Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons

David Galindo*†, Jia Liu*, Mihai Ordean†, Jin-Mann Wong*
*Fetch.ai, UK
†University of Birmingham, UK

***Abstract*—We provide the first systematic analysis of two multiparty protocols, namely (Non-Interactive) Fully Distributed Verifiable Random Functions (DVRFs) and Decentralised Random Beacons (DRBs), including their syntax and definition of robustness and privacy properties. We refine current pseudorandomness definitions for distributed functions and show that the privacy provided by *strong pseudorandomness*, where an adversary is allowed to make partial function evaluation queries on the challenge value, is strictly better than that provided by *standard pseudorandomness*, where such adversarial queries are disallowed. We provide two new DVRF instantiations, named DDH-DVRF and GLOW-DVRF, that meet strong pseudorandomness under widely accepted cryptographic assumptions. Additionally we show that a previously sketched DVRF construction that we name Dfinity-DVRF meets the weaker property of standard pseudorandomness. We show the usefulness of our DRB formalism in two different ways. Firstly, we give a rigorous treatment of a folklore generic construction that builds a Decentralized Random Beacon from any DVRF instance and prove that it satisfies robustness and pseudorandomness provided that the original DVRF protocol is secure. Secondly, we capture several existing DRB protocols from academy and industry using our framework, which serves as an evidence of its wider applicability. DRBs have recently gained a lot traction as a key component for leader(s) election in decentralized ledger technologies, and we provide a classification of some of the most prominent. By building on an an obvious connection between Dfinity-DVRF and Threshold BLS signatures we discuss how to transform GLOW-DVRF into a modified Threshold BLS signature scheme with stronger security as an additional contribution of independent value. Finally, we report on experimental evaluations of the three concrete DVRFs studied under several cryptographic libraries, and we also report preliminary benchmark results on two of the DRBs obtained from the generic DVRF-to-DRB transformation. Our findings are that our two new DRB instantiations are strongly pseudorandom and strongly unbiasable, while exhibiting high performance and linear communication complexity (as the random beacon values are computed in a non-interactive fashion). We conclude that our new DRB instantiations are to our knowledge the most efficient instantiations currently available while enjoying strong and formally proven security properties. Some of our benchmarks can be independently verified as we provide an open source C++ reference implementation of the DVRFs discussed in this work.**

***Index Terms*—Cryptography, Blockchain, Random Beacon, Distributed Computation, Implementation, Pseudorandom Functions, Threshold Signatures, Leader Election, Open Source**

I. Introduction

In recent years there has been prolific development in blockchain technologies [50], [62], and a plethora of platforms relying on blockchains have seen the light of day. The various platforms often differ in their design choices, and thus on the consensus protocol they rely on. Many consensus protocols involve allocating the creation of blocks with a block producer, whose selection procedure most often than not [37], [16], [44], [39], [45] requires a method for collective randomness sampling. Some approaches to consensus scalability rely on a sharding technique where access to a randomness source is also required [47], [59]. In order to avoid reliance on a trusted party, a common approach is to use a mechanism that allows the distributed computation of an unpredictable and unbiased source of randomness, verifiably.

a) Verifiable Random Function (VRF): This primitive was introduced by Micali, Rabin and Vadhan [48] and can be seen as the public-key version of a keyed cryptographic hash $F_{\mathsf{sk}}(\cdot)$, where a trusted party evaluates $F_{\mathsf{sk}}(x)$ on inputs x in such a way that the output can publicly be verified for correctness via an auxiliary proof π_x. The secret key sk allows i) evaluation of $F_{\mathsf{sk}}(\cdot)$ on any input x and ii) to compute and provide proof π_x of the correct evaluation $F_{\mathsf{sk}}(x)$. The proof correctness can be verified by means of an algorithm Verify that takes as inputs the public key pk corresponding to sk, the values x, $F_{\mathsf{sk}}(x)$, and proof π_x, and outputs accept or reject.

b) Unique Signatures: There are similarities between a VRF and a digital signature scheme with unique signatures. On inputs a string x and a secret signing key SK_S, a signing algorithm outputs a signature σ_x. If the signature scheme is unforgeable, the latter value σ_x is unpredictable given the public key PK_S. Despite its unpredictability, the signature σ_x on string x is *publicly verifiable*. There are two issues that may prevent signature schemes with unique signatures from being used straightforwardly as a VRF: i) signatures σ_x may not be unique given x and PK_S; and ii) σ_x is unpredictable but *not* pseudorandom (e.g. signatures could contain some bias and be distinguishable from a random distribution). VRFs derived from unique signatures present strong unbiasibility properties due to the uniqueness, even in the presence of active adversaries, of the corresponding pseudorandom value. More generally, the connection between pseudorandom functions with public verifiability and digital signatures is well-known [8], [34].

c) Our contributions.: We provide the first systematic analysis of (Non-Interactive) Fully Distributed Verifiable Random Functions (DVRFs), including their syntax and the definition of their integrity and privacy properties. We extend the

definitions of *standard and strong pseudorandomness* given in [1] in the context of Distributed Pseudorandom Functions with trusted setup and private correctness verification to trustless setups with publicly verifiable correctness. We provide further refinements to previous pseudorandomness definitions in distributed settings by parametrizing the strength of the privacy levels achieved with a triple (θ, t, ℓ), where ℓ is the total number of parties involved in the setup, t is the threshold, and $0 \leq \theta \leq t$ is the actual number of parties under adversarial control. We show that under widely accepted computational assumptions (i.e. DDH assumption) there exist DVRFs that are (θ, t, ℓ)-standard pseudorandom but are not (θ, t, ℓ)-strongly pseudorandom. Roughly speaking, strong pseudo-randomness strengthens the standard pseudorandomness property by allowing an adversary to make partial queries on the target pseudorandom value (Section III). This separation result implies that the privacy provided by strong pseudorandomness is a strictly better than that provided by standard pseudorandomness. In order to capture security against an active attacker that corrupts a subset of $\theta \leq t$ parties while providing public verifiability, we additionally define the properties of robustness and uniqueness.

In Section IV we provide *two new* DVRF *constructions* that achieve strong pseudorandomness, under well-known cryptographic assumptions. One of these constructions is called DDH-DVRF and can be implemented using standard elliptic curve cryptography (Section IV-A), whereas the other construction is named GLOW-DVRF and uses cryptographic pairings (Section IV-B). While the proof of correctness π_x in DDH-DVRF is non-compact, i.e., its size is linear in the threshold t, GLOW-DVRF is able to provide a compact proof, i.e., with constant-size. We additionally validate the security of Dfinity-DVRF (Section IV-C), a prominent DVRF construction in the blockchain space [39], by showing that it meets standard pseudorandomness.

In Section V we present a formalization of Decentralised Random Beacons with a distributed but non-interactive beacon computation. We introduce DRB's security and privacy properties inspired by those of DVRFs, namely strong/standard pseudorandomness, robustness and uniqueness. We show the usefulness of our newly introduced DRB formalism in two different ways. Firstly, we give a rigorous treatment of a folklore generic construction that builds a Decentralized Random Beacon from any DVRF instance and prove that it satisfies robustness and pseudorandomness provided that the original DVRFs are secure. Secondly,we capture several existing DRB protocols from academy and industry using our framework, which serves as an evidence of its wider applicability. DRBs have recently gained a lot traction as a key component for leader(s) election in decentralized ledger technologies, and we provide a classification of some of the most prominent (Section VI).

We provide an experimental evaluation of GLOW-DVRF, DDH-DVRF and Dfinity-DVRF, using the cryptographic libraries MCL [49], RELIC [2] and Libsodium [10]. Our experiments show that both GLOW-DVRF and DDH-DVRF outperform Dfinity-DVRF in running time, approximately by x3 and x5 respectively at the 128-bit security level. We provide a reference implementation in C++ and make it widely available with an open source license [33] (Apache 2.0 license). To further illustrate the superior performance and practicality of our DVRF-based DRB constructions, we have developed a ledger prototype with Tendermint-based state machine replication [16] using Cosmos SDK [41]. In our prototype validator nodes act as DRB nodes, and the block proposer stores in the block each DRB's round random value and corresponding proof using a special transaction. Preliminary benchmarks can be found in Section VII and allow us to conclude that our new DRB instantiations are the most efficient DRB constructions currently available while enjoying strong and formally proven security properties, as the random beacon values are computed in a non-interactive fashion.

In Section VIII we discuss how as a side-effect of our work we obtain a modified BLS signature scheme that is unforgeable in a more powerful attacker model. More concretely, by building on an an obvious connection between Dfinity-DVRF and Threshold BLS signatures, we discuss how to transform GLOW-DVRF into a modified Threshold BLS signature scheme with stronger security as an additional contribution of independent value.

A. Related Work

To our knowledge, the first distributed VRF construction was proposed by Dodis [29] and required the existence of a trusted dealer. The constructions described in this work dispose of this trusted dealer by using a Distributed Key Generation (DKG) sub-protocol: Dfinity-DVRF and DDH-DVRF use a protocol by Gennaro, Jarecki, Krawczyk and Rabin [35], whereas GLOW-DVRF required several modifications to the latter. Manulis and Kuchta [46] proposed a generic construction for interactive distributed VRFs based on unique aggregate signatures in the shared random string model. Compared to our DVRF syntax and new designs, the concrete constructions obtained from [46] are at least two orders of magnitude less efficient than ours, both in running time (as they use pairings and an inefficient generic transformation of pseudorandom functions from unpredictable functions [48]) and in latency (as they involve sequential interaction between a number of peers).

An influential DVRF (used e.g. in [24], [42], [23], [26], [54]) that we name Dfinity-DVRF was informally introduced in [39] with no formal security model nor analysis. We formalize Dfinity-DVRF and prove its security wrt to standard pseudorandomness under the co-CDH assumption in pairing groups in the random oracle model. We discuss how the techniques [11], [35] that have been used to prove standard unforgeability of threshold BLS signatures [15] do not trivially allow to prove strong pseudorandomness. In contrast, our new GLOW-DVRF also uses pairing groups, but achieves strong pseudorandomness in the random oracle model under the co-CDH and eXternal DDH assumptions. Perhaps

surprisingly, GLOW-DVRF shows not only stronger security than Dfinity-DVRF but also possesses better running times.

Our construction DDH-DVRF also achieves strong pseudorandomness but under the standard DDH assumption. The main drawback raised in [55] against DRBs built from DVRFs is the fact that they rely on computational assumptions on pairing groups, which are known to be easier hard problems to cryptoanalyse than those obtained from ordinary elliptic curves [6]. Our proposed DDH-DVRF resolves this issue as it builds on ordinary elliptic curves. A distributed pseudorandom function (DPRF) with trusted setup presented in [1, Figure 5] was claimed to enjoy public verifiability, but no hard evidence was provided [1, Remark 8.3]. Rigorously proving public verifiability requires to update the definitions of pseudorandomness and robustness in the presence of publicly available proofs of correctness, as well as defining and proving uniqueness. Furthermore, the DPRF construction above makes use of Pedersen commitments in the public key, while our construction DDH-DVRF provides verifiability without requiring the use of commitments, which improves computation efficiency. In this sense DDH-DVRF bears more resemblance to the DPRF construction [1, Figure 6], which was claimed to be only privately verifiable (while we update it to public verifiability and a trustless setup).

II. Building Blocks

We recall next the Chaum-Pedersen proof system for equality of discrete logarithms and the computational assumptions needed to build our concrete DVRFs constructions.

A. Equality of Discrete Logarithms NIZK

Our constructions use NIZK proof systems as an ingredient. Specifically, we need the *Equality of Discrete Logarithms* proof system $(\mathsf{PrEq}_H, \mathsf{VerifyEq}_H)$ [21] to show $k = \log_g x = \log_h y$:

- $\mathsf{PrEq}_H(g, h, x, y, k)$ chooses $r \xleftarrow{\$} \mathbb{Z}_q$, computes $com_1 = g^r, com_2 = h^r$ and sets $ch \leftarrow H(g, h, x, y, com_1, com_2)$. Output is $(ch, res = r + k \cdot ch)$.
- $\mathsf{VerifyEq}_H(g, h, x, y, ch, res)$ computes $com_1 \leftarrow g^{res}/x^{ch}$ and $com_2 \leftarrow h^{res}/y^{ch}$ and outputs $ch \stackrel{?}{=} H(g, h, x, y, com_1, com_2)$.

B. Cryptographic assumptions

Definition II.1 (Asymmetric Pairing Groups)**.** *Let $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$ and $\mathbb{G}_T$ be (cyclic) groups of prime order q. A map $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ to a group $\mathbb{G}_T$ is called a* bilinear *map, if it satisfies the following three properties:*

- *Bilinearity: $\mathbf{e}(g_1^x, g_2^y) = \mathbf{e}(g_1, g_2)^{xy}$ for all $x, y \in \mathbb{Z}_p$.*
- *Non-Degenerate: $\mathbf{e}(g_1, g_2) \neq 1$.*
- *Computable: $\mathbf{e}(g_1, g_2)$ can be efficiently computed.*

We assume there exists an efficient bilinear pairing instance generator algorithm $\mathcal{IG}$ that on input a security parameter 1^λ outputs the description of $\langle \mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q \rangle$.

Asymmetric pairing groups can be efficiently generated [36] and group exponentiations and pairing operations can also be efficiently computed [28]. The security of our pairing-based constructions may require the Computational co-Diffie-Hellman (co-CDH) assumption [13] which states CDH is hard in $\mathbb{G}_1$, and XDH assumption which states that DDH is hard in $\mathbb{G}_1$. Details of these assumptions can be found in Appendix B.

Definition II.2 (Lagrange coefficients)**.** *For a key reconstruction set Δ, we define the* Lagrange basis polynomials $\lambda_{j,\Delta}(x) = \prod_{k \in \Delta \setminus \{j\}} \frac{x-k}{j-k} \in \mathbb{Z}_q[X]$ *and the* Lagrange coefficients $\lambda_{i,j,\Delta} = \lambda_{j,\Delta}(i) \in \mathbb{Z}_q^*$. *For any polynomial $f \in \mathbb{Z}_q[X]$ of degree at most $|\Delta|-1$ this entails $\sum_{i \in \Delta} f(i)\lambda_{0,i,\Delta} = f(0)$.*

III. Distributed Verifiable Random Functions: Formal Definitions

This section introduces the syntax and properties of any (non-interactive fully) Distributed Verifiable Function (DVRF). Compared to a stand-alone Verifiable Random Function (VRF) [48], a DVRF can be seen as a generalisation of a VRF to a distributed setting with no trusted dealer. A DVRF can also be seen as adding public verifiability to the output of Distributed Pseudo-Random Functions [51], [1], although without the need of a trusted setup.

A. Syntax and Basic Properties

Let $a, b : \mathbb{N} \to \mathbb{N}$ be polynomial time functions, and where $a(\lambda), b(\lambda)$ both are bounded by a polynomial in λ. Let $F : \mathrm{Dom} \mapsto \mathrm{Ran}$ be a function with domain Dom and range Ran. Let Dom and Ran be sets of size $2^{a(\lambda)}$ and $2^{b(\lambda)}$ respectively. The goal of a DVRF is to initialize a pseudoRandom function and compute $F_{\mathsf{sk}}(x)$ for inputs x by a set of nodes $N_1, \ldots, N_\ell$ with no central party.

In the Setup phase ℓ nodes $N_1, \ldots, N_\ell$ communicate via pairwise private and authenticated channels. A setup interaction is then run between the ℓ nodes to build a global public key pk, individual nodes' secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell$, and individual nodes' public verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell$. The nodes' secret and verification keys $(\mathsf{sk}_i, \mathsf{vk}_i)$ for $i = 1, \ldots, \ell$ will later enable any subset of $t+1$ nodes to non-interactively compute the verifiable random value $F_{\mathsf{sk}}(x)$ on a plaintext $x \in \mathrm{Dom}$. On the contrary, any set of at most t nodes can not learn any information on $F_{\mathsf{sk}}(x)$ for any x not previously computed.

Definition III.1 (DVRF)**.** *A t-out-of-ℓ (Non-Interactive) Fully Distributed Verifiable Random Function (DVRF) $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ consists of the following algorithms:*

$\mathsf{DistKG}(1^\lambda, t, \ell)$ *is a fully distributed* key generation *protocol that takes as input a security parameter 1^λ, the number of participating nodes ℓ, and the threshold parameter t; it outputs a set of qualified nodes* QUAL*, a global public key pk, a list $\mathcal{VK} = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell\}$ of nodes' verification keys, and results in a list $\mathcal{SK} = \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell\}$ of nodes' secret keys where each secret key is only known to the corresponding node.*

$\mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ *is a* partial evaluation algorithm *that takes as input a plaintext* $x \in \mathrm{Dom}$*, secret key* sk_i *and verification key* vk_i*, and outputs a triple* $s_x^i = (i, v_i, \pi_i)$*, where* v_i *is the* i*-th evaluation share and* π_i *is a non-interactive proof of correct partial evaluation.*

$\mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \mathcal{E})$ *is a* combination *algorithm that takes as input the global public key* pk*, the verification keys* $\mathcal{VK}$*, a plaintext* $x \in \mathrm{Dom}$*, and a set* $\mathcal{E} = \{s_x^{i_1}, \ldots, s_x^{i_{|\mathcal{E}|}}\}$ *of partial function evaluations originating from* $|\mathcal{E}| \geq t+1$ *different nodes, and outputs either a pair* (v, π) *of pseudorandom function value* $v \in \mathrm{Ran}$ *and correctness proof* π*, or* $\perp$*.*

$\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v, \pi)$ *is a* verification algorithm *that takes as input the public key* pk*, a set of verification keys* $\mathcal{VK}$*, a plaintext* $x \in \mathrm{Dom}$*, a value* $v \in \mathrm{Ran}$ *and a proof* π*, and outputs 0/1.*

Admissibility. We say a DVRF is *admissible* if it satisfies three basic properties:

- *Consistency*: meaning that no matter which collection of correctly formed shares is used to compute the function on a plaintext x the same random value $v = F_{\mathsf{sk}}(x)$ is obtained.
- *Robustness*: that guarantees the availability of computing the random function value on any plaintext in the presence of an active adversary. Specifically, robustness demands that if the combine function does not return $\perp$ then its output must pass the verification test even in the presence of the adversary's inputs to the combine function. Robustness has been also called *guaranteed output delivery* (G.O.D.) in recent works [20], [44].
- *Uniqueness*: meaning that for every plaintext x a unique value $v = F_{\mathsf{sk}}(x)$ passes the verification test. It is infeasible for any adversary to compute two different output values v, v' and a plaintext $x \in \mathrm{Dom}$ such that both values pass the verification test w.r.t. x, even when the secret keys of the honest nodes are leaked.

