** To ensure that all participants utilize the same precomputed nonces for a specific round, Icy-DVRF employs a counter-based synchronization mechanism. The Combiner includes a unique single-use index j in the initial request, allowing provers to instantly retrieve the corresponding $(D_{i,j}, E_{i,j})$ pair from their local database. This index-based lookup eliminates the need for complex state-search operations. The computational overhead of this retrieval is several orders of magnitude lower than that of the subsequent cryptographic operations (e.g., exponentiations), ensuring that nonce management does not impact the protocol's real-time latency.

– **Large Committee Performance:** In scenarios involving large committee sizes (e.g., $n > 25$), the primary bottleneck in distributed systems often shifts from computation to network bandwidth. However, Icy-DVRF maintains a highly efficient communication profile owing to its fixed-size payloads. Based on the BLS12-381 specification [22], each participant broadcasts a total of **272 bytes** ($3\mathbb{G}_1 + 4\mathbb{Z}_q$) during the initial phases, and only **64 bytes** ($2\mathbb{Z}_q$) during the final proof generation. Unlike linear-proof constructions such as DDH-DVRF, where verification costs scale at approximately $70768t$ gas, Icy-DVRF preserves **constant-time on-chain verification**. This makes it a candidate for a highly practical engine for Distributed Randomness Beacons (DRBs) where committee size and gas economy must scale independently.

5.4 Strategic Positioning in the DRB Ecosystem

It is essential to distinguish between Distributed Verifiable Random Functions (DVRFs), which serve as cryptographic primitives, and Distributed Randomness Beacons (DRBs), which are decentralized systems that utilize such primitives to produce a continuous stream of randomness [16]. Following the recent systematization of the distributed randomness landscape by Choi et al. [16], DRB protocols were classified based on their entropy provision and recovery strategies.

Icy-DVRF is proposed as a high-performance engine for the category of DRBs that utilize "No Marginal Entropy" [16]. In these systems, such as **drand** or the **Internet Computer (Dfinity)**, the beacon output is generated through the collective computation of a DVRF or threshold signature primitive without requiring fresh external input in each epoch [16].

Choi et al. identified two primary existing primitives in this space: **DDH-DVRF**, which suffers from $O(t)$ linear proof sizes affecting verification scalability, and **GLOW-DVRF**, which offers constant-size proofs but incurs a high computational overhead due to bilinear pairing operations [16]. Icy-DVRF bridges this gap by offering a third alternative: a primitive that achieves constant-size proofs without using pairings. By optimizing the underlying cryptographic engine, Icy-DVRF offers an optimized cryptographic building block designed to support advanced DRB architectures that seek to minimize the on-chain settlement costs on high-throughput blockchains.

6 Real-World Viability and Deployment Considerations

The practical viability of Icy-DVRF in a decentralized network depends on the balance between the performance and fault tolerance. While Icy-DVRF follows the abort-and-identify model (similar to FROST), which lacks the robustness of protocols that guarantee output despite $t - 1$ faults, it is specifically optimized for environments in which on-chain verification costs are the primary bottleneck.

In real-world deployment (e.g., an Ethereum-based randomness beacon), the protocol's viability is maintained as follows.

- **Deterministic Fault Attribution:** The use of DLEQ proofs ensures that any participant who submits an incorrect share is identified immediately.
- **Slashing Conditions:** By integrating Icy-DVRF with a staking layer, malicious aborts can be penalized financially, thereby effectively deterring DoS attacks.
- **Optimistic Efficiency:** In the common case in which all participants are honest (the optimistic case), Icy-DVRF provides the fastest and cheapest path to verifiable randomness, saving approximately 67,000 gas per verification compared to GLOW-DVRF on the Sepolia testnet.

A key viability factor is that Icy-DVRF allows any threshold t out of n participants to produce a valid output. Unlike certain threshold schemes that require all participants to be active, this allows a subset of devices to remain offline, which is often desirable for security and high availability. Because the preprocessing stage is separate from the proving stage, participants can generate and publish nonces at their convenience.

The binding technique used in FROST is also used in Icy-DVRF, which allows unrestricted parallelism. In a deployment such as an Ethereum beacon, multiple randomness requests might be processed simultaneously and concurrently while remaining protected against forgery.

7 Conclusion

In this work, we introduce Icy-DVRF to address the communication scalability bottlenecks inherent in previous distributed verifiable random function protocols. By integrating the preprocessing model of the FROST signature scheme, we reduce the proof generation complexity with respect to the committee size, yielding a constant communication cost per participant and reducing the number of interaction rounds. This optimization significantly enhances scalability compared to DVRFwCP, but at the cost of robustness, as faults now require the protocol to re-run. Ultimately, by retaining the advantages of constant-size proofs and pairing-free verification, Icy-DVRF offers a highly efficient architectural alternative for decentralized networks where both on-chain gas economy and off-chain bandwidth are critical constraints.

A Proof of Security of Interactive T-DLEQ

We present the interactive version of our protocol and its security proof. Note that we once again use FROST-Interactive DKG and Preprocess verbatim; the PartialProofGen algorithm is modified to a two-round version in which binding commitments are computed via the one-time VRF and verified by the corresponding receiver, whereas the PartialEval and Combine algorithms remain unchanged from the non-interactive version. The Verify algorithm belongs solely to the Icy-DVRF algorithm, not to the T-DLEQ-Int.

A.1 Overview of the T-DLEQ-Int Security Proof

The core strategy for proving the security of Icy-DVRF's Interactive Threshold Discrete Logarithm Equality Protocol involves a reduction to the computational difficulty of the Discrete Logarithm Problem. At a high level, the proof embeds an arbitrary challenge $\omega \in \mathbb{G}$ into a group public key Y generated during the DistKeyGen protocol. If an adversary can successfully forge a proof, the reduction uses this capability to extract the discrete logarithm of ω.

The proof relies on the interaction of four distinct algorithms:

- **The Forger ($\mathcal{F}$):** A Probabilistic Polynomial-Time algorithm that controls up to $t-1$ corrupted participants and attempts to compute a forgery in time τ with probability ϵ for a given public key Y.
- **The Simulator ($\mathcal{A}$):** An adversary algorithm that invokes $\mathcal{F}$. It simulates the honest participants during DistKeyGen and proof generation, while also handling and simulating all random oracle queries.
- **The Generalized Forking Algorithm ($GF_{\mathcal{A}}$):** An algorithm that generates a random tape and outputs for random oracle queries, invoking $\mathcal{A}$ to produce two valid, distinct forgeries (π, π')
- **The Coordinator ($\mathcal{C}$):** The top-level algorithm that accepts the DLP instance ω, coordinates the execution of the other algorithms to obtain (π, π'), and ultimately extracts the discrete logarithm of ω.

During the forgery, $\mathcal{F}$ is granted the role of the combiner C. Without loss of generality, assume that $\mathcal{F}$ has complete control over the participants $P_1, \ldots, P_{t-1}$, including the secret and public values they generate. Additionally, assume that the honest party P_t is in the set S of provers.

Simulating the Protocols and Embedding the Challenge The Coordinator $\mathcal{C}$ executes the SimDistKeyGen algorithm to embed the challenge ω into the group public key Y without knowing its discrete logarithm. During this simulation, the Simulator $\mathcal{A}$ simulates the honest participant P_t. $\mathcal{C}$ extracts the secret values of the corrupted participants controlled by $\mathcal{F}$ using the zero-knowledge proofs they generate during the protocol.

In the subsequent protocols, $\mathcal{A}$ simulates both the honest party P_t and the random-oracle queries. During Preprocess-Interactive, $\mathcal{A}$ employs a forking strategy (rewinding $\mathcal{F}$) to extract the discrete logarithms $(a_{l,j}, b_{l,j})$ of the commitments generated by the dishonest parties. Because $\mathcal{A}$ learns these values, it can compute the group commitment R strictly before $\mathcal{F}$ during the PartialProofGen phase, allowing $\mathcal{A}$ to successfully program the random oracle to return the correct challenge value.

The Generalized Forking Strategy To extract the discrete logarithm of the challenge ω, the proof leverages the Generalized Forking Algorithm ($GF_{\mathcal{A}}$) to run the Simulator $\mathcal{A}$ twice on the same random tape with an *initial execution* and a *forked execution* mimicking the Pointcheval and Stern model [42]. If $\mathcal{F}$

successfully produces a second valid forgery π' at the same index J but with a distinct oracle output ($h_J \neq h'_J$), $GF_{\mathcal{A}}$ returns the tuple (π, π', h_J, h'_J).

Extracting the Discrete Logarithm As both forgeries are valid VRF proofs for γ, they share the same base relations. Using the two distinct challenges (h_J, h'_J) and responses (z, z'), Coordinator $\mathcal{C}$ computes the discrete logarithm of the group public key Y as:

$$\log_g(Y) = \frac{z' - z}{h'_J - h_J}$$

Since $\mathcal{C}$ already extracted the secret shares of all corrupted parties during DistKeyGen, it can isolate $a_{t,0}$, which is the discrete logarithm of the target challenge ω.

A.2 The Interactive Threshold Discrete Logarithm Equality Protocol

Within PartialProofGen, a one-time verifiable random function is used to instantiate the $\rho_i = a_{i,j} + b_{i,j} \cdot H_\rho(\alpha, B)$. Here, the hash function H_ρ, with inputs as (α, B) and outputs in $\mathbb{Z}_q$, serves as the interactive substitute for the static H_1 in the non-interactive variant. Additionally, to mitigate replay attacks, the protocol employs H_3, a hash function that maps a sequence of commitments to a context string Φ, which is used alongside the participant index i as input to another hash function H_4, with outputs in $\mathbb{Z}_q$.

A.3 The Algorithms

The Generalized Forking Algorithm ($GF_{\mathcal{A}}$) is designed to leverage an adversary $\mathcal{A}$ to produce two related forgeries from a single random tape Ξ. This process operates in two distinct phases:

- Initial Execution: $GF_{\mathcal{A}}$ first supplies $\mathcal{A}$ with a public key Y, the random Ξ, and a set of simulated random oracle outputs $(h_1, \ldots, h_{n_r})$. If $\mathcal{A}$ successfully outputs a forgery π and identifies the corresponding oracle query index J, the algorithm proceeds to the forking stage.
- Forked Execution: In the second run, $GF_{\mathcal{A}}$ re-executes $\mathcal{A}$ using the identical random tape and the same oracle responses up to index $J-1$. However, for indices J through n_r, it provides a fresh set of random oracle values.

This approach effectively *forks* the adversary at the moment of the attack, conceptually similar to the Pointcheval and Stern [42] model, without requiring explicit rewinding. If the adversary produces a second valid forgery π' at the same index J but with a distinct oracle output ($h_J \neq h'_J$), the algorithm successfully returns the tuple (π, π', h_J, h'_J). These values provide the necessary data for the coordinator $\mathcal{C}$ to extract the discrete logarithm of the challenge value ω.

Algorithm 1 Generalized Forking Algorithm $GF_{\mathcal{A}}$

Input A public key Y

Output $(1, h_J, h_{J'}, \pi, \pi')$ if $\mathcal{A}$ produces two forgeries; otherwise, $\perp$

1: Instantiate a random tape Ξ
2: $\{h_1, \ldots, h_{n_r}\} \xleftarrow{\$} H$
3: (J, π) or $\perp \leftarrow \mathcal{A}(Y, \{h_1, \ldots, h_{n_r}\}; \Xi)$
4: If $\perp$, return $\perp$
5: $\{h'_1, \ldots, h'_{n_r}\} \xleftarrow{\$} H$
6: (J', π') or $\perp \leftarrow \mathcal{A}(Y, \{h_1, \ldots, h_{J-1}, h'_J, \ldots, h'_{n_r}\}; \Xi)$
7: If $J = J'$ and $h_J \neq h'_J$, return $(1, h_J, h_{J'}, \pi, \pi')$. Else, return $\perp$.

