(R)Icy-DVRF: A Robust Distributed Verifiable Random Function Based on ROAST Signatures*

Ahmet Ramazan Ağırtaş[1,2][0000-0002-4574-0067],
Oğuz Yayla[2][0000-0001-8945-2780], and
Melis Berçin Yılmaz[2,3**][0000-0002-2588-3685]

[1] AYWARE, Ankara, Türkiye
[2] Institute of Applied Mathematics, Middle East Technical University, Ankara, Türkiye
[3] Department of Mathematics, Middle East Technical University, Ankara, Türkiye
a.r.agirtas@gmail.com
{oguz,mbercink}@metu.edu.tr

Abstract. Ensuring robustness and liveness in distributed verifiable random functions (DVRFs) allows the protocol to maintain correct operation and guarantee output generation, even in the presence of malicious actors attempting to disrupt the process, delay cryptographic shares, or remain unresponsive. Because existing DVRF protocols typically rely on synchronous or semi-synchronous network assumptions, simultaneously achieving these properties remains a challenge. To address this limitation, this paper enhances the FROST-based Icy-DVRF protocol to achieve both robustness and liveness. Specifically, we propose (R)Icy-DVRF, a novel protocol that operates over an asynchronous network while maintaining a constant-size proof. This is achieved by integrating the ROAST coordination layer into the underlying FROST-based threshold evaluation and proof generation mechanism.

Keywords: VRF · DVRF · Robustness.

1 Introduction

Traditional pseudo-random functions (PRFs) cannot verify the accuracy of their random outputs to third parties without sharing the secret key. This leads to a transferable trust vulnerability in decentralized systems. To address this vulnerability, Verifiable Random Functions (VRFs) were designed to allow public

* This is an expanded version of a paper to appear in SecITC 2026, Lecture Notes in Computer Science, Springer. Compared to the conference version, it additionally contains the full preliminaries (assumptions, secret sharing, the complete PedPoP and ROAST protocols, and a summary of Icy-DVRF), full security proofs, per-subroutine operation counts, and an extended comparison.

** Corresponding author.

verification of the accuracy of random outputs without disclosing the secret key. These functions process the input using a secret key, generating both a pseudo-random output and providing mathematical proof that the output was calculated correctly.

Introduced by Micali, Rabin and Vadhan [25] as a publicly verifiable analogue of pseudorandom functions, the original VRF construction is tree-based and therefore inefficient. Dodis and Yampolskiy [12] later gave a direct construction with short proofs and keys, under a q-type bilinear assumption. With the increasing use of technology in various fields, Papadopoulos et al. [26] introduced a suitable VRF standard for infrastructure under the NSEC5 protocol [18], which they believe could be a practical solution for regions concerned with enhancing DNS security (DNSSEC) and protecting sensitive information from offline zone enumeration attacks. Goldberg et al. [19], on the other hand, defined two separate VRF structures based on RSA and elliptic curves. They used the structure of asymmetric keys for dynamically verifiable data structures in cloud computing and database security.

Due to the rapid development of decentralized ecosystems, VRFs serve as a fundamental building block in modern blockchain protocols. In particular, unmanipulable and unpredictable sources of randomness are crucial for ensuring fair leader selection and achieving strong consensus security. In this context, the use of VRF architecture can be seen on platforms such as Algorand [17], Cardano [10], Polkadot [31], and Mina [5]. In addition to the consensus layer, decentralized oracle networks such as Chainlink [6] provide an infrastructure that delivers the computational resources needed by smart contracts based on the VRF mechanism. Furthermore, protocols like Aptos [8] and Sui [4] integrate VRFs directly into their core developer tooling and Move-based smart contract environments, enabling application developers to obtain verifiable randomness outputs directly via on-platform APIs without relying on external oracle services.

Traditional VRF architectures offer widespread use and undeniable benefits in maintaining cryptographic neutrality. However, they suffer from a critical systemic vulnerability: the single point of failure problem. The secret key plays a vital role in generating the pseudo-random output and its corresponding mathematical proof. If the node hosting this private key is compromised by an adversary, the entire network's forgery resistance and randomness guarantees instantly collapse. If that particular node is affected by network latency, hardware failure, or a targeted Denial-of-Service (DoS) attack, the entire consensus mechanism or smart contract implementation layer using the VRF service comes to a standstill. This creates a cryptographic bottleneck. Therefore, new VRF architectures have emerged where the secret key is distributed among multiple participants via threshold cryptography, ensuring that no single party can manipulate or stop the system.

1.1 Related Works

Distributed Verifiable Random Functions were developed as a solution to the single point of failure problem of VRFs. In its simplest form, these protocols

allow the input m to be transformed into a single VRF output, β, and the resulting proof Π to be computed by a group of participants. Using DKG protocols [28, 16], the secret key is shared among the participants so that the predetermined minimum number (threshold) of participants can honestly adhere to the protocol. The VRF output β can be verified by any third-party verifier using the group's public key.

Below, we will examine the DVRF literature in terms of the robustness, liveness, synchronicity characteristics and verification efficiency of each construct, which is the motivation of this paper.

To the best of our knowledge, the first DVRF construction was proposed by Dodis [11]. While this proposal stated that it could be verified locally without requiring interaction between servers and without needing ZKP, it was observed that it did require a trusted dealer. In this context, Kuchta and Manulis [23] presented a new model, shared random string (SRS), which eliminates the need for a trusted dealer. They used aggregate signature mechanisms with uniqueness. In this way, they established a distributed structure, collecting signatures from servers and combining them locally. However, this leaves the scheme vulnerability to rogue-key attacks. On the other hand, Hanke et al. [20] used a committee-based practical DKG and BLS threshold signature scheme to place a decentralized random beacon at the center of a globally operating blockchain consensus engine with near-instant finality.

In [15], Galindo et al. have introduced the DDH-DVRF and GLOW-DVRF schemes, based on the Decisional Diffie-Hellman assumption and bilinear pairings respectively, as fully distributed and non-interactive DVRF structures to the literature. The proof size of DDH-DVRF scales linearly with the threshold. GLOW-DVRF, on the other hand, is designed using bilinear pairings to keep the proof size constant. However, this time the verification cost has become quite expensive because of the pairing computation.

To have a constant size proof and cheaper verification, Ağırtaş et al. designed the DVRFwCP [1] protocol. In this design, a compact proof size is obtained without the need for bilinear pairing. However, to achieve this advantage, communication round complexity has been sacrificed, and two additional communication rounds are required. DVRFwCP is robust, but only under a fully synchronous network, in which identifiable aborts are resolved by re-running the affected round within a known time bound. To that end, the same authors proposed the Icy-DVRF [2] protocol which employs a FROST-based design [22], reducing the number of communication rounds and significantly increasing system speed, while keeping the proof size compact through aggregateable structures. However, this scheme completely compromises robustness because of the underlying FROST structure.

Subsequently, Kate et al. [21] developed a protocol that, unlike known DVRF protocols, ensures not only verifiability but also output privacy. While robustness and liveness are achieved, security relies on Co-CDH and threshold one-more Diffie Hellman assumptions. The liveness guarantee is built with a pairing-based construct derived from the static honest majority assumption. This output-

private DVRF concept proposed by Kate et al. was taken up by Ağırtaş et al. in IcyVeil [3]. In this paper, the privacy mechanism is integrated into a pairing-free Icy-DVRF [2] architecture, resulting in lower gas costs. However, the robustness feature remains unresolved, as it does in Icy-DVRF. On the other hand, Patel et al. [27] introduced an improved DVRF which can be used in the structure of decentralized randomness beacons. However, while leaving the robustness property as an open problem, the function they developed uses a synchronous structure. At the same time, Liu and Manulis [24] proposed a fast and non-interactive DVRF protocol using SNARK-based technology, which works directly and natively with Ethereum (EVM) smart contracts. However, their notion of robustness is weaker: the Combine transaction will yield a correct output if it is not cancelled. If participants withhold or delay their shares, there is no guarantee that Combine will succeed within the limited number of rounds. Furthermore, the introduced protocol is based on a SNARK proof system with reliable KZG commitments and depends on pairings.

As is evident, none of the above proposals simultaneously provides pairing-free efficiency, robustness, and complete asynchronous operation.

1.2 Our Contribution

In cryptographic protocols, robustness ensures that the protocol can execute and successfully complete its operation despite attacks or malicious actors. In distributed systems in particular, protocol robustness is of vital importance. Many existing cryptographic protocols abort when one of the participants goes offline or cheats. Robustness means detecting and removing malicious participants from the system or tolerating their absence. This prevents the process from being interrupted and ensures the system's liveness.

Considering decentralized ecosystems, it becomes clear that the same structural challenge arises when aiming to generate an unpredictable and unbiased random value. If participants go offline, act maliciously, or intentionally delay their cryptographic share, the protocol will weaken or fail.

To overcome the practical network bottlenecks and stalling problems given in Section 1.1, we propose a new protocol that prioritizes robustness and liveness characteristics, i.e., (R)Icy-DVRF (Robust Icy Distributed Verifiable Random Function). To ensure that this protocol provides full robustness and liveness in asynchronous environments, we redesign the existing Icy-DVRF [2] architecture by wrapping the FROST-based threshold evaluation and proof generation mechanism [22] in the ROAST [29] coordination layer, after adapting it to ROAST's preconditions. On the security side, we identify the property that the coordination layer requires of the evaluation layer. We formalise it as the concurrent security game of Definition 11 and show (Theorem 3) that the ROAST layer grants the adversary no power beyond this game, even when the adversary is itself the combiner. The unforgeability and pseudorandomness of (R)Icy-DVRF are therefore conditional on Definition 11, which is stronger than the single-session analysis of [2], and Remark 2 states in which respects. Robustness, in contrast, does not rely on Definition 11 (Theorem 4).

(R)Icy-DVRF preserves the dynamic and compact mathematical structure of the Icy-DVRF protocol while transitioning the system to a fully asynchronous network architecture. This protocol has a set of n participants and a combiner, who can be one of the participants. The protocol begins with a preprocessing step. In this step, each participant submits their share of the commitment, their h - basis twin required for the evaluation phase, and their partial evaluation along with the accompanying NIZK proof. The combiner initiates a session when it has collected a sufficient t number of such shares from active participants. If a participant submits an incorrect partial evaluation or invalid partial proof, the combiner identifies this participant via the PreAggregation and ShareVal subroutines and permanently excludes that participant from all subsequent sessions. However, a non-responsive participant is never detected. Instead, it is tolerated. That participant is not returned to the responsive set. The combiner does not wait and continues with the remaining participants. This prevents the system from locking up while allowing new sessions to continue with the remaining honest participants.

The protocol's communication rounds are expressed as asynchronous rounds. Because the commitment share of (R)Icy-DVRF is bound to the input, it cannot be produced before the input is known: a fresh plaintext m therefore always costs one preprocessing round, in which each participant sends its commitment share, its h-basis twin and its partial evaluation to the combiner. Each session then costs one round trip. Consequently, when all participants are honest and active, a fresh input is processed in 3 asynchronous rounds, and every further session on the same input costs only 2 additional rounds, since the replacement share ρ'_i is piggybacked on the reply of the preceding session and no new preprocessing round is needed. Let $f \leq n-t$ denote the maximum number of malicious nodes tolerated. Each session that fails to complete contains a distinct blocking participant, so at most $n-t+1$ sessions are initiated and the worst-case cost is $2(n-t)+3$ asynchronous rounds, which is at most $n+2$ under the threshold constraint, as established in Lemma 3. This provides the liveness property.

The verification of the final output generated by the (R)Icy-DVRF protocol is identical to a single-user DLEQ proof [7] for any third party. The verifier does not need to know the total number of participants in the system, which nodes are actively involved in the current session, or check the individual public keys of the participants one by one. Only the plaintext m, the group's public key Y, the final DVRF value β, and the combined proof (γ, c, z) are given as input to the Verify subroutine. The system checks the validity of this combined single proof in a single step. The final verification cost does not depend on the number of participants in the network (n) or the threshold value (t). It has a constant complexity.

While DVRFwCP [1] uses a fully synchronous network and Icy-DVRF uses a semi-synchronous model and is dependent on timeouts, (R)Icy-DVRF adopts a fully asynchronous network, in which no participant's delay or unresponsiveness can stall the protocol. Moreover, whereas the FROST infrastructure in Icy-DVRF causes the system to lock up and requires a complete restart of the protocol,

the ROAST wrapper mechanism isolates malicious actors in the background, allowing the final DVRF output generation with honest participants.

Due to the ROAST wrapper, the combiner in (R)Icy-DVRF acts as an active session manager, identifying malicious nodes, maintaining a dynamic blame list, and managing concurrent sub-sessions until a threshold of t valid shares is reached. Thus, increased computational and memory overhead for the combiner is an inherent trade-off. Another trade-off concern is one-time setup. Although our preference is to be fully asynchronous, PedPoP KeyGen [9] assumes a trusted broadcast channel or a semi-synchronous network for participants to agree on a joint public key.

The remainder of this paper is organized as follows. We give the preliminary information that covers the fundamental concepts forming the mathematical and cryptographic infrastructure of the proposed protocol in Section 2. Section 3 formally defines the (R)Icy-DVRF (Robust Icy Distributed Verifiable Random Function) protocol, which is the main contribution of this paper. This section details the operation of the subroutines that constitute the asynchronous and fault-tolerant structure of the scheme and the step-by-step execution flow of the protocol. Then, Section 4 theoretically compares the proposed (R)Icy-DVRF protocol with previous distributed verifiable random function schemes in the literature (especially DVRFwCP and Icy-DVRF) in terms of asynchronous round complexity, verification costs, proof size, and robustness guarantees. Finally, Section 5 summarizes the theoretical gains obtained, evaluates the contributions of the paper to the literature, and discusses possible future research directions.

2 Preliminaries

To establish a clear and rigorous framework for the security and viability analysis of our (R)Icy-DVRF protocol, this section presents the necessary cryptographic background and security abstractions. Our structure operates with threshold cryptography, non-interactive zero-knowledge proofs, and fault-tolerant asynchronous coordination. Therefore, we recall the standard discrete logarithm, decisional Diffie-Hellman, one-more discrete logarithm, and the Schnorr knowledge of exponent assumptions. By examining the fundamental mechanics of verifiable secret-sharing schemes, detailing the underlying key generation and the basic building blocks of robust signing protocols, we establish the structural framework for the formal proofs presented in the remainder of this paper.

$r \xleftarrow{\$} S$ is used to denote that the element r is sampled uniformly at random from the finite set S. We let $\mathbb{Z}_q$ represent the residue ring of prime order q, which serves as the scalar field for our cryptographic group operations. For any finite set of indices $\mathcal{I}$, we denote its cardinality as $|\mathcal{I}|$.