The formal definitions of the above three properties can be found in Appendix D.

B. Strong and Standard Pseudorandomness

Nex we give rigorous definitions for two *pseudorandomness* properties against active adversaries that generalise the pseudorandomness definitions in distributed scenarios [51], [1]. Roughly speaking, pseudorandomness ensures that no adversary controlling at most t nodes is able to distinguish the outputs of the function from random. Previous definitions allow an adversary to choose the set of parties to corrupt, obtain partial evaluations from the honest parties on the challenge plaintext (up to the threshold), and participate in computing the pseudorandom function on the challenge plaintext, in the presence of a trusted dealer.

Our definitions next dispense with the trusted setup phase in previous definitions, and highlight a separation between the strengths achieved by previous attacker models. More concretely, by parametrising attackers in terms of the *actual* number of nodes θ under adversarial control, with $0 \leq \theta \leq t$ and t being the recovery threshold, we are able to separate the pseudorandomness strength in [1], that we rename *strong pseudorandomness*, from [51], which we rename *standard pseudorandomness*.

Definition III.2 (Strong Pseudorandomness)**.** *A* DVRF *protocol* $\mathcal{V}$ *on nodes* $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ *is* (θ, t, ℓ)-strongly pseudorandom *with* $0 \leq \theta \leq t < \ell$ *if for all PPT adversaries* $\mathcal{A}$*,* $\mathbf{Adv}^{\mathrm{PRand}}_{\mathcal{V},\mathcal{A}} = \big|\Pr[\mathrm{PRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathrm{PRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, 1) = 1]\big| \leq \mathsf{negl}(\lambda)$*, where* $\mathsf{negl}()$ *is a negligible function and the experiment* $\mathrm{PRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, b)$ *is defined below:*

[Corruption] $\mathcal{A}$ *chooses a collection* C *of nodes to be corrupted with* $|C| \leq \theta$*. Adversary* $\mathcal{A}$ *acts on behalf of corrupted nodes, while the challenger acts on behalf of the remaining nodes, which behave honestly (namely they follow the protocol specification).*

[Initialization] *Challenger and adversary engage in running the distributed key generation protocol* $\mathsf{DistKG}(1^\lambda, t, \ell)$*. After this phase, the protocol establishes a qualified set of nodes* QUAL*. Every (honest) node* $N_j \in \mathrm{QUAL} \setminus C$ *obtains a key pair* $(\mathsf{sk}_j, \mathsf{vk}_j)$*. In contrast, (corrupted) nodes* $N_j \in C$ *end up with key pairs* $(\mathsf{sk}_j, \mathsf{vk}_j)$ *in which one of keys may be undefined (i.e. either* $\mathsf{sk}_j = \perp$ *or* $\mathsf{vk}_j = \perp$*). At the end of this phase, the global public key* pk *and the verification keys vector* $\mathcal{VK}$ *is known by both the challenger and the attacker.*

[Pre-Challenge Query] *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *for some honest node* $N_i \in \mathrm{QUAL} \setminus C$ *and plaintext* $x \in \mathrm{Dom}$*, the challenger returns* $s_x^i \leftarrow \mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$*. In any other case, the challenger returns* $\perp$*.*

[Challenge] *The challenger receives from the adversary* $\mathcal{A}$ *a set of evaluation shares* $\{s_{x^\star}^i\}_{N_i \in U}$ *with* $U \subseteq \mathrm{QUAL}$ *and* $|U| \geq t+1$*, and a plaintext* $x^\star \in \mathrm{Dom}$*, such that* $(\mathsf{Eval}, x^\star, i)$ *has been queried at most* $t - \theta$ *times for different honest nodes* $N_i \in \mathrm{QUAL} \setminus C$*. Let* $s_{x^\star}^j \leftarrow \mathsf{PartialEval}(x^\star, \mathsf{sk}_j, \mathsf{vk}_j)$ *for honest* $N_j \in U \setminus C$ *and* $(v^\star, \pi^\star) \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x^\star, \{s_{x^\star}^j\}_{N_j \in U \setminus C} \cup \{s_{x^\star}^i\}_{N_i \in U \cap C})$*. If* $v^\star = \perp$ *the experiment outputs* $\perp$*. Otherwise, if* $b = 0$ *the adversary receives* $v^\star$*; if* $b = 1$ *the adversary receives a uniform random value in* Ran*.*

[Post-Challenge Query] *In response to* $\mathcal{A}$*'s evaluation query* (Eval, x, i) *with* $x \neq x^\star$ *for some node* $N_i \in \mathrm{QUAL} \setminus C$ *and plaintext* $x \in \mathrm{Dom}$*, the challenger returns* $s_x^i \leftarrow \mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$*. In any other case, the challenger returns* $\perp$*.*

[Guess] *Finally* $\mathcal{A}$ *returns a guess* b'*. The experiment output is* b'*.*

In the above definition, we separate the upper limit θ on the number of corrupted nodes and the threshold t, which leads to a refinement of previous pseudorandomness definitions. Indeed, [51], [1], only consider the situation $\theta = t$, which is not always the case in practice. For example, the DKG protocol

in Dfinity-DVRF [39] requires that a super-majority of nodes are honest i.e., $\theta < \ell/3$, while the DVRF threshold is set to be $t = (\ell - 1)/2$, leaving a gap between θ and t. Moreover, the values of t and ℓ may greatly affect the communication and computation complexity of the DKG setup (e.g., [35]), and splitting θ and t can make the choice of t and ℓ more flexible for specific applications.

A weaker notion of pseudorandomness, that we refer to as *standard pseudorandomness*, where the adversary is not allowed to obtain any partial evaluation on the challenge plaintext, has been the standard up to now in the related literature on DVRF [30], [14], [46]. The usage of this weaker definition of pseudorandomness can also be found in the DRB literature e.g. [20]. Using our refined approach to defining (strong) pseudorandomness, we define (θ, t, ℓ)-*standard pseudorandomness* as follows.

Definition III.3 (Standard Pseudorandomness)**.** *A* DVRF *protocol $\mathcal{V}$ on nodes $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ satisfies* (θ, t, ℓ)-standard pseudorandomness *with* $0 \leq \theta \leq t < \ell$ *if for all PPT adversaries $\mathcal{A}$,* $\mathbf{Adv}^{\mathrm{StdPRand}}_{\mathcal{V},\mathcal{A}} = \left|\Pr[\mathrm{StdPRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathrm{StdPRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda)$*, where* $\mathsf{negl}(\cdot)$ *is a negligible function and the experiment* $\mathrm{StdPRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, b)$ *is defined as experiment* $\mathrm{PRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, b)$ *with the exception that the adversary is not allowed to obtain any partial evaluation on the challenge plaintext $x^\star$.*

C. Separation between Standard and Strong Pseudorandomness

We shall discuss the relation between standard and strong pseudorandomness. Perhaps unsurprisingly, we illustrate that strong pseudorandomness is a *strictly stronger* property than the standard pseudorandomness. As it is usual, we do so by constructing a DVRF protocol $\mathcal{V}$ that is (θ, t, ℓ)-standard pseudorandom, but it is not (θ, t, ℓ)-strongly pseudorandom. Let $\mathcal{V}' = (\mathsf{DistKG}', \mathsf{PartialEval}', \mathsf{Combine}', \mathsf{Verify}')$ be a DVRF which is (θ, t', ℓ)-standard pseudorandom with $\theta \leq t' < t < \ell$. The existence of such $\mathcal{V}'$ can be guaranteed under well-known computational assumptions with our DVRF constructions in Section IV. We shall construct $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ as follows:

- $\mathsf{DistKG}(1^\lambda, t, \ell)$ runs $\mathsf{DistKG}'(1^\lambda, t', \ell)$ among ℓ nodes to obtain a set QUAL of qualified nodes [1] and the global public key pk. Each node in $i \in \mathrm{QUAL}$ obtains a pair of $(\mathsf{sk}_i, \mathsf{vk}_i)$.
- $\mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ outputs $(i, v_i, \pi_i) \leftarrow \mathsf{PartialEval}'(x, \mathsf{sk}_i, \mathsf{vk}_i)$.
- $\mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \mathcal{E})$ when $|\mathcal{E}| > t$, it is also the case $|\mathcal{E}| > t'$. Thus we can run $(v, \pi) \leftarrow \mathsf{Combine}'(\mathsf{pk}, \mathcal{VK}, x, \mathcal{E})$. Output (v, π).
- $\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v, \pi)$ outputs 1 if $\mathsf{Verify}'(\mathsf{pk}, \mathcal{VK}, x, v, \pi) = 1$. Else outputs 0.

[1] A side condition should be $|\mathrm{QUAL}| > t$ in order to make sure the scheme is non-trivial. In Section IV, we can see that our constructions of DDH-DVRF and GLOW-DVRF both satisfy this condition.

Theorem III.1. *$\mathcal{V}$ is (θ, t, ℓ)-standard pseudorandom but it is not (θ, t, ℓ)-strongly pseudorandom.*

Proof. Firstly, we show that $\mathcal{V}$ does not satisfy (θ, t, ℓ)-strong pseudorandomness. By definition, an adversary $\mathcal{A}^{\mathrm{PRand}}_{\mathcal{V}}$ is allowed to compromise θ nodes, e.g. nodes $N_1, \ldots, N_\theta$ wlog. Additionally $\mathcal{A}^{\mathrm{PRand}}_{\mathcal{V}}$ can query $(\mathsf{Eval}, x^\star, i)$ for up to $t - \theta$ different nodes $N_{\theta+1}, \ldots, N_t \in \mathrm{QUAL}$ wlog on the challenge plaintext $x^\star$. Therefore $\mathcal{A}^{\mathrm{PRand}}_{\mathcal{V}}$ can obtain $t > t'$ valid partial evaluations $\{s^i_{x^\star}\}_{1 \leq i \leq t}$ which allows the adversary to run $\mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x^\star, \{s^i_{x^\star}\}_{1 \leq i \leq t})$ to recover $v^\star$ and trivially win the pseudorandomness game.

Secondly, we show that the (θ, t, ℓ)-standard pseudorandomness of $\mathcal{V}$ holds because an adversary $\mathcal{A}^{\mathrm{StdPRand}}_{\mathcal{V}}$ can only compromise up to θ nodes, where $\theta \leq t' < t$. If the adversary $\mathcal{A}^{\mathrm{StdPRand}}_{\mathcal{V}}$ can distinguish $v^\star$ from random for $\mathcal{V}^\star$, then an adversary $\mathcal{A}^{\mathrm{StdPRand}}_{\mathcal{V}'}$ that can distinguish $v^\star$ from random for $\mathcal{V}'$ would exist too, which contradicts the (θ, t, ℓ)-standard pseudorandomness of $\mathcal{V}'$. □

Corollary III.1.1. *If there exist DVRFs with standard pseudorandomness then strong pseudorandomness is a strictly stronger property than the standard pseudorandomness.*

Interestingly, we will show in Section VI that, in the context of Dencentralized Random Beacons, Algorand DRB [37] is $(0, t, \ell)$-standard pseudorandom, but is not $(0, t, \ell)$-strongly pseudorandom, which offers an independent and natural confirmation of the separation between pseudorandomness properties.

IV. DVRF Instantiations

In this section, we present two original DVRF constructions, that we name DDH-DVRF and GLOW-DVRF, that achieve strong pseudorandomness under widely accepted cryptographic assumptions. DDH-DVRF is described here for the first time and it can be seen as fully distributed version of the stand-alone DDH-VRF presented in [34] (the latter is being proposed for standardisation [38]). On the other hand, GLOW-DVRF is a pairing-based DVRF that achieves strong pseudorandomness with compact proofs (in contrast to DDH-DVRF). Of independent interest is our formalisation and analysis of Dfinity-DVRF, a construction sketched in [39]. We show that Dfinity-DVRF meets standard pseudorandomness and discuss why the current proof techniques do not seem to extend to show strong pseudorandomness. All security reductions in this section use the programmable Random Oracle Model [9].

A. DDH-DVRF: *DDH-based DVRF with non-compact proofs*

Let $(\mathbb{G}, g, q)$ be a multiplicative group where the DDH assumption holds (cf. Section II). Let $H_1 : \{0,1\}^* \mapsto \mathbb{G}$ and $H_2 : \{0,1\}^* \mapsto \mathbb{Z}_q$ be hash functions. Let $\mathcal{V}^{\text{DDH-DVRF}} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$, where:

- $\mathsf{DistKG}(1^\lambda, t, \ell)$ is run by ℓ participating nodes $\mathcal{N} = \{N_1, \ldots, N_\ell\}$. Each node N_i chooses a random polynomial $f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t$. The protocol outputs

a set of qualified nodes QUAL $\subseteq \mathcal{N}$, a secret key $\mathsf{sk}_i = \sum_{j\in\mathrm{QUAL}} f_j(i) \in \mathbb{Z}_q$ and a verification key $\mathsf{vk}_i = g^{\mathsf{sk}_i} \in \mathbb{G}$ for each $i \in$ QUAL, an implicit distributed secret value $\mathsf{sk} = \sum_{i\in\mathrm{QUAL}} a_{i,0} \in \mathbb{Z}_q$, and a global public key $\mathsf{pk} = \prod_{i\in\mathrm{QUAL}} g^{a_{i,0}}$. The full description of the protocol can be found in Figure 2 in Appendix I and it is a well-known DKG protocol due to [35].

$\mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ outputs $s_x^i = (i, v_i, \pi_i)$ for a plaintext x, where $v_i \leftarrow H_1(x)^{\mathsf{sk}_i}$ and $\pi_i \leftarrow \mathsf{PrEq}_{H_2}(g, \mathsf{vk}_i, H_1(x), v_i; r)$ for randomness $r \xleftarrow{\$} \mathbb{Z}_q$ (cf. Section II-A for the description of the NIZK proof system for equality of discrete logarithms $(\mathsf{PrEq}_H, \mathsf{VerifyEq}_H)$).

$\mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \mathcal{E})$ parses list $\mathcal{E} = \{s_x^{j_1}, \ldots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \geq t+1$ partial evaluation candidates originating from $|\mathcal{E}|$ different nodes, and obtains verification keys $\mathsf{vk}_{j_1}, \ldots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,

1) Identifies an index subset $I = \{i_1, \ldots, i_{t+1}\}$ such that $\mathsf{VerifyEq}_{H_2}(g, \mathsf{vk}_i, H_1(x), v_i, \pi_i) = \mathtt{accept}$ holds for every $i \in I$, where $(i, v_i, \pi_i) \leftarrow s_x^i$. If no such subset exists, outputs $\perp$.
2) Sets $v \leftarrow \prod_{i\in I} v_i^{\lambda_{0,i,I}}$ and $\pi \leftarrow \{s_x^i\}_{i\in I}$.
3) Outputs (v, π).

$\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v, \pi)$ parses $\pi = \{s_x^i\}_{i\in I}$ such that $|I| = t+1$ and $I \subseteq$ QUAL

1) Parses $s_x^i = (i, v_i, \pi_i)$ for $i \in I$.
2) Checks if $\mathsf{VerifyEq}_{H_2}(g, \mathsf{vk}_i, H_1(x), v_i, \pi_i) = 1$ for every $i \in I$; if some of the checks fail, outputs 0.
3) Checks if $v = \prod_{i\in I} v_i^{\lambda_{0,i,I}}$; if so outputs 1; otherwise outputs 0.

The full proofs of the following theorems can be found in Appendix J.

Theorem IV.1. $\mathcal{V}^{\text{DDH-DVRF}}$ *satisfies consistency, robustness and uniqueness.*

Theorem IV.2. $\mathcal{V}^{\text{DDH-DVRF}}$ *is* (θ, t, ℓ)*-strongly pseudorandom for any* θ *with* $\theta \leq t < \ell - \theta$ *under the* DDH *assumption in the random oracle model.*

Sketch of the proof. Assume an extended DDH instance (this can be easily derived from DDH instance) $(g^{\alpha_0}, g^{\alpha_1}, \ldots, g^{\alpha_w}, g^{\beta}, y_0, y_1, \ldots, y_w)$ where either $y_0 = g^{\alpha_0\beta}, y_1 = g^{\alpha_1\beta}, \ldots, y_w = g^{\alpha_w\beta}$ or $y_0 \xleftarrow{\$} \mathbb{G}, y_1 \xleftarrow{\$} \mathbb{G}, \ldots, y_w \xleftarrow{\$} \mathbb{G}$. Suppose the adversary corrupts nodes $C = \{1, \cdots, m\}$ with $m \leq \theta$ in the corruption phase. A simulator who represents all the $\ell - m$ honest nodes can simulate the running of the DKG protocol with the adversary who represents all the m corrupted nodes. At the end of the DKG protocol, the global public key is set to be $\mathsf{pk} = g^{\alpha_0}$ and the verification key $\mathsf{vk}_i = g^{\alpha_i}$ for each honest user $i \in [m+1, t]$. Because $\ell - m \geq \ell - \theta > t$, the simulator is able to derive all the secret keys of the adversary, i.e., sk_i for $i \in C$. In the random oracle model, the hash value of the challenge plaintext is set to be $H_1(x^\star) = g^\beta$. The adversary's partial evaluation queries on $x^\star$ can be answered directly using y_i for an honest user $j \in [m+1, t]$. For an honest user $j > t$, the partial evaluation on $x^\star$ can be computed using Lagrange coefficients and values of $\{y_i\}_{m+1\leq i\leq t}$ and the adversary's shares $g^{\beta\mathsf{sk}_i}$ for $i \in C$. When $y_i = g^{\alpha_i\beta}$ for each i, this simulates the real pseudorandomness game perfectly; but when y_i is a uniform random, the adversary receives randoms. If the adversary can break the pseudorandomness, then we can construct an adversary that breaks the DDH assumption.