Algorithm 2 The Coordinator Algorithm $\mathcal{C}(\omega)$

Input A challenge value ω

Output The discrete logarithm of ω, or $\perp$

1: Simulate DistKeyGen embedding the challenge value ω and extract the forger's secret values $\big(Y, (a_{1,0}, \ldots, a_{(t-1),0})\big) \leftarrow SimDistKeyGen(\omega)$
2: $(1, h_J, h_{J'}, \pi, \pi')$ or $\perp \leftarrow GF_{\mathcal{A}(Y)}$
3: If not $\perp$, ExtractDiscreteLog$(\omega, h_J, h_{J'}, \pi, \pi'\ (a_{1,0}, \ldots, a_{(t-1),0}))$

We now present the Generalized Forking Lemma by Bellare and Neven [6] in our setup. Let H be a set of size $h \geq 2$, and *adv* be the advantage of solving the discrete logarithm problem over some group $\mathbb{G}$.

Lemma 1. *Generalized Forking Lemma [6] If IG is a randomized input generator algorithm, define acc to be the probability of $J \geq 1$ when $x \xleftarrow{\$} IG$, $(h_1, \ldots, h_q) \xleftarrow{\$} H$, and $(J, \pi) \xleftarrow{\$} \mathcal{A}(x, h_1, \ldots, h_{n_r})$. Also, define frk as the probability of $b = 1$, when $x \xleftarrow{\$} IG$ and $(b, \pi, \pi') \xleftarrow{\$} GF_{\mathcal{A}}(x)$.*

Then,

$$adv \geq frk \geq acc \cdot \left(\frac{acc}{n_r} - \frac{1}{h} \right).$$

Lemma 1 shows the probability *adv* that the $GF_{\mathcal{A}}$ will produce two valid forgeries to enable the extraction of the desired discrete logarithm.

In Algorithm 3, for simplicity of notation, assume $t = n$ where n is the total number of participants and t is the threshold, and that $\mathcal{F}$ controls $P_1, \ldots P_{t-1}$, and $\mathcal{A}$ simulates the honest participant P_t to $\mathcal{F}$.

Note that in Line 8 of Algorithm 3, $\mathcal{A}$ cannot compute Y_t directly by g^{s_t} as both s_t and the discrete logarithm of ω are unknown to them. After the SimDistKeyGen Algorithm has completed, each party can follow the DistKeyGen protocol to derive the group public key as $Y = \Pi_{j=1}^{t} \phi_{j,0}$. Additionally, $\mathcal{C}$ must obtain the secret values $a_{1,0}, \ldots a_{(t-1),0}$ from $\mathcal{F}$ using the extractor for the zero-knowledge proofs that $\mathcal{F}$ generates, and in turn $\mathcal{C}$ can use these values to extract the discrete logarithm of ω from the discrete logarithm of the group public key Y.

Algorithm 3 Algorithm SimDistKeyGen(ω)

Input A challenge value ω and
$\{\boldsymbol{C}_i\}_{i=1}^{t-1} = \left\{\left\langle \phi_{i,0}, \ldots, \phi_{i,(t-1)} \right\rangle\right\}_{i=1}^{t-1}$
Output $\boldsymbol{C}_t, Y_t$

1: **Round 1**: $\mathcal{A}$ does the following.
2: $\bar{x}_{t,1}, \ldots, \bar{x}_{t,t-1} \xleftarrow{\$} \mathbb{Z}_q$ serving as the secret shares corresponding to $f_t(1), \ldots f_t(t-1)$
3: $\phi_{t,0} \leftarrow \omega$
4: Calculate $\phi_{t,1}, \ldots, \phi_{t,(t-1)}$ by performing Lagrange Interpolation in the exponent, i.e. $\phi_{t,j} = \omega^{\lambda_{j,t}} \cdot g^{\sum_{i=1}^{t-1} \lambda_{j,i} \cdot \bar{x}_{t,i}}$
5: Broadcasts $\boldsymbol{C}_t = \left\langle \phi_{t,0}, \ldots, \phi_{t,(t-1)} \right\rangle$.
6: **Round 2**: $\mathcal{A}$ sends $(1, \bar{x}_{t,1}), \ldots, (t-1, \bar{x}_{t,1})$ to the parties corrupted by $\mathcal{F}$.
7: $\mathcal{A}$ simulates proof of knowledge of $a_{t,0}$ by deriving σ_t as

$$c_t, \mu_t \xleftarrow{\$} \mathbb{Z}_q \; ; \; R_t = g^z \cdot \omega^{-\mu_t} \; ; \; \sigma_t = (R_t, \mu_t).$$

8: $\mathcal{A}$ derives the public key Y_t for the party P_t by following the same steps the other parties can calculate Y_t as

$$Y_t = \prod_{j=1}^{t} \prod_{k=0}^{t-1} (\phi_{j,k})^{t^k \bmod q}.$$

Algorithm 4 Algorithm
ExtractDiscreteLog($\omega, (a_{1,0}, \ldots a_{(t-1),0}), h_J, h'_J, \pi, \pi'$)

Input A challenge value ω, secret values $a_{1,0}, \ldots a_{(t-1),0}$, two distinct random oracle responses h_J, h'_J and their corresponding forgeries π, π'
Output The discrete logarithm of ω

1: Parse π, π' as $(\gamma, c, z), (\gamma, c', z')$, where $c = h_J$ and $c' = h'_J$
2: Compute discrete logarithm of Y as $\log_g(Y) = \frac{z'-z}{h'_J - h_J}$
3: Compute $a_{t,0} = \log_g(Y) - \sum_{i=1}^{t-1} a_{i,0}$
4: Return $\log_g(\omega) = a_{t,0}$

Regarding Line 1 of Algorithm 4, the Icy-DVRF consistency property guarantees that any two forgeries share the same γ. This applies to the general case ($t < n$) as well as the special case ($t = n$). In Line 2, because both forgeries are valid VRF proofs, it follows that

$$R = g^z \cdot Y^c = g^{z'} \cdot Y^{c'}$$

and since $c = h_J, c' = h'_J$ with $h_J \neq h'_J$, we can write $g^z \cdot g^{-z'} = Y^{h_J} \cdot Y^{-h'_J}$, which can be re-written as $g^{\frac{z'-z}{h'_J - h_J}} = Y$, incidentally yielding

$$\log_g(Y) = \frac{z' - z}{h'_J - h_J} \, .$$

Algorithm 5 Algorithm $\mathcal{A}(Y, \{h_1, \ldots, h_{n_r}\}; \Xi)$

Input A public key Y and random oracle outputs $\{h_1, \ldots, h_{n_r}\}$
Output An index J and a forgery π, or $\perp$

1: Initialize $ctr = 1, T_\rho = \{\}, T_\pi = \{\}, T_1 = \{\}$,
$T_2 = \{\}, T_3 = \{\}, T_4 = \{\}, J_2 = \{\}, C = \{\}, M = \{\}$
2: Run $\mathcal{F}$ on input Y, answering its queries as follows, until it outputs $(\alpha, \pi = (\gamma, c, z))$ or $\perp$
3: **On simulating** $H_\rho(\alpha, B)$:
4: If $T_\rho[\alpha, B] = \perp$, set $T_\rho[\alpha, B] = h_{ctr}$; $ctr = ctr + 1$. Return $T_\rho[\alpha, B]$.
5: **On simulating** $H_\pi(g, h, g^s, h^s, g^r, h^r)$:
6: If $T_\pi[g, h, g^s, h^s, g^r, h^r] = \perp$, set $T_\pi[g, h, g^s, h^s, g^r, h^r] = h_{ctr}$; $ctr = ctr+1$. Return $T_\pi[g, h, g^s, h^s, g^r, h^r]$.
7: **On simulating** $H_1(\alpha)$:
8: If $T_1[\alpha] = \perp$, sample $r \xleftarrow{\$} \mathbb{Z}_q$; $T_1[\alpha] = g^r$. Return $T_1[\alpha]$.
9: **On simulating** $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$:
10: If $T_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = \perp$, set $T_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = h_{ctr}$, $J_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = ctr$; $ctr = ctr + 1$. Return $T_2[g, h, Y, \gamma, R, \bar{R}, \alpha]$.
11: **On simulating** $H_3(\mathbf{X})$:
12: If $T_3[\mathbf{X}] = \perp$, set $T_3[\mathbf{X}] = h_{ctr}$; $ctr = ctr + 1$. Return $T_3[\mathbf{X}]$.
13: **On simulating** $H_4(i, \Phi)$:
14: If $T_4[i, \Phi] = \perp$, set $T_4[i, \Phi] = h_{ctr}$; $ctr = ctr + 1$. Return $T_4[i, \Phi]$.
15: **On simulating Preprocess**$^{\mathbf{I}}$:
16: **Round 1**:
17: For $1 \leq j \leq \Omega$, do:
18: Set $\bar{c}_j = h_{ctr}, C[j] = ctr, ctr = ctr + 1, \bar{z}_{t,j} \xleftarrow{\$} \mathbb{Z}_q, D_{t,j} = g^{\bar{z}_{t,j}} \cdot Y_t^{-\bar{c}_j}$.
19: Follow the protocol honestly to sample $e_{t,j}, a_{t,j}, b_{t,j} \xleftarrow{\$} \mathbb{Z}_q$ and derive
20: $E_{t,j} = g^{e_{t,j}}, A_{t,j} = g^{a_{t,j}}, B_{t,j} = g^{b_{t,j}}$.
21: Follow the protocol honestly to sample $k_{a_{t,j}}, k_{b_{t,j}} \xleftarrow{\$} \mathbb{Z}_q$ and derive
22: $R_{a_{t,j}} = g^{k_{a_{t,j}}}, R_{b_{t,j}} = g^{b_{t,j}}$.
23: Derive $K_t = H_3(K_{t,1}, \ldots, K_{t,\Omega})$, publish to $\mathcal{F}$, wait for all K_ℓ values from $\mathcal{F}$.

Round 2:
Derive L_t, Φ, J_t honestly. Send (t, L_t, J_t) to $\mathcal{F}$, wait to receive the (ℓ, L_ℓ, J_ℓ) tuples from $\mathcal{F}$, following the protocol for validation.
Reprogram $T_3[K_1, \ldots, K_t] = h_{ctr}$; set $ctr = ctr + 1$. Rederive c_t and J_t honestly.
Rewind $\mathcal{F}$ to Step 1 in Round 2 of Preprocess-Interactive, immediately before $\mathcal{F}$ queries H_3 with $(K_1, \ldots, K_t)$.
After $\mathcal{F}$ rewinds, use the two outputs to derive the discrete logarithm of each $A_{l,j}$ and $B_{l,j}$; store for use in the later protocols.
Complete the protocol honestly.
On simulating PartialEval:
Insert α into M.
$\mathcal{A}$, acting as P_t, samples $r \xleftarrow{\$} \mathbb{Z}_q$, sets $h = g^r = H_1(\alpha)$.
$\mathcal{A}$ simulates $\gamma_t = (Y_t)^r = (g^r)^{s_t} = h^{s_t}$.
$\mathcal{A}$ simulates a DLEQ proof by sampling $\bar{s}, \bar{c} \xleftarrow{\$} \mathbb{Z}_q$, deriving $R_t = g^{\bar{s}} \cdot (Y_t)^{-\bar{c}}$ and $\bar{R}_t = h^{\bar{s}} \cdot (\gamma_t)^{-\bar{c}}$ and programs the random oracle so that $\bar{c} = H_\pi(g, h, g^{\bar{s}}, h^{\bar{s}}, R_t, \bar{R}_t)$, hence $(\bar{s}, \bar{c})$ is a valid DLEQ proof.
Derive $\bar{D}_t = h^{\bar{z}_{t,j}} \cdot \gamma^{\bar{c}_j}$ and $\bar{E}_t = h^{e_{t,j}}$.
Output $(\gamma_t, \pi_t, \bar{D}_t, \bar{E}_t)$ by sending it to $\mathcal{F}$.
On simulating PartialProofGen:
Round 1:
Receive $(\gamma_\ell, \pi_\ell, \bar{D}_\ell, \bar{E}_\ell)$ for $\ell = 1, \ldots, t-1$.
Using $(a_{\ell,j}, b_{\ell,j})$ obtained during Preprocess-Interactive, derive ρ_ℓ for $\ell = 1 \ldots, t-1$.
Derive $\rho_t = a_{t,j} + b_{t,j} \cdot H_\rho(\alpha, B)$, R, $\bar{R}$, and γ, honestly following the protocol.
Program $T_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = \bar{c}_j, J_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = C[j]$.
Return ρ_t.
Round 2:
Validate ρ_l for $\ell = 1, \ldots, t-1$.
Let $z_t = \bar{z}_{t,j} + e_{t,j} \cdot \rho_t, c = \bar{c}_j$
Return (z_t, c) to $\mathcal{F}$
On simulating Combine:
If $\mathcal{F}$ outputs $\perp$, then return $\perp$. Else, $\mathcal{F}$ outputs $(\alpha, (\gamma, z, c))$.
If $T_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = \perp$, set $T_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = h_{ctr}$, $J_2[g, h, Y, \gamma, R, \bar{R}, \alpha] = ctr$, and $ctr = ctr + 1$.
$c = T_2[g, h, Y, \gamma, R, \bar{R}, \alpha]$. If $R \neq g^z \cdot Y^{-c}$, or $\bar{R} \neq h^z \cdot \gamma^{-c}$ or $\alpha \in M$, then return $\perp$
Let $J = J_2[g, h, Y, \gamma, R, \bar{R}, \alpha]$.
Return $J, \pi = (\gamma, z, c)$.