Definition 1 (Lagrange Interpolation [2]).
For any polynomial $f(X) \in \mathbb{Z}_q[X]$ of degree at most $t-1$ and let $\mathcal{I} = \{p_1, \ldots, p_t\}$, where $1 \leq p_i \leq n$ is a reconstruction index set, the polynomial $f(X)$ can be

uniquely reconstructed via Lagrange interpolation as follows:

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

where the Lagrange basis polynomials $\lambda_{p_i,\mathcal{I}}(X) \in \mathbb{Z}_q[X]$ *are defined as:*

$$\lambda_{p_i,\mathcal{I}}(X) = \prod_{p_k \in \mathcal{I} \setminus \{p_i\}} \frac{X - p_k}{p_i - p_k} \pmod{q}.$$

In threshold cryptographic primitives, one is primarily interested in evaluating or reconstructing the constant term $f(0)$*, which represents the joint secret key or shared secret. The constant term can be reconstructed using the static Lagrange coefficients* $\lambda_{p_i,\mathcal{I}} = \lambda_{p_i,\mathcal{I}}(0) \in \mathbb{Z}_q^*$ *as:*

$$f(0) = \sum_{p_i \in \mathcal{I}} \lambda_{p_i,\mathcal{I}} f(p_i),$$

where the coefficients are evaluated at zero:

$$\lambda_{p_i,\mathcal{I}} = \prod_{p_k \in \mathcal{I} \setminus \{p_i\}} \frac{-p_k}{p_i - p_k} = \prod_{p_k \in \mathcal{I} \setminus \{p_i\}} \frac{p_k}{p_k - p_i} \pmod{q}.$$

It is worth noting that these Lagrange coefficients depend merely on the organization of the reconstruction index set $\mathcal{I} \subset \{1, \ldots, n\}$ and are entirely independent of the actual shares or evaluations $f(p_1), \ldots, f(p_t)$. For the sake of simplicity and readability, we compress the notation and write $\lambda_j = \lambda_{p_i,\mathcal{I}}$ to represent these coefficients whenever the underlying reconstruction session context and the set $\mathcal{I}$ are clear.

Definition 2 (NIZK Proof for Equality of Discrete Logarithms (DLEQ) [2]).
Let $\mathbb{G} = \langle g \rangle$ *be a cyclic multiplicative group of prime order* q*. Following the non-interactive zero-knowledge (NIZK) framework originally introduced by Chaum and Pedersen [7] and adapted in the context of distributed random functions by [2], a Discrete Logarithm Equality (DLEQ) proof system allows a prover to demonstrate that two group elements share the same discrete logarithm over different bases without revealing the underlying witness.*

Formally, given the bases $g, h \in \mathbb{G}$ *and the public elements* $x, y \in \mathbb{G}$*, the system proves the relation* $k = \log_g x = \log_h y \in \mathbb{Z}_q$*. This primitive is formalized by a pair of probabilistic polynomial-time algorithms* $(\mathsf{ProveEq}_{H_2}, \mathsf{VerifyEq}_{H_2})$ *that utilize a cryptographic hash function* $\mathbf{H_2} : \{0,1\}^* \to \mathbb{Z}_q$ *modeled as a random oracle:*

ProveEq$_{H_2}(g, h, x, y; k)$*: The prover samples a random commitment nonce* $r \xleftarrow{\$} \mathbb{Z}_q$*, computes the group commitments* $u = g^r$ *and* $v = h^r$*, and derives the challenge value via the random oracle as* $c \leftarrow \mathbf{H_2}(g, h, x, y, u, v)$*. The algorithm outputs the non-interactive proof* $\pi = (c, z)$*, where the response scalar* z *is computed as:*

$$z = r - k \cdot c \pmod{q}.$$

VerifyEq$_{H_2}(g, h, x, y, \pi)$*: The verifier parses the proof as* $\pi = (c, z)$*, reconstructs the commitment approximations* $\bar{u} \leftarrow g^z x^c$ *and* $\bar{v} \leftarrow h^z y^c$*, and evaluates the hash condition. The algorithm outputs* 1 *(accept) if:*

$$c = \mathbf{H_2}(g, h, x, y, \bar{u}, \bar{v});$$

otherwise, it outputs 0 *(reject).*

Note that ProveEq uses the subtractive convention $z = r - kc$, whereas the partial proofs of Section 3.3 use the additive convention $z_i = k_i + \lambda_i s_i c$ inherited from FROST. The two are equivalent up to the sign of the verification exponent.

The proof system is honest-verifier zero-knowledge in the random oracle model: given only (g, h, x, y), a proof with exactly the distribution of a real one is obtained by sampling $(c, z) \xleftarrow{\$} \mathbb{Z}_q^2$, setting $\bar{u} \leftarrow g^z x^c$ and $\bar{v} \leftarrow h^z y^c$, and programming $H_2(g, h, x, y, \bar{u}, \bar{v}) \leftarrow c$, which fails only if that point has already been queried. We use this simulatability in Remark 2 to argue that the single copy of π_i that (R)Icy-DVRF publishes per input carries no information about the witness beyond the statement itself.

2.1 Underlying Hard Problems

Following the definitions established in [2] and [9], this section outlines the computational frameworks and standard hardness assumptions over prime-order cyclic groups upon which the cryptographic security and unforgeability of our proposed scheme are fundamentally based.

Definition 3 (Discrete Logarithm (DL) Assumption [2]).
Let $\mathbb{G} = \langle g \rangle$ *be a cyclic multiplicative group of a large prime order* q*. The Discrete Logarithm problem states that given the group parameters and a random element* $h \in \mathbb{G}$*, it is computationally infeasible to determine the exponent* $x \in \mathbb{Z}_q$ *such that* $g^x = h$*. Formally, the Discrete Logarithm (DL) assumption holds in* $\mathbb{G}$ *if for every probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$*, the advantage defined as:*

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

is negligible in the security parameter λ*.*

Definition 4 (Decisional Diffie-Hellman (DDH) Assumption [2]).
Let $\mathbb{G} = \langle g \rangle$ *be a cyclic multiplicative group of a large prime order* q*. The Decisional Diffie-Hellman problem addresses the computational hardness of distinguishing between a true Diffie-Hellman triplet and a completely random triplet in the same group. Formally, the DDH assumption holds in* $\mathbb{G}$ *if for all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$*, the advantage defined as:*

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

is negligible in the security parameter λ*, where the exponents are sampled uniformly at random such that* $\alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_q^*$*.*

Definition 5 (One-More Discrete Logarithm (OMDL) Assumption [9]). *Let GrGen be a group generator that outputs $\mathcal{G} = (\mathbb{G}, q, g)$, and let $\mathcal{O}^{dl}$ be a discrete logarithm oracle that can be called at most n times. The one-more discrete logarithm assumption holds with respect to $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$, there exists a negligible function ν such that*

$$\Pr\left[\begin{array}{l}\mathcal{G} \leftarrow GrGen(1^\lambda);\\ (X_0,\ldots,X_n) \xleftarrow{\$} \mathbb{G}^{n+1};\\ (x_0,\ldots,x_n) \xleftarrow{\$} \mathcal{O}^{dl}(\mathcal{G}, X_0,\ldots,X_n) :\\ X_i = g^{x_i}\ \forall\ 0 \le i \le n\end{array}\right] < \nu(\lambda).$$

$\underline{\textbf{MAIN}\ \mathrm{Game}^{\text{schnorr-koe}}_{\mathcal{A},\mathrm{Ext}}(\lambda)}$	$\underline{\mathcal{O}^{\mathrm{chal}}(X^*, \bar{R}^*, \bar{z}^*)}$
$\mathcal{G} \leftarrow \mathrm{GrGen}(1^\lambda)$	$\bar{c}^* \leftarrow \tilde{\mathrm{H}}_{\mathrm{reg}}(X^*, X^*, \bar{R}^*)$
$Q_{\text{sch-pop}}, Q_{\mathrm{reg}} \leftarrow \emptyset$	**if** $(X^*, \bar{R}^*, \bar{z}^*) \in Q_{\text{sch-pop}}$ **or**
$\omega \leftarrow_\$ \{0,1\}^{\mathrm{rl}_\mathcal{A}}$ // coins given to $\mathcal{A}$	$\bar{R}^*(X^*)^{\bar{c}^*} \neq g^{\bar{z}^*}$ **return** $\perp$
$\{0,1\}^* \leftarrow_\$ \mathcal{A}^{\mathcal{O}^{\text{sch-pop}}, \mathrm{chal}, \tilde{\mathrm{RO}}}(\mathcal{G}, \omega)$	**else**
// $\mathcal{A}$ outputs a bit string	$x^* \leftarrow_\$ \mathrm{Ext}(\mathbb{G}, \omega, Q_{\text{sch-pop}}, Q_{\mathrm{reg}})$
return win	**if** $g^{x^*} \neq X^*$ **then win** $\leftarrow$ **true**
	return x^*
$\underline{\mathcal{O}^{\text{sch-pop}}()}$	
// PoP: Schnorr signature on X	$\underline{\mathcal{O}^{\tilde{\mathrm{RO}}}(\theta)}$ // random oracle
$x, \bar{r} \leftarrow_\$ \mathbb{Z}_q$	**if** $\tilde{\mathrm{H}}(\theta) = \perp$ **then** $\tilde{\mathrm{H}}(\theta) \leftarrow_\$ \mathbb{Z}_q$
$X \leftarrow g^x; \bar{R} \leftarrow g^{\bar{r}}$	**return** $\tilde{\mathrm{H}}(\theta)$
$\bar{c} \leftarrow \tilde{\mathrm{H}}_{\mathrm{reg}}(X, X, \bar{R})$	
$\bar{z} \leftarrow \bar{r} + \bar{c}x$ // proof of knowledge of x	
$Q_{\text{sch-pop}} \leftarrow Q_{\text{sch-pop}} \cup \{(X, \bar{R}, \bar{z})\}$	
return $(X, \bar{R}, \bar{z})$	

Fig. 1. Game used to define the Schnorr knowledge of exponent assumption, where $\mathcal{G} = (\mathbb{G}, q, g)$ defines a cyclic group $\mathbb{G}$ of order q with generator g. PoP refers to "proof of possession," which in this setting means demonstrating knowledge of x. By $\mathrm{rl}_\mathcal{A}$ we denote the randomness length of $\mathcal{A}$. $\tilde{\mathrm{H}}$ is initialized to be an empty table. The hash functions $\tilde{\mathrm{H}}_{\mathrm{reg}}, \tilde{\mathrm{H}}_{\mathrm{cm}}, \tilde{\mathrm{H}}_{\mathrm{non}}, \tilde{\mathrm{H}}_{\mathrm{sig}}$ are computed as $\tilde{\mathrm{H}}(i, \cdot)$ for $i = 1, 2, 3, 4$ [9].

Definition 6 (The Schnorr Knowledge of Exponent (Schnorr-KoE) Assumption [9]). *GrGen be a group generator that outputs $\mathcal{G} = (\mathbb{G}, q, g)$ in which the discrete logarithm assumption holds, and let $\tilde{H}_{reg}$ be a hash function. Let $Adv^{schnorr\text{-}koe}_{\mathcal{A},Ext}(\lambda) = \Pr[Game^{schnorr\text{-}koe}_{\mathcal{A},Ext}(\lambda) = 1]$, where $Game^{schnorr\text{-}koe}_{A,Ext}(\lambda)$ is defined in Figure 1. The Schnorr knowledge of exponent assumption holds with respect to $\mathcal{G}$ and $\tilde{H}_{reg}$ if for all PPT adversaries $\mathcal{A}$, there exists a PPT extractor Ext and a negligible function ν such that $Adv^{schnorr\text{-}koe}_{\mathcal{A},Ext}(\lambda) < \nu(\lambda)$.*

2.2 Threshold Secret Sharing Schemes

This section provides definitions of the fundamental secret-sharing building blocks that govern our distributed key generation. These building blocks are adapted from frameworks presented in [30,14] and addressed by [1] in the context of asynchronous networks.

Definition 7 (Shamir's Threshold Secret Sharing Scheme [1]).
A (t, n)-threshold secret sharing scheme allows a dealer to divide a master secret $s \in \mathbb{Z}_q$ into n distinct private shares such that any subset of t or more parties can reconstruct the secret, while any coalition of at most $t-1$ parties gains zero information about it. The protocol consists of two primary phases:

Share Distribution: *The dealer chooses a random polynomial $f(X) \in \mathbb{Z}_q[X]$ of degree at most $t-1$, defined as $f(X) = \sum_{k=0}^{t-1} a_k X^k$, subject to the constraint that the constant term holds the secret, $f(0) = a_0 = s$. The dealer then securely transmits the private share $s_i = f(i) \pmod q$ to each participant P_i for $i = 1, \ldots, n$.*

Secret Reconstruction: *Upon invocation, any subset $\mathcal{I} \subseteq \{1, \ldots, n\}$ of active participants where $|\mathcal{I}| \geq t$ can uniquely compute the master secret $s = f(0)$ by executing Lagrange polynomial interpolation over their private shares as:*

$$s = \sum_{i \in \mathcal{I}} \lambda_{i,\mathcal{I}} s_i \pmod q.$$

Shamir's baseline scheme assumes an honest dealer and benign participants. To protect against malicious dealers distributing inconsistent shares or adversarial parties submitting erroneous shares during reconstruction, Feldman [14] extended the framework by introducing public cryptographic commitments.

Definition 8 (Feldman's Verifiable Secret Sharing (VSS) [1]).
The verification and fault-mitigation loop operates as follows:

1. ***Commitment Broadcast:*** *Alongside distributing the private shares $s_i = f(i) \pmod q$, the dealer computes and publicly broadcasts a vector of coefficient commitments $\mathbf{C} = (A_0, \ldots, A_{t-1}) \in \mathbb{G}^t$, where $A_k = g^{a_k}$ for $k = 0, \ldots, t-1$. Note that $A_0 = g^s$ implicitly represents the public key corresponding to the shared secret.*
2. ***Share Verification:*** *Upon receiving their private share s_i, each participant P_i independently verifies its mathematical consistency against the public commitments by checking if:*

$$g^{s_i} = \prod_{k=0}^{t-1} (A_k)^{i^k}. \tag{1}$$

3. ***Complaint and Disqualification Loop:***
   (a) *If the verification equation (1) fails for a participant P_i, they publicly broadcast a* complaint *against the dealer.*

(b) *In response to a valid complaint from P_i, the dealer is required to publicly reveal the explicit share s_i that satisfies equation* (1).
(c) *If any of the publicly revealed shares fail to satisfy the commitment check, or if the dealer refuses to broadcast a share for a complaining node, the dealer is immediately identified as malicious and* disqualified.

4. ***Robust Reconstruction:*** *At the time of secret reconstruction, equation* (1) *is re-evaluated to audit the shares submitted by the nodes. If at least t honest and verified shares exist, the underlying secret s can be securely reconstructed regardless of any adversarial disruptions.*

Following the above secret-sharing schemes, below we introduce the key generation protocol of our proposed protocol.

2.3 Pedersen PoP Key Generation

The Pedersen PoP key generation protocol requires each participant to provide polynomial commitments as well as a Schnorr-based proof of possession. This allows for the effective neutralization of rogue key attacks, even in a dishonest majority environment. As stated in [9], security of the Pedersen PoP key generation protocol depends the OMDL and Schnorr-KoE assumptions, which are given in 2.1.