Remark 1 (DDH-DVRF vs. DPRFs [1])**.** *The construction with trustless setup* DDH-DVRF *bears similarities to the distributed pseudorandom functions (DPRFs) with trusted setup [1, Figure 5] and [1, Figure 6]. The correctness of the latter is only privately verifiable, and although the authors claim that the former construction enjoys public verifiability, no hard evidence was provided in [1]. As we have seen rigorously proving public verifiability requires to update pseudorandomness and robustness (called correctness in [1]) in the presence of publicly available proofs of correctness, as well as defining and proving uniqueness. Futhermore, the DPRF construction that admits public verifiability in [1] makes use of Pedersen commitments in the public key, while our construction* DDH-DVRF *provides verifiability without requiring the use of commitments, which improves computation efficiency. In this sense* DDH-DVRF *bears more resemblance to the DPRF construction [1, Figure 6], which was claimed to be only privately verifiable.*

B. GLOW-DVRF: a strongly pseudorandom pairing-based DVRF with compact proofs

DDH-DVRF has the advantages of admitting very fast implementations and relying on a long-standing cryptographic assumption, as it can be built using e.g. elliptic curves. DDH-DVRF's proofs π are however non-compact, i.e. their size is linear in the reconstruction threshold t. Next we describe a pairing-based GLOW-DVRF (cf. Definition II.1) that achieves *strong pseudorandomness and compact proofs* simultaneously.

Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, h_1, h_2)$ be a bilinear pairing, where g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively (same applies to h_1, h_2). Let $H_1 : \{0,1\}^* \mapsto \mathbb{G}_1$, $H_2 : \mathbb{G}_1 \mapsto \{0,1\}^{b(\lambda)}$ and $H_3 : \{0,1\}^* \mapsto \mathbb{Z}_q$ be hash functions. Let $\mathcal{V}^{\text{GLOW-DVRF}} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ be a DVRF for a pseudorandom function defined as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ proceeds in the same way as in the non-compact case, with the difference described as below:

- In the Generating phase, g, h are replaced with $g_1, h_1 \in \mathbb{G}_1$.
- In Step 4, each node N_i with $i \in$ QUAL exposes $B_{i,0} = g_2^{a_{i,0}}$ via Feldman-VSS.
  - In Step 4(a), each node N_i with $i \in$ QUAL broadcasts $A_{i,k} = g_1^{a_{i,k}}$ for $0 \leq k \leq t$ and $B_{i,0} = g_2^{a_{i,0}}$.

- In Step 4(b), for each $i \in \text{QUAL}$, N_j checks if $g_1^{s_{i,j}} = \prod_{k=0}^{t}(A_{i,k})^{j^k}$ and $e(A_{i,0}, g_2) = e(g_1, B_{i,0})$.
- In Step 4(c), set the global public key as $\mathsf{pk} = \prod_{i \in \text{QUAL}} B_{i,0}$.

To summarise, the verification keys vk_i are generated in $\mathbb{G}_1$ using the generator g_1 and the global public key pk is generated in $\mathbb{G}_2$ using the generator g_2. Full details can be found in Figure 3 in Appendix K.

$\mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ computes $v_i = H_1(x)^{\mathsf{sk}_i}$ and $\pi_i \leftarrow \mathsf{PrEq}_{H_3}(g, \mathsf{vk}_i, H_1(x), v_i; r)$ for randomness $r \xleftarrow{\$} \mathbb{Z}_q$, and outputs share $s_x^i = (i, v_i, \pi_i)$.

$\mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \mathcal{E})$ parses list $\mathcal{E} = \{s_x^{j_1}, \ldots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \geq t+1$ partial evaluation candidates originating from $|\mathcal{E}|$ different nodes, and obtain verification keys $\mathsf{vk}_{j_1}, \ldots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,

1) Identifies an index subset $I = \{i_1, \ldots, i_{t+1}\}$ such that for every $i \in I$ it holds that $\mathsf{VerifyEq}_{H_3}(g, \mathsf{vk}_i, H_1(x), v_i, \pi_i) = 1$, where $(i, v_i, \pi_i) \leftarrow s_x^i$. If no such subset exists, outputs $\perp$.
2) Sets $\pi \leftarrow \prod_{j \in I} v_j^{\lambda_{0,j,I}}$ and $v = H_2(\pi)$.
3) Outputs (v, π).

$\mathsf{Verify}(\mathsf{pk}, x, v, \pi)$**:** output 1 if the relation holds: $e(\pi, g_2) = e(H_1(x), \mathsf{pk})$ and $v = H_2(\pi)$. Otherwise output 0.

The DistKG protocol in GLOW-DVRF generates the verification keys vk_i in $\mathbb{G}_1$ but the global public key pk in $\mathbb{G}_2$. The function evaluation shares are validated using NIZKs, while the well-formedness of the reconstructed pseudorandom value v is validated using a pairing equation, which in turn provides the compact proof as intended.

Using NIZKs to validate partial evaluations has two benefits. Firstly, it is crucial for proving strong pseudorandomness for GLOW-DVRF. The main technical difficulty of proving strong pseudorandomness is to simulate the answers to the oracle queries $\mathsf{PartialEval}(\mathsf{sk}_i, \mathsf{vk}_i, x^\star)$ on the challenge plaintext $x^\star$ when the secret key sk_i is unknown. Placing verification keys vk_i's on $\mathbb{G}_1$ results in an adversary being unable to check the validity of the i-th partial evaluation (provided that the XDH assumption holds in the bilinear pairing group), which facilitates exhibiting a security reduction to strong pseudorandomness. Secondly, by validating evaluation shares verifying NIZKs instead of pairing equations speeds up the Combine algorithm by computing $4 \cdot |\mathcal{E}|$ exponentiations in $\mathbb{G}_1$ instead of computing $2 \cdot |\mathcal{E}|$ pairings, as the latter are more computationally expensive.

The full proofs of the following theorems are given in Appendix K.

Theorem IV.3. $\mathcal{V}^{\text{GLOW-DVRF}}$ *satisfies consistency, robustness and uniqueness.*

Theorem IV.4. $\mathcal{V}^{\text{GLOW-DVRF}}$ *achieves* (θ, t, ℓ)*-standard pseudorandomness for any* θ *with* $\theta \leq t < \ell - \theta$ *under the co-CDH assumption in the random oracle model.*

Sketch of the proof. Assume a co-CDH instance $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\alpha)$ with $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$. Suppose the adversary corrupts nodes $C = \{1, \cdots, m\}$ in the corruption phase. A simulator who represents all the $\ell - m$ honest nodes can simulate the running of the DKG protocol with the adversary who represents all the m corrupted nodes. At the end of the DKG protocol, the global public key is set to be $\mathsf{pk} = g_2^\alpha$. The secret keys sk_i for honest nodes $i \in [m+1, t]$ are chosen by the simulator. The secret keys sk_i for honest nodes $i > t$ cannot be computed by the simulator because α is unknown, but the corresponding verification keys vk_i for $i > t$ can be computed using Lagrange coefficients. In the random oracle model, the hash value of the challenge plaintext is set as $H_1(x^\star) = g_1^\beta$. When the adversary queries the random oracle H_2 on a value y, we check if $e(y, g_2) = e(g_1^\beta, g_2^\alpha)$. If true, it means $y = g_1^{\alpha\beta}$ and y can be output as a solution to the co-CDH instance. We can show the probability that the adversary queries such a y is at least the same as the advantage the adversary distinguishes the standard pseudorandomness game.

Theorem IV.5. $\mathcal{V}^{\text{GLOW-DVRF}}$ *is* (θ, t, ℓ)*-strongly pseudorandom under the* XDH *assumption and* co-CDH *assumption in the random oracle model.*

Sketch of the proof. Suppose the adversary corrupts nodes $C = \{1, \cdots, m\}$ in the corruption phase. When $m = t$, the adversary has already compromised t nodes and is not allowed to issue any partial evaluation query on the challenge plaintext $x^\star$. Therefore, the proof for this case is similar to Theorem IV.4. When $m < t$, the partial evaluation queries on $x^\star$ can be simulated using values in XDH instance which is similar to Theorem IV.2.

C. Formalisation and analysis of Dfinity-DVRF

In Appendix L we provide a detailed description of a pairing-based construction that was sketched in [39] and that we call Dfinity-DVRF. We notice that no formal treatment nor security proofs have been given to our knowledge. We show that Dfinity-DVRF *achieves standard pseudorandomness*. The full proofs of the following theorems are given in Appendix L, where we also discuss how standard proof techniques [11], [35] do not seem to trivially extend to prove strong pseudorandomness for Dfinity-DVRF.

Theorem IV.6. $\mathcal{V}^{\text{Dfinity-DVRF}}$ *satisfies consistency, robustness and uniqueness.*

Theorem IV.7. $\mathcal{V}^{\text{Dfinity-DVRF}}$ *achieves* (θ, t, ℓ)*-standard pseudorandomness for any* θ *such that* $\theta \leq t < \ell - \theta$ *under the co-CDH assumption in the random oracle model.*

Remark 2 (DVRFs imply DPRFs)**.** *It is easy to see that DVRFs impliy Distributed Pseudorandom Functions by simply dropping the verification algorithm. As a consequence the three concrete trustless DVRF constructions presented in this section imply three new* trustless *DPRF constructions as per [1].*

V. Decentralised Random Beacons

A Decentralised Random Beacon (DRB) provides a way to collaboratively agree on a (pseudo)random value [23] without the involvement of a central party. A prominent application of DRBs is to randomly select a leader in Proof-of-Stake blockchains (e.g. Dfinity [39], Ethereum 2.0 [17], OmniLedger [45], Tendermint [16]), without the need for a coordinator.

A. Formalisation of DRB

We start by presenting a syntax for DRBs with non-interactive randomness generation and formally define its relevant security properties, including strong/standard pseudorandomness, robustness and uniqueness. As we will argue, we believe our definitions have merits not present in recent definitions in the literature [20], [5]. In particular we are able to analyse with our framework several academic and industry constructions (including ours): Algorand [37], HERB [22], Ouroboros Praos [27], Elrond [32], Orbs [4] and Harmony [40]. Last but not least our new DRB formalisation also allows us to capture the generic DVRF-to-DRB construction and study its security, bringing the total of instantiations capture by our model to nine.

Definition V.1 (Decentralised random beacon (DRB))**.** *A t-out-of-ℓ* DRB *on a set of nodes $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ is specified as a tuple $\mathcal{R} = (\mathsf{CmteGen}, \mathsf{PartialRand}, \mathsf{CombRand}, \mathsf{VerifyRand}, \mathsf{UpdState})$ of polynomial algorithms as follows:*

- $\mathsf{CmteGen}(1^\lambda, t, \ell)$**:** *this is an interactive protocol run by the nodes in $\mathcal{N}$ to set up a random beacon committee. The protocol determines a set of qualified nodes $\mathrm{QUAL} \subseteq \mathcal{N}$, outputs a committee public key cpk and results in a list of secret keys $\mathcal{SK} = \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell\}$, where each secret key is only known to the corresponding node in the committee.*
- $\mathsf{PartialRand}(\mathsf{st}_{n-1}, \mathsf{sk}_i, \mathsf{cpk})$**:** *on input the state $\mathsf{st}_{n-1} \in \mathrm{Dom}$ from round $n-1$, a secret key sk_i and a committee public key cpk, the algorithm computes a partial value $\sigma_{n,i}$ and a proof $\pi_{n,i}$ for round n. Output is $(i, \sigma_{n,i}, \pi_{n,i})$.*
- $\mathsf{CombRand}(\mathsf{st}_{n-1}, \mathcal{E}, \mathsf{cpk})$ *: on input a state $\mathsf{st}_{n-1} \in \mathrm{Dom}$, a set $\mathcal{E} = \{(i, \sigma_{n,i}, \pi_{n,i})\}_{i \in I}$ of partial values from $|I| \geq t+1$ different nodes, and a committee public key cpk, this algorithm outputs a pair (σ_n, π_n) of a beacon value and a proof, or $\perp$.*
- $\mathsf{VerifyRand}(\mathsf{st}_{n-1}, \sigma_n, \pi_n, \mathsf{cpk})$**:** *on input a state $\mathsf{st}_{n-1} \in \mathrm{Dom}$, a beacon value $\sigma_n \in \mathrm{Ran}$, a proof π_n and a committee public key cpk, this algorithm verifies if (σ_n, π_n) is valid. The algorithm outputs 0/1.*
- $\mathsf{UpdState}(\mathsf{st}_{n-1}, \sigma_n, \pi_n, \mathsf{cpk})$**:** *on input the current state st_{n-1}, a beacon value $\sigma_n \in \mathrm{Ran}$ and its proof π_n generated at the end of round $n-1$, the algorithm outputs the updated state st_n for round n, or $\perp$.*

Before giving security definitions for DRB, we first describe a set of oracles that model the adversarial capabilities. The oracles are introduced below and their formal definitions are given in Figure 1. The oracles update the following global variables: a total round number rn; the current state st; a variable ck that stores committee information.

$\mathsf{Init}(C)$
if $C \not\subseteq \mathcal{N}$ or $|C| > t$ then return $\perp$
$(\mathrm{QUAL}, \mathsf{cpk}, \mathcal{K}) \leftarrow \mathsf{CmteGen}(1^\lambda, t, \ell)$
$\mathsf{ck} = (\mathcal{N}, C, \mathrm{QUAL}, \mathsf{cpk}, \mathcal{K})$
$\mathsf{rn} = 0$, $\mathsf{st} = \mathsf{st}_0$
return 1

$\mathcal{O}^{\mathsf{KeyRev}(i)}$
if $i \notin \mathrm{QUAL} \setminus C$ return $\perp$
parse $\mathcal{K} = \{\mathsf{sk}_j\}_{j \in \mathrm{QUAL} \setminus C}$
return sk_i

$\mathcal{O}^{\mathsf{Update}(\sigma, \pi)}$
parse $\mathsf{ck} = (\mathcal{N}, C, \mathrm{QUAL}, \mathsf{cpk}, \mathcal{K})$
if $\mathsf{VerifyRand}(\mathsf{st}, \sigma, \pi, \mathsf{cpk}) = 0$ return $\perp$
$x \leftarrow \mathsf{UpdState}(\mathsf{st}, \sigma, \pi, \mathsf{cpk})$
if $x = \perp$ then return $\perp$
$\mathsf{rn} = \mathsf{rn} + 1$; $\mathsf{st} = x$
return $(\mathsf{rn}, \mathsf{st})$

$\mathcal{O}^{\mathsf{PartialRand}(i)}$
parse $\mathsf{ck} = (\mathcal{N}, C, \mathrm{QUAL}, \mathsf{cpk}, \mathcal{K})$
if $i \notin \mathrm{QUAL}$ or $i \in C$ return $\perp$
$(i, \sigma_i, \pi_i) \leftarrow \mathsf{PartialRand}(\mathsf{st}, \mathsf{sk}_i, \mathsf{cpk})$
return (i, σ_i, π_i)

Figure 1: Oracles used in the different security games associated to any t-out-of-ℓ DRB.

- $\mathsf{Init}(C)$: this oracle initialises a committee from nodes in $\mathcal{N}$. The oracle runs the interactive protocol $\mathsf{CmteGen}$ on behalf of the honest nodes $\mathcal{N} \setminus C$ and the adversary runs on behalf of the bad nodes C. The adversary is allowed to control up to t bad nodes in $\mathcal{N}$. The protocol determines a set of qualified nodes QUAL and generates a committee public key cpk. Each qualified node $i \in \mathrm{QUAL}$ obtains a secret key sk_i at the end of the protocol. Let $\mathcal{K}$ be the set of secret keys of the honest nodes run by the oracle in the committee. We slightly abuse notation and write $(\mathrm{QUAL}, \mathsf{cpk}, \mathcal{K}) \leftarrow \mathsf{CmteGen}(1^\lambda, U)$ for the oracle's output after running the interactive protocol $\mathsf{CmteGen}$ where the adversary's secret keys are not included in $\mathcal{K}$.
- $\mathcal{O}^{\mathsf{KeyRev}(i)}$: this oracle returns the secret key of an honest node i in the committee.
- $\mathcal{O}^{\mathsf{PartialRand}(i)}$: this oracle computes the i-th node contribution of the committee to the random beacon by using the current state st and the i-th honest node's secret key and running $\mathsf{PartialRand}(\mathsf{st}, \mathsf{sk}_i, \mathsf{cpk})$.
- $\mathcal{O}^{\mathsf{Update}}(\sigma, \pi)$: this oracle extends the current ledger with (σ, π) as the official randomness output for the current round. The oracle runs $\mathsf{UpdState}(\mathtt{s.t.}, \sigma, \pi, \mathsf{cpk})$ to obtain the state for the next round and increases the round number rn by 1.

Pseudorandomness for DRB. Pseudorandomness guarantees that the random beacon outputs are indistinguishable from uniformly random values in the presence of active adver-

saries, which implies the properties of unpredictability and bias-resistance [57], [55]. Below we shall give the formal definitions of the standard and strong pseudorandomness for DRB.

Definition V.2 (Strong Pseudorandomness of DRB)**.** *A* DRB $\mathcal{R} = (\mathsf{CmteGen}, \mathsf{PartialRand}, \mathsf{CombRand}, \mathsf{VerifyRand}, \mathsf{UpdState})$ *on a set of nodes* $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ *is* (θ, t, ℓ)-strongly pseudorandom *with* $\theta \leq t < \ell$ *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\mathbf{Adv}^{\mathrm{PRand}}_{\mathcal{R},\mathcal{A}} = \left| \begin{array}{l} \Pr\left[\mathrm{PRand}_{\mathcal{R},\mathcal{A}}(1^\lambda, 0) = 1\right] \\ -\Pr\left[\mathrm{PRand}_{\mathcal{R},\mathcal{A}}(1^\lambda, 1) = 1\right] \end{array} \right| \leq \mathsf{negl}(\lambda)$$

where the experiment $\mathrm{PRand}_{\mathcal{R},\mathcal{A}}(1^\lambda, b)$ *with* $b \in \{0, 1\}$ *is defined as below:*

[Corruption] *A correctness adversary* $\mathcal{A}$ *chooses a collection* C *of nodes to be corrupted with* $|C| \leq \theta$*.*

[Initialization] *The challenger runs the oracle* $\mathcal{O}^{\mathsf{Init}(C)}$ *to set up a committee by interacting with the adversary.*

[Pre-Challenge Queries] $\mathcal{A}$ *is given access to oracles* $\mathcal{O}^{\mathsf{PartialRand}}$ *and* $\mathcal{O}^{\mathsf{Update}}$ *described above.*

[Challenge] *The challenger receives from the adversary* $\mathcal{A}$ *a set* U *of size at least* $t + 1$ *and* $U \subseteq \mathrm{QUAL}$*, and a set of partial values* $\{(i, \sigma_i, \pi_i)\}_{i \in U \cap C}$*. Let the current state be* $\mathsf{st} = x^\star$ *and the current round number be* $\mathsf{rn} = j^\star$*.* $\mathcal{A}$ *is not allowed to query the oracle* $\mathcal{O}^{\mathsf{PartialRand}(i)}$ *for more than* $t - |C|$ *different* i *in the* $j^\star$*-th round. The challenger proceeds as follows:*

- $(i, \sigma_i, \pi_i) \leftarrow \mathsf{PartialRand}(x^\star, \mathsf{sk}_i, \mathsf{cpk})$ *for* $i \in U \setminus C$*. Let* $(\sigma^\star, \pi^\star) \leftarrow \mathsf{CombRand}(x^\star, \{(i, \sigma_i, \pi_i)\}_{i \in U}, \mathsf{cpk})$*. If* $\sigma^\star = \perp$*, then the experiment outputs* $\perp$*.*
- *Otherwise, if* $b = 0$*, the challenger sets* $\delta = \sigma^\star$*; if* $b = 1$*, the challenger chooses a uniform random* $\delta \xleftarrow{\$} \mathrm{Ran}$*.*
- *The challenger moves to the next round by computing* $\mathsf{st} \leftarrow \mathsf{UpdState}(\delta, x^\star)$ *and increasing the round number* $\mathsf{rn} = j^\star + 1$*. The challenger gives* $(\delta, \mathsf{rn}, \mathsf{st})$ *to the adversary.*

[Post-Challenge Queries] $\mathcal{A}$ *can continue to query the oracles* $\mathcal{O}^{\mathsf{PartialRand}}$ *and* $\mathcal{O}^{\mathsf{Update}}$*.*

[Guess] *Finally* $\mathcal{A}$ *returns a guess* b'*. Output* b'*.*

In the above experiment, the adversary is allowed to compromise m nodes with $m \leq \theta$ and is allowed to query $\mathcal{O}^{\mathsf{PartialRand}(\cdot)}$ up to $t - m$ times in the challenge round (i.e., the $j^\star$-th round). To answer the challenge query, the challenger combines the paritial values from the adversary and the honest nodes to compute $\sigma^\star$. Depending on the value of b, the challenger sets $\delta = \sigma^\star$ or a uniform random. Then the challenger uses the value of δ to update the current state and move to the next round.