Let n_h, n_p, and n_v denote the numbers of queries to the random oracle, the preprocess queries, and the VRF-creation queries, respectively.

Theorem 3. *If the discrete logarithm problem in $\mathbb{G}$ is (τ', ϵ')-hard, then the T-DLEQ-Int scheme over $\mathbb{G}$ with n proving participants, a threshold of t, and a*

preprocess batch size of Ω is $(\tau, n_h, n_p, n_v, \epsilon)$-secure whenever

$$\epsilon' \leq \frac{\epsilon^2}{2n_h + \Omega \cdot n_p}$$

and

$$\tau' \leq 4 \cdot \tau + [(8t+4) \cdot \Omega \cdot n_p + (10t+12) \cdot n_v + (n+t-1)t + 6]\, t_{exp} + O(\Omega \cdot n_p + n_v + n_h),$$

where t_{exp} is the time of an exponentiation in $\mathbb{G}$, assuming the number of participants compromised by the adversary is less than the threshold t.

Proof. Assume that $\mathcal{F}$ is a $(\tau, n_h, n_p, n_v, \epsilon)$-forger for T-DLEQ-Int. We show that the algorithm $\mathcal{C}$ presented in Algorithm 2 can (τ', ϵ')-solve the discrete logarithm of an arbitrary challenge $\omega \in \mathbb{G}$.

$\mathcal{A}$ from Algorithm 5 simulates T-DLEQ-Int to $\mathcal{F}$, who controls the dishonest participants $P_1, \ldots, P_{t-1}$ while $\mathcal{A}$ simulates the single honest participant P_t in the proof creation. Let $n_r = 2n_h + \Omega \cdot n_p$ be the maximum number of random oracle outputs $\mathcal{A}$ may require.

Once SimDistKeyGen from Algorithm 3 is completed, $\mathcal{A}$ invokes $\mathcal{F}$ by simulating both the role of P_t and the responses to the random oracle queries of $\mathcal{F}$ in all of the protocols to follow.

To simulate random oracle queries for $H_\rho, H_\pi, H_1, H_2, H_3$, and H_4, the algorithm $\mathcal{A}$ utilizes a stateful-table-based approach. For any fresh query, $\mathcal{A}$ assigns the successor value from a set $\{h_1, \ldots, h_{n_r}\}$ supplied by $GF_{\mathcal{A}}$ and records the result to ensure consistency. The interactive protocol diverges from the original by replacing the static binding hash H_1 with an interactive one-time VRF that accepts $H_\rho(\alpha, B)$ as input.

For the simulation of Preprocess-Interactive, $\mathcal{A}$ simulates P_t, faithful to the standard protocol with a specific deviation during the generation of $D_{t,j}$: $\mathcal{A}$ selects a controlled hash value $\bar{c}_j$ and a random scalar $\bar{z}_{t,j}$ to derive $D_{t,j} = g^{\bar{z}_{t,j}} \cdot Y_t^{-\bar{c}_j}$. Crucially, to extract the discrete logarithms $(a_{\ell,j}, b_{\ell,j})$ from dishonest parties controlled by $\mathcal{F}$, $\mathcal{A}$ employs a forking strategy during the second round. By rewinding $\mathcal{F}$ to the point prior to the query $\Phi = H_3(K_1, \ldots, K_t)$ and reprogramming the random oracle with a distinct output Φ', $\mathcal{A}$ forces $\mathcal{F}$ to republish commitment values, thereby solving the underlying discrete logarithms.

During the simulation of PartialEval, $\mathcal{F}$ initiates T-DLEQ-Int as the combiner by sending (j, α, B). Once again, $\mathcal{A}$ simulates P_t, staying true to the protocol except that it simulates γ_t and the associated DLEQ proof π_t. Having possession of the values $\bar{c}_j$ and $\bar{z}_{t,j}$, $\mathcal{A}$ derives $(\bar{D}_t, \bar{E}_t)$ and sends them to $\mathcal{F}$.

To simulate PartialProofGen, $\mathcal{F}$ initiates T-DLEQ-Int as the combiner once again. This time, leveraging the discrete logarithms $(a_{\ell,j}, b_{\ell,j})$ extracted previously, $\mathcal{A}$ uses them to compute not only its own ρ_t, but also the binding values ρ_ℓ

for all other participants. Now $\mathcal{A}$ can derive $R, \bar{R}$, and γ, honestly following the protocol. $\mathcal{A}$ programs the random oracle $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha) = \bar{c}_j$ and saves $C[j] = ctr$ so that $\bar{c}_j = h_{ctr}$, in the table J_2.

Finally, in the simulation phase for Combine, $\mathcal{F}$ initiates T-DLEQ-Int as the combiner, yet again. $\mathcal{A}$ is capable of computing the commitments $R, \bar{R}$ and γ prior to $\mathcal{F}$, thereby guaranteeing the successful programming of the random oracle entry $H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$.

Note that the success probability of the adversary $\mathcal{F}$ in a single execution of the simulator $\mathcal{A}$ is denoted by ϵ, given that $\mathcal{A}$ is capable of returning the correct challenge for every signing query. By applying the General Forking Lemma, which entails running $\mathcal{A}$ twice, the discrete logarithm of ω can be extracted. The probability of successful extraction is greater than $\frac{\epsilon^2}{n_r}$; this bound neglects the negligible term $\frac{\epsilon}{h}$ under the assumption that the hash output space h is sufficiently large (typically $h \geq 2^{256}$).

The runtime complexity of $\mathcal{F}$ is derived from several factors. The base runtime of $\mathcal{F}$ is scaled by a factor of four due to the nested forking strategy, wherein $\mathcal{A}$ forks $\mathcal{F}$, and the reduction itself forks $\mathcal{A}$. The following number of exponentiations is needed during the simulations to produce two forgeries.

- During SimDistKeyGen: $(t-1)t$, 2, and nt to compute $\boldsymbol{C}_t$, R, and Y_t, respectively.
- In each of the two executions of $\mathcal{A}$:
  - During Preprocess-Int. Round 1: 1 for each of $D_{t,j}, E_{t,j}, A_{t,j}, B_{t,j}, R_{a_{t,j}}$, and $R_{b_{t,j}}$, Ω times.
  - During Preprocess-Int. Round 2: 2 for each of R'_a and R'_b, for $t-1$ parties, Ω times.
  - During PartialEval: $1, 2, 2, 2$, and 1 for derivations of $\gamma_t, R_t, \bar{R}_t, \bar{D}_t$, and $\bar{E}_t$, respectively.
  - During PartialProofGen Round 1: t, t, and t for derivations of R, $\bar{R}$, and γ, respectively.
  - During PartialProofGen Round 2: 2 for each validation of ρ_l, for $t-1$ parties.
  - During Verification of Forgery: 2 for each of T and $\bar{T}$.

Accounting for the $O(\Omega \cdot n_p + n_v + n_h)$ minor operations, such as table lookups, the total running time of $\mathcal{F}$ is given as

$$\tau' \leq 4 \cdot \tau + [(8t+4) \cdot \Omega \cdot n_p + (10t+12) \cdot n_v + (n+t-1)t + 6]\, t_{exp} + O(\Omega \cdot n_p + n_v + n_h).$$

□

B Knowledge-Soundness of T-DLEQ

In this appendix, we present the proof of the knowledge-soundness clause of Theorem 2: any PPT adversary that produces a verifying, non-trivial, T-DLEQ proof

can be turned into a solver for the one-more discrete logarithm (OMDL) problem, with the distributed key generation handled by the Schnorr knowledge-of-exponent assumption (Schnorr-KoE), which holds under the discrete logarithm assumption in the algebraic group model (AGM) [17, Thm. 1].

B.1 Overview of Proof

We present our reduction in the *single-honest-party* convention as in the interactive-security appendix (Appendix A). We assume the adversary $\mathcal{F}$ controls $P_1, \ldots, P_{t-1}$ and the OMDL challenge is embedded in the one honest prover P_t. We mimic the proof of security of FROST1 in the modular style of Crites-Komlo-Maller [17] and Bellare et al. [4]. Instead of rewinding the full protocol, we introduce an intermediate *binonce DLEQ* (Bi-DLEQ) assumption (Definition 5) that abstracts the preprocessing structure of Icy-DVRF, so that the T-DLEQ knowledge-soundness is reduced to Bi-DLEQ by a straight-line simulation of P_t that runs the adversary only once (Lemma 7). Furthermore, Bi-DLEQ reduces to OMDL (Lemma 4), which keeps all rewinding inside a single forking step.

The proof is assembled from five components. Lemma 5 justifies the single-honest-party reduction and isolates its soundness factor κ, which is 1 under the fixed-quorum convention and at most $\min\{t, n-t+1\} \leq n$ in the general setting. Using Lemma 6, one can extract the corrupt key shares via Schnorr-KoE and write the honest share in the affine form $Y_t = Y^{\alpha_t} g^{\beta_t}$ that the simulation relies on, while using Lemma 3, we show that the second verification equation (in base h) is redundant, so the extraction reduces to the single base g.

Then, a winning forgery is split into two disjoint events: an *insufficient-quorum* forgery ($\mathcal{E}_{\mathsf{quo}}$), handled by a general forking at the challenge query, and a *token-substitution* forgery ($\mathcal{E}_{\mathsf{sub}}$), handled by a local forking (in the sense of Bellare-Dai-Li [5]) of the single binding output ρ_t, which yields a two-equation linear system in the embedded discrete logarithms. The freshness predicate is based on the aggregate commitment pair $(R^*, \bar{R}^*)$.

While [17] proves the security of the three-round interactive FROST1 scheme under the DL assumption, the security of their two-round preprocessed variant relies on OMDL. Because Icy-DVRF inherits this preprocessing structure, OMDL, along with Schnorr-KoE/AGM for the DKG, is the appropriate assumption set. This remains consistent with Theorem 3 (a DL statement), demonstrating that avoiding the interactive one-time VRF in the interactive version of Icy-DVRF trades the DL assumption for OMDL.

B.2 Setting, Assumptions, and the Bi-DLEQ Assumption

Icy-DVRF is a (t, n)-threshold scheme: any t of the n provers reconstruct, and a static adversary corrupts a set Cor with $|\mathrm{Cor}| \leq t-1$, leaving honest set $\mathrm{Hon} = [n] \setminus \mathrm{Cor}$. The OMDL game gives a challenge oracle $\mathsf{Chal}()$ returning a fresh g^x (x secret) and a discrete logarithm oracle $\mathsf{Dlog}(\cdot)$; with c_{id} challenge and ℓ discrete logarithm queries, the adversary wins iff $\ell < c_{\mathrm{id}}$ and it outputs all challenge logarithms; $\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B})$ is the advantage of this adversary [17, Def. 2].

Schnorr-KoE provides an extractor Ext that recovers the secret key behind any verifying Schnorr proof of possession, except with probability $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\cdot)$ [17, Assumption 4], [4, Theorem 8].

In the remainder of this document, $\mathbb{G} = \langle g \rangle$ has prime order q, and $H_1 : \{0,1\}^* \to \mathbb{G}$ and $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ are random oracles, with $h = H_1(\alpha)$, $\rho_\ell = H_2(\ell, \alpha, B)$, and $c = H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$, observing the domain separation conventions of Section 3.1. Also, let n_h and n_{pr} be the number of random-oracle and proving queries, respectively, and $Q = n_h + n_{\mathsf{pr}} + 1$.