1) Every participant P_i chooses random polynomial $f_i(X)$ over $\mathbb{Z}_q$ of degree $t-1$.
$$f_i(X) = \sum_{k=0}^{t-1} a_{i,k} X^k$$
and computes $A_{i,k} = g^{a_{i,k}}$ for $k = 0, \ldots, t-1$. Each P_i computes **a proof of possession** of $A_{i,0}$ as a Schnorr POK on $A_{i,0}$ as follows. They sample $\bar{r_i} \xleftarrow{\$} \mathbb{Z}_q$, and set $\bar{R_i} \leftarrow g^{\bar{r_i}}$. They compute $\bar{c_i} \leftarrow \mathbf{H_{reg}}(A_{i,0}, A_{i,0}, \bar{R_i})$ and set $\bar{z_i} \leftarrow \bar{r_i} + \bar{c_i} a_{i,0}$. They then derive a commitment $\mathbf{C_i} = (A_{i,0}, \ldots, A_{i,t-1})$ and broadcast $((\bar{R_i}, \bar{z_i}), \mathbf{C_i})$.
2) After receiving commitments from all other parties, each participant verifies the Schnorr POK's by computing $\bar{c_j} \leftarrow \mathbf{H_{reg}}(A_{j,0}, A_{j,0}, \bar{R_j})$ and checking that
$$\bar{R_j} A_{j,0}^{\bar{c_j}} = g^{\bar{z_j}} \text{ for } j = 1, \ldots, n$$
If any checks fail, they disqualify the corresponding participant. Otherwise, they continue to next step.
3) Each P_i computes secret shares $s_{i,j} = f_i(j)$ for $j = 1, \ldots, n$, where j is the participant identifier, and sends $s_{i,j}$ secretly to party P_j.
4) Upon receiving the shares from other participants, every participant P_j verifies the shares by checking that
$$g^{s_{i,j}} = \prod_{k=0}^{t-1} (A_{i,k})^{j^k}$$
If the check fails for an index i, then P_j broadcasts a complaint against P_i.

5) For each of the complaining parties P_j against P_i, P_i broadcasts the share $s_{i,j}$. If any of the revealed shares fails to satisfy the equation, or should P_i not broadcast anything for a complaining player, then P_i is disqualified. The share of disqualified party P_i is set to 0.
6) The secret share for each P_j is $s_j = \sum_{i=1}^{n} s_{i,j}$, and the corresponding public verification share is $Y_j = g^{s_j}$.
7) If $A_{i,0} = A_{j,0}$ for any $i \neq j$, then abort. Else, output is the joint public key $Y = \prod_{i=1}^{n} A_{i,0}$.

Since the share of the disqualified P_i is set to 0 in step 5, the summation and product expressions in steps 6 and 7 are effectively computed only over the non-disqualified parties.

2.4 ROAST: Robust Asynchronous Schnorr Threshold Signatures

If a signing protocol is semi-interactive, which means that it involves a preprocessing round and an actual signing round, provides identifiable cancellations, and is unforgeable under concurrent sessions, ROAST [29] acts as a simple wrapper that transforms this threshold signature scheme into a robust and asynchronous signing protocol. When applied to FROST [22], a state-of-the-art Schnorr threshold signature scheme that meets these requirements, a simple, efficient, and highly practical Schnorr threshold signature scheme is obtained in [29].

In the FROST signing protocol, a session fails if there is a disruptive signer. Therefore, FROST is not a robust signing protocol. The wrapper protocol ROAST overcomes this by allowing a maximum of $n-t+1$ simultaneous signing sessions.

Since we will be using this structure in our proposed protocol in Section 3, this section presents the ROAST algorithm and its subroutines in detail to the reader.

There are 8 subroutines that occur during the execution of the ROAST algorithm. These subroutines assume a prime-order group $(\mathbb{G}, q, g)$, where q is the order of $\mathbb{G}$ and g is the generator. Two hash functions are used, $\mathrm{H}_{\mathrm{non}}$ and $\mathrm{H}_{\mathrm{sig}}$. These hash functions are mapping to $\mathbb{Z}_q$. It is stated in [29] that ROAST algorithm assumes the use of the PedPoP KeyGen [9] subroutine that we provided in the previous subsection. The other subroutines occur as follows.

1. $\mathsf{PreRound}(\mathsf{PK}) \rightarrow (state_i, \rho_i)$:
   Let $\mathrm{PK} = (Y, (Y_1, \ldots, Y_n))$. Each signer is denoted as S_i, and they follow the steps below.
   - Samples $d_i \xleftarrow{\$} \mathbb{Z}_q$ and $e_i \xleftarrow{\$} \mathbb{Z}_q$.
   - Sets $D_i \leftarrow g^{d_i}$ and $E_i \leftarrow g^{e_i}$.
   - Returns $\mathbf{state}_i = (d_i, e_i)$ and $\rho_i = (D_i, E_i)$.

2. $\mathsf{PreAgg}(\mathsf{PK}, \{\rho_i\}_{i\in T}) \to \rho$:
   T[sid] represents the set of signers in session sid. It is called as T for short.
   - Combiner parses $\{(D_i, E_i)\}_{i\in T} \leftarrow \{\rho_i\}_{i\in T}$
   - Computes $D = \prod_{i\in T} D_i$ and $E = \prod_{i\in T} E_i$
   - Sets $\rho \leftarrow (D, E)$
   - Returns ρ

3. $\mathsf{Lagrange}(T, i) \to \lambda_i$:
   The Lagrange coefficient will be computed as follows:

$$\lambda_i \leftarrow \prod_{j\in T\setminus\{i\}} \frac{j}{j-i}$$

   - Returns λ_i

4. $\mathsf{SignRound}(sk_i, \mathsf{PK}, T, state_i, \rho, m) \to (\sigma_i, c)$:
   - Each signer S_i in T parses the following:
     - $s_i \leftarrow sk_i$
     - $(Y, (Y_1, \ldots, Y_n)) \leftarrow PK$
     - $(D, E) \leftarrow \rho$
     - $(d_i, e_i) \leftarrow state_i$
   - Computes $b \leftarrow \mathbf{H}_{non}(Y, T, \rho, m)$
   - Computes $R \leftarrow DE^b$
   - Evaluates $c \leftarrow \mathbf{H}_{sig}(Y, m, R)$
   - Evaluates $\lambda_i \leftarrow \text{Lagrange}(T, i)$
   - Computes the signature share $\sigma_i \leftarrow d_i + be_i + c\lambda_i s_i$
   - Returns σ_i

5. $\mathsf{ShareVal}(\mathsf{PK}, T, i, \rho, \rho_i, \sigma_i, m) \to 1/0$:
   The combiner follows these steps:
   - Parses
     - $(D_i, E_i) \leftarrow \rho_i$
     - $(D, E) \leftarrow \rho$
     - $(Y, (Y_1, \ldots, Y_n)) \leftarrow \text{PK}$
   - Obtains
     - $b \leftarrow \mathbf{H}_{non}(Y, T, \rho, m)$
     - $R \leftarrow DE^b$
     - $c \leftarrow \mathbf{H}_{sig}(Y, R, m)$
     - $\lambda_i \leftarrow \text{Lagrange}(T, i)$
   - Checks whether $g^{\sigma_i} = D_i E_i^b Y_i^{c_i \lambda_i}$

6. $\mathsf{SignAgg}(\mathsf{PK}, \rho, \{\sigma_i\}_{i\in T}, m) \to (s, \sigma)$:
   The combiner follows these steps:
   - Parses
     - $(D, E) \leftarrow \rho$
     - $(Y, (Y_1, \ldots, Y_n)) \leftarrow \text{PK}$
   - Obtains

- $b \leftarrow \mathbf{H}_{non}(Y, T, \rho, m)$
- $R \leftarrow DE^b$

– Computes

$$s \leftarrow \sum_{i \in T} \sigma_i$$

– Returns $\sigma \leftarrow (R, s)$

7. $\mathsf{Verify}(\mathsf{PK}, m, \sigma) \rightarrow 1/0$:
   – The verifier parses
   - $(Y, (Y_1, \ldots, Y_n)) \leftarrow \mathrm{PK}$
   - $(R, s) \leftarrow \sigma$
   – The verifier computes $c \leftarrow \mathbf{H}_{sig}(Y, m, R)$ and checks whether $g^s = RY^c$. If these hold, outputs 1, or 0 otherwise.

To better understand the Algorithm 1, which represents the ROAST wrapper, and Algorithm 2, which represents the signer protocol within it, we want to clarify a few points.

The combiner initiates the ROAST algorithm with $\mathrm{PK} = (Y, (Y_1, \ldots, Y_n))$ published after key generation, the number of participants n, the threshold t, and the message m. Initially, since the n participants may not be available at all times, the set of participants available for signature generation is denoted by W. At the beginning of the algorithm, W is initialized as an empty set. If $i \in W$, S_i is referred to as a responsive participant. The set of malicious participants is denoted by M. It is initialized as an empty set at the beginning of the algorithm. $P[\sim]$ is defined as an array of length n. The latest presignature share "ρ_i'" of participant S_i is stored in the i-th coordinate of P, namely $P[i]$. *sidctr* is a counter function. It counts the sessions and is initialized as 0 at the beginning of the algorithm. $SID[\sim]$ is defined as an array of length n. The session ID including participant S_i is maintained in the i-th coordinate, $SID[i]$. Since ROAST allows at most $n-t+1$ concurrent sessions, the array $T[\sim]$ of length $n-t+1$ is defined to store the set of signers in session *sid* at its corresponding coordinate $T[sid]$. The array $N[\sim]$ of length $n-t+1$ is defined to store the aggregate presignature "ρ" of session *sid* at the coordinate $N[sid]$. Finally, the array $S[\sim]$ of length $n-t+1$ is defined to store the set of signature shares in session *sid* at the coordinate $S[sid]$.

Note that at the very beginning of the ROAST algorithm, since none of the n participants are yet involved in a session, $\sigma_i = \perp$. Consequently, S_i provides the value $(\perp, \rho_i')$ to the combiner. At this point, the algorithm performs a check. The procedure MarkMalicious(i) is introduced in step 37 of the Algorithm 1. Accordingly, it is stated that if the i-th participant is malicious, it is included in the set M, and if $|M| > n - t$, the algorithm fails. Therefore, we see steps 10 through 12. If a participant is in W but the session has not started yet, they cannot submit any signature shares. Therefore, this control protects the system from non-protocol actions. This participant is identified as malicious. If $i \notin M$

and $i \notin R$, then the presignature shared by S_i after the PreRound algorithm is stored by the combiner for the next sessions.

In the Algorithm 1, the step 25 shows us that S_i maintains the protocol's liveness property. After creating and delivering the participant signature, the process is not finished, as it regains its active status to join a new session. It has already created and worked with the signature up to step 24. Therefore, it is in a state to work again, create a signature, and should be added to the list.

If we look at steps 27 through 29 in the Algorithm 1, we will see that when the t participants in W respond, the system starts a new session. Thus, the session count is increased by 1. For each element in this set, the combiner accesses the pre-signatures and generates pre-signatures using PreAgg. The materials used in step 28 are actually the materials that participants submitted as an investment for the future at the end of the previous round. The step that speeds up the protocol is that the combiner, thanks to the materials from step 28, calculates the total pre-signature ρ locally in step 29 without needing to contact the participants again.

Finally, when we examine the steps 30-35, we will see that ρ and W are sent to each element in the responsive set. The purpose is for that participant to recognize other members of the same session. Now, since S_i will be in a new session, $SID[i]$ is renewed, a new set is assigned to T, and N holds the presignature for this set. Finally, the responsive set W is cleared.

2.5 Icy-DVRF: A DVRF based on FROST signatures

To better understand the new protocol we are proposing, we remind the reader of the Icy-DVRF [2] protocol, in which we have replaced the FROST [22] signature mechanism with the ROAST [29] architecture.

We summarise below the subroutines of Icy-DVRF that are modified in this work. The full specification is given in [2].

Let $\mathbb{G}$ be a multiplicative group of order q with a generator g, and let the DDH assumption hold on $\mathbb{G}$. 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 the hash functions, where Λ denotes the desired output length. Also, $\mathcal{P} = \{P_1, \ldots, P_n\}$ is defined as the set of participants.

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

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

   (b) 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.

(c) Every participant P_i computes a public commitments $\mathbf{C}_i = \langle \phi_{i,0}, \ldots, \phi_{i,(t-1)} \rangle$, where $\phi_{i,j} = g^{a_{i,j}}, 0 \leq j \leq t-1$. In particular, note that $\phi_{i,0} = g^{a_{i,0}}$.
(d) Every P_i broadcasts $\mathbf{C}_i, \sigma_i$ to all other participants, or to the public.
(e) Upon receiving $\mathbf{C}_\ell, \sigma_\ell$ from participants $1 \leq \ell \leq n, \ell \neq i$, each participant P_i verifies $\sigma_\ell = (R_\ell, \mu_\ell)$, aborting on failure, by checking $R_\ell \stackrel{?}{=} g^{\mu_\ell} \cdot \phi_{\ell,0}^{c_\ell}$, where $c_\ell = H_2(\ell, \Phi, \phi_{\ell,0}, R_\ell)$. Upon success, participants delete $\{\sigma_\ell : 1 \leq \ell \leq n\}$.

Round 2

(a) 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.
(b) Each P_i verifies their shares by calculating

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

aborting if the check fails.
(c) 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 $f_\ell(i)$.
(d) 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

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

2. 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.
(a) Create an empty list L_i. Then, for $1 \leq j \leq \Omega$, perform the following:
   i. Sample single-use nonces $(d_{i,j}, e_{i,j}) \stackrel{\$}{\leftarrow} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.
   ii. Derive commitment shares $(D_{i,j}, E_{i,j}) = (g^{d_{i,j}}, g^{e_{i,j}})$.
   iii. Append $(D_{i,j}, E_{i,j})$ to L_i. Store these values for later use.
(b) Publish (i, L_i) to a predetermined location, as specified by the implementation.

3. PartialEval:
Let C be the combiner (who might be chosen amongst the participants). Let K 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.

(a) C sends (m, j, K') to P_i, $i \in K$, where K' is the ordered list of parties in K and j is the index of the single-use commitments from the Preprocess stage to be used.
(b) Each P_i checks if K' is valid and if m is the message they want to use. If the checks fail, abort.
(c) Each P_i performs the following.
   i. Compute $h \leftarrow H_1(m) \in \mathbb{G} \setminus \{1\}$.
   ii. Compute partial evaluation $\gamma_i = h^{s_i} \in \mathbb{G}$.
   iii. Compute partial evaluation proof $\pi_i \leftarrow \text{ProveEq}_{H_2}(g, h, Y_i, \gamma_i; s_i)$.
   iv. Compute $\bar{D}_i = h^{d_{i,j}}$ and $\bar{E}_i = h^{e_{i,j}}$.
   v. Outputs $(\gamma_i, \pi_i, \bar{D}_i, \bar{E}_i)$.