Standard pseudorandomness for DRBs is obtained by restricting the adversary in the above experiment to make no queries in the $j^\star$-th round to the oracle $\mathcal{O}^{\mathsf{PartialRand}}$.

DRB Uniqueness. Uniqueness requires that the random beacon values are unique per round, even in the presence of an adversary that obtains the secret keys of the honest parties.

Definition V.3 (Uniqueness)**.** *A* t*-out-of-*ℓ DRB *protocol* $\mathcal{R} = (\mathsf{CmteGen}, \mathsf{PartialRand}, \mathsf{CombRand}, \mathsf{VerifyRand}, \mathsf{UpdState})$ *on a set of nodes* $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ *satisfies* uniqueness *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following experiment outputs 1 with probability at most* $\mathsf{negl}(\lambda)$*.*

[Corruption] *Adversary* $\mathcal{A}$ *chooses a collection* C *of nodes to be corrupted with* $|C| \leq t$*.*

[Initialization] *The challenger runs the initialisation phase* $\mathsf{Init}(C)$ *as described in Figure 1 to set up a committee by interacting with the adversary, and also initialises the global variables.*

[Queries] $\mathcal{A}$ *is given access to oracles* $\mathcal{O}^{\mathsf{PartialRand}}$*,* $\mathcal{O}^{\mathsf{Update}}$ *and* $\mathcal{O}^{\mathsf{KeyRev}}$ *as described in Figure 1.*

[Challenge] *The challenger receives from the adversary* $\mathcal{A}$*, two beacon values* $\sigma, \sigma' \in \mathrm{Ran}$ *and two proofs* π, π'*. Let the current state be* $\mathsf{st} = x^\star$*. If* $\sigma \neq \sigma'$ *and* $\mathsf{VerifyRand}(x^\star, \sigma, \pi, \mathsf{cpk}) = \mathsf{VerifyRand}(x^\star, \sigma', \pi', \mathsf{cpk}) = 1$ *then output 1; else output 0.*

The uniqueness of the pseudorandom output of a DRB provides *strong bias resistance*, as it stands against any adversary independently from the number of corrupted nodes that the adversary controls. This prevents an adversary from manipulating the beacon values to obtain e.g. financial advantage and can be an useful block to obtain fairness in DeFi applications [43].

Similarly to DVRFs an additional security property is needed for DRBs, namely *robustness*. Robustness, also called Guaranteed Output Delivery, ensures that the computation of beacon outputs cannot be stopped by an adversary once $t + 1$ partial beacons are emitted by nodes from the committee, even if some of the beacon shares have been generated by malicious nodes. A definition of DRB robustness is given in Appendix E.

B. Generic DVRF-to-DRB Construction and Security Analysis

Instantiation. We now describe a general implementation of DRB using DVRF:

$\mathsf{CmteGen}(1^\lambda, t, \ell)$**:** a set $\mathcal{N}$ of ℓ nodes jointly run $\mathsf{DistKG}(1^\lambda, t, \ell)$ to obtain a set QUAL of qualified nodes, a public key pk. Each node i obtains a secret key sk_i and a verification key vk_i. Let $\mathsf{cpk} = (\mathsf{pk}, \mathcal{VK})$ where $\mathcal{VK} = \{\mathsf{vk}_i\}_{i \in \mathrm{QUAL}}$.

$\mathsf{PartialRand}(\mathsf{st}_{n-1}, \mathsf{sk}_i, \mathsf{cpk})$**:** parse $\mathsf{cpk} = (\mathsf{pk}, \mathcal{VK})$. Output $(\sigma_i, \pi_i) \leftarrow \mathsf{PartialEval}(\mathsf{st}_{n-1}, \mathsf{sk}_i, \mathsf{vk}_i)$.

$\mathsf{CombRand}(\mathsf{st}_{n-1}, \mathcal{E}, \mathsf{cpk})$**:** parse $\mathsf{cpk} = (\mathsf{pk}, \mathcal{VK})$. Output $(\sigma, \pi) \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, \mathsf{st}_{n-1}, \mathcal{E})$.

$\mathsf{VerifyRand}(\mathsf{st}_{n-1}, \sigma_n, \pi_n, \mathsf{cpk})$**:** parse $\mathsf{cpk} = (\mathsf{pk}, \mathcal{VK})$. Output $\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, \mathsf{st}_{n-1}, \sigma_n, \pi_n)$.

$\mathsf{UpdState}(\mathsf{st}_{n-1}, \sigma_n, \pi_n, \mathsf{cpk})$**:** Output the state $\mathsf{st}_n = \sigma_n \| n$ for the next round n. The round number is of ϱ-bit and the

restriction placed here is that the total number of rounds is at most 2^{ϱ}. We assume the state $\mathsf{st}_0 = \sigma_0 \| 0$ where the choice of $\sigma_0 \in \mathcal{R}$ is left open.

The full proofs of the following theorems are given in Appendix E.

Theorem V.1. *$\mathcal{V}^{\mathrm{DRB}}$ is correct if $\mathcal{V}^{\mathrm{DVRF}}$ is robust.*

Theorem V.2. *$\mathcal{R}^{\mathrm{DRB}}$ achieves (θ, t, ℓ)-standard (resp. strong) pseudorandomness if $\mathcal{V}^{\mathrm{DVRF}}$ achieves (θ, t, ℓ)-standard (resp. strong) pseudorandomness.*

Theorem V.3. *$\mathcal{V}^{\mathrm{DRB}}$ satisfies uniqueness if $\mathcal{V}^{\mathrm{DVRF}}$ satisfies uniqueness.*

Remark 3. *With our DVRF constructions from Section IV the initial seed σ_0 can be set to be* pk*, the global public key of the underlying DVRF construction. Indeed* pk *as output by our concrete DVRFs is uniformly distributed at random in the corresponding DH group, a property inherited by our DKG protocols from [35]. It must be noted that other DRB protocols from the literature [37], [32], [40] need to use an out-of-band channel to agree on the initial seed, which is not needed with our DVRF-to-DRB constructions.*

VI. Comparison with state-of-art DRB Instantiations

Our DRB framework focus on the DRBs which can generate a random beacon value for each round in a non-interactive manner. We stress that non-interactive protocols are the most desirable solutions for decentralised applications. Although our DRB syntax bears a lot of similarity to our modelisation of DVRFs, and hence could seem to lack in generality, we show below how to use our definitions to formalise Algorand DRB [37], Ouroboros-Praos DRB [27], Elrond DRB [32] and Harmony DRB [40] which are non-interactive. Then we shall describe and compare with other interactive DRBs which may run across multiple rounds to create a beacon value.

A. Capturing non-interactive DRBs with our formalisation

Algorand, Ouroboros-Praos and Elrond. Algorand [37] defines a DRB with the aim to create a pseudorandom seed that is fed into a cryptographic sortition algorithm that assigns certain roles to nodes belonging to a committee (such as being block proposer in a given round). Ouroboros-Praos and Elrond [27], [32] also implement DRBs that are similar to Algorand's. These DRBs are based on stand-alone verifiable random functions $\mathrm{VRF} = (\mathrm{VRF.KeyGen}, \mathrm{VRF.Eval}, \mathrm{VRF.Verify})$ [48] (the syntax of VRF can be found in Appendix C), and can be formalised as the tuple $\mathcal{R} = (\mathsf{CmteGen}, \mathsf{UpdState}, \mathsf{PartialRand}, \mathsf{CombRand}, \mathsf{VerifyRand})$ as follows:

$\mathsf{CmteGen}(1^\lambda, t, \ell)$**:** This is a non-interactive protocol, in which each node in $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ creates a VRF key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathrm{VRF.KeyGen}(1^\lambda)$ for $i = 1, \ldots, \ell$. The set of qualified nodes $\mathrm{QUAL} \subseteq U$ are those that contribute a VRF public key, and the committee public key $\mathsf{cpk} = \{\mathsf{pk}_i\}_{i \in \mathrm{QUAL}}$.

$\mathsf{PartialRand}(\mathsf{st}_{n-1}, \mathsf{sk}_i, \mathsf{cpk})$**:** Compute a partial value for round n by running the VRF evaluation function $(\rho_{n,i}, \pi_{n,i}) \leftarrow \mathrm{VRF.Eval}(\mathsf{sk}_i, \mathsf{st}_{n-1})$. Output $(i, \rho_{n,i}, \pi_{n,i})$.

$\mathsf{CombRand}(\mathsf{st}_{n-1}, \mathcal{E}, \mathsf{cpk})$**:** Parse $\mathcal{E} = \{(i, \rho_{n,i}, \pi_{n,i})\}_{i \in I}$ with $|I| > t$ and $I \subseteq \mathrm{QUAL}$. Run a selection algorithm $j \leftarrow \mathsf{Sel}(\mathsf{st}_{n-1}, I)$, where Sel is a pseudorandom permutation and returns the node from I with the highest priority. If $\mathrm{VRF.Verify}(\mathsf{pk}_j, \mathsf{st}_{n-1}, \rho_{n,j}, \pi_{n,j}) = 1$, set $\sigma_n = \rho_{n,j}$ and $\pi_n = (j, \pi_{n,j}, I)$; else set $\sigma_n = H(\mathsf{st}_{n-1} \| n)$ and $\pi_n = \perp$. Output (σ_n, π_n).

$\mathsf{VerifyRand}(\mathsf{st}_{n-1}, \sigma_n, \pi_n, \mathsf{cpk})$**:** If $\pi_n = \perp$, output the result of whether $H(\mathsf{st}_{n-1} \| n) == \sigma_n$. Otherwise, parse $\pi_n = (j, \pi_{n,j}, I)$. If $|I| > t$ and $I \subseteq \mathrm{QUAL}$ and $\mathrm{VRF.Verify}(\mathsf{pk}_j, \mathsf{st}_{n-1}, \sigma_n, \pi_{n,j}) = 1$ with $j \leftarrow \mathsf{Sel}(\mathsf{st}_{n-1}, I)$, output 1; else output 0.

$\mathsf{UpdState}(\mathsf{st}_{n-1}, \sigma_n)$**:** Output $\mathsf{st}_n = \sigma_n \| n$. The initial state $\mathsf{st}_0 = \sigma_0 \| 0$ where $\sigma_0 \in \mathrm{Ran}$ is a random chosen by the initial participants in the protocol.

The intuition behind this construction is that in each round n a node N_j is chosen as a beacon leader based on the randomness st_{n-1} agreed on round $n-1$. The next state is defined as the output of the VRF computation $\mathrm{VRF.Eval}(\mathsf{sk}_j, \mathsf{st}_{n-1})$ by node N_j (provided it passes the verification test wrt public key pk_j); otherwise if node N_j failed to provide a value, then the state for round n is updated as $H(\mathsf{st}_{n-1} \| n)$.

Algorand DRB does not satisfy either (θ, t, ℓ)-standard or strong pseudorandomness when $\theta > 0$. Intuitively, this is because the combine algorithm $\mathsf{CombRand}$ relies on selecting a single leader node to compute the next beacon value σ_n, facilitating an adversary the possibility of introducing bias whenever that node is corrupted. Assume the adversary corrupts nodes $C = \{N_1, \ldots, N_m\}$ with $0 < m \leq \theta$. The probability that a corrupted node is selected as leader for each round is at least $p = m/\ell$ which is non-negligible. Indeed, assume N_j is a corrupted node controlled by the adversary. When N_j is selected as the leader in the challenge query, the adversary is able to distinguish $\sigma^\star$ from a uniform random which breaks the standard/strong pseudorandomness. The above attack does not depend on whether the adversary knows the identity of the leader or not. The adversary can always guess successfully with probability at least p.

When $\theta = 0$, i.e., the adversary is not allowed to corrupt any node, the $(0, t, \ell)$-standard pseudorandomness of Algorand DRB holds because of the pseudorandomness of VRF. However, no corrupted node is a very strong assumption for any decentralised application where any node can participate. Furthermore, even if $\theta = 0$, Algorand DRB does not satisfy $(0, t, \ell)$-strong pseudorandomness. This is because strong pseudorandomness allows the adversary to query $\mathcal{O}^{\mathsf{PartialRand}}$ oracle in the challenge round. When i is the current leading node and i is honest, the adversary can query $\mathcal{O}^{\mathsf{PartialRand}(i)}$ to obtain the next beacon value which breaks the strong pseudorandomness. Note that this attack also does not rely on the knowledge of the leader. The total number of nodes is polynomial and the adversary can issue up to t queries $\mathsf{PartialRand}$ which gives the adversary a successful guessing

probability t/ℓ.

Finally, Algorand does not enjoy strong bias resistance. When the secret key of any honest node is leaked to the adversary, despite the fact that the adversary cannot change the values of $\rho_{n,i}$ for each honest node i, the adversary can adjust the set $I \subseteq \mathrm{QUAL}$ by including/excluding the partial outputs corresponding to corrupted nodes to bias the output of σ_n to the adversary's advantage.

The arguments above also apply to Ouroboros-Praos [27, Section 5] and Elrond [32, Section 11] DRBs.

Harmony. Harmony [40, Section 3.1] is based on verifiable random functions $\mathrm{VRF} = (\mathrm{VRF.KeyGen}, \mathrm{VRF.Eval}, \mathrm{VRF.Verify})$ [48], and verifiable delay function $\mathrm{VDF} = (\mathrm{VDF.Eval}, \mathrm{VDF.Verify})$ [12] and can be formalised as follows:

$\mathsf{CmteGen}(1^\lambda, t, \ell)$**:** This is a non-interactive protocol, in which each node in $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ creates a VRF key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathrm{VRF.KeyGen}(1^\lambda)$ for $i = 1, \ldots, \ell$. The set of qualified nodes $\mathrm{QUAL} \subseteq \mathcal{N}$ are those that contribute a VRF public key, and the committee public key $\mathsf{cpk} = (\{\mathsf{pk}_i\}_{i \in \mathrm{QUAL}}, \mathsf{pp})$, where pp is the global parameter for the VDF function.

$\mathsf{PartialRand}(\mathsf{st}_{n-1}, \mathsf{sk}_i, \mathsf{cpk})$**:** The algorithm computes a partial value for round n by running the VRF evaluation function $(\rho_{n,i}, \pi_{n,i}) \leftarrow \mathrm{VRF.Eval}(\mathsf{sk}_i, \mathsf{st}_{n-1})$. Output is $(i, \rho_{n,i}, \pi_{n,i})$.

$\mathsf{CombRand}(\mathsf{st}_{n-1}, \mathcal{E}, \mathsf{cpk})$**:** Parse $\mathcal{E} = \{(i, \rho_{n,i}, \pi_{n,i})\}_{i \in I}$ with $|I| > t$ and $I \subseteq \mathrm{QUAL}$.

1) Identify the maximum index subset $J \subseteq I$ such that $|J| > t$ and $\mathrm{VRF.Verify}(\mathsf{pk}_j, \mathsf{st}_{n-1}, \rho_{n,j}, \pi_{n,j}) = 1$ for each $j \in J$. Let $\pi_n^r = \{(\rho_{n,j}, \pi_{n,j})\}_{j \in J}$.
2) Compute $\rho_n = \oplus_{j \in J} \rho_{n,j}$ and $(\sigma_n, \pi_n^d) \leftarrow \mathrm{VDF.Eval}(\mathsf{pp}, \rho_n)$.
3) Output (σ_n, π_n) with $\pi_n = (\pi_n^r, \pi_n^d)$.

$\mathsf{VerifyRand}(\mathsf{st}_{n-1}, \sigma_n, \pi_n, \mathsf{cpk})$**:** Parse $\pi_n = (\pi_n^r, \pi_n^d)$ with $\pi_n^r = \{(\rho_{n,j}, \pi_{n,j})\}_{j \in J}$. If $|I| \leq t$ or $I \not\subseteq \mathrm{QUAL}$, output 0. If $\mathrm{VRF.Verify}(\mathsf{pk}_j, \mathsf{st}_{n-1}, \rho_{n,j}, \pi_{n,j}) = 1$ for all $j \in J$ and $\mathrm{VDF.Verify}(\mathsf{pp}, \oplus_{j \in J} \rho_{n,j}, \sigma_n, \pi_n^d) = 1$, output 1; else output 0.

$\mathsf{UpdState}(\mathsf{st}_{n-1}, \sigma_n)$**:** Output $\mathsf{st}_n = \sigma_n \| n$. The initial state $\mathsf{st}_0 = \sigma_0 \| 0$ where $\sigma_0 \in \mathrm{Ran}$ is a random chosen by the initial participants in the protocol.

Roughly speaking, in each round Harmony's DRB computes the XOR of at least $t + 1$ VRF evaluations from pairwise different nodes that is then fed into a verifiable delay function (VDF) to delay the computation of the final randomness for k blocks to prevent the last revealer attack [31]. Once ρ_n is committed into the blockchain, it cannot be modified by any node. To simplify the description, we let $k = 1$.