Definition 5. *Let $\mathbb{G} = \langle g \rangle$ be a (cyclic) group of prime order q. The Bi-nonce Discrete Logarithm Equality (Bi-DLEQ) assumption holds if for all PPT adversaries $\mathcal{A}$,*

$$\mathsf{Adv}^{\text{bi-dleq}}_{\mathbb{G}}(\mathcal{A}) = \Pr[\, \mathsf{G}^{\text{bi-dleq}}_{\mathbb{G}}(\mathcal{A}) \Rightarrow \mathsf{true} \,]$$

is negligible, where $\mathsf{G}^{\text{bi-dleq}}_{\mathbb{G}}$ is the game of Algorithm 6.

Algorithm 6 The Bi-DLEQ game $\mathsf{G}^{\text{bi-dleq}}_{\mathbb{G}}$ (binonce DLEQ; two bases)

Init: $\dot{x} \xleftarrow{\$} \mathbb{Z}_q$; $Y \leftarrow g^{\dot{x}}$; $Q_{\text{bin}}, Q_{\text{used}}, Q_{\text{prove}} \leftarrow \emptyset$;
return Y

$\mathsf{BinonceO}()$: $d, e \xleftarrow{\$} \mathbb{Z}_q^*$; $(D, E) \leftarrow (g^d, g^e)$; $Q_{\text{bin}} = Q_{\text{bin}} \cup \{(D, E, d, e)\}$; **return** (D, E)

$\mathsf{BiproveO}\big(k, \alpha, S, \{(\psi_i, D_i, E_i, \bar{D}_i, \bar{E}_i)\}_{i \in S}, \gamma\big)$:
  require $(D_k, E_k, d_k, e_k) \in Q_{\text{bin}}$ **and** $(D_k, E_k) \notin Q_{\text{used}}$
  $Q_{\text{used}} = Q_{\text{used}} \cup \{(D_k, E_k)\}$; $h \leftarrow H_1(\alpha)$
  require $\bar{D}_k = h^{d_k} \wedge \bar{E}_k = h^{e_k}$
  for $i \in S$: $\rho_i \leftarrow H_2(i, \alpha, B_S)$; $R \leftarrow \prod_i D_i E_i^{\rho_i}$; $\bar{R} \leftarrow \prod_i \bar{D}_i \bar{E}_i^{\rho_i}$
  $c \leftarrow H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$; $z_k \leftarrow d_k + \rho_k e_k + c\, \psi_k\, \dot{x}$
  $\mathsf{ctr}(\alpha, R, \bar{R}) = \mathsf{ctr}(\alpha, R, \bar{R}) \cup \{(k, B_S)\}$; **return** z_k

$\mathsf{RO}_1(x)$: **return** $H_1(x)$ $\qquad$ $\mathsf{RO}_2(x)$: **return** $H_2(x)$

Fin$(\alpha^*, \gamma^*, c^*, z^*)$: $h^* \leftarrow H_1(\alpha^*)$; $T \leftarrow g^{z^*} Y^{-c^*}$; $\bar{T} \leftarrow (h^*)^{z^*} (\gamma^*)^{-c^*}$
  require $c^* = H_2(g, h^*, Y, \gamma^*, T, \bar{T}, \alpha^*)$
  require $\big|\{k : (k, \cdot) \in \mathsf{ctr}(\alpha^*, T, \bar{T})\}\big| < t - |\mathrm{Cor}|$
  return true

The Bi-DLEQ game of Algorithm 6 gives an adversary access to two oracles, $\mathsf{BinonceO}$ and $\mathsf{BiproveO}$, two random oracles H_1, H_2, and a challenge public key $Y = g^{\dot{x}}$, and asks it to forge a fresh two-base DLEQ proof with respect to Y. The $\mathsf{BinonceO}$ oracle takes no input and returns a "token", i.e., a one-time commitment pair, $(D, E) = (g^d, g^e)$, recording the discrete logarithms (d, e) for later use. The $\mathsf{BiproveO}$ oracle takes an index k, an input α, a prover set S, the per-prover commitments $\{(\gamma_i, D_i, E_i, \bar{D}_i, \bar{E}_i)\}_{i \in S}$, and the aggregate evaluation

γ. It checks if (D_k, E_k) is a prior BinonceO response that has not been used before, returning an error otherwise. Then, it forms the bindings $\rho_i = H_2(i, \alpha, B_S)$, the aggregate commitments $R = \prod_i D_i E_i^{\rho_i}$ and $\bar{R} = \prod_i \bar{D}_i \bar{E}_i^{\rho_i}$, the challenge $c = H_2(g, h, Y, \gamma, R, \bar{R}, \alpha)$, and returns z_k such that $(R, \bar{R}, c, z_k)$ is a valid partial response for slot k. The single-use restriction ensures that BiproveO may be called at most once per commitment pair (D, E), and limits the reduction to one Dlog query per challenge. This is the source of the OMDL budget of Lemma 4. The scalar ψ_k lets the response be given as $z_k = d_k + \rho_k e_k + c\,\psi_k\,\dot{x}$ rather than $d_k + \rho_k e_k + c\,\dot{x}$; in the threshold setting ψ_k absorbs the Lagrange coefficient λ_k together with the affine key factor α_k of the honest-share representation (Lemma 6). The adversary wins if it outputs a verifying $(\alpha^*, \gamma^*, c^*, z^*)$ whose aggregate commitment pair $(T, \bar{T})$ records fewer than $t - |\mathrm{Cor}|$ honest contributions, i.e. a proof object not produced by a full honest quorum.

We present our proof of Theorem 2 in a modular fashion using a sequence of lemmas. Lemmas 2, 3, and 4 are independent of which parties are corrupt.

Lemma 2. *For any statement (g, h, Y_k, γ_k) with $\gamma_k = h^{s_k}$ and α such that $h = H_1(\alpha)$, the simulator that samples $\bar{c}, \bar{z} \xleftarrow{\$} \mathbb{Z}_q$, sets $R_k = g^{\bar{z}} Y_k^{-\bar{c}}$, $\bar{R}_k = h^{\bar{z}} \gamma_k^{-\bar{c}}$, and programs $H_2(g, h, Y_k, \gamma_k, R_k, \bar{R}_k, \alpha) \leftarrow \bar{c}$, yields a proof $(\bar{c}, \bar{z})$ identically distributed to an honest one, provided the programmed point was undefined.*

Proof. In an honest generation, we sample $r \xleftarrow{\$} \mathbb{Z}_q$ and output $\bar{c} = H_2(g, h, Y_k, \gamma_k, R_k, \bar{R}_k, \alpha)$, which is uniform since H_2 is a random oracle, and $\bar{z} = r - s_k \bar{c}$, which is also uniformly distributed and independent of $\bar{c}$, since r is uniformly distributed. The simulator draws $(\bar{c}, \bar{z})$ uniformly and independently, and defines $(R_k, \bar{R}_k)$ by the verification relations $R_k = g^{\bar{z}} Y_k^{-\bar{c}}$, $\bar{R}_k = h^{\bar{z}} \gamma_k^{-\bar{c}}$, which fix them identically; programming the challenge oracle at exactly the point the verifier reads makes $(\bar{c}, \bar{z})$ pass verification. Hence the two output distributions coincide. □

Lemma 3. *Let $h = H_1(\alpha^*) = g^\eta$ with $\eta \in \mathbb{Z}_q^*$ known and $\varphi(x) = x^\eta$, a bijection on $\mathbb{G}$. When $\gamma^* = h^s$, we have $\gamma^* = Y^\eta$, therefore $g^z = R^* Y^{c^*}$ if and only if $h^z = \bar{R}^* (\gamma^*)^{c^*}$. Hence, the twin equation carries no information independent of the base-g equation; any discrete log extraction valid in the single-base setting transfers verbatim, with $\bar{D}_i = D_i^\eta$, $\bar{E}_i = E_i^\eta$.*

Proof. Note that φ is a homomorphism and a bijection since $\gcd(\eta, q) = 1$. Under φ, g^z is mapped to h^z, $R^* = \prod_i D_i\, E_i^{\rho_i}$ is mapped to $\prod_i D_i^\eta\, E_i^{\eta\rho_i} = \prod_i \bar{D}_i\, \bar{E}_i^{\rho_i} = \bar{R}^*$, and Y^{c^*} is mapped to $(Y^\eta)^{c^*} = (\gamma^*)^{c^*}$. Since applying a bijection to an equation results in an equivalence, the two verification equations are logically equivalent under the uniqueness premise. Therefore, the system a forger must satisfy, and the system an extractor solves, are identical to the single-base case. □

Lemma 4. *For every Bi-DLEQ adversary $\mathcal{A}$ making at most n_b queries to* BinonceO*, n_{pr} to* BiproveO*, and n_h random-oracle queries, there is an OMDL*

adversary $\mathcal{B}$ with

$$\mathsf{Adv}^{\text{bi-dleq}}_{\mathbb{G}}(\mathcal{A}) \le \sqrt{Q\big(\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3(n+1)^2Q^2/q\big)},$$

where $Q = n_h + n_{\mathsf{pr}} + 1$, $\mathcal{B}$ making at most $n_{\mathsf{pr}} + 1$ challenge and n_{pr} discrete logarithm queries (so $\ell < c_{\text{id}}$ in the OMDL game), where $\mathcal{B}$ runs in time roughly twice that of $\mathcal{A}$ plus the time to perform $O\big((n+1)(n_b+n_{\mathsf{pr}}+n_h)\big)$ group operations.

Proof. $\mathcal{B}$ plays OMDL and simulates $\mathsf{G}^{\text{bi-dleq}}_{\mathbb{G}}$ to $\mathcal{A}$.

Embedding the challenge. $\mathcal{B}$ calls $\mathsf{Chal}()$ once for $Y = g^{\dot{x}}$, never queries $\mathsf{Dlog}(Y)$, and programs $h = H_1(\alpha) = g^{\eta_\alpha}$ with known η_α for each fresh α. Thus, a single base g defines both the g- and h-commitments. Hence, anything simulatable at base g lifts to base h by exponentiating by η_α, which is what lets a two-base forgery yield a single-base OMDL solution.

$\mathsf{BinonceO}$. $\mathcal{B}$ answers with self-sampled $D_j = g^{a_j}, E_j = g^{e_j}$ (with known discrete logarithms) and the corresponding twins $\bar{D}_j = D_j^{\eta_\alpha}, \bar{E}_j = E_j^{\eta_\alpha}$. Since no Chal is drawn here, a binonce that is never used in a subsequent $\mathsf{BiproveO}$ query carries only known discrete logarithms and is never handed to the OMDL game.

$\mathsf{BiproveO}$. On the first use of binonce k, $\mathcal{B}$ draws $\mathsf{Chal}() \to U_k = g^{u_k}$ (with u_k unknown), overwrites $D_k \leftarrow U_k$, and sets $\bar{D}_k \leftarrow U_k^{\eta_\alpha}$, since $g^{z_k} = D_k E_k^{\rho_k} Y^{c\psi_k} = U_k g^{e_k\rho_k} Y^{c\psi_k}$ for an honest response. To produce z_k, $\mathcal{B}$ needs the discrete logarithm of $U_k Y^{c\psi_k}$, but both U_k and Y have unknown discrete logarithms. So, $\mathcal{B}$ makes one query $\mathsf{Dlog}(W_k)$ on $W_k := U_k Y^{c\psi_k}$, obtaining $w_k = u_k + c\psi_k\dot{x}$, and returns $z_k \leftarrow w_k + \rho_k e_k$; the twin holds as $h^{z_k} = (g^{z_k})^{\eta_\alpha}$. By the single-use requirement, each $\mathsf{BiproveO}$ consumes a distinct binonce, hence one Chal and one Dlog.

Budget. Writing $n'_{\mathsf{pr}} \le n_{\mathsf{pr}}$ for the number of $\mathsf{BiproveO}$ queries issued, the query counts are as follows.

phase	# Chal	# Dlog
key embedding	1	0
each $\mathsf{BinonceO}$	0	0
each $\mathsf{BiproveO}$	1	1
forking / Fin	0	0
total	$1 + n'_{\mathsf{pr}}$	n'_{pr}

The key embedding draws one Chal to obtain $Y = g^{\dot{x}}$ and never queries $\mathsf{Dlog}(Y)$. Each $\mathsf{BinonceO}$ reply uses self-sampled D_j, E_j with known discrete logs, so no oracle is called. Each $\mathsf{BiproveO}$ on its first use draws one Chal to get U_k, sets $D_k \leftarrow U_k$, and makes one $\mathsf{Dlog}(W_k)$ query. The forking and Fin steps recover all discrete logarithms purely arithmetically, without further oracle calls. Thus, in the notation of the OMDL game, $c_{\text{id}} = 1 + n'_{\mathsf{pr}}$ and $\ell = n'_{\mathsf{pr}}$: the single added challenge Y is supplied by the forgery.