4. PartialProofGen:
Now that each participant P_i published $(\gamma_i, \pi_i, D_i, E_i, \bar{D}_i, \bar{E}_i)$, each P_i produces their partial DVRF proof, using the following procedure.
(a) Compute the set of binding values $\rho_\ell = H_2(\ell, m, K')$ for all $\ell \in K$.
(b) Compute $k_i = d_i + \rho_i \cdot e_i$.
(c) Compute the (public) group commitments $R = \prod_{\ell \in K} D_\ell \cdot (E_\ell)^{\rho_\ell} = g^k$ and $\bar{R} = \prod_{\ell \in K} \bar{D}_\ell \cdot (\bar{E}_\ell)^{\rho_\ell} = h^k$, where $k = \sum_{\ell \in K} k_\ell$, unknown to individual parties.
(d) Compute $\gamma = \prod_{\ell \in K} \gamma_\ell^{\lambda_\ell}$, where λ_ℓ is the ℓth Lagrange coefficient in K.
(e) Compute the group challenge $c = H(g, h, Y, \gamma, R, \bar{R}, m)$.
(f) Computes their 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 K.
(g) Securely deletes d_i, e_i from their local storage, and broadcasts (z_i, c).

5. Combine:
Upon receiving (z_ℓ, c) from P_ℓ for $\ell \in K$, the combiner C performs the following actions.
(a) Computes the following.[4]
   i. $\rho_\ell = H_2(\ell, m, K')$ for $\ell \in K$.
   ii. $R_\ell = D_\ell \cdot (E_\ell)^{\rho_\ell}$ for $\ell \in K$.
   iii. $\bar{R}_\ell = \bar{D}_\ell \cdot (\bar{E}_\ell)^{\rho_\ell}$ for $\ell \in K$.
   iv. The group commitment $R = \prod_{\ell \in K} R_\ell$.
   v. The group commitment $\bar{R} = \prod_{\ell \in S} \bar{R}_\ell$.
   vi. $\gamma = \prod_{\ell \in K} \gamma_\ell^{\lambda_\ell}$, where λ_ℓ is the ℓth Lagrange coefficient in K.
   vii. The group challenge $c = H_2(g, h, Y, \gamma, R, \bar{R}, m)$.
(b) Verifies the following.
   i. Verifies that each c received from P_ℓ matches the c value the combiner computed.

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

ii. Verifies each partial evaluation proof by checking

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

for each $(\gamma_\ell, \pi_\ell), \ell \in K$, where Y_ℓ is the partial public key of P_ℓ.

iii. Verifies the validity of the response of $P_\ell, \ell \in K$, 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 $z_\ell, \ell \in K$.

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 K are not qualified, abort the protocol. Otherwise, continue.

(c) Compute the group response $z = \sum_{\ell \in K} z_\ell$.

(d) Output the VRF value $\beta \leftarrow H_3(\gamma)$ and the proof as $\Pi = (\gamma, c, z)$.

6. Verify:

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

(a) Compute $T = g^z \cdot Y^{-c}$.

(b) Compute $h = H_1(m)$ and $\bar{T} = h^z \cdot \gamma^{-c}$.

(c) Check if $c \stackrel{?}{=} H_2(g, h, Y, \gamma, T, \bar{T}, m)$ and $\beta \stackrel{?}{=} H_3(\gamma)$. If both are satisfied, output 1; otherwise, output 0.

3 Protocol Description

While the FROST [22] design eliminates extra communication rounds and prioritizes a short, turn-efficient protocol over robustness, ROAST [29] initiates a maximum of $n - t + 1$ asynchronous signing sessions to transform FROST [22] into a robust protocol.

FROST's adopted model is called an abort and identify model, which cancels the session and identifies the misbehaving participant when any participant sends an invalid partial signature. Therefore, it requires honest participants to retry. Because Icy-DVRF inherits this feature from FROST, it also suffers from the same lack of robustness. This section replaces the abort and identify model with our central technical change, ROAST's concurrent session mechanism.

The wrapper's coordination logic initiates at most $n - t + 1$ concurrent sessions, since a participant that is pending in a session is not a member of the responsive set W and therefore cannot be assigned to a new one. We make this precise in Lemma 1 and convert it into round complexity in Lemma 3.

Regarding round and communication efficiency, Icy-DVRF uses a preprocessing scheme, similar to FROST, to reduce the number of interaction rounds between participants.It improves upon DVRFwCP [1] and offers a protocol that

preserves a fixed-size proof while eliminating the quadratic computational overhead of PartialProofGen. It bases its safety and efficiency model on abort on error. This creates a significant practical problem in asynchronous or real-world networks.

Building upon fault-tolerant asynchronous design principles, we formally define the (R)Icy-DVRF protocol below.

Definition 9. *((R)Icy-Distributed Verifiable Random Function) A t-out-of-n (R)Icy-DVRF, i.e., $\mathcal{V}_R$, consists of one interactive key-generation protocol and seven algorithms, i.e., $\mathcal{V}_R$ = (KeyGen, PreRound, PartialEval, PreAggregation, PartialProofGen, ShareVal, ProofAgg, Verify), described below.*

KeyGen$(1^\lambda, t, n)$*: An interactive protocol run by all n participants on input a security parameter λ, the threshold t and the number of participant nodes n. Each P_i obtains its partial secret-public key pair (s_i, Y_i), and all parties obtain the verification shares $\{Y_i\}_{i\in[n]}$ and the group public key Y.*

PreRound$(m) \to (state_i, \rho_i)$*: It is run by each participant, which takes the plaintext m as input and outputs a secret state $state_i$ and commitment share ρ_i.*

PartialEval$(m, s_i, Y_i) \to (\gamma_i, \pi_i)$*: It is run by each participant once per plaintext, which takes the plaintext m, the partial secret key s_i and the verification key Y_i as inputs and outputs a partial evaluation γ_i and partial evaluation NIZK proof π_i.*

PreAggregation$(m, \{(\rho_i, \gamma_i, \pi_i, Y_i)\}_{i\in S}) \to (B)$*: It is run by the combiner, which takes the plaintext m, the set of commitment shares, partial evaluations, their partial evaluation NIZK proofs and verification keys as inputs, where S is the set of participants of the session, and outputs the ordered list B containing the tuples $(D_i, E_i, \bar{D}_i, \bar{E}_i, \gamma_i)$ for all $i \in S$.*

PartialProofGen$(m, B, state_i, s_i, Y) \to (\Pi_i, \rho'_i)$*: It is run by each participant in a session, which takes the plaintext m, the ordered list B, $state_i$, the partial secret key s_i and the group public key Y as inputs and outputs the partial proof $\Pi_i = (z_i, c)$ and the next commitment share $\rho'_i = (D'_i, E'_i, \bar{D}'_i, \bar{E}'_i)$.*

ShareVal$(m, B, Y, Y_l, \Pi_l) \to 0/1$*: It is run by the combiner, which takes a plaintext m, the ordered list B, the group public key Y, the verification key Y_l and the partial proof Π_l as inputs and outputs $0/1$.*

ProofAgg$(\gamma, (\Pi_i)_{i\in S}) \to (\beta, \Pi)$*: It is run by the combiner on the aggregated evaluation value $\gamma = \prod_{l\in S} \gamma_l^{\lambda_l}$, which the combiner computes from B, and on the partial proofs. It outputs the DVRF value β and the combined proof $\Pi = (\gamma, c, z)$.*

Verify$(m, Y, \beta, \Pi) \to 0/1$*: It takes a plaintext m, the group public key Y, a DVRF value β, a combined proof $\Pi = (\gamma, c, z)$ as inputs, it outputs $0/1$.*

We define the security of the (R)Icy-DVRF as follows.

Definition 10 (Security of (R)Icy-DVRF). *For an input m, let $Q(m)$ denote the number of distinct participants for which the adversary holds a valid partial evaluation of m, that is, the participants it has corrupted together with the honest participants it has queried on m either in PartialEval or in a session of PartialProofGen on m. We call m a* fresh input *if $Q(m) < t$ (Remark 1 explains why this is the right condition for (R)Icy-DVRF). Session responses count, since*

$\gamma_i = \left(h^{z_i}\bar{R}_i^{-1}\right)^{(c\lambda_i)^{-1}}$ *whenever* $c \neq 0$*, which fails only with the probability* $1/q$*, where* q *is the prime order of* $\mathbb{G}$*.*

The (R)Icy-DVRF protocol is said to be secure if it satisfies the following properties.

- ***Correctness:*** *For every plaintext* m *and every honest execution of the protocol in which a session of size* $|S| = t$ *completes,* VERIFY *outputs* 1 *on the resulting pair* (β, Π)*.*
- ***Consistency:*** *For every plaintext* m *and every pair of qualified sets* $S, S' \subseteq [n]$ *of size* t*, the honest executions on* S *and on* S' *produce the same output* β*.*
- ***Public verifiability:*** *Anyone should be able to verify the proof for the correctness of the pseudorandom value* β *from the public data* (m, Y, β, Π) *alone, removing the need to trust the parties involved in the generation of the VRF value or of the proof.*
- ***Uniqueness:*** *For any given plaintext* m*, no adversary making at most* q_H *queries to the random oracle can compute two pairs* (β_1, Π_1) *and* (β_2, Π_2) *with* $\beta_1 \neq \beta_2$ *that both pass* VERIFY*, except with negligible probability.*
- ***Unforgeability:*** *For every fresh input* m*, no PPT adversary corrupting at most* $t-1$ *participants can produce a pair* (β, Π) *that passes the* VERIFY *subroutine, except with negligible probability.*
- ***Pseudorandomness:*** *For every fresh input* m*, no PPT adversary corrupting at most* $t-1$ *participants can distinguish the output* β *from a uniformly random element of* $\{0,1\}^\Lambda$ *with non-negligible advantage.*
- ***Robustness [29]:*** *Let* $F \subseteq [n]$ *with* $|F| = f$ *denote the set of corrupted participants. The protocol is* (n, t, f)*-robust if, for every plaintext* m *and every key tuple generated by* KEYGEN*, in any execution in which the adversary controls the timing and delivery of all network messages but ultimately delivers every message sent between honest participants, including the combiner, an honest combiner terminates and outputs a pair* (β, Π) *with* $\text{VERIFY}(m, Y, \beta, \Pi) = 1$*. If the protocol is* $(n, t, n-t)$*-robust, we say that the protocol is* robust*.*

3.1 Model and Assumptions

Network. Our assumption is an asynchronous network with authenticated point-to-point channels between every pair of parties. Although the adversary plans and delays messages optionally, it must eventually deliver every message between honest parties, including the combiner. Our protocol does not use timeouts or a broadcast channel during the evaluation. The only component using a reliable broadcast channel is the one-time PEDPOP setup of Section 2.3. We measure latency in asynchronous rounds by following ROAST [29] in the standard sense: messages sent in parallel, such as C's message to all members of a session, count as one round, so a request and its reply count as two. As is standard for round complexity under asynchrony, the maximum delay of a message that the adversary does deliver is normalised to one round. The adversary may

still schedule deliveries arbitrarily within that normalisation, and no party uses a timeout.

Adversary. Corruption is treated statically. The validity of unforgeability and pseudorandomness is assumed to hold against an adversary capable of corrupting at most $t-1$ participants. Robustness is assumed to be valid in the presence of at most $f = n - t$ faulty participants, who may be Byzantine or simply be crashed. Both bounds are required simultaneously, so our assumption throughout the paper is the following:

$$n - t \leq t - 1, \qquad \text{equivalently} \qquad t \geq \tfrac{n+1}{2}. \tag{2}$$

We refer to (2) as the *threshold constraint*. The combiner holds no secret shares and is trusted only for liveness. Definition 10 accordingly quantifies robustness over executions with an honest combiner, and Section 3.5 discusses how a faulty combiner is tolerated.

Input. The plaintext m, the DVRF input, is a value publicly determined and known by all participants before PreRound and PartialEval for that plaintext begins—for example, an epoch counter, a block hash, or a beacon round. This matches the applications discussed in Section 1.1 and the ROAST model, since the coordinator $\mathcal{C}(\mathrm{PK}, n, t, m)$ is invoked once per plaintext and all of its concurrent sessions share the same m. Because commitment shares are bound to m, the state the combiner keeps for a plaintext is local to that instance and is discarded when it terminates, and each new plaintext starts from a fresh PreRound.

3.2 Applicability of ROAST Wrapper to Icy-DVRF

ROAST [29] is a black-box robustness wrapper that can be applied to any threshold protocol that satisfies the conditions that the protocol is semi-interactive, provides identifiable cancellations, and remains unforgeable under concurrent sessions. In (R)Icy-DVRF the first two are structural properties, which we establish in Proposition 1together with the absence of replay of a partial re sponse across sessions (Lemma 2). The third is a computational property of the evaluation layer, which we formalise as Definition 11 and state as a hypothesis, as discussed in Remark 2. Our technical contribution is to redesign the DVRF's evaluation and proof subroutines so that these preconditions can be met, while leaving the coordination logic intact. That logic is that of Algorithm 1: the responsive set W, the session counter, the pending discipline and the malicious list are unchanged. Three local modifications are needed at the combiner. The aggregate presignature ρ is replaced by the ordered list B, which is not aggregated because the Lagrange coefficients depend on the session committee. ShareVal also verifies two equations per participant instead of one, and PreAggregation is inserted once $|W|$ reaches t, before B is sent. None of these alters the pending discipline on which Lemma 1 depends. We address the three preconditions in turn, together with the key-generation protocol that ROAST assumes.

- **Dual verification in ShareVal.** The ROAST signature session verifies a single equation per participant. However, the evaluation of (R)Icy-DVRF is defined via a second, message-dependent basis with $h = H_1(m)$ and $\gamma_i = h^{s_i}$. Therefore, ShareVal verifies two equations per participant: $g^{z_\ell} \stackrel{?}{=} R_\ell \cdot Y_\ell^{c\lambda_\ell}$ and $h^{z_\ell} \stackrel{?}{=} \bar{R}_\ell \cdot \gamma_\ell^{c\lambda_\ell}$. The second equation ensures that the h-basis twin is subsequently verifiable and provides exact attribution in Proposition 1.
- **Message-dependent presignature shares.** In ROAST, a nonce commitment (D_i, E_i) is message-independent and can be generated entirely offline, but this is impossible in our design. Neither the twin $(\bar{D}_i, \bar{E}_i) = (h^{d_i}, h^{e_i})$ nor the partial evaluation $\gamma_i = h^{s_i}$ can be derived from (D_i, E_i) or Y_i by anyone other than the shareholder, and neither exists before $h = H_1(m)$ is fixed. For these reasons, we bind the presignature share to m, meaning that each participant publishes $(D_i, E_i, \bar{D}_i, \bar{E}_i)$ in PreRound and the session-independent pair (γ_i, π_i) in PartialEval. Due to the assumption of Section 3.1 that m remains fixed throughout the protocol, the new share ρ_i' created at the end of a session carries its own twin. Therefore, each subsequent session of the same input does not require an additional preprocessing round. This satisfies the first prerequisite of ROAST by keeping the evaluation phase a single round. It also strengthens identifiability, since π_i arrives before a session starts, a participant providing a faulty partial evaluation is excluded before any session message is sent to it.
- **Single-use nonce state.** The nonce state machine of step 5 of Section 3.3 prevents any partial response from being replayed across sessions, as stated in Lemma 2.
- **Compatible key generation.** We use the PedPoP [9] protocol recommended by ROAST instead of the FROST-style DKG used by Icy-DVRF. This key generation protocol provides a proper fit in terms of underlying secret sharing.