The fact that Harmony's DRB directly uses the result of VDF as beacon value guarantees that the output is unpredictable but *probably not pseudo-random* [12], [61]. As suggested in [12], a random oracle should be applied to the output of the VDF to obtain the beacon value. However, it is still a open question whether this combination is provably indifferentiable from a Random Delay Oracle [12]. We conclude then that Harmony's DRB does not satisfy either standard or strong pseudorandomness.

Interestingly, Harmony's DRB does not satisfy strong bias resistance neither, even with the VDF function in place. When these secret keys are leaked to the adversary, the adversary can pre-compute the partial values from the honest nodes and the VDF function. The adversary can choose the partial values to be included into the beacon value leading to a biased beacon value. We conclude that Harmony's DRB does not offer strong bias resistance.

HERB. The so-called Homomorphic Encryption Random Beacon construction [22], defines a DRB using DKG and ElGamal homomorphic properties, similarly to the random beacon used in Orbs' Helix consensus protocol [3], [4]. HERB first runs DKG among n nodes to establish a public key and the shares of the corresponding secret key spread among n nodes. Each node publishes an ElGamal encryption share of a secret along with NIZK of correct encryption. Once the publication phase is over, any node can check and aggregate the encryption shares into an aggregate ciphertext which can be subsequently decrypted by threshold of nodes. The decryption result is the final beacon value. HERB's beacon value is pseudorandom as long as one participating node is honest and selects the secret share randomly. Our preliminary analysis indicated that HERB satisfies both standard and strong pseudorandomness. However, HERB does not offer strong bias resistance when the secret keys of the honest nodes are leaked, since the adversary is able to perform decryption and obtain other nodes' secret shares before choosing his own secret share. A detailed description of HERB using our DRB framework is in Appendix G.

B. Comparison with interactive DRBs

SCRAPE [20], RandShare/RandHound [57] and Ouroboros [44] are interactive DRBs constructed using Publicly Verifiable Secret Sharing (PVSS). These protocols directly run PVSS protocol across multiple rounds to produce one beacon value at a time, and belong to a line of work initiated in [7]. We stress that this is a rather expensive approach to create beacon values since PVSS's communication and computation complexity is similar to a DKG protocol. In comparison, the DKG protocol in our DRB is initiated once and generates multiple random beacon values using non-interactive algorithms PartialRand and CombRand.

HydRand [55] builds upon SCRAPE and also uses PVSS protocol. HydRand's secret reconstruction process runs across multiple rounds. In each round n, a leader node is selected to open its PVSS secret value s_r previously committed in a round r where the node was last chosen as the leader, create a new commitment for a new secret s_n and use PVSS distribution protocol to generate encrypted shares for s_n. When the round leader does not reveal its previous secret s_r, then the other nodes jointly reconstruct the secret using PVSS reconstruction process. HydRand has a similar problem to Algorand: both

Random Beacon Protocol	Standard Pseudorandomness	Strong Pseudorandomness	Strong Bias Resistance	Unpredictability
Algorand [37]	(–)	✘	✘	✔
Elrond [32]	(–)	✘	✘	✔
Harmony [40]	✘	✘	✘	✔
HERB [22]	✔	✔	✘	✔
Orbs [4]	✔	✔	✘	✔
Ouroboros Praos [27]	(–)	✘	✘	✔
Dfinity-DRB [39]	✔	✘	✔	✔
DDH-DRB [This work]	✔	✔	✔	✔
GLOW-DRB [This work]	✔	✔	✔	✔

Table I: Comparison of Random Beacons with non-interactive beacon computation. (–) means the corresponding property holds when all nodes are honest.

protocols rely on selecting a leader to produce a random beacon value. The pseudorandomness of the beacon values are therefore not guaranteed when some of the nodes are corrupted. In comparison, our DRB can allow up to t corrupted nodes while still achieves standard/strong pseudorandomness. The tolerance of a certain number of corrupted nodes is essential for any decentralised application to avoid single-point of failures.

The results of our DRBs comparison is summarized in Table I.

VII. Performance Evaluation

We compare the efficiency of different DVRF implementations by benchmarking the time required to generate a single random value for DRB purposes. The protocols, and associated curves, studied in the reference implementation [33] are shown in Table II, where the protocols are implemented using mcl library [49], RELIC [2], Libsodium [10].

The results are summarised in Table II, which shows the average time for each protocol to generate a single random value based on the average execution time of the functions PartialEval and Combine. Benchmarking was done using Catch2 [52] on a laptop running Ubuntu 18.04. LTS with 64bit Intel Core i7-8550U processor, with 4GHz processor speed, and 16GiB of memory. We observe that the DDH-DVRF protocol with Ristretto255 outperforms the Dfinity-DVRF protocol with curves offering the same 128-bit security level [2] by approximately a factor of 5. In the case of BN256, which has the same prime field size that Ristretto255 but lower security level, the times for random value generation are 1.5 slower if using the mcl library, and slower by over a factor of 10 if using RELIC. Between the protocols with compact proofs, the GLOW-DVRF protocol outperforms the Dfinity-DVRF implementation on randomness generation for the same curve by at least a factor of 2.5, and the highest performing implementation, disregarding other factors, is therefore the GLOW-DVRF protocol with curve BN256. However, comparing implementations with the same security level the DDH-DVRF protocol produces the fastest times. The benchmarks discussed in this section can be reproduced using the reference implementation [33] licensed under Apache 2.0. Further details on the implementation are given in Appendix A.

To further illustrate the superior performance and practicality of our DVRF-based DRB constructions, we have developed a ledger prototype with Tendermint-based state machine replication [16] using Cosmos SDK [41]. In our prototype validator nodes act as DRB nodes, and the block proposer stores in the block each DRB's round random value and corresponding proof using a special transaction. The first random value appearing in a block is used to select the leader for Tendermint consensus for the next block. We evaluate how long it takes on average for at least 2/3 of the total number of validators to agree on a random value (which guarantees that the corresponding beacon outputs will be stored eventually in a block), where the beacon evaluation shares are propagated amongst the nodes using the underlying P2P network of Tendermint/Cosmos. The experiments have been run using the GLOW-DVRF and Dfinity-DVRF-based DRB protocols on Amazon EC2 t2.micro instances (1 GiB of RAM, one virtual CPU core, 60-80 Mbit/s network bandwidth) in the same AWS region but an artificial delay of 200ms was added to each of the nodes. Experiments were performed with correct nodes only, as well as considering up to 15% simulated node failures.

The benchmarks obtained show in particular how the DRBs obtained through the DVRF-based generic construction outperform random beacons where randomness generation typically takes several rounds of communication [20], [57], [44], [55]. For instance, while the recently presented Hydrand DRB [55] generates for a total of 64 nodes an average of 13 random values per minute in a platform similar to the one used in our experiments, our GLOW-DVRF-based DRB generates approximately 100 random outputs per minute, which represents approximately x8 better throughput. With regards to verification performance, Hydrand reports that an external client can publicly verify the correctness of a round's random beacon in 57ms with a proof size of 26624 bytes in a setting with 128 nodes, whereas for both our pairing-based DRBs verification takes approximately 1.8ms with a proof size of 48 bytes (independently of the number of nodes). Even for our DDH-DVRF-based DRB with non-compact proofs, the proof size is only 6464 bytes for a simlar number of nodes.

[2]See https://tools.ietf.org/id/draft-yonezawa-pairing-friendly-curves-00.html for a discussion of revised security strength of pairing-based cryptographic assumptions.

Protocol	Curve	Library	Security Level	Proof size (bytes)	Randomness Generation (ms)	Time Ratio	Assumption
GLOW-DVRF	BN256	mcl	100	32	7.38	0.69	co-CDH XDH
	BLS12-381	mcl	128	48	18.67	1.75	
	BN384	mcl	128	48	21.39	2.00	
	BN_P256	RELIC	100	33	33.16	3.10	
DDH-DVRF	Ristretto255	Libsodium	128	1664	10.70	1	DDH
	Curve25519	RELIC	128	1664	65.97	6.17	
Dfinity-DVRF	BN256	mcl	100	32	18.81	1.76	co-CDH
	BLS12-381	mcl	128	48	55.79	5.22	
	BN384	mcl	128	48	60.73	5.68	
	BN_P256	RELIC	100	33	138.36	12.94	

Table II: Time ratios per individual node for each protocol to generate one round of entropy for total number of nodes $\ell = 50$, with threshold value $t = 25$. These figures do not take into account latency due to communicating partial evaluations between nodes.

VIII. Extension to Threshold BLS signatures with Improved Security and Efficiency

Our technique of using NIZKs to validate partial evaluations in GLOW-DVRF can also be applied to modify the well-known threshold BLS signature scheme [11] in order to achieve stronger security and better efficiency in the most time-consuming operation, i.e. reconstruction of a signature from individual signature shares.

Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ be a bilinear pairing, where g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively. Let $H_1 : \{0,1\}^* \mapsto \mathbb{G}_1$ and $H_3 : \{0,1\}^* \mapsto \mathbb{Z}_q$ be hash functions. The syntax of threshold digital signatures and description of the corresponding unforgeability definitions are deferred to Appendix H. The modified threshold BLS signature that we call *threshold* GLOW *signature* works as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ runs in the same way as GLOW-DVRF (see Figure 3 in the Appendix) to generate individual party's signing key $\mathsf{sk}_i \in \mathbb{Z}_q$, verification key $\mathsf{vk}_i \in \mathbb{G}_1$ for $i = 1, \ldots, \ell$, and the global public key $\mathsf{pk} \in \mathbb{G}_2$.

$\mathsf{PartialSign}(\mathsf{sk}_i, x)$ outputs a partial signature (i, σ_i, π_i) with $\sigma_i = H_1(x)^{\mathsf{sk}_i} \in \mathbb{G}_1$ and a NIZK proof $\pi_i \leftarrow \mathsf{PrEq}_{H_3}(g, \mathsf{vk}_i, H_1(x), v_i)$ to show that partial signature σ_i has been correctly computed.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ verifies a set of partial signatures $\mathcal{E} = \{s_x^{j_1}, \ldots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \geq t+1$ originating from $|\mathcal{E}|$ different servers, and obtains from vk individual verification keys $\mathsf{vk}_{j_1}, \ldots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,

1) Identifies an index subset $I = \{i_1, \ldots, i_{t+1}\}$ such that for every $i \in I$ it holds that $\mathsf{VerifyEq}_{H_3}(g, \mathsf{vk}_i, H_1(x), \sigma_i, \pi_i) = \texttt{accept}$, where $(i, \sigma_i, \pi_i) \leftarrow s_x^i$. If no such subset exists, outputs `reject`.
2) Outputs $\sigma \leftarrow \prod_{j \in I} \sigma_j^{\lambda_{0,j,I}}$.

$\mathsf{Verify}(\mathsf{pk}, x, \sigma)$ validates σ by checking $e(\sigma, g_2) = e(H_1(x), \mathsf{pk})$.

In contrast to threshold BLS signatures [11] where individual verification keys vk_i belong to $\mathbb{G}_2$, individual verification keys vk_i belong to $\mathbb{G}_1$ in threshold GLOW, where $i = 1, \ldots, \ell$. This presents two main benefits: firstly, partial signatures validity is checked by a NIZK proof of equality of discrete logarithms, instead of computing a pairing equation as in threshold BLS, which makes the computation for the Combine phase to be between x2.5 to x4.5 faster than threshold BLS. Secondly, we can extend the definition of unforgeability [11] to *strong unforgeability*, by allowing the adversary to query partial signatures on challenge message $x^\star$, and show that threshold GLOW signatures meet this stronger security notion. In contrast, it is not known whether threshold BLS signatures satisfy strong unforgeability: similary to the strong pseudorandomness case of Dfinity-DVRF, the techniques used in the unforgeability proof of threshold-BLS do not trivially extend to prove strong unforgeability.

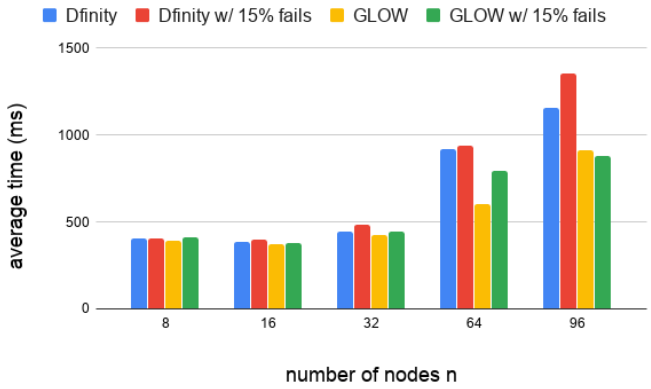

Table III: Average randomness generation time for different numbers of nodes n with no failures and 15% simulated failures on a Cosmos/Tendermint-compatible ledger for Dfinity-DVRF and GLOW-DVRF-based DRBs

We note that our improved unforgeability definition generalises the unforgeability definition for threshold signatures in the presence of a trusted dealer that can be traced back to Shoup [56]. Furthermore, it is possible to show a separation result for strong/standard unforgeability, i.e. there exist schemes that satisfy standard unforgeability but do not satisfy strong unforgeability. Hence, strong unforgeability offers a *stronger* security level than standard unforgeability. The corresponding separation result is given in Appendix H.

The security reduction of GLOW-DVRF can be easily adapted to show that the threshold GLOW signature scheme satisfies strong unforgeability under XDH assumption and co-CDH assumption. The proof of strong unforgeability is similar to the proof of strong pseudorandomness of GLOW-DVRF in

Theorem IV.5: the answers to the queries of partial signatures on $x^\star$ can be simulated under XDH assumption and when the adversary found a forgery $H(x^\star)^{\mathsf{sk}} = g_1^{\alpha\beta}$ it breaks the co-CDH assumption.

Acknowledgments. We thank Jérome Maloberti for his decisive contribution to architecting our VRF implementations, Nathan Hutton for providing us with the experimental data on our DVRF-based DRBs and Jonathan Ward for comments on this manuscript.

REFERENCES

[1] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed Symmetric-key Encryption. In *CCS 2018*, pages 1993–2010. ACM, 2018.

[2] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient Library for Cryptography. https://github.com/relic-toolkit/relic, 2014.

[3] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. A fair consensus protocol for transaction ordering. In *2018 IEEE 26th International Conference on Network Protocols, ICNP 2018*, pages 55–65. IEEE Computer Society, 2018.

[4] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. Helix: A scalable and fair consensus algorithm resistant to ordering manipulation. *IACR Cryptol. ePrint Arch.*, 2018:863, 2018.

[5] Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Winning the caucus race: Continuous leader election via public randomness. *CoRR*, abs/1801.07965, 2018.

[6] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *J. Cryptology*, 32(4):1298–1336, 2019.

[7] Mihir Bellare, Juan A. Garay, and Tal Rabin. Distributed pseudo-random bit generators - A new way to speed-up shared coin tossing. In *Proceedings of PODC 1996*, pages 191–200. ACM, 1996.

[8] Mihir Bellare and Shafi Goldwasser. New paradigms for digital signatures and message authentication based on non-interative zero knowledge proofs. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 194–211. Springer, 1989.

[9] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

[10] Daniel J. Bernstein and Frank Denis. Libsodium - a modern, portable, easy to use crypto library. https://github.com/jedisct1/libsodium, 2019.

[11] Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *PKC 2003*, pages 31–46, 2002.

[12] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable Delay Functions. In *CRYPTO 2018*, volume 10991, pages 757–788, 2018.

[13] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT 2018*, pages 435–464, 2018.

[14] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key Homomorphic PRFs and Their Applications. In *CRYPTO 2013*, pages 410–428, 2013.

[15] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT 2001*, volume 2248, pages 514–532, 2001.

[16] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[17] Vitalik Buterin. Ethereum 2.0 Mauve Paper. https://cdn.hackaday.io/files/10879465447136/MauvePaperVitalik.pdf, 2018.

[18] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO 2001*, pages 524–541, 2001.

[19] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *EUROCRYPT 2005*, pages 302–321, 2005.

[20] Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *ACNS*, pages 537–556, 2017.

[21] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *CRYPTO' 92*, pages 89–105, 1993.

[22] Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. Homomorphic encryption random beacon. *IACR Cryptol. ePrint Arch.*, 2019:1320, 2019.

[23] Cloudflare. Decentralized Verifiable Randomness Beacon. https://developers.cloudflare.com/randomness-beacon/, 2019.

[24] Corestar. Corestar Arcade: Tendermint-based Byzantine Fault Tolerant (BFT) middleware with an embedded BLS-based random beacon. https://github.com/corestario/tendermint, 2019.

[25] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed ElGamal à la Pedersen: Application to Helios. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013*, pages 131–142. ACM, 2013.

[26] DAOBet. DAOBet (ex — DAO.Casino) to Deliver On-Chain Random Beacon Based on BLS Cryptography. https://daobet.org/blog/on-chain-random-generator/, 2019.

[27] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018*, volume 10821, pages 66–98, 2018.

[28] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing Cryptographic Pairings over Barreto-Naehrig Curves. In *Pairing*, pages 197–207, 2007.

[29] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In *Public Key Cryptography - PKC 2003*, volume 2567, pages 1–17. Springer, 2003.

[30] Yevgeniy Dodis and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In *Public Key Cryptography - PKC 2005*, pages 416–431, 2005.

[31] Paul Dworzanski. A note on committee random number generation, commit-reveal, andlast-revealer attacks. http://paul.oemm.org/commit_reveal_subcommittees.pdf.

[32] Team Elrond. Elrond: A highly scalable public blockchain via adaptive state shardingand secure proof of stake, 2019. https://elrond.com/assets/files/elrond-whitepaper.pdf.

[33] Fetch.ai. Distributed Verifiable Random Functions: an Enabler of Decentralized Random Beacons. https://github.com/fetchai/research-dvrf, 2020.

[34] Matthew K. Franklin and Haibin Zhang. Unique Group Signatures. In *ESORICS 2012*, volume 7459, pages 643–660, 2012.

[35] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.

[36] C. C. F. Pereira Geovandro, Marcos A. Simplício Jr., Michael Naehrig, and Paulo S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319–1326, 2011.

[37] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. SOSP '17, pages 51–68, 2017.

[38] Sharon Goldberg, Leonid Reyzin, Dimitrios Papadopoulos, and Jan Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-irtf-cfrg-vrf-03, Internet Engineering Task Force, September 2018. Work in Progress.