Forking and extraction. Let J index the query for H_2 at $(g, h^*, Y, \gamma^*, R^*, \bar{R}^*, \alpha^*)$ whose output equals c^* in $\mathcal{A}$'s forgery. By the

Generalized Forking Lemma, rerunning $\mathcal{A}$ on the same coins and oracle answers up to index $J-1$ yields, with the stated probability, a second forgery with the same $(\alpha^*, \gamma^*, R^*, \bar{R}^*)$ but $c^{*\prime} \neq c^*$, and hence

$$\dot{x} = \log_g Y = \frac{z^* - z^{*\prime}}{c^* - c^{*\prime}} \bmod q\,.$$

Then, $\mathcal{B}$ outputs $\dot{x}$ and each consumed $u_j \leftarrow w_j - c_j \psi_j \dot{x}$ to win the OMDL game.

Bad-event slack. A single proving query writes, besides the one challenge entry, up to n binding entries $\rho_i = H_2(i, \alpha, B)$. Over the complete run, the game touches at most $(n+1)Q$ distinct H_2 entries. The simulation is perfect unless one of four bad events occurs.

- "programming collision": when $\mathcal{B}$ programs one of the fewer than $(n+1)Q$ touched H_2 entries, its sampled value coincides with an entry already defined, which happens with probability at most $\binom{(n+1)Q}{2}/q$.
- "replacement detection": the overwrite $D_k \leftarrow U_k$ in $\mathsf{BiproveO}$ is observable only if $\mathcal{A}$ has already queried H_2 at a point depending on the replaced binonce, which is bounded by $(n+1)\, n_h\, n_{\mathsf{pr}}/q$.
- "output collision": two of the fewer than $2(n+1)Q$ consumed random-oracle outputs collide, with probability at most $\binom{2(n+1)Q}{2}/q$.
- "premature challenge-hash query": $\mathcal{A}$ queries the forgery's challenge point $H_2(g, h^*, Y, \gamma^*, R^*, \bar{R}^*, \alpha^*)$ before $\mathcal{B}$ has programmed it, which occurs with probability at most n_h/q.

Taking the union bound over these four events and using $n_h \le Q^2$ and $n_h\, n_{\mathsf{pr}} \le Q^2$, we obtain

$$\Pr[\mathrm{BAD}] \le 3(n+1)^2 Q^2/q.$$

Under the condition $\neg$BAD, the simulation is perfect, and the forking lemma gives $\mathrm{acc}^2 \le Q(\mathrm{frk} + 1/q)$ with $\mathrm{frk} \le \mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}) + \Pr[\mathrm{BAD}]$, which is the stated bound.

□

B.3 Knowledge-Soundness in the Single-Honest-Party Presentation

Our reduction treats the worst-case scenario when $|\mathrm{Cor}| = t-1$, in which the adversary corrupts the maximum permitted number of parties and the honest set reduces to a single prover P_t, into whose key we embed the OMDL challenge. We prioritize isolating the soundness of the single-honest-party reduction, since it dictates the tightness of the subsequent argument. If the forgery quorum contains multiple honest provers, the reduction must guess which one is substituted, incurring a multiplicative loss proportional to the quorum size. When we collapse the honest set to a single party, we are removing that guess, so the token-substitution extraction (the event $\mathcal{E}_{\mathsf{sub}}$ in the proof of Theorem 4) proceeds

directly. This is precisely why this collapse cannot be treated as a free "without loss of generality", and its cost must be charged explicitly, which is the content of Lemma 5.

Lemma 5. *It suffices to bound* $\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A})$ *for adversaries that corrupt exactly* $|\mathrm{Cor}| = t-1$ *parties, relabeled* $P_1, \ldots, P_{t-1}$*, and produce a forgery whose quorum* S^* *contains a designated honest party* P_t *whose key contains the embedded challenge. Concretely, for every PPT* $\mathcal{A}'$ *corrupting* Cor' *with* $|\mathrm{Cor}'| \le t-1$*, there is a PPT* $\mathcal{A}$ *of the restricted form with*

$$\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}') \;\le\; \kappa \cdot \mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}),$$

where $\kappa = 1$ *under the fixed-quorum convention of the interactive proof: the quorum is* $\{P_1, \ldots, P_t\}$ *with corrupt set* $\{P_1, \ldots, P_{t-1}\}$*, so* P_t *is the single honest party and* $|S^* \cap \mathrm{Hon}| = 1$*. For general adversary-chosen quorums,* $\kappa \le \min\{t,\, n-t+1\} \le n$*.*

Proof. If $|\mathrm{Cor}'| = c < t-1$, the reduction corrupts $t-1-c$ further parties itself; it can do so at no cost, because by the affine representation of Lemma 6 applied to those parties, it already knows their secret keys, and granting the adversary additional corruptions only enlarges its view. Hence $\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}') \le \mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}'')$ for an $\mathcal{A}''$ with $|\mathrm{Cor}''| = t-1$.

After the self-corruption we have $|\mathrm{Cor}''| = t-1$, and since a valid forgery quorum has $|S^*| = t$, at least one party of S^* must be honest: $|S^* \cap \mathrm{Hon}''| = t - |S^* \cap \mathrm{Cor}''| \ge 1$. We must embed the OMDL challenge in the specific honest party on which the forgery relies: the one whose published token is reused, or, if no token is reused, the one at whose challenge query the extraction forks. We designate that party as P_t. Under the fixed-quorum convention the quorum is exactly $\{P_1, \ldots, P_t\}$ with $P_1, \ldots, P_{t-1}$ corrupt, so S^* contains the single honest party P_t, which is therefore determined without any guessing and $\kappa = 1$. For a general adversary-chosen quorum, however, S^* may contain several honest parties; we then guess in advance which one the forgery exercises, choosing uniformly among the $|S^* \cap \mathrm{Hon}''| \le \min\{t, n-t+1\}$ honest members of S^*, and the guess is correct with probability at least $1/\kappa$.

When we relabel $P_1, \ldots, P_{t-1}$ as the corrupt set, and P_t as the embedded honest party, we do not break soundness, since nothing in the proof system or in either assumption depends on the actual party indices: H_2 uses an index only as a binding tag inside $\rho_\ell = H_2(\ell, \alpha, B)$, so permuting the indices permutes the tags consistently. OMDL and Schnorr-KoE are stated without reference to party identities, and Lagrange interpolation is equivariant under any permutation of $\{1, \ldots, n\}$, so the reconstruction remains unchanged when the parties are renamed. Therefore, without loss of generality, we may assume that the designated honest party is P_t. □

Under the convention $\mathrm{Cor} = \{1, \ldots, t-1\}$, $\mathrm{Hon} = \{t\}$, the quorum threshold is $t - |\mathrm{Cor}| = 1$, so the non-triviality predicate reduces to "zero honest contributors": a forgery wins precisely when the single honest party P_t made no genuine

contribution to the aggregate. Consequently, when the forgery reuses an honest party's token, that party is necessarily P_t, and there is nothing further to guess. The following two lemmas instantiate the shared simulation of Section B.2 at the single-honest-party setting $\mathrm{Hon} = \{t\}$: Lemma 6 fixes the algebraic form of P_t's key, and Lemma 7 shows that every message P_t must produce can be simulated from that form together with the Bi-DLEQ oracles, without ever rewinding.

Lemma 6. *Let f be the sharing polynomial of* DistKeyGen *of degree $t-1$, with group secret $f(0) = s$, shares $f(i) = s_i$, and the group public key $Y = g^s$. Let $\mathcal{B}_{\mathsf{koe}}$ denote the Schnorr-KoE adversary that runs the Pedersen-DKG-with-proofs-of-possession simulator (PedPoP), embeds the OMDL challenge as the group key by setting $f(0) = \dot{x}$ with $Y = g^{\dot{x}}$, and applies the extractor* Ext *to each corrupted proof of possession to recover the corrupt shares $\{x_j\}_{j\in\mathrm{Cor}}$, where $x_j = f(j)$. Let L_0 and $\{L_j\}_{j\in\mathrm{Cor}}$ denote the Lagrange basis polynomials for the node set $\{0\} \cup \mathrm{Cor}$, and define the scalars*

$$\alpha_t = L_0(t), \qquad \beta_t = \sum_{j\in\mathrm{Cor}} L_j(t)\, x_j,$$

both of which are computable by the reduction. Then the honest share and public share of P_t admit the affine representation

$$s_t = \alpha_t\, \dot{x} + \beta_t, \qquad Y_t = Y^{\alpha_t} g^{\beta_t}.$$

The extraction of $\{x_j\}_{j\in\mathrm{Cor}}$ fails with probability at most $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$.

Proof. At the $t-1$ corrupt nodes $j \in \mathrm{Cor}$, $\mathcal{B}_{\mathsf{koe}}$ recovers the shares $f(j) = x_j$ through the Schnorr-KoE extractor applied to the corrupted proofs of possession. At the node 0, it embeds the OMDL challenge as the secret: the DKG is simulated so that $f(0) = \dot{x}$ with $Y = g^{\dot{x}}$, a value $\mathcal{B}_{\mathsf{koe}}$ carries symbolically, since learning it would itself solve the discrete logarithm challenge. We include the node 0 precisely because it carries the master secret, where the challenge is planted, and the corrupt nodes are the only additional evaluations $\mathcal{B}_{\mathsf{koe}}$ can obtain. These two sources supply t values of f in total, hence f, and in particular the honest share $f(t)$, is fixed.

The Lagrange basis polynomials for $\{0,\dots,t-1\}$ satisfy $L_0(0) = 1$ and $L_0(j) = 0$ for $j \in \mathrm{Cor} = \{1,\dots,t-1\}$, while each L_j satisfies $L_j(j) = 1$ and vanishes on $\{0,\dots,t-1\} \setminus \{j\}$. Therefore, Lagrange interpolation evaluated at t gives

$$\begin{aligned} s_t = f(t) &= L_0(t)\, f(0) + \sum_{j\in\mathrm{Cor}} L_j(t)\, f(j) \\ &= L_0(t)\, \dot{x} + \sum_{j\in\mathrm{Cor}} L_j(t)\, x_j = \alpha_t\, \dot{x} + \beta_t. \end{aligned}$$

The basis polynomials depend only on the node set $\{0,\dots,t-1\}$ and the evaluation point t, all known to $\mathcal{B}_{\mathsf{koe}}$, so α_t and β_t are known scalars. The challenge $\dot{x}$ appears merely in the single term $\alpha_t\, \dot{x}$, restricting the entire dependence on the unknown to the one coefficient α_t.

Raising the affine relation to the generator g removes the need to know $\dot{x}$:

$$Y_t = g^{s_t} = g^{\alpha_t \dot{x} + \beta_t} = (g^{\dot{x}})^{\alpha_t} g^{\beta_t} = Y^{\alpha_t} g^{\beta_t},$$

with both α_t, β_t known.

The embedding and extraction are carried out by $\mathcal{B}_{\mathsf{koe}}$ as described above, and the extraction fails with probability at most $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$. The Schnorr-KoE assumption it relies on follows from the discrete logarithm (DL) assumption in the algebraic group model (AGM) [17, Thm. 1], [4, Thm. 8]: any algebraic adversary that produces a verifying Schnorr proof of possession also yields its discrete logarithm, so no assumption beyond DL is needed for the extractor. The AGM is invoked only in this DKG phase of the reduction.

□

Lemma 7. *Let* Sim *be the simulator that plays the honest party P_t using the key representation of Lemma 6 and the Bi-DLEQ oracles of Definition 5, while the adversary $\mathcal{F}$ runs the corrupted parties $P_1, \ldots, P_{t-1}$. Then* Sim *answers every* Preprocess*,* PartialEval*,* PartialProofGen*, and* Combine *message of P_t in a single execution of $\mathcal{A}$, without rewinding and without knowledge of the secret share s_t or the nonces of P_t. The resulting transcript is identically distributed to a real one, except on a programming-collision event of probability at most Q^2/q.*

Proof. We exhibit, for each message P_t must send, a simulation that uses only public values, the affine key representation $Y_t = Y^{\alpha_t} g^{\beta_t}$, $s_t = \alpha_t \dot{x} + \beta_t$ of Lemma 6, and the Bi-DLEQ oracles. None of the four steps uses s_t itself, the secret nonces of P_t, or any rewinding, and the corrupted parties need no simulation since $\mathcal{F}$ runs them.