3.3 (R)Icy-DVRF Protocol

Let $(\mathbb{G}, g, q)$ be a cyclic multiplicative group where the DDH assumption holds. Let $H_{reg} : \{0,1\}^* \to \mathbb{Z}_q$, $H_1 : \{0,1\}^* \to \mathbb{G}$, $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ and $H_3 : \{0,1\}^* \to \{0,1\}^\Lambda$ be the hash functions, where the desired output length is denoted as Λ. The oracle H_2 is used for three unrelated purposes, so we domain-separate it and write

- $H_2^{\mathsf{eq}}(\cdot) = H_2(\texttt{"RICY/eq"} \,\|\, \cdot)$,
- $H_2^{\mathsf{bind}}(\cdot) = H_2(\texttt{"RICY/bind"} \,\|\, \cdot)$,
- $H_2^{\mathsf{chal}}(\cdot) = H_2(\texttt{"RICY/chal"} \,\|\, \cdot)$.

Assume that we have the set of participants as $\mathcal{P} = \{P_1, \ldots, P_n\}$.

Before delving into technical details, we explain the notation used in the subroutines. While the group public key is denoted as Y, the combiner is denoted as C. $W \subseteq \mathcal{P}$ denotes the responsive set, the participants currently available to be assigned to a new session. S represents the set of participants in each session,

and the set B represents the ordered list containing the tuples $(D_i, E_i, \bar{D}_i, \bar{E}_i, \gamma_i)$ for all $i \in S$, sorted deterministically by the participant identifiers i. Also, we must note that $|S| = t$, and without loss of generality, C can be one of the participants. Each participant maintains a local state $state_i = (d_i, e_i, \mathsf{status}_i)$, where (d_i, e_i) is the current nonce pair and $\mathsf{status}_i \in \{\mathsf{FREE}, \mathsf{BOUND}\}$ records whether that pair has been committed to a session.

1. KeyGen:
   Instead of the FROST Distributed Key Generation algorithm used in Icy-DVRF [2], our protocol primarily uses the PedPoP key generation protocol [9], which we presented in 2.3. In our protocol, each user runs this protocol as a subroutine and generates the secret-public key pairs and the group public key.

2. PreRound
   Each of the participants P_i for $i \in \{1, \ldots, n\}$ performs the steps below.
   (a) Computes $h \leftarrow H_1(m) \in \mathbb{G} \setminus \{1\}$
   (b) Sample single-use nonces $(d_i, e_i) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$ and sets
   $$state_i = (d_i, e_i, \mathsf{FREE})$$
   (c) Derive g-basis commitment share $(D_i, E_i) = (g^{d_i}, g^{e_i})$
   (d) Derive h-basis commitment share $(\bar{D}_i, \bar{E}_i) = (h^{d_i}, h^{e_i})$ and let $\rho_i = (D_i, E_i, \bar{D}_i, \bar{E}_i)$

3. PartialEval
   Each P_i executes the following steps once per m plaintext. The result is session-independent and reused in every session of that entry.
   (a) Computes partial evaluation $\gamma_i = h^{s_i} \in \mathbb{G}$
   (b) Computes partial evaluation proof $\pi_i \leftarrow \mathrm{ProveEq}_{H_2^{\mathrm{eq}}}(g, h, Y_i, \gamma_i; s_i)$
   (c) Each P_i sends $(\rho_i, \gamma_i, \pi_i)$ to C.

4. PreAggregation
   C selects $S \subseteq W$ with $|S| = t$ from the responsive set and, using the share ρ_i received from each $P_i \in S$ (if this is the first session for m, then C has also received (γ_i, π_i)), performs the following steps:
   (a) Computes $h \leftarrow H_1(m)$ and parses the list to obtain $\{(D_i, E_i, \bar{D}_i, \bar{E}_i)\}_{i \in S}$.
   (b) Verifies $\mathrm{VerifyEq}_{H_2^{\mathrm{eq}}}(g, h, Y_i, \gamma_i, \pi_i) \stackrel{?}{=} 1$ for each $i \in S$.

   When C receives (γ_i, π_i) for an m for the first time, it verifies them and caches them, so they are not sent again. In subsequent sessions of the same m, this check is applied only to participants whose pair has not yet been verified.

(c) Any participant failing this check is marked as malicious and is permanently excluded. Since $|S| = t$, a single failure leaves fewer than t verified participants, so the remaining participants of S are returned to the responsive set W and the session is cancelled without consuming a session identifier and without any message being sent by C. Otherwise,
   i. Create B as a sorted list of tuples of participants who passed the verification, based on their identifier.
   ii. Sends the ordered list B and the plaintext m to each participant in S.

5. PartialProofGen
Upon receiving (B, m) from C, each P_i follows these steps:
(a) Performs the following checks and aborts if any of them fails
   i. Obtains the set S from the identities appearing in B.
   ii. Checks if its own share ρ_i and its own partial evaluation γ_i match the ones in B.
   iii. Checks if $|S| = t$.
   iv. Checks if m is the intended plaintext.
(b) **Nonce binding.** If $\mathsf{status}_i = \mathsf{FREE}$, it sets $\mathsf{status}_i \leftarrow \mathsf{BOUND}$ and continues. Otherwise the message is ignored.
(c) Computes the set of binding values $b_l = H_2^{\mathsf{bind}}(l, m, B)$ where $l \in S$.
(d) Computes $k_i = d_i + b_i e_i$.
(e) Computes $R = \prod_{l \in S} D_l E_l^{b_l} = g^k$ and $\bar{R} = \prod_{l \in S} \bar{D}_l \bar{E}_l^{b_l} = h^k$ (implicitly $k = \sum_{l \in S} k_l$).
(f) Computes $\gamma = \prod_{l \in S} \gamma_l^{\lambda_l}$ where λ_l is the appropriate Lagrange coefficient.
(g) Computes the group challenge $c = H_2^{\mathsf{chal}}(g, h, Y, \gamma, R, \bar{R}, m)$.
(h) Computes its response using its long-lived partial secret key s_i by computing the partial proof

$$z_i = k_i + \lambda_i s_i c = d_i + b_i e_i + \lambda_i s_i c.$$

(i) Runs PreRound again to obtain a fresh pair $(state_i', \rho_i')$ with $\rho_i' = (D_i', E_i', \bar{D}_i', \bar{E}_i')$, securely deletes d_i and e_i, and sets $state_i \leftarrow state_i'$.
(j) Sends (Π_i, ρ_i') to C, where $\Pi_i = (z_i, c)$.

6. ShareVal
Upon receiving Π_l from P_l for $l \in S$ after PartialProofGen subroutine, C performs the following actions:
(a) Computes the following
   i. $b_l = H_2^{\mathsf{bind}}(l, m, B)$ for $l \in S$
   ii. $R_l = D_l (E_l)^{b_l}$ for $l \in S$ and $\bar{R}_l = \bar{D}_l (\bar{E}_l)^{b_l}$ for $l \in S$
   iii. $R = \prod R_l$ and $\bar{R} = \prod \bar{R}_l$
   iv. $\gamma = \prod_{l \in S} \gamma_l^{\lambda_l}$, where λ_l is the appropriate Lagrange coefficient
   v. $c = H_2^{\mathsf{chal}}(g, h, Y, \gamma, R, \bar{R}, m)$

Although this subroutine is defined as a calculation of R, $\bar{R}$, γ and c independently for each $l \in S$, these values have no dependence on l or Π_l. C calculates them once per session, immediately after PreAggregation returns B, and reuses them in all t calls, as reflected in the cost accounting of Table 1. This γ is the one passed to ProofAgg.

(b) and then, separately for each $l \in S$, outputs 1 if and only if both of the following hold:
   i. Each c received from P_l matches the c it computes.
   ii. The validity of partial proofs for $l \in S$

$$g^{z_l} \stackrel{?}{=} R_l \cdot Y_l^{c\lambda_l} \qquad h^{z_l} \stackrel{?}{=} \bar{R}_l \cdot \gamma_l^{c\lambda_l}$$

(c) If ShareVal outputs 0 for P_l, C identifies and marks P_l as malicious, since its partial evaluation proof π_l is already verified in PreAggregation.

7. ProofAgg
Once ShareVal gives the output 1 for all t participants of the session, C follows these steps:
   (a) Computes the group response $z = \sum_{l \in S} z_l$.
   (b) Outputs the DVRF value $\beta \leftarrow H_3(\gamma)$ and the proof as $\Pi = (\gamma, c, z)$.
   If fewer than t partial proofs are accepted, no output is produced for the session, and the protocol proceeds with other sessions.

8. Verify
Given the group public key Y, the proof $\Pi = (\gamma, c, z)$, the DVRF value β and the plaintext m, the verifier performs the verification as follows:
   (a) Compute $T = g^z Y^{-c}$
   (b) Compute $h \leftarrow H_1(m)$ and $\bar{T} = h^z \gamma^{-c}$
   (c) Check if $c \stackrel{?}{=} H_2^{\mathsf{chal}}(g, h, Y, \gamma, T, \bar{T}, m)$ and $\beta = H_3(\gamma)$. If both are satisfied, output 1; otherwise, output 0.

3.4 Security

The security of the proposed (R)Icy-DVRF scheme is established by evaluating its correctness, consistency, public verifiability, uniqueness, unforgeability, pseudorandomness, and robustness. Throughout this subsection, we work in the random oracle model and under the model and assumptions of Section 3.1. We give the results with their proofs. Only Theorem 3 is conditional on Definition 11. In particular, Theorem 4 does not use it and relies only on combinatorial arguments and the soundness of the DLEQ proof system, while Theorem 2 is unconditional. Lemma 1, Lemma 3 and Theorem 4 assume an honest combiner, whereas Theorem 2 and Theorem 3 allow the combiner to be the adversary itself.

Lemma 1 (Session bound). *In any execution of (R)Icy-DVRF with an honest combiner and at most $n-t$ disruptive participants, the combiner initiates at most $n-t+1$ sessions, where a session is said to be initiated when the combiner sends the ordered list B to its members.*

Proof. A pending participant belongs to exactly one session and is outside W, so it is never assigned to a newly initiated session. A session cancelled in PreAggregation returns its verified participants to W and permanently excludes one participant, so at most $n-t$ such cancellations occur and they leave no participant pending. An initiated session that has not completed contains a *blocking* member, which has either not yet replied and is therefore pending, or has replied with a share rejected by ShareVal, and is therefore marked malicious. Either way, it is outside W and blocks no other session, since a marked participant never re-enters W, while members that replied with valid shares return to W whatever the fate of the session is. If $n-t+1$ sessions had been initiated and none had completed, their blocking members would be pairwise distinct and outside W, so $|W| \leq n-(n-t+1) = t-1 < t$ and no further session could be initiated. Since an honest combiner stops at the first completed session, at most $n-t+1$ sessions are initiated in total. Only the bound on the number of cancellations uses $f \leq n-t$. □

Theorem 1 (Correctness). *For every plaintext m and honest executions of (R)Icy-DVRF where a session of size $|S| = t$ completes, the resulting pair (β, Π) satisfies $\textsc{Verify}(m, Y, \beta, \Pi) = 1$.*

Proof (Proof Sketch). Let $S \subseteq \mathcal{P}$ denote the session committee of size $|S| = t$ that completed the evaluation phase. In PedPoP, participant P_i holds $s_i = \sum_j f_j(i)$, the evaluation at i of the aggregate polynomial $f = \sum_j f_j$, which has degree at most $t-1$ and satisfies $f(0) = \sum_j a_{j,0} = s$ with $Y = g^s$. By Lagrange interpolation applied to f, we therefore have $\sum_{i \in S} \lambda_i s_i = s$, where the λ_i are the Lagrange coefficients for S. Recomputing T with the group public key $Y = g^s$, we obtain

$$T = g^z Y^{-c} = g^{\sum_{i \in S}(d_i + b_i e_i)}\, g^{c \sum_{i \in S} \lambda_i s_i}\, Y^{-c} = g^k Y^c Y^{-c} = g^k = R, \tag{3}$$

where $k = \sum_{i \in S}(d_i + b_i e_i)$ represents the unique joint session nonce hidden from all individual participants. An identical computation, replacing g, Y with h, γ and using $\gamma = \prod_{l \in S} \gamma_l^{\lambda_l} = h^s$ from the same interpolation, gives $\bar{T} = h^z \gamma^{-c} = h^k = \bar{R}$. Substituting $T = R$ and $\bar{T} = \bar{R}$ into the challenge of Verify gives

$$c = H_2^{\mathsf{chal}}(g, h, Y, \gamma, T, \bar{T}, m) = H_2^{\mathsf{chal}}(g, h, Y, \gamma, R, \bar{R}, m), \tag{4}$$

which is exactly the challenge computed in step 5. At the end, since $\beta = H_3(\gamma)$ by construction in ProofAgg, both checks of Verify succeed. □

Consistency follows from the same interpolation: $h = H_1(m)$ is fixed by m, and Lagrange interpolation in the exponent gives $\gamma = \prod_{l \in S} \gamma_l^{\lambda_l} = h^s$ for every qualified S, independently of S, so $\beta = H_3(\gamma)$ is the same for every qualified set.

Public verifiability is immediate, since VERIFY reads only (m, Y, β, Π) and no per-participant data.

Theorem 2 (Uniqueness). *In the random oracle model, for every adversary making at most q_H queries to H_2^{chal}, the probability that it outputs a plaintext m and two pairs (β_1, Π_1), (β_2, Π_2) with $\beta_1 \neq \beta_2$ that both pass VERIFY is at most $(q_H+1)/q$. The bound is unconditional, requiring no computational assumption.*

Proof (Proof Sketch). Fix m, let $h = H_1(m)$ and $s = \log_g Y$, and call $\Pi = (\gamma, c, z)$ *false* if $\gamma \neq h^s$. If two distinct challenges $c \neq c'$ admitted accepting responses z, z' for the same oracle query $(g, h, Y, \gamma, T, \bar{T}, m)$, then $g^{z-z'} = Y^{c-c'}$ and $h^{z-z'} = \gamma^{c-c'}$, so $\log_g Y = (z - z')/(c - c') = \log_h \gamma$ and $\gamma = h^s$. Hence, for each query with a false γ, at most one challenge value can be completed to an accepting proof. Union bound over the q_H queries and the one made at verification demonstrates that a false accepting proof exists with probability at most $(q_H+1)/q$ because H_2^{chal} answers each fresh query uniformly in $\mathbb{Z}_q$. Finally, two accepting pairs with $\beta_1 \neq \beta_2$ have $\gamma_1 \neq \gamma_2$, since VERIFY checks $\beta = H_3(\gamma)$, so at least one of them is false. Hence γ, and therefore β, is unique except with negligible probability. □

It should be noted that uniqueness follows from the (statistical) soundness of the Fiat–Shamir compiled Chaum–Pedersen proof system in the random oracle model, not from the determinism of the evaluation layer and not from any computational assumption.