[39] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

[40] Team Harmony. Technical Whitepaper - version 2.0. https://harmony.one/whitepaper.pdf.

[41] Tendermint Inc. Cosmos SDK Documentation. Cryptology ePrint Archive, Report 2013/177. https://docs.cosmos.network/.

[42] Keep. The Keep Random Beacon: An Implementation of a Threshold Relay. http://docs.keep.network/random-beacon/, 2019.

[43] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. *Advances in Cryptology - CRYPTO 2020*.

[44] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 357–388, 2017.

[45] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *IEEE Symposium on Security and Privacy*, pages 583–598, 2018.
[46] Veronika Kuchta and Mark Manulis. Unique aggregate signatures with applications to distributed verifiable random functions. In *CANS 2013*, volume 8257, pages 251–270, 2013.
[47] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.
[48] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS '99*, pages 120–130. IEEE Computer Society, 1999.
[49] Shigeo Mistunari. mcl - A portable and fast pairing-based cryptography library. https://github.com/herumi/mcl, 2019.
[50] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. http://bitcoin.org/bitcoin.pdf.
[51] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and kdcs. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346. Springer, 1999.
[52] Phil Nash. Catch2. https://github.com/catchorg/Catch2, 2019.
[53] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO 1991*, pages 129–140, 1991.
[54] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Ethdkg: Distributed key generation with ethereum smart contracts. 2019. https://eprint.iacr.org/2019/985.
[55] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. Hydrand: Practical continuous distributed randomness. *IEEE S&P*, 2020.
[56] Victor Shoup. Practical threshold signatures. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000.
[57] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy, SP 2017*, pages 444–460. IEEE Computer Society, 2017.
[58] Hong Wang, Yuqing Zhang, and Dengguo Feng. Short threshold signature schemes without random oracles. In *Progress in Cryptology - INDOCRYPT 2005*, volume 3797 of *Lecture Notes in Computer Science*, pages 297–310. Springer, 2005.
[59] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019*, pages 95–112. USENIX Association, 2019.
[60] Rhys Weatherley. Noise-C, a plain C implementation of the Noise protocol. https://github.com/rweather/noise-c, 2016.
[61] Benjamin Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT 2019*, pages 379–407, 2019.
[62] Gavin Wood. Ethereum: Secure decentralised generalised and transaction ledger, 2019. https://ethereum.github.io/yellowpaper/paper.pdf.

APPENDIX

A. Performance Evaluation (Continued)

1) Theoretical performance: In Table IV, we give analytical measurements for the size of the output of the partial evaluation and the combine algorithms of the DVRFs studied in this paper. The size is represented using the number of group elements.

In Table V, we give analytical measurements for the computational costs of the secure DKG protocol, the partial evaluation algorithm, the combine algorithm and the verification of the pseudorandom value. The costs are computed using the number of group operations $\exp_{\mathbb{G}}, \mathsf{mul}_{\mathbb{G}}, \mathsf{pairing}$ which represent the exponentiation in $\mathbb{G}$, the multiplication in $\mathbb{G}$ and the pairing operation, respectively.

2) Implementation: The benchmarks presented in Table VI can be independently verified by using the reference implementation [33] licensed under Apache 2.0. The source code provided also allows for separate benchmarking of the DistKG phase and randomness generation with message passing for all curves listed in Table VI. The communication between nodes is implemented either locally, by means of a scheduler, or using network connections. For the former all nodes must reside within the same process, and can be used for running the DistKG and DRB with simulated network latency. A simple implementation of the latency, where each node's latency is sampled from a Gamma distribution with some chosen mean, is currently available. In the latter case of network connections the nodes can be run on different computers identified by their IP address. Each node has an ECDSA and Diffie-Hellman key pair, which are used to sign all outgoing messages and to set up the point-to-point secure channels using Noise-C [60], respectively. We assume both the ECDSA and Diffie-Hellman public keys for each node are synchronised using a trusted third party. In addition to the secure point-to-point channels, the DistKG additionally provides a secure bulletin for broadcasting information to all participants, that has been implemented using the reliable broadcast protocol described in [18], and that is of independent interest.

B. Cryptographic assumptions

In the following we recall the definitions of two variants of the Decisional Diffie-Hellman assumption, the Discrete Logarithm assumption and Lagrange coefficients for polynomial interpolation.

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

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

is negligible in λ.

Definition A.2 (Computational co-CDH assumption [13])**.** *Let*

$$X \leftarrow \left(\mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^{\alpha}, g_1^{\beta}, g_2^{\alpha}\right)$$

where $\langle \mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q\rangle \leftarrow \mathcal{IG}(1^{\lambda})$ and $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q^*$ and $g_1 \xleftarrow{\$} \mathbb{G}_1, g_2 \xleftarrow{\$} \mathbb{G}_2$. We say that $\mathcal{IG}$ satisfies the Computational co-CDH assumption in $\mathbb{G}_1$ *if*

$$\mathbf{Adv}^{\text{co-CDH}}_{\mathcal{IG},\mathcal{A}}(\lambda) := \Pr\left[\mathcal{A}(X) = g_1^{\alpha\beta}\right]$$

is negligible in λ.

Definition A.3 (XDH assumption [19])**.** *Let $\langle \mathbf{e}(\cdot,\cdot), \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q\rangle \leftarrow \mathcal{IG}(1^{\lambda})$ be a a bilinear mapping. The XDH assumption states that DDH is hard in $\mathbb{G}_1$.*

Protocol	GLOW-DVRF	DDH-DVRF	Dfinity-DVRF
PartialEval	$1 \cdot \mathbb{G}_1 + 2 \cdot \mathbb{Z}_q$	$1 \cdot \mathbb{G} + 2 \cdot \mathbb{Z}_q$	$1 \cdot \mathbb{G}_1$
Combine	$1 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{Z}_q$	$1 \cdot \mathbb{G} + 2(t+1) \cdot \mathbb{Z}_q$	$1 \cdot \mathbb{G}_1 + 1 \cdot \mathbb{Z}_q$

Table IV: Size of the output of PartialEval and Combine algorithm

Operations		GLOW-DVRF	DDH-DVRF	Dfinity-DVRF
DistKG	Gen.	$(2t+3+\ell(t+2)) \cdot \mathsf{exp}_{\mathbb{G}_1}$ $+(t+1)(\ell+1) \cdot \mathsf{mul}_{\mathbb{G}_1}$	$(2t+3+\ell(t+2)) \cdot \mathsf{exp}_{\mathbb{G}}$ $+(t+1)(\ell+1) \cdot \mathsf{mul}_{\mathbb{G}}$	$(2t+3+\ell(t+2)) \cdot \mathsf{exp}_{\mathbb{G}_2}$ $+(t+1)(\ell+1) \cdot \mathsf{mul}_{\mathbb{G}_2}$
	Extra.	$(2nt+n+1) \cdot \mathsf{exp}_{\mathbb{G}_1}$ $+(3nt+2n-2t-2) \cdot \mathsf{mul}_{\mathbb{G}_1}$ $+(n-1) \cdot \mathsf{mul}_{\mathbb{G}_2} + 2n \cdot \mathsf{pairing}$	$(2nt+n+1) \cdot \mathsf{exp}_{\mathbb{G}}$ $+(3nt+2n-2t-2) \cdot \mathsf{mul}_{\mathbb{G}}$	$(2nt+n+1) \cdot \mathsf{exp}_{\mathbb{G}_2}$ $+(3nt+2n-2t-2) \cdot \mathsf{mul}_{\mathbb{G}_2}$
PartialEval		$3 \cdot \mathsf{exp}_{\mathbb{G}_1}$	$3 \cdot \mathsf{exp}_{\mathbb{G}}$	$1 \cdot \mathsf{exp}_{\mathbb{G}_1}$
Combine		$(4k+t+1) \cdot \mathsf{exp}_{\mathbb{G}_1} + t \cdot \mathsf{mul}_{\mathbb{G}_1}$	$(4k+t+1) \cdot \mathsf{exp}_{\mathbb{G}} + t \cdot \mathsf{mul}_{\mathbb{G}}$	$(t+1) \cdot \mathsf{exp}_{\mathbb{G}_1} + t \cdot \mathsf{mul}_{\mathbb{G}_1}$ $+2k \cdot \mathsf{pairing}$
Verify		$2 \cdot \mathsf{pairing}$	$5(t+1) \cdot \mathsf{exp}_{\mathbb{G}} + t \cdot \mathsf{mul}_{\mathbb{G}}$	$2 \cdot \mathsf{pairing}$

Table V: Theoretical computational costs. Notations: ℓ represents the total number of participating nodes. $n = |\mathrm{QUAL}|$ is the number of qualified nodes. $k = |\mathcal{E}|$ is the number of partial evaluations with $k \geq t+1$ given to the Combine algorithm. For the DistKG sub-protocol, we compare the number of computations that each node performs in the Generating phase and the Extracting phase.

Protocol	Curve	Library	Proof size (bytes)	Randomness Generation (ms)	Time Ratio
GLOW-DVRF	BN256	mcl	32	7.38	0.69
	BLS12-381	mcl	48	18.67	1.75
	BN384	mcl	48	21.39	2.00
	BN_P256	RELIC	33	33.16	3.10
DDH-DVRF	Ristretto255	Libsodium	1664	10.70	1
	Curve25519	RELIC	1664	65.97	6.17
Dfinity-DVRF	BN256	mcl	32	18.81	1.76
	BLS12-381	mcl	48	55.79	5.22
	BN384	mcl	48	60.73	5.68
	BN_P256	RELIC	33	138.36	12.94

Protocol	Curve	Library	Proof size (bytes)	Randomness Generation (ms)	Time Ratio
GLOW-DVRF	BN256	mcl	32	29.06	0.68
	BLS12-381	mcl	48	71.91	1.68
	BN384	mcl	48	80.15	1.87
	BN_P256	RELIC	33	132.92	3.10
DDH-DVRF	Ristretto255	Libsodium	6464	42.91	1
	Curve25519	RELIC	6464	285.59	6.66
Dfinity-DVRF	BN256	mcl	32	74.01	1.72
	BLS12-381	mcl	48	215.34	5.02
	BN384	mcl	48	228.54	5.33
	BN_P256	RELIC	33	566.21	13.20

Table VI: Time ratios per individual node for each protocol to generate one round of entropy for total number of nodes $\ell = 50$ (upper table) and $\ell = 200$ (lower table), with threshold value $t = \ell/2$. The time ratio is computed as the time to generate a single random value for each protocol divided by the observed time for DDH-DVRF, which is the DVRF offering higher security, implemented using LibSodium.

C. Verifiable Random Function (VRF)

Formally, a Verifiable Random Function (VRF) $\mathcal{V} = (\mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Verify})$ consists of the following algorithms [48]:

$\mathsf{KeyGen}(1^\lambda)$ is a *key generation* algorithm that takes as input a security parameter 1^λ; it outputs a public key pk and a secret key sk.

$\mathsf{Eval}(x, \mathsf{sk})$ is an *evaluation algorithm* that takes as input secret key sk, a message x, and outputs a triple $(x, F_{\mathsf{sk}}(x), \pi)$, where $F_{\mathsf{sk}}(x)$ is the function's evaluation on input x and π is a non-interactive proof of correctness.

$\mathsf{Verify}(\mathsf{pk}, x, v, \pi)$ is a *verification algorithm* that takes as input the public key pk, a message x, a value v and a proof π and outputs 0/1.

D. Additional DVRF Definitions

Definition A.4 (Consistency)**.** *A t-out-of-ℓ* DVRF $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ *is said to be* consistent *if: for any* $(\mathrm{QUAL}, \mathsf{pk}, \mathcal{VK}, \mathcal{SK}) \leftarrow \mathsf{DistKG}(1^\lambda, t, \ell)$ *run by ℓ nodes among which up to t nodes are corrupted, any plaintext* $x \in \mathrm{Dom}$*, any two subsets* $U, U' \subseteq \mathrm{QUAL}$ *of size at least* $t+1$*, it holds that* $(v, \pi) \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \{(i, v_i, \pi_i)\}_{i \in U})$*,* $(v', \pi') \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \{(j, v'_j, \pi'_j)\}_{j \in U'})$ *and* $v = v' \neq \perp$*, where* $(i, v_i, \pi_i) \leftarrow \mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ *for each* $i \in U$*,* $(j, v'_j, \pi'_j) \leftarrow \mathsf{PartialEval}(x, \mathsf{sk}_j, \mathsf{vk}_j)$ *for each* $j \in U'$*.*

Robustness ensures the availability of computing the random function value on any admissible plaintext in an adversarial environment. Robustness has been also called *guaranteed output delivery* (G.O.D.) in recent works [20], [44]:

Definition A.5 (Robustness)**.** *A t-out-of-ℓ* DVRF *protocol*

$\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ *on nodes* $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ *satisfies* robustness *if for all PPT adversaries* $\mathcal{A}$*, the following experiment outputs 1 with negligible probability.*

[Corruption] *$\mathcal{A}$ chooses a collection C of nodes to be corrupted with $|C| \leq t$. Adversary $\mathcal{A}$ acts on behalf of corrupted nodes, while the challenger acts on behalf of the remaining nodes, which behave honestly (namely they follow the protocol specification).*

[Initialization] *Challenger and adversary engage in running the distributed key generation protocol* $\mathsf{DistKG}(1^\lambda, t, \ell)$*. After this phase, the protocol establishes a qualified set of nodes* QUAL*. Every (honest) node $N_j \in \mathrm{QUAL} \setminus C$ obtains a key pair $(\mathsf{sk}_j, \mathsf{vk}_j)$. In contrast, (corrupted) nodes $N_j \in C$ end up with key pairs $(\mathsf{sk}_j, \mathsf{vk}_j)$ in which one of keys may be undefined (i.e. either $\mathsf{sk}_j = \perp$ or $\mathsf{vk}_j = \perp$). At the end of this phase, the global public key pk and the verification keys vector $\mathcal{VK}$ is known by both the challenger and the attacker.*

[Query] *In response to $\mathcal{A}$'s evaluation query (Eval, x, i) for some honest node $N_i \in \mathrm{QUAL} \setminus C$ and plaintext $x \in \mathrm{Dom}$, the challenger returns $s_x^i \leftarrow \mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$. In any other case, the challenger returns $\perp$.*

[Challenge] *The challenger receives from $\mathcal{A}$ a set $U \subseteq \mathrm{QUAL}$, of size at least $t+1$, a plaintext $x^\star \in \mathrm{Dom}$ and a set of evaluation shares $\{(i, v_i, \pi_i)\}_{i \in U \cap C}$ corresponding to nodes under adversarial control. Challenger proceeds to compute the partial evaluations corresponding to honest nodes as $(i, v_i, \pi_i) \leftarrow \mathsf{PartialEval}(x^\star, \mathsf{sk}_i, \mathsf{vk}_i)$ for $i \in U \setminus C$. Let $(v^\star, \pi^\star) \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x^\star, \{(i, v_i, \pi_i)\}_{i \in U})$. If $v^\star \neq \perp$ and $\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x^\star, v^\star, \pi^\star) = 0$ then output 1; else, output 0.*

In the above experiment the output value $v^\star$ is obtained by running the combine function on a set that contains adversarial inputs. Robustness demands that if the combine function does not return $\perp$ then its output must pass the verification test even in the presence of the adversary's inputs.

Finally, *uniqueness* guarantees that it is infeasible for any adversary to compute two different output values v, v' and a plaintext $x \in \mathrm{Dom}$ such that both values pass the verification test wrt x, even when the secret keys of the honest nodes are leaked.

Definition A.6 (Uniqueness)**.** *A t-out-of-ℓ* DVRF *protocol $\mathcal{V}$ satisfies* uniqueness *if for all PPT adversaries $\mathcal{A}$, the following experiment outputs 1 with negligible probability.*

[Corruption and Initialization] *these two phases are defined exactly as in Definition A.5 (Robustness).*

[Query] *The adversary $\mathcal{A}$ can issue evaluation query and key revealing query.*

- *In response to $\mathcal{A}$'s evaluation query (Eval, x, i) for some honest node $N_i \in \mathrm{QUAL} \setminus C$ and plaintext $x \in \mathrm{Dom}$, the challenger returns $s_x^i \leftarrow \mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$. In any other case, the challenger returns $\perp$.*
- *In response to $\mathcal{A}$'s key revealing query (KeyRev, j) for some honest node $N_j \in \mathrm{QUAL} \setminus C$, the challenger returns sk_j.*

[Challenge] *The challenger receives from the adversary $\mathcal{A}$, a plaintext $x^\star \in \mathrm{Dom}$, two values $v, v' \in \mathrm{Ran}$ and two proofs π, π'. If $v \neq v'$ and $\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v, \pi) = \mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v', \pi') = 1$ then output 1; else, output 0.*

E. Additional DRB Security Definitions and Proofs

Correctness for DRB**.** We shall define the correctness for DRB below. In the definition, the adversary is given access to the oracles $\mathcal{O}^{\mathsf{PartialRand}}, \mathcal{O}^{\mathsf{Update}}$ as described above. The correctness guarantees that the output of CombRand is valid in the presence of the adversary's inputs.

Definition A.7 (Correctness of DRB)**.** *A t-out-of-ℓ* DRB *protocol $\mathcal{R} = (\mathsf{CmteGen}, \mathsf{PartialRand}, \mathsf{CombRand}, \mathsf{VerifyRand}, \mathsf{UpdState})$ on a set of nodes $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ satisfies* correctness *if for any PPT adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following experiment outputs 1 with probability at most $\mathsf{negl}(\lambda)$.*

[Corruption] *A correctness adversary $\mathcal{A}$ chooses a collection C of nodes to be corrupted with $|C| \leq t$.*

[Initialization] *The challenger runs the oracle $\mathcal{O}^{\mathsf{Init}(C)}$ to set up a committee by interacting with the adversary, and also initialises the global variables.*

[Queries] *$\mathcal{A}$ is given access to oracles $\mathcal{O}^{\mathsf{PartialRand}}$ and $\mathcal{O}^{\mathsf{Update}}$ described above.*

[Challenge] *The challenger receives from the adversary $\mathcal{A}$ a set U of size at least $t+1$ and $U \subseteq \mathrm{QUAL}$, and a set of partial values $\{(i, \sigma_i, \pi_i)\}_{i \in U \cap C}$. Let the current state be $\mathsf{st} = x^\star$. Compute $(i, \sigma_i, \pi_i) \leftarrow \mathsf{PartialRand}(x^\star, \mathsf{sk}_i, \mathsf{cpk})$ for $i \in U \setminus C$. Let $(\sigma^\star, \pi^\star) \leftarrow \mathsf{CombRand}(x^\star, \{(i, \sigma_i, \pi_i)\}_{i \in U}, \mathsf{cpk})$. If $\sigma^\star \neq \perp$ and $\mathsf{VerifyRand}(x^\star, \sigma^\star, \pi^\star, \mathsf{cpk}) = 0$ then output 1; else, output 0.*

Theorem A.1. *$\mathcal{V}^{\mathrm{DRB}}$ is correct if $\mathcal{V}^{\mathrm{DVRF}}$ is robust.*

Proof. For correctness, suppose there is a PPT adversary $\mathcal{A}$ that breaks the DRB correctness experiment with non-negligible probability. We shall construct a PPT adversary $\mathcal{B}$ which breaks the DVRF provability.