(i) Preprocess *and* PartialEval*.* The token of P_t is obtained as a fresh $\mathsf{BinonceO}() \to (D_t, E_t)$, so Sim does not need the nonce discrete logarithms. To handle the input α, Sim programs the random oracle as $h = H_1(\alpha) = g^{\eta_\alpha}$ with a self-chosen, known exponent η_α, and this makes the base h a known power of g and lets every base-h value be computed from its base-g counterpart. Then, the partial evaluation γ_t of P_t becomes public, since by Lemma 6

$$\gamma_t = h^{s_t} = (g^{s_t})^{\eta_\alpha} = Y_t^{\eta_\alpha} = (Y^{\alpha_t} g^{\beta_t})^{\eta_\alpha},$$

which Sim evaluates from the known $\alpha_t, \beta_t, \eta_\alpha$ and the public Y, without ever using s_t. The twin commitments are computed as $\bar{D}_t = D_t^{\eta_\alpha}$ and $\bar{E}_t = E_t^{\eta_\alpha}$, and the accompanying DLEQ proof π_t is produced by the honest-verifier zero-knowledge simulator of Lemma 2. Here Sim treats the Bi-DLEQ oracles abstractly, as the black box of Definition 5, by querying BinonceO and BiproveO and using their outputs directly. The embedding of the OMDL challenge into those oracles happens inside the reduction of Lemma 4 and it is invisible at the level of the present simulation, where the oracles are assumed to behave as specified.

(ii) PartialProofGen*.* The honest response of P_t is $z_t = d_t + \rho_t e_t + \lambda_t s_t c$, which Sim cannot compute directly because s_t is unknown. Instead, Sim recovers z_t in two pieces: the $\dot{x}$-dependent part is produced by a single BiproveO call, and the

known remainder is added by hand. Setting the Bi-DLEQ scalar to $\psi_t := \lambda_t \alpha_t$, the oracle returns

$$z_t^{(\mathrm{bi})} = d_t + \rho_t e_t + c\,\lambda_t \alpha_t\,\dot{x},$$

in which the unknown challenge $\dot{x}$ appears only in the last term, $c\lambda_t\alpha_t\dot{x}$. The remaining part of the honest response is the known base-g correction $c\lambda_t\beta_t$, which Sim adds itself:

$$\begin{aligned} z_t &\leftarrow z_t^{(\mathrm{bi})} + c\,\lambda_t\beta_t \\ &= d_t + \rho_t e_t + c\,\lambda_t(\alpha_t\dot{x} + \beta_t) \\ &= d_t + \rho_t e_t + c\,\lambda_t s_t\ , \end{aligned}$$

where the last equality uses the $s_t = \alpha_t\dot{x} + \beta_t$ representation of Lemma 6. The result is exactly the honest response, so the simulated z_t is distributed identically to the real one.

(iii) $\mathsf{Combine}$. This step is public and needs no further secret material. The aggregate evaluation satisfies

$$\gamma = \prod_{i\in S} \gamma_i^{\lambda_i} = h^{\sum_{i\in S}\lambda_i s_i} = h^s,$$

with the corrupt partial evaluations γ_j supplied by $\mathcal{F}$ from its known keys. Also, the group response is the public sum $z = \sum_{\ell\in S} z_\ell$, which the combiner now can form because the honest z_t that is not otherwise publicly computable was provided by step (ii) on top of the corrupt z_j coming from $\mathcal{F}$. Since the simulated z_t is identically distributed to the honest one and the aggregation is a deterministic public function of the per-party values, the combined proof (γ, c, z) and the output $\beta = H_3(\gamma)$ are distributed exactly as in a real execution.

Concurrency and the collision bound. Because Sim simulates only P_t, the family of simulated proofs is over a single party, and Sim neither rewinds $\mathcal{A}$ nor depends on the number of concurrently open sessions. Therefore, Sim runs $\mathcal{A}$ exactly once. The only way the transcript can veer from a real one is a programming collision (PC): a point Sim wishes to program in the H_2 table is already defined. This happens in two ways:

- through a proof Sim has previously simulated, contributing at most $\binom{n_{\mathsf{pr}}}{2}$ pairs, each colliding with probability at most $1/q$, since $R_t = g^{\tilde{z}} Y_t^{-\tilde{c}}$ is uniform;
- through a prior query of $\mathcal{A}$ to H_2, contributing at most $n_h n_{\mathsf{pr}}$ pairs, each colliding with probability at most $1/q$.

The per-proof challenge points $(g, h, Y_t, \gamma_t, \cdot, \cdot)$ that Sim programs are disjoint from the aggregate points $(g, h, Y, \gamma, \cdot, \cdot)$ because $Y_t \neq Y$, so no collision arises between the two kinds. Summing the two contributions,

$$\Pr[\mathrm{PC}] \le \left(\tbinom{n_{\mathsf{pr}}}{2} + n_h\, n_{\mathsf{pr}}\right)/q \le Q^2/q.$$

Conditioned on the complement of this event, the simulation by Sim is perfect, which gives the claimed bound.

□

Algorithm 7 combines all of the reduction steps for Theorem 4.

Algorithm 7 Composed reduction $\mathcal{R}'$ for Theorem 4

1: **Reduction to a single honest party.** Using Lemma 5, reduce to $|\mathrm{Cor}| = t-1$ by self-corruption; designate the honest party P_t of the forgery quorum (factor κ).
2: **Challenge embedding and key extraction.** Receive $Y = g^{\dot{x}}$; run Lemma 6 to embed Y and extract $\{x_j\}_{j=1}^{t-1}$; obtain (α_t, β_t). **If** the KoE extractor fails, abort (cost $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$).
3: **Straight-line simulation.** Simulate P_t by Lemma 7 where $\mathcal{F}$ runs $P_1, \ldots, P_{t-1}$. (Gap $\leq Q^2/q$, one run, no rewinding).
4: **Forgery.** $\mathcal{A}$ outputs $(\alpha^*, \gamma^*, c^*, z^*)$; abort if it does not verify or is trivial.
5: **Freshness dichotomy.** In the event where quorum size is insufficient ($\mathcal{E}_{\mathsf{quo}}$): general fork at the c^*-query. In the event of token substitution ($\mathcal{E}_{\mathsf{sub}}$): local-fork ρ_t with no slot guessing, solve the two-equation system.
6: **Discrete logarithm recovery.** Recover all challenge discrete logarithms arithmetically and submit to Fin, or output the KoE break if the extraction step is triggered.

We are finally in a position to prove the knowledge-soundness of the T-DLEQ proof system.

Theorem 4 (Knowledge-soundness of non-interactive T-DLEQ, single-honest-party presentation). *In the random oracle model, the non-interactive T-DLEQ proof system of Icy-DVRF is knowledge-sound, assuming the one-more discrete logarithm (OMDL) assumption for the proof system and the Schnorr knowledge-of-exponent (Schnorr-KoE) assumption for the distributed key generation, the latter implied by the discrete logarithm assumption in the algebraic group model. Under the fixed-quorum convention of the interactive proof, for which $\kappa = 1$ in Lemma 5, for every PPT adversary $\mathcal{A}$ corrupting up to $t-1$ parties, there exist a Schnorr-KoE adversary $\mathcal{B}_{\mathsf{koe}}$ and OMDL adversaries $\mathcal{B}, \mathcal{B}'$ such that*

$$\begin{aligned}\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}) \;\leq\; & \mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}}) \\ & + \; 2\max\Bigg\{\sqrt{Q\big(\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3(n+1)^2Q^2/q\big)}, \\ & \qquad Q\sqrt{2\big(\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}') + 1/q\big)}\Bigg\} \;+\; \frac{Q^2}{q},\end{aligned}$$

where $Q = n_h + n_{\mathsf{pr}} + 1$ and n is the number of provers. The first maximand bounds the insufficient-quorum event $\mathcal{E}_{\mathsf{quo}}$ and the second bounds the token-substitution event $\mathcal{E}_{\mathsf{sub}}$. Without the fixed-quorum convention, the $\mathcal{E}_{\mathsf{sub}}$ term is multiplied by $\kappa \leq \min\{t, n-t+1\} \leq n$, which recovers, and never exceeds, the explicit-set bound.

Proof. We run the composed reduction $\mathcal{R}'$ of Algorithm 7, and assemble the bound in five steps below.

Step 1. By Lemma 6, the adversary $\mathcal{B}_{\mathsf{koe}}$ embeds the OMDL challenge as the group key Y and extracts the $t-1$ corrupt shares through the Schnorr-KoE extractor. This extraction succeeds except with probability at most $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$. If successful, the honest party's key has the known affine representation $Y_t = Y^{\alpha_t} g^{\beta_t}$ with $s_t = \alpha_t \dot{x} + \beta_t$, where $\dot{x} = \log_g Y$ is carried symbolically. This is the only place the AGM is used, and it is confined to the distributed key generation phase.

Step 2. By Lemma 7, the simulator answers every message of the honest party P_t in a single run of $\mathcal{A}$, without rewinding and without access to the secret share, and the simulated transcript is identically distributed to a real one except on a programming collision event of probability at most Q^2/q. Unless that collision occurs, the simulation is perfect, and the probability of $\mathcal{A}$ producing a forgery equals its advantage in the real game. Lemma 5 further reduces a general adversary to one of the single-honest-party form at the cost of the factor κ, which is 1 under the fixed-quorum convention. We adopt this convention so that the displayed bound has $\kappa = 1$, and restore the general factor in Step 5.

Step 3. Suppose $\mathcal{A}$ outputs a verifying, non-trivial forgery $(\alpha^*, \gamma^*, c^*, z^*)$. Uniqueness of the VRF output forces $\gamma^* = (h^*)^s$, so the forgery is a fresh *proof object* for an output that is already fixed. The verifier recomputes the aggregate commitments $R^* = g^{z^*} Y^{-c^*}$ and $\bar{R}^* = (h^*)^{z^*} (\gamma^*)^{-c^*}$, which are exactly the values $T, \bar{T}$ formed in the Fin stage of the Bi-DLEQ game (Algorithm 6). For a verifying forgery the challenge satisfies $c^* = H_2(g, h^*, Y, \gamma^*, R^*, \bar{R}^*, \alpha^*)$, so $(R^*, \bar{R}^*)$ are exactly the commitments the forged challenge binds. Therefore, *freshness* is defined in terms of this aggregate $(R^*, \bar{R}^*)$, which a forgery can genuinely vary, rather than the fixed pair (α^*, γ^*). Since $t - |\mathrm{Cor}| = 1$, the winning condition

$$|\{k : (k, \cdot) \in \mathsf{ctr}(\alpha^*, R^*, \bar{R}^*)\}| < t - |\mathrm{Cor}|$$

requires fewer than one recorded honest slot, that is, none at all, meaning P_t made no genuine contribution to the forged aggregate. In other words, either P_t's published token does not appear in the aggregate at all, or it appears but was driven on an aggregate P_t never authorized. These two cases are mutually exclusive, and they cover all possibilities: if neither held, P_t's token would appear and would carry a genuine P_t response, making P_t a true contributor, so together with the $t-1$ corrupt parties, a full quorum would hold consistent contributions, which by correctness forces the honest proof and contradicts non-triviality. Hence, a winning forgery falls into exactly one of the two disjoint events

- *Insufficient quorum* ($\mathcal{E}_{\mathsf{quo}}$): P_t did not contribute to $(R^*, \bar{R}^*)$.
- *Token substitution* ($\mathcal{E}_{\mathsf{sub}}$): P_t's published token was reused in an aggregate that it never authorized.