Lemma 2 (No cross-session nonce reuse). *For every honest participant P_i and nonce pair (d_i, e_i) generated by PREROUND, P_i outputs at most one partial proof z_i computed from (d_i, e_i).*

Proof. According to step (b) of 5, P_i computes a partial proof only after changing status_i from FREE to BOUND, and a message failing the checks of step (a) is aborted before this point. Subsequent messages are ignored because $\mathsf{status}_i =$ BOUND. Thus, at most one z_i is computed using the same (d_i, e_i) pair. The status returns to FREE only in step (i), where P_i securely deletes the current (d_i, e_i) and replaces $state_i$ with a freshly generated pair. Hence, (d_i, e_i) is never current again, and P_i outputs at most one z_i computed from it. □

Lemma 2 is a barrier to key-recovery attacks. If two different commitment lists $B \neq B'$ were to generate two different z_i and z_i' using the same (d_i, e_i) nonce pair, they would form two different linear equations in terms of d_i, e_i, and s_i. Therefore, if a third use of the same (d_i, e_i) were to occur, a third equation would be given to the adversary, and solving it would lead to s_i. Since a nonce pair can enter one session at most, two different sessions can never carry the same B. Therefore, B itself already distinguishes sessions, and no separate session identifier needs to be stored.

Definition 11 (Concurrent security of the evaluation layer). *By the* evaluation layer *of (R)Icy-DVRF we mean its algorithms KEYGEN, PREROUND,*

PartialEval, PartialProofGen, ShareVal, ProofAgg and Verify, that is, the protocol of Section 3.3 stripped of the coordination logic of [29]. In $\mathsf{Game}^{\mathsf{conc}}_{\mathcal{A}}$ *the adversary* $\mathcal{A}$ *statically corrupts a set* $F \subset [n]$ *with* $|F| \leq t-1$ *and runs KeyGen with the challenger, which plays every honest* P_i*. Acting as the combiner and as the network scheduler,* $\mathcal{A}$ *may then query, in any interleaving and any polynomial number of times, the random oracles and, for honest* i*, the following three oracles. The oracle* $\mathcal{O}^{\mathsf{Eval}}(i, m)$ *runs PartialEval for* P_i *on* m *at the first query on the pair* (i, m) *and returns that same pair* (γ_i, π_i)*, unchanged, at every later query on* (i, m)*. The oracle* $\mathcal{O}^{\mathsf{Pre}}(i, m)$ *returns the current commitment share* ρ_i *of* P_i *for* m*, generated by PreRound, and the oracle* $\mathcal{O}^{\mathsf{Sess}}(i, m, B)$ *runs step 5 for* P_i *on the list* (B, m) *and returns* (Π_i, ρ'_i)*, or* $\perp$ *when a check of step 5(a) fails or the nonce state machine of step 5(b) refuses the call. Let* $Q(m)$ *be* $|F|$ *plus the number of honest indices* i *queried on* m *through* $\mathcal{O}^{\mathsf{Eval}}$ *or* $\mathcal{O}^{\mathsf{Sess}}$*, as in Definition 10. For* unforgeability*,* $\mathcal{A}$ *outputs* (m^*, β^*, Π^*) *and wins if* $Q(m^*) < t$ *and* $\textsc{Verify}(m^*, Y, \beta^*, \Pi^*) = 1$*. For* pseudorandomness*, the challenger samples a hidden bit* b *uniformly at random,* $\mathcal{A}$ *names an input* m^* *and receives* $H_3(H_1(m^*)^s)$ *if* $b = 1$ *and a uniform element of* $\{0,1\}^{\Lambda}$ *otherwise, and* $\mathcal{A}$ *wins if it outputs* b *while* $Q(m^*) < t$ *at the end of the game. The evaluation layer is* concurrently secure *if every PPT* $\mathcal{A}$ *wins the first game with negligible probability and the second with probability negligibly close to* $1/2$*.*

Definition 11 has two games playing different roles. As in [2], pseudorandomness is the standalone DVRF property for a fresh input, and it ultimately rests on DDH in $\mathbb{G}$. Concurrent unforgeability is what the coordination layer needs. Indeed, ROAST opens several sessions on the same input, possibly with different committees, and the adversary must not combine the responses of these sessions into an accepting proof for an input m^* with $Q(m^*) < t$. Both games share the oracles $\mathcal{O}^{\mathsf{Eval}}$, $\mathcal{O}^{\mathsf{Pre}}$ and $\mathcal{O}^{\mathsf{Sess}}$, so that a single hypothesis on the evaluation layer covers the concurrent sessions in both.

Theorem 3 (Unforgeability and Pseudorandomness). *If the evaluation layer of (R)Icy-DVRF is concurrently secure in the sense of Definition 11, then (R)Icy-DVRF satisfies the unforgeability and the pseudorandomness of Definition 10, against an adversary corrupting at most* $t-1$ *participants and, in addition, controlling the combiner and the delivery schedule of the network. The reduction is perfect and independent of the number of sessions the adversary opens, i.e., the coordination layer adds no adversarial power beyond that captured by Definition 11.*

Proof (Proof Sketch). Let $\mathcal{A}$ corrupt at most $t-1$ participants and, in addition, play the combiner and schedule the network. The reduction $\mathcal{B}$ plays $\mathsf{Game}^{\mathsf{conc}}$ of Definition 11 and runs $\mathcal{A}$, relaying KeyGen between $\mathcal{A}$ and the challenger and answering for each honest P_i as follows: its first message $(\rho_i, \gamma_i, \pi_i)$ on m is assembled from $\mathcal{O}^{\mathsf{Pre}}(i, m)$ and $\mathcal{O}^{\mathsf{Eval}}(i, m)$, and its reply to a list (B, m) delivered by $\mathcal{A}$ is $\mathcal{O}^{\mathsf{Sess}}(i, m, B)$. Finally $\mathcal{B}$ forwards the output, respectively the guess, of $\mathcal{A}$.

Three observations show that the simulation is perfect. *(i) Honest messages.* Every message an honest participant sends is the output of PreRound, PartialEval or step 5 on inputs chosen by $\mathcal{A}$, which is exactly what the three oracles compute, and $\mathcal{O}^{\mathsf{Sess}}$ enforces the nonce state machine exactly as P_i does. *(ii) Reuse of* (γ_i, π_i). An honest P_i computes the randomised proof π_i once per input and sends this single copy, which the combiner caches, so no session causes a second proof on (i, m), and $\mathcal{O}^{\mathsf{Eval}}$ returns exactly this copy at every query on (i, m). The view of $\mathcal{A}$ thus contains one proof per pair (i, m), both in the protocol and in the game, so the simulation never has to replace a repeated proof $(\pi_i, \ldots, \pi_i)$ by independent proofs $(\pi_{i,1}, \ldots, \pi_{i,\ell})$, and no hybrid over proofs is needed. The reuse is part of the hypothesis of Definition 11 rather than a step of the argument, and Remark 2 explains why a single copy is harmless. *(iii) Coordination.* The responsive set W, the malicious list, the session counter and the committees are local to the combiner and are carried out by $\mathcal{A}$ itself, which may open any polynomial number q_S of sessions on the same input and deliver, drop or reorder messages at will. Lemma 1, which concerns an honest combiner, is not used here.

Hence the view of $\mathcal{A}$ is identical to its view in a real execution and the two counters $Q(m)$ coincide, so $\mathcal{B}$ wins whenever $\mathcal{A}$ does, with no loss and independently of q_S. Unforgeability and pseudorandomness of Definition 10 follow from the corresponding games of Definition 11. □

Remark 1 (Early determination of β). Since PartialEval is session independent, any participant holding t of the shares γ_i for an input m, in particular C, obtains $\beta = H_3(\gamma)$ one round before an accepting proof Π exists. This is why Definition 10 quantifies unforgeability and pseudorandomness over fresh inputs (an initiated session already yields $Q(m) = t$). This grants no bias, since $\gamma = h^s$ is fixed by m and Y alone. It does mean β is not confidential from the combiner, so deployments requiring this should adopt the output-private variant of [3], which is out of the scope of this work.

Remark 2. [2] analyses a single evaluation session per input. There, the h-basis twins are released only after the session committee is fixed, (γ_i, π_i) is recomputed per session, and the FROST-style DKG is used rather than PedPoP. Thus, Definition 11 is strictly stronger than the one established in [2]. We state it as an assumption on the evaluation layer and do not claim to derive it from the single-session analysis of [2]. It is, however, the natural analogue of the hypothesis under which the ROAST wrapper itself is analysed [29], and its signing-layer analogue is what [9,22] establish for the FROST layer underlying Icy-DVRF, under the Schnorr knowledge-of-exponent and OMDL assumptions. Three observations support it. First, the single copy of π_i that Definition 11 hands out is a Fiat–Shamir compiled Chaum–Pedersen proof, simulatable from (g, h, Y_i, γ_i) alone, so it carries no information about s_i beyond the statement $\gamma_i = h^{s_i}$, a deterministic function of (m, s_i) that any session on m reveals anyway and that is therefore counted in $Q(m)$. Second, the concurrent sessions of the coordination layer all run on inputs with $Q(m) \geq t$, whose output is already fixed by Theorem 2. What they must not do is help the adversary on a fresh input m^*, and that

is precisely what $\mathsf{Game}^{\mathsf{conc}}$ forbids. Finally, by Lemma 2 each honest response consumes a fresh nonce pair, and two sessions with $B \neq B'$ carry distinct binding factors $b_l = H_2^{\mathsf{bind}}(l, m, B)$, which is the situation two-round Schnorr-type threshold schemes must handle [13,22] and exactly the situation $\mathcal{O}^{\mathsf{Sess}}$ models.

Proposition 1 (ROAST preconditions). *(R)Icy-DVRF, as specified in Section 3.3, is semi-interactive in the sense of Section 3.2, with the presignature share bound to the input, provides identifiable cancellations, and admits no replay of a partial response across two sessions.*

Proof (Proof Sketch). ***Semi-interactivity.*** Since at preprocessing time, by the assumptions of Section 3.1, $h = H_1(m)$ is available, PreRound produces h-basis commitments and PartialEval generates (γ_i, π_i). Therefore, the values R, $\bar{R}$, γ, z_i, and c calculated in step 5 of PartialProofGen, originate only from state_i and (B, m) from C. No participant waits for another participant's message, and step (j) of PartialProofGen sends a fresh ρ_i' for the next session in the same message along with the reply. The presignature is message-dependent, unlike ROAST's, but each session still costs a single round-trip, which is the property ROAST's analysis uses. ***Identifiable cancellations.*** If a participant submits a corrupted/invalid γ_l, this will be caught in the DLEQ verification in PreAggregation. $\bar{R}_l = \bar{D}_l \bar{E}_l^{b_l}$ is calculated only with $(\bar{D}_l, \bar{E}_l)$ sent by P_l itself. Thus, if a corrupted $(\bar{D}_l, \bar{E}_l)$ is sent, this is caught by the $h^{z_l} = \bar{R}_l \gamma_l^{c\lambda_l}$ check in ShareVal. A malformed z_l is caught by $g^{z_l} = R_l Y_l^{c\lambda_l}$, while the mismatched challenge c is caught in step (b) at item (i). Because the values that C uses to calculate c are exactly the same as the values used by an honest participant, no check can unfairly exclude an honest participant, so acceptance is a deterministic, publicly computable predicate that every honest share satisfies, and every failure is attributable to exactly one l with no computational assumption. ***No replay.*** Lemma 2. □

Theorem 4 (Robustness). *(R)Icy-DVRF is $(n, t, n-t)$-robust in the sense of Definition 10 under Proposition 1.*

Proof (Proof Sketch). By Proposition 1 the coordination logic of [29] applies unchanged, and we give the termination argument directly. Let C be honest and let $H = [n] \setminus F$ be the set of honest participants, so $|H| = n - f \geq t$, as $f \leq n - t$. By Proposition 1 no member of H is ever marked malicious or removed by a cancellation, and since C sends only well-formed lists and messages between honest parties are eventually delivered, every member of H eventually replies to each session it is assigned to and returns to W.

Suppose, for contradiction, that C never outputs. A session that never completes contains a member that never replies or whose share is rejected. This member lies in F and, by the pending discipline, blocks that session alone, so at most $f \leq n - t$ sessions are blocked. Likewise each cancellation in PreAggregation removes a member of F for good, so at most $n - t$ cancellations occur. After the last one, every non-pending member of H eventually re-enters W, so $|W|$ reaches t and a further session is initiated. Repeating this, the number of

initiated sessions reaches the bound $n-t+1$ of Lemma 1, and as at most $n-t$ of them are blocked, at least one of them receives t shares accepted by ShareVal, a contradiction. This session need not consist exclusively of honest members.

It remains to see that the output is accepted. Multiplying the accepted equations of step (b) item (ii) of ShareVal over $l \in S$ and using $\prod_{l\in S} Y_l^{\lambda_l} = Y$ gives $g^z = RY^c$ and $h^z = \bar{R}\gamma^c$, whence $T = R$ and $\bar{T} = \bar{R}$. Since C computed $c = H_2^{\mathsf{chal}}(g, h, Y, \gamma, R, \bar{R}, m)$ and $\beta = H_3(\gamma)$, both checks of Verify succeed unconditionally. As every γ_l in B passed $\mathsf{VerifyEq}$ in PreAggregation, the soundness of the DLEQ proof system gives $\gamma = h^s$ except with negligible probability, so the accepted output is the unique DVRF value of m in the sense of Theorem 2. The argument uses only Proposition 1, Lemma 1 and the soundness of the DLEQ proof system, and in particular does not rely on the concurrent-security hypothesis of Definition 11. □

Lemma 3 (Round complexity). *Under the assumptions of Section 3.1, an honest combiner C outputs (β, Π) after at most $2(n-t)+3$ asynchronous rounds on the critical path, which is $O(n)$ and, under the threshold constraint of Section 3.1, at most $n+2$ rounds.*

Proof (Proof Sketch). In the round notion of Section 3.1, every delivered message arrives within one round of being sent, so the initial messages of the at least t honest participants reach C by time 1 and the first session is initiated at a time $\tau_1 \leq 1$. A cancellation is local to C, happens before B is sent and, by Proposition 1, never removes an honest participant, so it adds no round to the critical path, consumes no session index and returns its verified members to W immediately. Let session j be initiated at time τ_j and suppose no output has been produced by $\tau_j + 2$. Every honest participant either belongs to a session initiated by τ_j, whose reply reaches C by $\tau_j + 2$, or had its initial message delivered by time $1 \leq \tau_j$. Hence, by $\tau_j + 2$ every honest participant not yet assigned to session $j + 1$ has returned to W, and as there are at least t honest participants, C initiates session $j + 1$, so $\tau_{j+1} \leq \tau_j + 2$. By Lemma 1 no session of index $n - t + 2$ is ever initiated, and by Theorem 4 (proved independently above, without Definition 11) some initiated session completes, so C outputs at the latest by $\tau_{n-t+1} + 2 \leq 1 + 2(n - t) + 2 = 2(n - t) + 3$, matching the bound of [29]. Under the threshold constraint of Section 3.1, $n - t \leq (n - 1)/2$, so this is at most $n + 2$ rounds. □

3.5 Resistance to Malicious Combiners

As an active session coordinator, combiner C's hostile actions are limited to temporary service delays (denial of service). Therefore, it can delay the protocol but cannot produce or bias its output. We formalize this resilience through the following features:

- **Unforgeability:** The combiner C does not hold private key shares or long-lived secret material, all it holds are session logs, which are publicly available.