$\mathcal{A}$ chooses to corrupt a set of nodes C with $C \subseteq \mathcal{N}$ and $|C| \leq t$. $\mathcal{B}$ forwards C to its DVRF correctness challenger to run the interactive protocol DistKG to create a committee. $\mathcal{B}$'s DVRF challenger runs on behalf of the honest nodes in $\mathcal{N}$ and $\mathcal{A}$ runs on behalf of the bad nodes C. $\mathcal{B}$ acts as a message forwarder between $\mathcal{B}$'s challenger and $\mathcal{A}$. After that, $\mathcal{A}$ can continue to query the oracles $\mathcal{O}^{\mathsf{PartialRand}}$ and $\mathcal{O}^{\mathsf{Update}}$. Any $\mathcal{O}^{\mathsf{PartialRand}}$ query by $\mathcal{A}$ can be answered by $\mathcal{B}$'s DVRF evaluation oracle. $\mathcal{A}$ outputs $(U, \mathcal{E} = \{(i, \sigma_i, \pi_i)\}_{i \in U \cap C})$. Let the current state be $\mathsf{st} = x^\star$. $\mathcal{B}$ issues a challenge query $(x^\star, U, \mathcal{E})$ to its DVRF challenger. From the definitions of correctness of DRB and DVRF, we can easily see that when

$\mathcal{A}$'s challenger outputs 1, $\mathcal{B}$'s challenger will also output 1. This concludes the proof. □

Theorem A.2. *$\mathcal{R}^{\mathrm{DRB}}$ achieves (θ, t, ℓ)-standard (resp. strong) pseudorandomness if $\mathcal{V}^{\mathrm{DVRF}}$ achieves (θ, t, ℓ)-standard (resp. strong) pseudorandomness.*

Proof. Given a PPT adversary $\mathcal{A}$ that breaks the standard pseudorandomness of $\mathcal{R}^{\mathrm{DRB}}$ with non-negligible probability, we shall construct a PPT adversary $\mathcal{B}$ that breaks the standard pseudorandomness of $\mathcal{V}^{\mathrm{DVRF}}$ using $\mathcal{A}$ as a subroutine.

$\mathcal{A}$ chooses to corrupt a collection C of nodes in $\mathcal{N}$ with $|C| \leq \theta$. $\mathcal{B}$ forwards C to its DVRF challenger who will start the interactive DistKG protocol. The DVRF challenger runs on behalf of the honest nodes in $\mathcal{N}$, and $\mathcal{A}$ runs on behalf of the bad nodes in C. $\mathcal{B}$ acts as a message forwarder between the DVRF challenger and the adversary $\mathcal{A}$ in the DistKG protocol. At the end of the protocol, $\mathcal{B}$ can learn the set QUAL of qualified nodes, the verification keys $\mathcal{VK}$ and the global public key pk. $\mathcal{B}$ sets $\mathsf{ck} = (\mathcal{N}, C, \mathrm{QUAL}, \mathsf{cpk}, \perp)$ with $\mathsf{cpk} = (\mathsf{pk}, \mathcal{VK})$.

$\mathcal{B}$ can answer $\mathcal{A}$'s queries of $\mathcal{O}^{\mathsf{Update}}$ easily since these queries do not require any secret information that $\mathcal{B}$ does not have. $\mathcal{B}$ can answer $\mathcal{O}^{\mathsf{PartialRand}}(i)$ queries by querying $\mathcal{B}$'s DVRF evaluation oracle on $(\mathsf{Eval}, \mathsf{st}, i)$ to obtain the partial value and forward the result to $\mathcal{A}$.

In the challenge query, $\mathcal{B}$ receives $\mathcal{E} = \{(i, \sigma_i, \pi_i)\}_{i \in U \cap C}$ from $\mathcal{A}$. $\mathcal{B}$ answers the challenge query as below:

- Let $\mathsf{st} = x^\star$ and $\mathsf{rn} = j^\star$. $\mathcal{B}$ issues a challenge query on $(x^\star, \mathcal{E})$ to its DVRF challenger and receives a value δ from the DVRF challeger.
- $\mathcal{B}$ then uses δ to update the state and increases the round number.

Since the state for the $j^\star$-th round is of the form $x^\star = \sigma' \| j^\star$ which is an unique value and has never been used before. After the challenge round, the state is updated to a new value $\mathsf{st} = \delta^\star \| (j^\star + 1)$ which is also a unique value and never repeats. Therefore, the adversary $\mathcal{A}$ does not query $\mathsf{PartialRand}$ on state $x^\star$ before or after the round $j^\star$. Within the round $j^\star$, $\mathcal{A}$ is only allowed to query $\mathsf{PartialRand}(i)$ for at most $t - |C|$ distinct i, which is consistent with the requirement that $\mathcal{B}$ can only query $(\mathsf{Eval}, x^\star, i)$ for at most $t - |C|$ distinct i to his DVRF evaluation oracle.

Finally $\mathcal{A}$ outputs b' and $\mathcal{B}$ outputs b'. Therefore we have $\mathbf{Adv}_{\mathrm{DVRF},\mathcal{B}} = \mathbf{Adv}_{\mathrm{DRB},\mathcal{A}}$. This concludes the proof. □

Theorem A.3. *$\mathcal{V}^{\mathrm{DRB}}$ satisfies uniqueness if $\mathcal{V}^{\mathrm{DVRF}}$ satisfies uniqueness.*

Proof. Suppose there is a PPT adversary $\mathcal{A}$ that breaks the DRB uniqueness experiment with non-negligible probability. We shall construct a PPT adversary $\mathcal{B}$ which breaks the DVRF uniqueness.

$\mathcal{A}$ chooses to corrupt a set of nodes C with $C \subseteq \mathcal{N}$ and $|C| \leq t$. $\mathcal{B}$ forwards C to its DVRF uniqueness challenger to run the interactive protocol DistKG to create a committee. $\mathcal{B}$'s DVRF challenger runs on behalf of the honest nodes in $\mathcal{N}$ and $\mathcal{A}$ runs on behalf of the bad nodes C. $\mathcal{B}$ acts as a message forwarder between $\mathcal{B}$'s challenger and $\mathcal{A}$. After that, $\mathcal{A}$ can query the oracles $\mathcal{O}^{\mathsf{PartialRand}}$, $\mathcal{O}^{\mathsf{KeyRev}}$, $\mathcal{O}^{\mathsf{Update}}$. Queries $\mathcal{O}^{\mathsf{PartialRand}}, \mathcal{O}^{\mathsf{KeyRev}}$ by $\mathcal{A}$ can be answered by $\mathcal{B}$'s DVRF evaluation and key-revealing oracle respectively. $\mathcal{A}$ outputs $(\sigma, \sigma', \pi, \pi')$. Let the current state be $\mathsf{st} = x^\star$. $\mathcal{B}$ issues a challenge query $(x^\star, \sigma, \sigma', \pi, \pi')$ to its DVRF challenger. We can easily see that when $\mathcal{A}$'s challenger outputs 1, $\mathcal{B}$'s challenger will also output 1. This concludes the proof. □

Corollary A.3.1. *$\mathcal{V}^{\mathrm{DRB}}$ satisfies fairness if $\mathcal{V}^{\mathrm{DVRF}}$ satisfies uniqueness.*

F. Fully Distributed Threshold Cryptosystem

A fully distributed t-out-of-ℓ threshold cryptosystem with non-interactive threshold decryption consists of the following algorithms:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ is a fully distributed *key generation* algorithm that takes as input a security parameter 1^λ, the number of decryption nodes ℓ, and the threshold parameter t; it outputs a public key pk, a list $\mathbf{sk} = \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell\}$ of nodes' private keys, a list $\mathbf{vk} = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell\}$ of verification keys.

$\mathsf{Enc}(\mathsf{pk}, m)$ is an *encryption algorithm* that takes as input the public key pk and a plaintext m, and outputs a ciphertext C. We may write $\mathsf{Enc}(\mathsf{pk}, m; r)$ when we want to make explicit the random coins used for encrypting.

$\mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{vk}_i, C)$ is a *share decryption algorithm* that takes as input the public key pk, the private key sk_i, the verification key vk_i, a ciphertext C, and outputs a partial decryption (i, μ_i, π_i) where π is the proof of the correctness of the decryption share μ_i.

$\mathsf{ShareVer}(\mathsf{pk}, C, \mu_i, \pi_i, \mathsf{vk}_i)$ is a *share verification algorithm* that takes as input the public key pk, a ciphertext C, a decrytion share μ_i, a proof π_i, and outputs 0/1.

$\mathsf{Recovery}(\mathsf{pk}, \mathbf{vk}, C, \mathcal{C})$ is a *recovery* algorithm that takes as input the public key pk, a ciphertext C, and a list $\mathcal{C}$ of $t+1$ decryption shares, together with the verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell$, and outputs a message m or $\perp$.

G. HERB

The so-called Homomorphic Encryption Random Beacon construction [22] defines a DRB using DKG and ElGamal homomorphic properties, similary to the random beacon used in Orbs' Helix consensus protocol [3], [4]. A necessary building block is ElGamal-based Threshold Cryptosystem $\mathrm{EG} = (\mathrm{EG.DistKG}, \mathrm{EG.Enc}, \mathrm{EG.ShareDec}, \mathrm{EG.ShareVer}, \mathrm{EG.Recovery})$ from [25, Section 3.2] which is non-malleable (the syntax of threshold decryption protocols is described in Appendix F). Herb's DRB is specified as:

- $\mathsf{CmteGen}(1^\lambda, t, \ell)$: a set $\mathcal{N} = \{N_1, \ldots, N_\ell\}$ of ℓ nodes jointly run $\mathrm{EG.DistKG}(1^\lambda, t, \ell)$ to obtain a set QUAL of qualified nodes, a global public key pk, and each node N_i obtains a secret key sk_i and a verification key vk_i. Let $\mathsf{cpk} = (\mathsf{pk}, \mathcal{VK})$ where $\mathcal{VK} = \{\mathsf{vk}_i\}_{i \in \mathrm{QUAL}}$.

- $\mathsf{PartialRand}(\mathsf{st}_{n-1}, \mathsf{sk}_i, \mathsf{cpk})$: the state $\mathsf{st}_{n-1} \in \mathrm{Dom}$ consists of a collection $\{(i, C_{n-1,i}, \pi^{eq}_{n-1,i})\}_{i\in I}$ of ElGamal ciphertexts under the global public key pk and encryption correctness proofs. Each ciphertext encrypts a random element $R_{n-1,i} \in \mathbb{G}$ contributed by node N_i for $i \in I$. Compute the ciphertext production $C = \prod_{i\in I} C_{n-1,i}$ and run the ElGamal partial decryption $(\sigma_{n,i}, \pi^{\mathrm{dec}}_{n,i}) \leftarrow \mathrm{EG.ShareDec}(\mathsf{sk}_i, \mathsf{vk}_i, C)$. Finally, choose a fresh random $R_{n,i} \in \mathbb{G}$ for the next round and encrypt the random $(C_{n,i}, \pi^{eq}_{n,i}) \leftarrow \mathrm{EG.Enc}(\mathsf{pk}, R_{n,i})$ under the global public key pk. Output is $\big((i, C, \sigma_{n-1,i}, \pi^{\mathrm{dec}}_{n-1,i}), (i, C_{n,i}, \pi^{eq}_{n,i})\big)$.
- $\mathsf{CombRand}(\mathsf{st}_{n-1}, \mathcal{E}, \mathsf{cpk})$: Parse $\mathcal{E} = \{(i, C, \sigma_{n-1,i}, \pi^{\mathrm{dec}}_{n-1,i})\}_{i\in I}$ with $|I| > t$ and $I \subseteq \mathrm{QUAL}$. Identify an index subset $J \subset I$ such that $|J| = t + 1$ and for each $j \in J$, the partial decryption $\sigma_{n-1,j}$ is correct, i.e., $\mathrm{EG.ShareVer}(\mathsf{pk}, C, \sigma_{n-1,j}, \pi^{\mathrm{dec}}_{n-1,j}, \mathsf{vk}_i) = 1$. Obtain the decryption $\sigma_n \leftarrow \mathrm{EG.Recovery}(\mathsf{pk}, C, \{\sigma_{n-1,j}\}_{j\in J})$ such that $\sigma_n = \prod_{j\in J} R_{n-1,j}$. Output (σ_n, π_n) where $\pi_n = (C, \{(j, \sigma_{n-1,j}, \pi^{\mathrm{dec}}_{n-1,j})\}_{j\in J})$.
- $\mathsf{VerifyRand}(\mathsf{st}_{n-1}, \sigma_n, \pi_n, \mathsf{cpk})$: Parse $\pi_n = (C, \{(j, \sigma_{n-1,j}, \pi^{\mathrm{dec}}_{n-1,j})\}_{j\in J})$. Verify that σ_n is the decryption of C by checking whether $\mathrm{EG.ShareVer}(\mathsf{pk}, C, \sigma_{n-1,j}, \pi^{\mathrm{dec}}_{n-1,j}, \mathsf{vk}_i) = 1$ for each $j \in J$ and $\sigma_n = \mathrm{EG.Recovery}(\mathsf{pk}, C, \{\sigma_{n-1,j}\}_{j\in J})$.
- $\mathsf{UpdState}(\sigma_n, \pi_n, \mathsf{st}_{n-1})$: Set st_n to be $\{(i, C_{n,i}, \pi^{eq}_{n,i}))\}_{i\in I}$.

H. Fully Distributed Threshold Signatures

A fully distributed t-out-of-ℓ threshold signature with non-interactive threshold signing consists of the following algorithms [11]:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ is a fully distributed *key generation* algorithm that takes as input a security parameter 1^λ, the number of participating nodes ℓ, and the threshold parameter t; it outputs a set of qualified nodes QUAL, a global public key pk, a list $\mathcal{VK} = \{\mathsf{vk}_1, \ldots, \mathsf{vk}_\ell\}$ of nodes' verification keys, and results in a list $\mathcal{SK} = \{\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell\}$ of nodes' secret keys where each secret key is only known to the corresponding sever.

$\mathsf{PartialSign}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ is a *partial signing algorithm* that takes as input a plaintext $x \in \mathrm{Dom}$, secret key sk_i and verification key vk_i, and outputs a triple $s^i_x = (i, v_i, \pi_i)$, where v_i is the i-th signature share and π_i is a non-interactive proof of correct partial signing.

$\mathsf{ShareSigVer}(\mathsf{pk}, x, \sigma_i, \mathsf{vk}_i)$ is a *signature share algorithm* that takes as input the public key pk, a plaintext x, a signature share σ_i, and outputs 0/1.

$\mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \mathcal{E})$ is a *combination* or reconstruction algorithm that takes as input the global public key pk, the verification keys $\mathcal{VK}$, a plaintext $x \in \mathrm{Dom}$, and a set $\mathcal{E} = \{s^{i_1}_x, \ldots, s^{i_{|\mathcal{E}|}}_x\}$ of signature shares originating from $|\mathcal{E}| \geq t+1$ different nodes, and outputs either a signature σ or $\perp$.

$\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v, \pi)$ is a *verification algorithm* that takes as input the public key pk, a set of verification keys $\mathcal{VK}$, a plaintext $x \in \mathrm{Dom}$, a signature σ, and outputs 0/1.

Formally, let us define

Definition A.8 (Strong Unforgeability)**.** *A distributed signature scheme* $\S = (\mathsf{DistKG}, \mathsf{PartialSign}, \mathsf{Combine}, \mathsf{Verify})$ *is* (θ, t, ℓ)-strongly unforgeable *if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following experiment outputs 1 with probability at most* $\mathsf{negl}(\lambda)$.

[Corruption] *On input the Servers list* $S = \{S_1, \ldots, S_\ell\}$ *and threshold* $0 \leq t < \ell$*, a adversary* $\mathcal{A}$ *chooses a collection* C *of servers to be corrupted with* $|C| \leq \theta$*. Adversary* $\mathcal{A}$ *acts on behalf of corrupted servers, while the challenger acts on behalf of the remaining servers, which behave honestly.*

[Initialization] *Challenger and adversary engage in running the distributed key generation protocol* $\mathsf{DistKG}(1^\lambda, t, \ell)$*. After this phase, the protocol establishes a qualified set of servers* QUAL*. Every (honest) server* $S_j \in \mathrm{QUAL} \setminus C$ *obtains a key pair* $(\mathsf{sk}_j, \mathsf{vk}_j)$*. In contrast, (corrupted) servers* $S_j \in C$ *end up with key pairs* $(\mathsf{sk}_j, \mathsf{vk}_j)$ *in which one of keys may be undefined (i.e. either* $\mathsf{sk}_j = \perp$ *or* $\mathsf{vk}_j = \perp$*). At the end of this phase, the global public key* pk *and the verification keys vector* vk *is known by both the challenger and the attacker.*

[Partial signature queries] *In response to* $\mathcal{A}$*'s evaluation query* (Sign, x, i) *for some honest server* $S_i \in \mathrm{QUAL} \setminus C$ *and plaintext* $x \in \mathcal{D}$*, the challenger returns* $s^i_x \leftarrow \mathsf{PartialSign}(\mathsf{sk}_i, \mathsf{vk}_i, x)$*. In any other case, the challenger returns* $\perp$.

[Output] *The adversary outputs a plaintext-signature pair* $(x^\star, \sigma^\star)$.

The output of the experiment is 1 is $\mathsf{Verify}(\mathsf{pk}, x^\star, \sigma^\star) = \mathtt{accept}$ *and plaintext* $x^\star \in \mathcal{D}$ *is such that* $(\mathsf{Sign}, x^\star, i)$ *has been queried by the adversary at most* $t - \theta$ *times for different* $S_i \in \mathrm{QUAL} \setminus C$.