Because the two events require different forking strategies, a general fork for $\mathcal{E}_{\mathsf{quo}}$ and a local one for $\mathcal{E}_{\mathsf{sub}}$, we bound each one separately and take the

maximum. Since the events are disjoint, $\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}) \le \Pr[\mathcal{E}_{\mathsf{quo}}] + \Pr[\mathcal{E}_{\mathsf{sub}}]$, and hence

$$\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}) \le 2\max\{\Pr[\mathcal{E}_{\mathsf{quo}}], \Pr[\mathcal{E}_{\mathsf{sub}}]\} .$$

Step 4. We perform extraction in each branch separately.
Branch $\mathcal{E}_{\mathsf{quo}}$. In this event, P_t did not contribute, so the forged response z^* is not a $\mathsf{BiproveO}$ output. Thus, the random-oracle query $H_2(g, h^*, Y, \gamma^*, R^*, \bar{R}^*, \alpha^*)$ whose answer is c^* is a genuine forking point, not one programmed by the simulator. Forking at that point and extracting $\log_g Y$ is exactly the reduction of Lemma 4, so

$$\Pr[\mathcal{E}_{\mathsf{quo}}] \le \sqrt{Q\big(\mathsf{Adv}^{\text{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3(n+1)^2Q^2/q\big)}.$$

Branch $\mathcal{E}_{\mathsf{sub}}$. In this branch, P_t's token is reused in an aggregate for which it was never authorized. By Lemma 3, given the base-g verification, the base-h verification equation is redundant, so the extraction is single-base. The reduction $\mathcal{B}'$ plays OMDL, embedding one challenge in the honest party's E-component ($E_t \leftarrow V_t = g^{v_t}$ fresh, $D_t = g^{a_t}$ with a_t known, twins supplied by Lemma 3) and the key $Y = g^{\dot{x}}$. The challenge is placed in E_t rather than D_t because only the E_t factor is weighted by the binding ρ_t: in the aggregate $R^* = \prod_i D_i E_i^{\rho_i}$, the half D_t enters unweighted while E_t enters as $E_t^{\rho_t}$, so resampling ρ_t perturbs the aggregate through E_t alone and leaves D_t untouched. Keeping $D_t = g^{a_t}$ known lets $\mathcal{B}'$ answer P_t's genuine signing query without a further Dlog call, while the unknown $v_t = \log_g E_t$ is the term that the fork exposes. $\mathcal{B}'$ is not required to guess the slot, since by Lemma 5 the only honest slot is t; this is where the single-honest-party convention is useful. Because the binding $\rho_t = H_2(t, \alpha^*, B)$ contributes to the aggregate $R^* = \prod_i D_i E_i^{\rho_i}$ *only* via the factor $E_t^{\rho_t}$, resampling the single random-oracle output ρ_t perturbs the aggregate in exactly that one factor,

$$R^{*\prime} = R^* \cdot E_t^{\rho_t' - \rho_t}, \qquad \bar{R}^{*\prime} = \varphi(R^{*\prime}),$$

where φ is the base-g-to-base-h map of Lemma 3, and the newly acquired challenge becomes a fresh $c^{*\prime}$. Subtracting the two verifying transcripts gives

$$z^* - z^{*\prime} = (\rho_t - \rho_t')\, v_t + (c^* - c^{*\prime})\, \dot{x}, \tag{1}$$

a single equation in the two unknowns $\dot{x} = \log_g Y$ and $v_t = \log_g E_t$. A single equation does not determine two unknowns, which is why *local* forking (the Local Forking Lemma of Bellare-Dai-Li [5]) rather than general forking is essential: local forking resamples only the ρ_t entry and leaves the genuine response that P_t gave in the session where it did contribute intact. The reduction answered that response with one Dlog query and so knows a second, independent linear relation

$$w_t = \rho_t^{(0)}\, v_t + \lambda_t c^{(0)}\, \dot{x}, \tag{2}$$

where $\rho_t^{(0)} = H_2(t, \alpha^{(0)}, B^{(0)})$ and $c^{(0)}$ are the binding and challenge of that honest session, a *distinct* H_2 entry untouched by the fork. Equations (1) and (2)

form a linear system in $(v_t, \dot{x})$ with determinant

$$\Delta = (\rho_t - \rho'_t)\,\lambda_t c^{(0)} - \rho_t^{(0)}\,(c^* - c^{*\prime}).$$

Since the four random-oracle outputs $\rho_t, \rho'_t, c^{(0)}, c^* - c^{*\prime}$ are uniform and independent, $\Delta = 0$ with probability at most $1/q$. Whenever $\Delta \neq 0$, the system determines $\dot{x} = \log_g Y$, from which $\mathcal{B}'$ recovers every consumed E-component challenge arithmetically and wins OMDL. Combining the local-forking success probability with the $1/q$ degeneracy gives

$$\Pr[\mathcal{E}_{\mathsf{sub}}] \leq Q\sqrt{2\big(\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}') + 1/q\big)}.$$

The single-base system above is exactly the one solved by the FROST1 local-forking reduction [4, Lem. 7], under the correspondence $d_i \mapsto \rho_i$, $H_{\mathrm{non}}(vk, lr, i) \mapsto H_2(i, \alpha, B)$, $n_s \mapsto 1$ (the slot forced to $m = t$), $q \mapsto Q$, and $p \mapsto q$. The three conditions of that lemma are satisfied in our case:

- $\rho_\ell = H_2(\ell, \alpha, B)$ is keyed by the index ℓ and enters the aggregate only through $E_\ell^{\rho_\ell}$, so resampling ρ_t localizes to a single factor.
- $H_2(t, \alpha^*, B)$ is a distinct, identifiable table entry, so the single-output resampling condition of [5] holds.
- The OMDL challenge is embedded in the E-component of P_t, so after Lemma 3 removes the twin equation, the residual system is precisely the one [4, Lem. 7] solves, with target $\log_g Y$.

Therefore, the cited constant transfers with the slot factor set to 1.

Step 5. Combining the setup loss $\mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}})$ in Step 1, the Step-2 simulation gap Q^2/q, and the per-branch bounds of Step 4 using the maxima of Step 3, we get

$$\begin{aligned}\mathsf{Adv}^{\text{t-dleq}}(\mathcal{A}) \;\leq\; & \mathsf{Adv}^{\text{sch-koe}}_{\mathbb{G},\mathsf{Ext}}(\mathcal{B}_{\mathsf{koe}}) \\ & +\; 2\max\Bigg\{\sqrt{Q\big(\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}) + 3(n+1)^2Q^2/q\big)}, \\ & \qquad\qquad Q\sqrt{2\big(\mathsf{Adv}^{\mathrm{omdl}}_{\mathbb{G}}(\mathcal{B}') + 1/q\big)}\Bigg\} \;+\; \frac{Q^2}{q}.\end{aligned}$$

Reinstating the general factor of Lemma 5 requires multiplying the $\mathcal{E}_{\mathsf{sub}}$ term by κ, which is 1 under the fixed-quorum convention and at most $\min\{t, n-t+1\} \leq n$ in general. Finally, since Schnorr-KoE is implied by the discrete logarithm assumption in the algebraic group model, the distributed key generation introduces no assumption beyond the discrete logarithm assumption and the algebraic group model, and the proof system itself rests on OMDL. □

Remark 1. Two random-oracle roles appear in the proof, and only the first separates the FROST variants introduced in [4,17]. The bindings $\rho_\ell = H_2(\ell, \alpha, B)$

are *per-prover* and tied to the ordered set B: this is the original FROST design, later named FROST1, which Icy-DVRF uses and never replaces by the single shared binding of FROST2. This choice is what fixes the constants in Theorem 4. In the $\mathcal{E}_{\mathsf{quo}}$ branch, a single proving query evaluates up to n per-prover bindings in addition to the one challenge, so the honest collision count carries the factor $(n+1)^2$, matching the FROST1 reduction of [4, Lem. 6]. Equivalently, one may absorb the factor by defining $Q := n_h + (n+1)n_{\mathsf{pr}} + 1$ so that internal binding evaluations are counted directly. We keep the explicit $(n+1)^2$ for transparency. In the $\mathcal{E}_{\mathsf{sub}}$ branch, the per-prover binding is exactly what makes token substitution extractable by the local fork of [5], and the resulting bound is the FROST1 single-base local-forking bound of [4, Lem. 7] with the slot factor specialized to 1 by the convention of Lemma 5. The OMDL-based intermediate assumption is independent of the choice of binding: it was introduced for FROST2 in [4, 17] and applies verbatim to our case.

Remark 2. The non-triviality condition of Theorem 4, that fewer than $t - |\mathrm{Cor}|$ honest provers contributed to the *exact* aggregate $(R^*, \bar{R}^*)$, is the VRF analog of the strongest unforgeability level, TS-SUF-3, of the threshold-signature hierarchy of [4]. Because the VRF output is unique, the strong-unforgeability target of a "new signature on an already-signed message" is vacuous here: there is only one valid output per input. What our reduction rules out instead is a new *proof object*, namely a fresh aggregate commitment pair, so freshness is defined in terms of $(R^*, \bar{R}^*)$ rather than (α^*, γ^*). This level 3 guarantee is precisely resistance to *token substitution* (the event $\mathcal{E}_{\mathsf{sub}}$): the per-prover binding prevents an honest prover's published commitment from being reused in a set it did not authorize. The single-shared-binding optimization of FROST2 would weaken this to level 2 [4, Fig. 8], which is the concrete reason Icy-DVRF utilizes the per-prover binding of Remark 1.

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). https://doi.org/10.1007/978-3-031-76934-4_8, doi: 10.1007/978-3-031-76934-4_8
2. 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, accessed: Jul. 7, 2026
3. Aptos Labs: The Aptos Blockchain: Safe, Scalable, and Upgradeable Web3 Infrastructure. Aptos Labs Whitepaper (2022), https://aptos.dev/assets/files/Aptos-Whitepaper-47099b4b978e19d89d971ed53e572070.pdf, accessed: Jul. 7, 2026
4. Bellare, M., Crites, E., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than Advertised Security for Non-interactive Threshold Signatures. In: Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceed-

ings, Part IV. pp. 517–550. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-15985-5_18, doi: 10.1007/978-3-031-15985-5_18
5. Bellare, M., Dai, W., Li, L.: The Local Forking Lemma and Its Application to Deterministic Encryption. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. pp. 607–636. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_21, doi: 10.1007/978-3-030-34618-8_21
6. Bellare, M., Neven, G.: Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma. In: Proceedings of the 13th ACM conference on Computer and communications security. pp. 390–399 (2006). https://doi.org/10.1145/1180405.1180453, doi: 10.1145/1180405.1180453
7. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)Security of ROS. J. Cryptol. **35**(4), 25 (2022). https://doi.org/10.1007/s00145-022-09436-0, doi: 10.1007/s00145-022-09436-0
8. Boneh, D., Lynn, B., Shacham, H.: Short Signatures from the Weil Pairing. J. Cryptol. **17**(4), 297–319 (2004). https://doi.org/10.1007/s00145-004-0314-9, doi: 10.1007/s00145-004-0314-9
9. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Mina: Decentralized Cryptocurrency at Scale. Technical Whitepaper (2020), https://eprint.iacr.org/2020/352, accessed: Jul. 7, 2026
10. Breidenbach, L., et al.: Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks. Whitepaper (2021), https://research.chain.link/whitepaper-v2.pdf, accessed: Jul. 7, 2026
11. Buser, M., et al.: Post-Quantum Verifiable Random Function from Symmetric Primitives in PoS Blockchain. In: European Symposium on Research in Computer Security. pp. 25–45. Springer (2022). https://doi.org/10.1007/978-3-031-17140-6_2, doi: 10.1007/978-3-031-17140-6_2
12. Buterin, V., Merriam, P., contributors: py_ecc: Python implementation of ECC pairing and BLS12-381 curve operations, https://github.com/ethereum/py_ecc, accessed: Jul. 7, 2026
13. Camenisch, J., et al.: The Internet Computer for Geeks. DFINITY Foundation Whitepaper (2022), https://eprint.iacr.org/2022/087, accessed: Jul. 7, 2026
14. Chainlink Labs: Chainlink VRF: Verifiable random function, https://chain.link/vrf, accessed: Jul. 7, 2026
15. 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, doi: 10.1007/3-540-48071-4_7
16. 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, doi: 10.1109/SP46215.2023.10179419
17. Crites, E., Komlo, C., Maller, M.: How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures. Cryptology ePrint Archive, Paper 2021/1375 (2021), https://eprint.iacr.org/2021/1375, accessed: Jul. 7, 2026
18. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 66–98. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_3, doi: 10.1007/978-3-319-78375-8_3
19. Dodis, Y.: Efficient Construction of (Distributed) Verifiable Random Functions. In: Public Key Cryptography – PKC 2003. Lecture Notes in Computer Science,

vol. 2567, pp. 1–17. Springer (2003). https://doi.org/10.1007/3-540-36288-6_1, doi: 10.1007/3-540-36288-6_1