Therefore, it is impossible for a compromised or malicious combiner to generate valid proof Π, or to manipulate $\beta = H_3(\gamma)$. By Remark 1, the combiner may learn β before the proof exists, but since $\gamma = h^s$ is fixed by m and Y it cannot choose among several outputs, as its only power is to delay. This is covered formally by Theorem 3, whose adversary is itself the combiner.

- **Attribution of Misconduct:** PreAggregation and ShareVal evaluate all participant shares (z_i, c) and DLEQ proof π_i. According to Proposition 1, an honest participant will not fail these subroutines. Therefore, a malicious C cannot cause a false outcome to be accepted or fabricate proof to convince a third party against an honest participant, and by Theorem 2 it cannot have a different output accepted. However, it can silently drop valid shares. This type of behaviour is a liveness concern and is addressed by the redundancy discussed below rather than being detected.
- **Malformed session messages.** A malicious combiner can send a list B that fails the checks of step $5(a)$. An honest participant then aborts without replying and is indistinguishable from an unresponsive one. This costs liveness only, because the checks of step (a) precede the nonce binding of step (b), so the participant's nonce pair remains FREE and can be used in a later, well-formed session. A list that is well formed for P_i but corrupts the entry of another member $l \in S$ is a separate case, since step (a) inspects only P_i's own entry, the size of S and the plaintext. Here P_i does bind and consume its nonce pair on a session that cannot complete, because P_l rejects the same list at its own step (a) and never replies. The cost is again liveness only, and it is bounded. By step (i) the participant immediately generates a fresh pair and piggybacks it on its reply, so it is available to the next session, and by Lemma 2 the consumed pair is never reused. No partial proof on a well-formed list is lost, and the robustness statement of Theorem 4 is unaffected, since it is stated for an honest combiner.
- **Semi-trusted coordination, redundancy, and the scope of Theorem 4.** The robustness guarantee of Theorem 4 is stated with respect to an *honest* combiner, as it is in ROAST [29]: a combiner that crashes or withholds valid shares can stall the protocol without being detected, since withholding is indistinguishable from network delay under an asynchronous scheduler. Accordingly, C is trusted only for liveness, never for unforgeability. Robustness against a faulty combiner is recovered by redundancy. Either an alternative combiner is invoked, or an honest participant takes over, or the protocol is run in a fully leaderless mode in which every participant simultaneously runs a local combiner instance and the output is produced by the first instance to collect t valid partial proofs. The last option eliminates the single point of failure entirely, at the cost of an n-fold increase in communication [29].

4 Comparison

Our quantitative comparison is restricted to the Schnorr-based, non-pairing DVRF structure in this section, since pairing-based and privacy-focused struc-

tures are based on different assumptions and are not directly comparable in terms of the communication complexity and robustness properties discussed here. Therefore, we evaluate the theoretical performance of our proposed (R)Icy-DVRF protocol by comparing it directly to the DVRFwCP [1] and its architectural predecessor, the round-optimized Icy-DVRF [2]. We focus on exact computational operations within comparable subroutines, communication overhead, proof size, and robustness. Table 1 details the computational complexity of each phase by counting the exact group exponentiations, multiplications, and hashing operations required per session.

We begin our comparison from the operations in the first row, which includes the KeyGen phase. Compared to DVRFwCP, (R)Icy-DVRF cuts the heaviest computational costs, i.e., exponentiations and multiplications, almost in half, reducing them from $2nt$ to nt. Compared to Icy-DVRF, the increase of 2 in exponentiations, 1 in multiplications, and 1 in hash operations is due to the use of the PedPoP protocol during the setup phase. It is worth noting that this is a one-time and negligible cost, paid before the DVRF production phase, in exchange for rogue-key-secure key generation.

In Table 1, the PreRound/Preprocess row is empty for DVRFwCP because that scheme uses no preprocessing. Icy-DVRF generates a pool of Ω nonce pairs at a cost of 2Ω exponentiations, that is, two exponentiations per session, and must store the pool until it is consumed.

(R)Icy-DVRF spends four exponentiations and one hash to $\mathbb{G}$ per session, two of them for the h-basis twin, but keeps no pool, exactly one nonce pair is live at any time and its replacement is piggybacked on the reply of the current session. The trade-off is therefore two additional exponentiations per session in exchange for constant participant state, the removal of the pool-exhaustion failure mode, and the elimination of the shared index j that the combiner and the participants of Icy-DVRF must agree upon. The further cost of the twin is that preprocessing becomes message-dependent and cannot be performed before the input is known.

In the PartialEval subroutine, (R)Icy-DVRF has 2 fewer exponentiations and 1 fewer hash to $\mathbb{G}$ compared to Icy-DVRF. However, this difference actually stems from moving the twin generation to the PreRound subroutine. The main gain here is amortization. Since the partial evaluation is session independent, the 3 exponentiations are performed only once, regardless of how many sessions are run.

It is worth noting that while the next subroutine, PreAggregation, does not appear as a separate step, its function is not entirely new. The $4t$ exponentiations come from the DLEQ verification carried over from the aggregation phase performed by Icy-DVRF, and the $2t$ multiplications come from the same check. The single hash to $\mathbb{G}$ is the computation of $h = H_1(m)$, performed once per plaintext by C, and the t hashes to $\mathbb{Z}_q$ are the challenges recomputed inside VerifyEq, while the binding values b_l are counted in the ShareVal row.

The EvalCombine subroutine is unique to DVRFwCP. This subroutine does not occur in the Icy-DVRF and our designed (R)Icy-DVRF schemes. We notice

that this subroutine costs DVRFwCP an additional $5t$ exponentiations, $3t-1$ multiplications, and 1 hash to $\mathbb{Z}_q$.

In the PartialProofGen row, our protocol eliminates the quadratic increase of the $3nt$ computational overhead found in DVRFwCP, just like Icy-DVRF. Furthermore, compared to Icy-DVRF, this subroutine is arithmetically identical. It has the same t binding values, the same R and $\bar{R}$ calculations, and the same γ and z_i calculations. The ROAST layer adds no local arithmetic here. Therefore, robustness costs nothing extra. Since the commitment for the next session generated at the end of this subroutine is accounted for in the PreRound, it is not counted twice.

The most computationally intensive operations occur in the Combine/ShareVal–ProofAgg phase. This row shows three improvements compared to Icy-DVRF: exponentiations decrease from $11t$ to $7t$, multiplications from $9t-3$ to $7t-3$, and hashes to $\mathbb{Z}_q$ from $2t+1$ to $t+1$. The hash to $\mathbb{G}$ also leaves this row, but it does not disappear, as it moves to PreAggregation, where C computes $h = H_1(m)$ once per plaintext. All of these shifts follow from moving the $4t$ exponentiation portion of the DLEQ verification into PreAggregation. The comparison is most meaningful at the level of the whole combiner-side computation, adding PreAggregation back gives $4t+7t = 11t$ exponentiations and $2t+7t-3 = 9t-3$ multiplications, exactly the counts of the Combine subroutine of Icy-DVRF. Robustness is therefore achieved without any additional local computation, but its real cost lies in structural complexity, namely the worst-case round count, the session state maintained by C, and the early detection performed in PreAggregation.

The Verify row shows the cost of verifying the final DVRF proof generated by third-party verifiers. As can be seen in Table 1, one of the greatest achievements of (R)Icy-DVRF is that it generates the DVRF output and verifiable proof fully asynchronously, without imposing any additional cost on the outside world to verify the final proof.

Finally, we explain the **Synchrony**, **Interactions**, **Proof size**, **Verification cost**, **Robustness**, **Messages**, **Message size** and **Total communication** rows. (R)Icy-DVRF possesses robustness that Icy-DVRF lacks, and it achieves this under a fully asynchronous network, unlike the synchronous network of DVRFwCP. Under realistic network conditions, adversarial nodes can intentionally delay their messages or become unresponsive, which causes protocols relying on timeouts or on the cooperation of a fixed signer set to lose liveness. (R)Icy-DVRF uses no timeouts and, with an honest combiner and at least t honest and responsive participants, terminates under any asynchronous scheduler (Theorem 4). The Interactions row shows the price of this guarantee. A fresh input costs 3 asynchronous rounds and each further session on the same input 2, while in the worst case, where up to $n-t$ sessions fail before one succeeds, the combiner needs $2(n-t)+3$ rounds, which is at most $n+2$ under the threshold constraint (Lemma 3). Since the first two columns count synchronous broadcast rounds, these figures are not directly comparable.

The Proof size and Verification cost rows show that none of this cost reaches the verifier. Despite the transition to a fully asynchronous model, the combiner acting as a session manager and the larger number of rounds, the final proof still consists of 1 $\mathbb{G}$ element and 2 $\mathbb{Z}_q$ elements and is verified with 4 exponentiations and 2 multiplications, as in the previous schemes, independently of n and t. The protocol therefore keeps the compactness required in space-constrained settings such as blockchain data structures and smart contracts.

The true cost of robustness lies in communication. When participants are honest and responsive, a fresh input needs $n + 2t = O(n)$ messages, asymptotically the same as DVRFwCP and Icy-DVRF, and each further session on the same input only $2t$. In the worst case, since at most $n - t + 1$ sessions are attempted, this grows to $n+2t(n-t+1) = O(n^2)$. The largest single message has $O(t)$ group elements, as in the other two schemes, and the total communication is analysed below. In other words, the trade-off for robustness lies in communication, not in local computation.

Table 1: Mathematical operations of DVRFwCP [1], Icy-DVRF [2] and (R)Icy-DVRF

	DVRFwCP	Icy-DVRF	(R)Icy-DVRF
KeyGen	$2nt + 3n + 2t$ exp. $2nt + t - 1$ mul.	$nt + 3n - 1$ exp. $nt + n - t - 1$ mul. n hash to $\mathbb{Z}_q$	$nt + 3n + 1$ exp. $nt + n - t$ mul. $n + 1$ hash to $\mathbb{Z}_q$
Preprocess/ PreRound[a]	–	2Ω exp.	4 exp. 1 hash to $\mathbb{G}$
PartialEval[b]	3 exp. 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$	5 exp. 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$	3 exp. 1 hash to $\mathbb{Z}_q$
PreAggregation[c]		–	$4t$ exp. $2t$ mul. 1 hash to $\mathbb{G}$ t hash to $\mathbb{Z}_q$
EvalCombine[d]	$5t$ exp. $3t - 1$ mul. 1 hash to $\mathbb{Z}_q$	–	–
PartialProofGen	$3nt + 4n + 3t$ exp. $3nt + t - 2$ mul. 1 hash to $\mathbb{Z}_q$	$3t$ exp. $5t - 3$ mul. $t + 1$ hash to $\mathbb{Z}_q$	$3t$ exp. $5t - 3$ mul. $t + 1$ hash to $\mathbb{Z}_q$
Combine[e] / ShareVal- ProofAgg	$6t$ exp. $2t$ mul. 1 hash to $\{0,1\}^\Lambda$	$11t$ exp. $9t - 3$ mul. 1 hash to $\mathbb{G}$ 1 hash to $\{0,1\}^\Lambda$ $2t + 1$ hash to $\mathbb{Z}_q$	$7t$ exp. $7t - 3$ mul. 1 hash to $\{0,1\}^\Lambda$ $t + 1$ hash to $\mathbb{Z}_q$

Table 1 (continued)

	DVRFwCP	Icy-DVRF	(R)Icy-DVRF
Verify	4 exp. 2 mul. 1 hash to $\mathbb{G}$ 2 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^\Lambda$	4 exp. 2 mul. 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^\Lambda$	4 exp. 2 mul. 1 hash to $\mathbb{G}$ 1 hash to $\mathbb{Z}_q$ 1 hash to $\{0,1\}^\Lambda$
Robustness	Robust	Not robust	**Robust**
Synchrony	Fully synchronous	Semi-synchronous	**Asynchronous**
Interactions[f),(g)] (#rounds)	2	1	3[h)] (fresh input) 2 (further sessions) $2(n-t)+3 \leq n+2$ (worst case)
Proof size	$1\,\mathbb{G} + 2\,\mathbb{Z}_q$	$1\,\mathbb{G} + 2\,\mathbb{Z}_q$	$1\,\mathbb{G} + 2\,\mathbb{Z}_q$
Verification cost	4 exp., 2 mul.	4 exp., 2 mul.	4 exp., 2 mul. (independent of n and t)
Messages	$O(n)$	$O(n)$	$n + 2t = O(n)$ (fresh input) $n + 2t(n-t+1) = O(n^2)$ (worst case)
Message size[i)]	$O(t)$ group elements	$O(t)$ group elements	$O(t)$ group elements
Total communication[j)]	$O(nt)$	$O(n+t^2)$	$O(n+t^2)$ (fault-free) $O(n+t^2(n-t+1))$ (worst case)

Rows PreRound and PartialProofGen are counted per participant per session, PartialEval is counted per participant per plaintext, PreAggregation and ShareVal-ProofAgg are counted per session at the combiner, and KeyGen is counted per participant over the whole one-time key-generation protocol, while Verify is counted per external verifier per proof. Operation counts for Icy-DVRF are recomputed here under this convention and differ from [2, Table 1] in the PartialProofGen, Combine and Verify rows, which omitted the per-participant binding-value hashes and the h-basis multiplications.

[a)] For (R)Icy-DVRF this subroutine is re-executed once per session, at the end of PartialProofGen, to obtain the share used by the next session.

[b)] For (R)Icy-DVRF the partial evaluation is session independent and is computed once per plaintext m, and the hash to $\mathbb{G}$ is accounted for in the PreRound row.

[c)] The DLEQ verification counted here is paid once per participant per plaintext. Later sessions only re-assemble B, and the per-session aggregation of $R,\bar{R}$, γ and c is counted in the ShareVal-ProofAgg row.

[d)] This subroutine is not present in the other two schemes.

[e] The corresponding algorithm is named ProofCombine in DVRFwCP.
[f] The rounds of the one-time distributed key generation are excluded.
[g] Round metrics are not directly comparable, since the first two columns count synchronous broadcast rounds while (R)Icy-DVRF counts asynchronous rounds in the sense of Section 3.1
[h] Commitment shares are bound to m through $h = H_1(m)$ and cannot be precomputed, so a fresh input always incurs one preprocessing round, and the 2-round figure is the marginal cost of each further session on the same input.
[i] Largest single message. DVRFwCP: no combiner, each participant broadcasts $2t$ group elements in PartialProofGen. Icy-DVRF: as in FROST, the combiner sends the commitment list of the t signers, i.e. $O(t)$ group elements, to each signer, and each signer replies with $3|\mathbb{G}| + 2|\mathbb{Z}_q|$. (R)Icy-DVRF: C's list B carries $O(t)$ group elements, while a participant's first message per plaintext carries $5|\mathbb{G}| + 2|\mathbb{Z}_q|$ and later ones $4|\mathbb{G}| + 2|\mathbb{Z}_q|$, namely $\Pi_i = (z_i, c)$ and ρ'_i.
[j] Total number of group elements and scalars sent over a whole execution on one input, excluding key generation. A broadcast is counted once.

Total Communication. A fresh input costs $n(5|\mathbb{G}| + 2|\mathbb{Z}_q|)$ for the initial messages, and each session $5t^2|\mathbb{G}|$ for sending B to its t members plus $t(4|\mathbb{G}| + 2|\mathbb{Z}_q|)$ for the replies, giving $O(n + t^2)$ group elements without faults and, by Lemma 1, $O(n+t^2(n-t+1))$ in the worst case. The quadratic term is inherited from ROAST, where the coordinator likewise sends the committee to each of its t members, and is not specific to the DVRF setting. Table 2 instantiates these expressions for several (n, t), in the fault-free and in the worst case, to quantify the practical overhead of the robustness mechanism.

Concrete figures. Table 2 uses $|\mathbb{G}| = 33$ and $|\mathbb{Z}_q| = 32$ bytes (compressed points of a 256-bit prime-order curve) and $t = \lceil (n+1)/2 \rceil$, the smallest threshold allowed by the threshold constraint of Section 3.1. A fresh input costs n initial messages of $5|\mathbb{G}| + 2|\mathbb{Z}_q|$, and each session $5t^2|\mathbb{G}|$ for B plus $t(4|\mathbb{G}| + 2|\mathbb{Z}_q|)$ for the replies. The worst case has $n - t$ faulty participants, each blocking one session, and attains the bounds of Lemma 1 and Lemma 3. More generally, an execution with $f \leq n - t$ unresponsive participants takes at most $f + 1$ sessions, hence at most $2f + 3$ rounds, $n + 2t(f + 1)$ messages and $n(5|\mathbb{G}| + 2|\mathbb{Z}_q|) + (f + 1)\big(5t^2|\mathbb{G}| + t(4|\mathbb{G}| + 2|\mathbb{Z}_q|)\big)$ bytes. The two blocks of Table 2 are the endpoints $f = 0$ and $f = n-t$, and every intermediate fraction of unresponsive participants interpolates linearly between them.

Table 2: Analytic communication and latency of one execution of (R)Icy-DVRF on a fresh input, excluding key generation

		Fault-free			Worst case			
n	t	msgs	rounds	KiB	faults	msgs	rounds	KiB
3	2	7	3	1.7	1	11	5	2.7
5	3	11	3	3.1	2	23	7	7.2
7	4	15	3	4.9	3	39	9	14.9
11	6	23	3	9.4	5	83	13	44.2
21	11	43	3	26.3	10	263	23	242.3
51	26	103	3	125.3	25	1403	53	2972.9
101	51	203	3	451.5	50	5303	103	21894.9

In the fault-free case an execution takes three asynchronous rounds and a few tens of kilobytes for committees of up to about twenty participants, and the dominant term is the list B, which grows as $5t^2|\mathbb{G}|$ per session. The worst case multiplies the per-session cost by the $n - t + 1$ sessions of Lemma 1. It is reached only when every faulty participant blocks exactly one session, and its cubic growth in n is the price of robustness without timeouts, inherited from ROAST [29]. The proof size and the third-party verification cost are the same in all cases. An implementation measuring wall-clock latency under realistic delay distributions is left for future work (Section 5).

5 Conclusion

In this paper, we present (R)Icy-DVRF, a novel protocol that integrates the asynchronous coordination layer of ROAST [29] and the PedPoP key generation protocol [9] into the FROST-based Icy-DVRF architecture [2]. By executing up to $n - t + 1$ parallel asynchronous sessions, our design guarantees full robustness and liveness against up to $n - t$ malicious or unresponsive participants, for an honest combiner or in the leaderless mode of Section 3.5, without relying on network timeouts or protocol aborts. Furthermore, the combiner collects future commitment shares during active rounds, preventing execution stalls and eliminating the need for communication restarts. The price of binding the preprocessing to the input is that a fresh plaintext always costs one preprocessing round, as only subsequent sessions on the same input are pipelined.

As demonstrated by our theoretical analysis, (R)Icy-DVRF provides a highly favorable balance between local computation, round complexity, and communication overhead. Compared to DVRFwCP [1], it substantially reduces the quadratic overheads in the key generation, and entirely eliminates the n -dependent term in partial proof generation phases and removes the interactive EvalCombine step entirely. Compared to our predecessor Icy-DVRF, switching to an asynchronous model does not lead to an increase in local verification operations. It simply involves moving the DLEQ verification that Icy-DVRF performs in the Combine phase to PreAggregation (before the session starts) in our design. The combined number of operations in PreAggregation, ShareVal, and ProofAgg is the same as Icy-DVRF's Combine phase. The benefit of this early movement is gaining the identifiable cancellations feature before the session starts without any additional cost. Moreover, this is complemented by a preprocessing layer that keeps participant state constant: a single live nonce pair, refreshed at a cost of four exponentiations per session, replaces Icy-DVRF's pool of Ω precomputed pairs. Crucially, despite tolerating active background disruptions within a worst-case bound of $2(n - t) + 3 \leq n + 2$ asynchronous rounds, the final proof size remains constant and compact, comprising exactly 1 $\mathbb{G}$ and 2 $\mathbb{Z}_q$ elements.

Future work will include an implementation measuring latency and total communication for several (n, t) and fractions of unresponsive participants, compared with Icy-DVRF in the fault-free case, and expanding this framework to

incorporate output privacy, creating a fully robust, asynchronous, and private distributed randomness architecture.

References

1. Ağırtaş, A.R., Özer, A.B., Saygı, Z., Yayla, O.: Distributed verifiable random function with compact proof. In: Dolev, S., Elhadad, M., Kutyłowski, M., Persiano, G. (eds.) Cyber Security, Cryptology, and Machine Learning. pp. 119–134. Springer Nature Switzerland, Cham (2025)
2. Ağırtaş, A.R., Özer, A.B., Saygı, Z., Yayla, O.: Icy-DVRF: A distributed verifiable random function based on FROST signatures. IEEE Access **14**, 105882–105909 (2026). https://doi.org/10.1109/ACCESS.2026.3712805
3. Ağırtaş, A.R., Özer, A.B., Saygı, Z., Yayla, O.: IcyVeil: An output-private DVRF based on Icy-DVRF. Cryptology ePrint Archive, Paper 2026/1138 (2026), https://eprint.iacr.org/2026/1138
4. Alonso, A., Au, M., Baftiu, D., Bhatia, N., Chaliasos, S., et al.: The Sui smart contracts platform: Economics and incentives. Mysten Labs Whitepaper (2022)
5. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Mina: Decentralized cryptocurrency at scale. O(1) Labs Whitepaper (2020)
6. Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., et al.: Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. Chainlink Labs Whitepaper (2021)
7. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Advances in Cryptology – CRYPTO '92. pp. 89–105. Springer (1992)
8. Ching, A., Shaikh, M., et al.: The Aptos blockchain: Safe, scalable, and upgradeable Web3 infrastructure. Aptos Labs Whitepaper (2022)
9. 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
10. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Advances in Cryptology – EUROCRYPT 2018. pp. 66–98. Springer (2018)
11. Dodis, Y.: Efficient construction of (distributed) verifiable random functions. In: Desmedt, Y.G. (ed.) Public Key Cryptography — PKC 2003. Lecture Notes in Computer Science, vol. 2567, pp. 1–17. Springer, Heidelberg (2003)
12. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) Public Key Cryptography — PKC 2005. Lecture Notes in Computer Science, vol. 3431, pp. 416–431. Springer, Heidelberg (2005)
13. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the security of two-round multi-signatures. Cryptology ePrint Archive, Paper 2018/417 (2018), https://eprint.iacr.org/2018/417
14. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: FOCS 1987. pp. 427–438. IEEE (1987)
15. Galindo, D., Liu, J., Ordean, M., Wong, J.M.: Fully distributed verifiable random functions and their application to decentralised random beacons. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 88–102. IEEE (2021)
16. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. In: Advances in Cryptology – EUROCRYPT '99. pp. 295–310. Springer (1999)

17. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling Byzantine agreements for cryptocurrencies. In: SOSP 2017. pp. 51–68 (2017)
18. Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., Ziv, A.: NSEC5: Provably preventing DNSSEC zone enumeration. Cryptology ePrint Archive, Paper 2014/582 (2014), https://eprint.iacr.org/2014/582
19. Goldberg, S., Reyzin, L., Papadopoulos, D., Včelák, J.: Verifiable random functions (VRFs). RFC 9381, Internet Engineering Task Force (2023). https://doi.org/10.17487/RFC9381
20. Hanke, T., Movahedi, M., Williams, D.: DFINITY technology overview series, consensus system. arXiv preprint arXiv:1805.04548 (2018)
21. Kate, A., Mangipudi, E.V., Maradana, S., Mukherjee, P.: FlexiRand: Output private (distributed) VRFs and application to blockchains. In: ACM CCS 2023. pp. 1776–1790 (2023)
22. Komlo, C., Goldberg, I.: FROST: Flexible round-optimized Schnorr threshold signatures. In: SAC 2020. pp. 34–65. Springer (2020)
23. Kuchta, V., Manulis, M.: Unique aggregate signatures with applications to distributed verifiable random functions. In: CANS 2013. pp. 251–270. Springer (2013)
24. Liu, J., Manulis, M.: Fast SNARK-based non-interactive distributed verifiable random function with Ethereum compatibility. In: ACM AsiaCCS 2025. pp. 807–822 (2025)
25. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science (FOCS). pp. 120–130. IEEE (1999)
26. Papadopoulos, D., Wessels, D., Huque, S., Naor, M., Včelák, J., Reyzin, L., Goldberg, S.: Making NSEC5 practical for DNSSEC. Cryptology ePrint Archive, Paper 2017/099 (2017), https://eprint.iacr.org/2017/099
27. Patel, S., Persiano, G., Seo, J.Y., Yeo, K.: Efficient secret sharing for large-scale applications. In: ACM CCS 2024. pp. 3065–3079 (2024)
28. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Advances in Cryptology – EUROCRYPT '91. pp. 522–526. Springer (1991)
29. Ruffing, T., Ronge, V., Jin, E., Schneider-Bensch, J., Schröder, D.: ROAST: Robust asynchronous Schnorr threshold signatures. In: ACM CCS 2022. pp. 2551–2564 (2022)
30. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
31. Wood, G.: Polkadot: Vision for a heterogeneous multi-chain framework. White paper (2016)

Algorithm 1 ROAST Protocol [29]

procedure $\mathcal{C}(\mathrm{PK}, n, t, m)$
  $W \leftarrow \emptyset$ ▷ $\mathcal{S}_i$ is responsive if $i \in W$
  $M \leftarrow \emptyset$ ▷ $\mathcal{S}_i$ is known to be malicious if $i \in M$
  $P[\sim] \leftarrow \mathbf{array}(n)$ ▷ $P[i]$ is the latest presignature share of $\mathcal{S}_i$
  $sidctr \leftarrow 0$ ▷ Session counter
  $SID[\sim] \leftarrow \mathbf{array}(n)$ ▷ $SID[i]$ is the session that includes $\mathcal{S}_i$
  $T[\sim] \leftarrow \mathbf{array}(n-t+1)$ ▷ $T[sid]$ is the set of signer indices of session sid
  $N[\sim] \leftarrow \mathbf{array}(n-t+1)$ ▷ $N[sid]$ is the presignature of session sid
  $S[\sim] \leftarrow \mathbf{array}(n-t+1)$ ▷ $S[sid]$ is the set of signature shares for session sid

  upon receive (σ_i, ρ'_i) **from** $\mathcal{S}_i, i \notin M$ **do**
    if $i \in R$ **then**
      MarkMalicious(i) ; **break** ▷ Unsolicited reply
    if $SID[i] \neq \bot$ **then** ▷ Unless this is the initial message from $\mathcal{S}_i$
      $sid \leftarrow SID[i]$ ▷ Look up session of $\mathcal{S}_i$
      $T_{sid} \leftarrow T[sid]$ ▷ Look up signers of session sid
      $\rho \leftarrow N[sid]$ ▷ Look up (aggregate) presignature of session sid
      $\rho_i \leftarrow P[i]$ ▷ Look up presignature share of $\mathcal{S}_i$
      if $\neg$ShareVal($\mathrm{PK}, T_{sid}, i, \rho, \rho_i, \sigma_i, m$) **then**
        MarkMalicious(i) ; **break** ▷ Invalid signature share from $\mathcal{S}_i$
      $S[sid] \leftarrow S[sid] \cup \{\sigma_i\}$ ▷ Store valid signature share
      if $|S[sid]| = t$ **then** ▷ If we have t valid signature shares:
        $\sigma \leftarrow$ SignAgg($\mathrm{PK}, \rho, S[sid], m$) ▷ Aggregate them
        return σ ▷ and output the final signature.
    $P[i] \leftarrow \rho'_i$ ▷ Store received presignature share of $\mathcal{S}_i$
    $W \leftarrow W \cup \{i\}$ ▷ Mark $\mathcal{S}_i$ as responsive
    if $|W| = t$ **then** ▷ If we now have t responsive signers:
      $sidctr \leftarrow sidctr + 1$ ▷ Initiate a new session with them
      $\{\rho_i\}_{i \in R} \leftarrow \{P[i]\}_{i \in R}$ ▷ Look up presignature shares
      $\rho \leftarrow$ PreAgg($\mathrm{PK}, \{\rho_i\}_{i \in R}$) ▷ Build the presignature
      for all $i \in W$ **do**
        send (ρ, W) **to** $\mathcal{S}_i$ ▷ Send the presignature to the signers
        $SID[i] \leftarrow sidctr$ ▷ Remember the session of $\mathcal{S}_i$
      $T[sidctr] \leftarrow W$ ▷ Remember the signers
      $N[sidctr] \leftarrow \rho$ ▷ Remember the presignature
      $W \leftarrow \emptyset$ ▷ Mark signers as pending again

procedure MarkMalicious(i)
  $M \leftarrow M \cup \{i\}$
  if $|M| > n - t$ **then**
    fail ▷ Too many malicious signers

Algorithm 2 Signer Protocol [29]

procedure $\mathcal{S}_i(sk_i, \mathrm{PK}, m)$
  $(\rho_i, \mathrm{state}_i) \leftarrow \textsc{PreRound}(\mathrm{PK})$
  send $(\perp, \rho_i)$ **to** $\mathcal{C}$ ▷ Send initial message with presignature share only

  upon receive (ρ, R) **from** $\mathcal{C}$ **do**
    $\sigma_i \leftarrow \textsc{SignRound}(sk_i, \mathrm{PK}, R, \mathrm{state}_i, \rho, m)$
    $(\rho_i, \mathrm{state}_i) \leftarrow \textsc{PreRound}(\mathrm{PK})$
    send (σ_i, ρ_i) **to** $\mathcal{C}$