20. Dodis, Y., Yampolskiy, A.: A Verifiable Random Function with Short Proofs and Keys. In: Public Key Cryptography – PKC 2005. pp. 416–431. Springer (2005). https://doi.org/10.1007/978-3-540-30580-4_28, doi: 10.1007/978-3-540-30580-4_28
21. Drijvers, M., et al.: On the Security of Two-Round Multi-Signatures. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 1084–1101 (2019). https://doi.org/10.1109/SP.2019.00050, doi: 10.1109/SP.2019.00050
22. Edgington, B.: BLS12-381 For The Rest Of Us (2022), https://hackmd.io/@benjaminion/bls12-381, accessed: Jul. 7, 2026
23. Esgin, M.F., et al.: Practical Post-quantum Few-Time Verifiable Random Function with Applications to Algorand. In: Borisov, N., Diaz, C. (eds.) Financial Cryptography and Data Security. pp. 560–578. Springer Berlin Heidelberg, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64331-0_29, doi: 10.1007/978-3-662-64331-0_29
24. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). pp. 427–438 (Oct 1987). https://doi.org/10.1109/SFCS.1987.4, doi: 10.1109/SFCS.1987.4
25. Fuchsbauer, G., Kiltz, E., Loss, J.: The Algebraic Group Model and its Applications. In: Advances in Cryptology – CRYPTO 2018, Part II. Lecture Notes in Computer Science, vol. 10992, pp. 33–62. Springer (2018). https://doi.org/10.1007/978-3-319-96881-0_2, doi: 10.1007/978-3-319-96881-0_2
26. 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, doi: 10.1109/EuroSP51992.2021.00017
27. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. J. Cryptol. **20**, 51–83 (May 2007). https://doi.org/10.1007/s00145-006-0347-3, doi: 10.1007/s00145-006-0347-3
28. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles (SOSP). pp. 51–68. ACM (2017). https://doi.org/10.1145/3132747.3132757, doi: 10.1145/3132747.3132757
29. Goldberg, S., Reyzin, L., Papadopoulos, D., Včelák, J.: Verifiable Random Functions (VRFs). RFC 9381 (Aug 2023). https://doi.org/10.17487/RFC9381, doi: 10.17487/RFC9381
30. 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, doi: 10.1145/3576915.3616601
31. Kılınç Alper, H.: BABE: Blind assignment for blockchain extension. Web3 Foundation Research (2019), https://research.parity.io/Polkadot/protocols/block-production/Babe, accessed: Jul. 7, 2026
32. 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. pp. 34–65. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-81652-0_2, doi: 10.1007/978-3-030-81652-0_2

33. 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, doi: 10.1007/978-3-319-02937-5_14
34. League of Entropy: Cryptography: An Overview of drand's Cryptographic Building Blocks. drand Documentation, https://docs.drand.love/docs/cryptography/, accessed: Jul. 7, 2026
35. Liu, J., Manulis, M.: Fast SNARK-based Non-Interactive Distributed Verifiable Random Function with Ethereum Compatibility. In: Proceedings of the 20th ACM Asia Conference on Computer and Communications Security (ASIA CCS '25). pp. 807–822. ACM, Hanoi, Vietnam (2025). https://doi.org/10.1145/3708821.3710835, doi: 10.1145/3708821.3710835
36. 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, doi: 10.1109/SFFCS.1999.814584
37. Mysten Labs: The Sui smart contracts platform. Mysten Labs Whitepaper (2022), https://docs.sui.io/paper/sui.pdf, accessed: Jul. 7, 2026
38. Papadopoulos, D., et al.: Making NSEC5 Practical for DNSSEC. Cryptology ePrint Archive, Paper 2017/099 (2017), https://eprint.iacr.org/2017/099, accessed: Jul. 7, 2026
39. Patel, S., Persiano, G., Seo, J.Y., Yeo, K.: Efficient Secret Sharing for Large-Scale Applications. In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24). pp. 3065–3079. ACM, Salt Lake City, UT, USA (2024). https://doi.org/10.1145/3658644.3670379, doi: 10.1145/3658644.3670379
40. Pedersen, T.P.: A Threshold Cryptosystem without a Trusted Party. In: Davies, D.W. (ed.) Advances in Cryptology — EUROCRYPT '91. pp. 522–526. Springer Berlin Heidelberg, Berlin, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_47, doi: 10.1007/3-540-46416-6_47
41. Pedersen, T.P.: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91. pp. 129–140. Springer Berlin Heidelberg, Berlin, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9, doi: 10.1007/3-540-46766-1_9
42. Pointcheval, D., Stern, J.: Security Arguments for Digital Signatures and Blind Signatures. J. Cryptol. **13**(3), 361–396 (2000). https://doi.org/10.1007/s001450010003, doi: 10.1007/s001450010003
43. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176, doi: 10.1145/359168.359176
44. Supra Research: Supra VRF Service. Technical Whitepaper (2023), https://supra.com/documents/VRF-Service-Whitepaper.pdf, accessed: Jul. 7, 2026
45. 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, accessed: Jul. 7, 2026

Table 1. Mathematical operations of DVRFwCP and Icy-DVRF schemes

	DVRFwCP	**Icy-DVRF**
DistKeyGen	$2nt + 3n + 2t$ exp. in $\mathbb{G}$ $2nt + t - 1$ mul. in $\mathbb{G}$	$nt + 3n - 1$ exp. in $\mathbb{G}$ $nt + n - t - 1$ mul. in $\mathbb{G}$ n hash to $\mathbb{Z}_q$
Preprocess	–	2Ω exp. in $\mathbb{G}$
PartialEval	3 exp. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$	5 exp. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$
EvalCombine[(a)]	$5t$ exp. in $\mathbb{G}$ $3t - 1$ mul. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$	–
PartialProofGen	$3nt + 4n + 3t$ exp. in $\mathbb{G}$ $3nt + t - 2$ mul. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$	$3t$ exp. in $\mathbb{G}$ $3t - 3$ mul. in $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$
Combine[(b)]	$6t$ exp. in $\mathbb{G}$ $2t$ mul. in $\mathbb{G}$ 1 hash to $\{0,1\}^\Lambda$	$9t$ exp. in $\mathbb{G}$ $8t - 3$ mul. in $\mathbb{G}$ 1 hash to $\{0,1\}^\Lambda$ $t + 2$ hash to $\mathbb{Z}_q$
Verify	4 exp. in $\mathbb{G}$ 2 mul. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 2 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^\Lambda$	4 exp. in $\mathbb{G}$ 2 mul. in $\mathbb{G}$ 1 hash to $\mathbb{G}$ 2 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^\Lambda$
Interactions[(c)]	2 round	1 round
Proof size	1 $\mathbb{G}$ element 2 $\mathbb{Z}_q$ elements	1 $\mathbb{G}$ element 2 $\mathbb{Z}_q$ elements

(a) These algorithms are not present in the corresponding DVRFs. (b) The corresponding algorithm is named ProofCombine in DVRFwCP. (c) The one-time DistKeyGen is not included.

Table 2. Communication Cost of DVRFwCP and Icy-DVRF (Per Participant)

Phase	**DVRFwCP**	**Icy-DVRF**
PartialEval	$1\mathbb{G} + 2\mathbb{Z}_q$	$3\mathbb{G} + 2\mathbb{Z}_q$
PartialProofGen	$2t\mathbb{G} + 2\mathbb{Z}_q$	$0\mathbb{G} + 2\mathbb{Z}_q$
Total Broadcast	$(2t + 1)\mathbb{G} + 4\mathbb{Z}_q$	$3\mathbb{G} + 4\mathbb{Z}_q$

Table 3. Theoretical gas cost estimation comparison of Verify on BLS12-381.

Gas cost	Verification cost	Calldata cost	Total cost
DDH-DVRF	$\approx 70000\ t$	$768\ t$	$\approx$ **70768 t**
GLOW-DVRF	≈ 200000	1280	$\approx$ **201280**
DVRFwCP	≈ 50000	1792	$\approx$ **51792**
Icy-DVRF	≈ 50000	1792	$\approx$ **51792**

Table 6. Off-chain per-request cost: DVRFwCP vs. Icy-DVRF, excluding one-time DistKeyGen and (for Icy-DVRF) offline Preprocess.

	DVRFwCP			**Icy-DVRF**				
(n, t)	**Comp.** (ms)	**Total** (ms)	**Per-node** (ms)	**Comp.** (ms)	**Total** (ms)	**Per-node** (ms)	**Speedup**	**Per-node Speedup**
$(5, 3)$	420.2	780.2	444.0	315.5	555.5	303.1	1.40×	1.46×
$(7, 4)$	620.3	980.3	448.6	447.8	687.8	304.0	1.43×	1.48×
$(10, 7)$	1378.2	1738.2	497.8	936.0	1176.0	333.6	1.48×	1.49×
$(25, 17)$	9019.3	9379.3	720.8	3644.4	3884.4	385.8	2.41×	1.87×
$(50, 34)$	90433.5	90793.5	2168.7	12471.1	12711.1	489.4	7.14×	4.43×

(a) The figures are from a single-process simulation that runs all n parties sequentially on one machine. In a real world deployment, the per-party computation runs in parallel while the network rounds are incurred once, so the realistic per-node latency is approximately Comp./n + (rounds × 120 ms) (b) Comp. is cryptographic computation time per request; Total adds 120 ms per interaction round (3 rounds for DVRFwCP, 2 for Icy-DVRF). (c) Per-node is Comp./n + (rounds × 120 ms), estimating realistic per-node latency when parties compute in parallel. (d) Total Speedup is the ratio of aggregate Total latencies; Per-node Speedup is the ratio of the two Per-node columns.

Table 7. Per-node post-setup broadcast (compressed BLS12-381, $1\mathbb{G}_1 = 48$ B, $1\mathbb{Z}_q = 32$ B), instantiating the formulas of Table 2 at concrete committee sizes.

(n, t)	**DVRFwCP** (bytes)	**Icy-DVRF** (bytes)	**Improvement**
$(5, 3)$	464	272	1.71×
$(7, 4)$	560	272	2.06×
$(10, 7)$	848	272	3.12×
$(25, 17)$	1808	272	6.65×
$(50, 34)$	3440	272	12.65×

Per-node totals: DVRFwCP $(2t{+}1)\mathbb{G}_1 + 4\mathbb{Z}_q$, Icy-DVRF $3\mathbb{G}_1 + 4\mathbb{Z}_q$.

Table 8. One-time setup cost. DistKeyGen runtime is shown for both protocols. Preprocess is offline and unique to Icy-DVRF.

	DistKeyGen (ms)		**Icy-DVRF Preprocess**		
(n, t)	**DVRFwCP**	**Icy**	**Comp.** (ms)	**Storage** (KB)	**Amort.** (ms/req)
$(5, 3)$	180.3	243.2	4234.3	46.9	42.3
$(7, 4)$	335.8	439.0	5886.5	65.6	58.9
$(10, 7)$	780.4	984.8	8328.8	93.8	83.3
$(25, 17)$	8330.6	12712.1	20648.0	234.4	206.5
$(50, 34)$	104506.4	197054.0	42421.4	468.8	424.2

(a) $\Omega = 100$ pairs/party. (b) Amort. is Preprocess compute divided over a batch of $\Omega = 100$ requests. (c) Storage $= n \times \Omega \times 96\,\mathrm{B}$ (compressed BLS12-381).

Table 9. Measured per-phase computation time (ms) at $(n, t) = (10, 7)$, complementing the operation counts of Table 1.

Phase	**DVRFwCP**	**Icy-DVRF**
DistKeyGen	780.4	984.8
Preprocess[(a)]	–	8328.8
PartialEval	147.3	162.4
EvalCombine[(b)]	186.3	–
PartialProofGen	874.7	441.6
Combine[(b)]	150.9	312.9
Verify	18.9	19.1

(a) Absent in DVRFwCP. (b) Absent in Icy-DVRF. (c) Named ProofCombine in DVRFwCP.