If the adversary *is not allowed* to obtain any partial signatures on the challenge plaintext in the above experiment, the security definition collapses to the standard unforgeability definition for pairing-based threshold signatures [11], [58]. We note that our improved unforgeability definition generalises the unforgeability definition for threshold signatures in the presence of a trusted dealer that can be traced back to Shoup [56].

1) Separation between Standard and Strong Unforgeability: We shall construct a non-interactive threshold signature protocol $\mathcal{S}$ that is (θ, t, ℓ)-unforgeable, but not (θ, t, ℓ)-strongly unforgeable. Let $\mathcal{S}' = (\mathsf{DistKG}', \mathsf{PartialSign}', \mathsf{Combine}', \mathsf{Verify}')$ be a non-interactive threshold signature protocol which is (θ, t', ℓ)-unforgeable with $0 < t' < t$. We construct $\mathcal{S}$ as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ Run $\mathsf{DistKG}(1^\lambda, t', \ell)$ among ℓ nodes to obtain the qualified set QUAL of nodes and the global public key pk. Each node in $i \in \mathrm{QUAL}$ obtains a pair of $(\mathsf{sk}_i, \mathsf{vk}_i)$. Abort if $|\mathrm{QUAL}| \leq t$

$\mathsf{PartialSign}(\mathsf{sk}_i, x)$ outputs (i, σ_i), where $(i, \sigma_i) \leftarrow \mathsf{PartialSign}'(\mathsf{sk}_i, x)$.

$\mathsf{Combine}(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$ When $|\mathcal{E}| > t$, it is also the case $|\mathcal{E}| > t'$. Thus we can run $\sigma \leftarrow \mathsf{Combine}'(\mathsf{pk}, \mathsf{vk}, x, \mathcal{E})$. Output σ.

$\mathsf{Verify}(\mathsf{pk}, x, \sigma)$ outputs 1 if $\mathsf{Verify}'(\mathsf{pk}, x, \sigma) = 1$. Else outputs 0.

Theorem A.4. *$\mathcal{S}$ is (θ, t, ℓ)-unforgeable but it is not (θ, t, ℓ)-strongly unforgeable.*

Proof. $\mathcal{S}$ does not satisfy (θ, t, ℓ)-strong unforgeability. Suppose an adversary $\mathcal{A}$ compromises θ nodes. Then $\mathcal{A}$ can query $\mathsf{PartialSign}(\mathsf{sk}_i, x^\star)$ for up to $t-\theta$ different $i \in \mathrm{QUAL}$ on the challenge input $x^\star$. Therefore the adversary can obtain $t > t'$ valid partial signatures $\{(i, \sigma_i)\}_i$ which allows the adversary to run Combine to recover $\sigma^\star$.

The standard unforgeability of $\mathcal{S}$, i.e., (θ, t, ℓ)-unforgeability, holds because the adversary can only compromise up to $\theta \leq t' \leq t$ nodes. If the adversary can output a forgery $\sigma^\star$ for $\mathcal{S}$, then $\sigma^\star$ is also a forgery for $\mathcal{S}'$. □

I. Secure DKG protocols

We recall in Figure 2 the DKG protocol originally proposed in [35]. This protocol is used in our proposed DDH-based DVRF called DDH-DVRF. The DKG protocol used in Dfinity-DVRF can be found in Figure 4, whereas the protocol used in GLOW-DVRF is given in Figure 3.

J. Security proofs for DDH-DVRF

Theorem IV.1. $\mathcal{V}^{\text{DDH-DVRF}}$ satisfies consistency, robustness and uniqueness.

Proof. First, we show that $\mathcal{V}^{\text{DDH-DVRF}}$ is consistent. It suffices to see that the following equality holds:

$$\sum_{j\in\Delta} \mathsf{sk}_j \lambda_{0,j,\Delta} = \sum_{j\in\Delta} \lambda_{0,j,\Delta} \left(\sum_{i\in \mathrm{QUAL}} s_{i,j} \right)$$
$$= \sum_{i\in\mathrm{QUAL}} \left(\sum_{j\in\mathcal{I}} \lambda_{0,j,\Delta} \cdot s_{i,j} \right)$$
$$= \sum_{i\in\mathrm{QUAL}} \left(\sum_{j\in\Delta} \lambda_{0,j,\Delta} \cdot f_i(j) \right) = \sum_{i\in\mathrm{QUAL}} a_{i,0} = \mathsf{sk} \quad (3)$$

Then $\prod_{j\in\Delta} (H_1(x)^{\mathsf{sk}_j})^{\lambda_{0,j,\Delta}} = H_1(x)^{\left(\sum_{j\in\Delta} \lambda_{0,j,\Delta}\cdot \mathsf{sk}_j\right)} = H_1(x)^{\mathsf{sk}}$ holds for every subset $\Delta \subseteq \mathrm{QUAL}$ with $|\Delta| \geq t+1$.

Robustness is straightforward since the NIZK proofs included in $\pi^\star$ are all valid proofs that have been checked by the Combine function. This implies Verify outputs 1.

We are left to show the uniqueness of DDH-DVRF. The proof relies on the extractibility property of NIZKs. We first describe the original uniqueness game in the random oracle model as below. W.l.o.g., we assume the list of nodes $\mathcal{N} = \{1, \ldots, \ell\}$.

$\underline{\mathsf{Unique}_{\mathcal{A}}(b)}$:

1) Give the public parameters $(\mathbb{G}, q, g)$ to the adversary $\mathcal{A}$
2) $\mathcal{A}$ chooses to corrupt a collection C of nodes with $|C| \leq t$. Without loss of generality, let $C = \{1, \cdots, m\}$. Get the set C from $\mathcal{A}$.
3) Run the distributed key generation protocol $\mathsf{DistKG}(1^k, t, \ell)$ with ℓ nodes $\mathcal{N}$. The protocol outputs a set QUAL of qualified nodes, a key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ for every honest node $N_i \in \mathrm{QUAL} \setminus C$, and outputs key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ in which one of keys may be undefined for corrupted nodes $N_i \in C$ if $i \notin \mathrm{QUAL}$.
4) The random oracle H_1 is programmed as follows: Define a list $\mathcal{L}_{H_1} = \emptyset$. For a query on x:
   a) If there exists $(x, r, h) \in \mathcal{L}_{H_1}$, then the oracle outputs h.
   b) Otherwise, choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $h = g^r$ and update the list $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$. The oracle outputs h.
5) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$. The oracle outputs c.
6) On an evaluation query (Eval, x, i) for an honest node $N_i \in \mathrm{QUAL} \setminus C$, compute $H_1(x)^{\mathsf{sk}_i}$ and run PrEq_{H_2} to generate a proof π_i, and return $(i, H_1(x)^{\mathsf{sk}_i}, \pi_i)$. Otherwise return $\perp$.
7) On the challenge query, the challenger receives from the adversary $\mathcal{A}$, a plaintext $x^\star \in \mathrm{Dom}$, two values $v, v' \in \mathrm{Ran}$ and two proofs π, π'. If $v \neq v'$ and $\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v, \pi) = \mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v', \pi') = 1$ then output 1; else, output 0.

Suppose there exists an adversary $\mathcal{A}$ leading the challenger to output 1 with non-negligible probability. We will show that this leads to a contradiction. If the challenger outputs 1, then $v \neq v'$ but $\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v, \pi) = \mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x, v', \pi') = 1$. The Verify algorithm checks all the proofs $\pi = \{(i, v_i, \pi_i)\}_{i\in I}$ and $\pi' = \{(i, v'_i, \pi_i)\}_{i\in I'}$ where $|I| = |I'| = t+1$. In addition $v = \prod_{i\in I} v_i^{\lambda_{0,i,I}}$ and $v' = \prod_{i\in I} v_i'^{\lambda_{0,i,I'}}$. Due to the consistency property, we can derive that when $v \neq v'$, there exists $j \in I$ (or $j \in I'$) such that $v_j \neq H_1(x)^{\mathsf{sk}_j}$ (or $v'_j \neq H_1(x)^{\mathsf{sk}_j}$)but $\mathsf{vk}_j = g^{\mathsf{sk}_j}$ (otherwise $v = v' = H_1(x)^{\mathsf{sk}}$). W.l.o.g., assume $j \in I$. However, the proof $\pi_j = (ch_j, res_j)$ is a valid NIZK proof where ch_j is the hash value obtained from querying the random oracle H_2. Thus we can use the forking lemma to rewind $\mathcal{A}$ to the point ch_j is generated and extract a k such that $v_j = H_1(x)^k$ and $\mathsf{vk}_j = g^k$ which contradicts to the hypothesis.

□

We shall prove the strong pseudorandomness of DDH-based DVRF. Our proof is constructed using the simulator for proving secrecy of the secure DKG protocol in [35] and the proofs of Theorem 8.1 and 8.2 in [1]. In addition, we use standard zero-knowledge property to simulate NIZKs. Below we show in color where our proofs for DDH-DVRF diverge from the proofs in [1].

- **Generating phase:**
  1) Each node N_i performs a Pedersen-VSS of a random value $a_{i,0}$:
     a) N_i chooses two random t-degree polynomials f_i, f'_i over $\mathbb{Z}_q$:
     $$f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t \qquad f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t}z^t$$
     N_i broadcasts $C_{i,k} = g^{a_{i,k}}h^{b_{i,k}}$ for $0 \le k \le t$. N_i computes the shares $s_{i,j} = f_i(j), s'_{i,j} = f'_i(j)$ for $1 \le j \le \ell$ and sends $s_{i,j}, s'_{i,j}$ to node N_j.
     b) Each node N_j verifies the shares he received from the other nodes. N_j checks if
     $$g^{s_{i,j}}h^{s'_{i,j}} = \prod_{k=0}^{t}(C_{i,k})^{j^k} \tag{1}$$
     for $1 \le i \le \ell$. If the check fails for an index i, N_j broadcasts a complaint against N_i.
     c) Each node N_i who received a complaint from node N_j broadcasts the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 1.
     d) Each node marks as *disqualified* any node that either
     - Received more than t complaints in Step 1b, or
     - Answered to a complaint in Step 1c with values that falsify Equation 1.
  2) Each node then builds the set of non-disqualified players QUAL.
  3) Each node N_i sets his share of the secret as $\mathsf{sk}_i = \sum_{j\in\mathrm{QUAL}} s_{j,i}$, the verification key $\mathsf{vk}_i = g^{\mathsf{sk}_i}$ and the value $r_i = \sum_{j\in\mathrm{QUAL}} s'_{j,i}$.
- **Extracting phase:**
  4) Each node $i \in$ QUAL exposes $A_{i,0} = g^{a_{i,0}}$ via **Feldman-VSS**:
     a) Each node N_i with $i \in$ QUAL, broadcasts $A_{i,k} = g^{a_{i,k}}$ for $0 \le k \le t$.
     b) Each node N_j verifies the values broadcast by the other nodes in QUAL, formally, for each $i \in$ QUAL, N_j checks if
     $$g^{s_{i,j}} = \prod_{k=0}^{t}(A_{i,k})^{j^k} \tag{2}$$
     If the check fails for some index i, N_j complains against N_i by broadcasting the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 1 but not Equation 2.
     c) For node N_i who receives at least one valid complaint, i.e., values which satisfy Equation 1 but not Equation 2, the other nodes run the re-construction phase of **Pedersen-VSS** to compute $a_{i,0}, f_i, A_{i,k}$ for $0 \le k \le t$. Compute the global public key as $\mathsf{pk} = \prod_{i\in\mathrm{QUAL}} A_{i,0}$. The distributed secret value $\mathsf{sk} = \sum_{i\in\mathrm{QUAL}} a_{i,0}$ is not explicitly computed by any party.
     d) Each node N_i can reconstruct the verification keys of other nodes:
     - Compute $v_k = \prod_{i\in\mathrm{QUAL}} A_{i,k}$ for $0 \le k \le t$.
     - For each $j \in$ QUAL such that $j \ne i$, compute $\mathsf{vk}_j = \prod_{k=0}^{t} v_k^{j^k}$.

Figure 2: Secure distributed key generation (DKG) protocol [35] used in DDH-DVRF.

Theorem IV.2. For any θ such that $\theta \le t < \ell - \theta$, $\mathcal{V}^{\text{DDH-DVRF}}$ is (θ, t, ℓ)-strongly pseudorandom under the *DDH* assumption in the random oracle model.

Proof. For any PPT adversary, we shall first describe the real game $\mathrm{PRand}_{\mathcal{A}}(b)$ as below. W.l.o.g., we assume the list of nodes $\mathcal{N} = \{1, \ldots, \ell\}$.

$\underline{\mathrm{PRand}_{\mathcal{A}}(b)}$:

1) Give the public parameters $(\mathbb{G}, q, g)$ to the adversary $\mathcal{A}$
2) $\mathcal{A}$ chooses to corrupt a collection C of nodes with $|C| \le \theta$. Without loss of generality, let $C = \{1, \cdots, m\}$. Get the set C from $\mathcal{A}$.
3) Run the distributed key generation protocol $\mathsf{DistKG}(1^k, t, \ell)$ with ℓ nodes $\mathcal{N}$. The protocol outputs a set QUAL of qualified nodes with QUAL $\subseteq \mathcal{N}$, key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ for every honest node $i \in \mathrm{QUAL} \setminus C$, and outputs key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ in which one of keys may be undefined for corrupted nodes $i \in C$.
4) The random oracle H_1 is programmed as follows: Define a list $\mathcal{L}_{H_1} = \emptyset$. For a query on x:
   a) If there exists $(x, r, h) \in \mathcal{L}_{H_1}$, then the oracle outputs h.
   b) Otherwise, choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $h = g^r$ and update the list $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$. The oracle outputs h.
5) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$. The oracle outputs c.
6) On an evaluation query (Eval, x, i) for an honest node $i \in \mathrm{QUAL} \setminus C$, compute $H_1(x)^{\mathsf{sk}_i}$ and run PrEq_{H_2} to generate

a proof π_i, and return $(H_1(x)^{\mathsf{sk}_i}, \pi_i)$. Otherwise return $\perp$.

7) On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, v_i^\star, \pi_i)\}_{i\in U}, V)$ with $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$,
   a) If $\mathcal{A}$ has made at least $t + 1 - |C|$ queries of the form $(\mathsf{Eval}, x^\star, *)$, then output 0 and stop.
   b) Otherwise do as follows:
      i) Run $\mathsf{VerifyEq}_{H_1}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
      ii) Set $v_i^\star = H_1(x^\star)^{\mathsf{sk}_i}$ for $i \in V \setminus U$.
      iii) Compute $v^\star = \prod_{i\in V} v_i^{\star\lambda_{0,i,V}}$.
      iv) If $v^\star = \perp$ then return $\perp$. Otherwise depending on b do as follows:
         A) If $b = 0$ then return $v^\star$;
         B) Else return a uniform random.
8) Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in \mathrm{QUAL} \setminus C$ and i is the $(t + 1 - |C|)$-th honest node that $\mathcal{A}$ contacted then output 0 and stop.
9) Receive a guess b' from $\mathcal{A}$ and output b'.

a) First hybrid - $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$*:* Define a hybrid $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ which is similar to $\mathrm{PRand}_{\mathcal{A}}(b)$ except that real proofs π_i in Step 6 and 8 are replaced with simulated proofs. $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ is indistinguishable from $\mathrm{PRand}_{\mathcal{A}}(b)$ for any PPT $\mathcal{A}$ and $b \in \{0, 1\}$ due to the zero-knowledge property of NIZKs.

b) Second hybrid - $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$*:* Define a hybrid $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ which is similar to $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ except that the running of the secure DKG protocol $\mathsf{DistKG}(1^k, t, \ell)$ in Step 3 is replaced with a simulator described in [35] which sets the public key pk to a given element $Y = g^\alpha$ with α unknown. We give a brief description of the construction of the simulator below:

1) Generating phase: The simulator runs on behalf of all the honest nodes. It performs the Generating phase exactly as in the secure DKG protocol. The simulator chooses random polynomials $f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t$ and $f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t}z^t$ for each honest node $i \in \mathcal{N} \setminus C$. At the end of the Generating phase of the secure DKG protocol, the simulator obtains a well-defined set of non-disqualified nodes QUAL which contains all the honest nodes and some corrupted nodes. The simulator receives all the shares $s_{i,j}, s'_{i,j}$ from the corrupted nodes. Since the simulator runs all the $\ell - \theta$ honest nodes and $\ell - \theta > t$, the simulator can recover all the polynomials $f_i(z), f'_i(z)$ for $i \in \mathrm{QUAL} \cap C$.
2) Extracting phase:
   - For the honest nodes $i \in \mathrm{QUAL} \setminus (C \cup \{\ell\})$, compute $A_{i,j} = g^{a_{i,j}}$ for $0 \le j \le t$.
   - For the ℓ-th honest node, the simulator replaces f_ℓ with a new polynomial $f^\star_\ell$ which is determined by the values: $\alpha - \sum_{j\in\mathrm{QUAL}\setminus\{\ell\}} f_j(0), f_\ell(1), \ldots, f_\ell(t)$. The simulator can compute $A^\star_{\ell,j}$ for $0 \le j \le t$ based on the new polynomial $f^\star_\ell(x)$ without knowing the value of α:
     - $A^\star_{\ell,0} = Y \cdot \prod_{i\in\mathrm{QUAL}\setminus\{\ell\}} A^{-1}_{i,0}$
     - $A^\star_{\ell,j} = (A^\star_{\ell,0})^{\delta_{0,j}} \prod_{i=1}^{t} g^{f_\ell(i)\cdot\delta_{i,j}}$ where $\delta_{i,j}$ is the coefficient of x^j in the lagrange basis polynomial $\lambda_{i,T}(x)$ with $T = \{0, 1, \ldots, t\}$.

   Broadcast $A_{i,j}$, $A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$.

Therefore, the final polynomial $f^\star$ takes the values $f^\star(0) = \alpha$, $f^\star(1) = f(1)$, $\ldots$, $f^\star(t) = f(t)$. The global public key $\mathsf{pk} = A^\star_{\ell,0} \prod_{i\in\mathrm{QUAL}\setminus\{\ell\}} A_{i,0} = g^\alpha$.

For each honest node $j \in \mathrm{QUAL} \setminus C$ and $j \le t$, the simulator can compute j's secret key $\mathsf{sk}_j = f^\star(j)$ as well as the corresponding verification key vk_j. For each honest node $j \in \mathrm{QUAL} \setminus C$ and $j > t$, the simulator is not able to computer j's secret key because $f^\star(0) = \alpha$ is unknown. This means the simulator cannot compute $f^\star(j)$ for $j > t$. However, the simulator can still compute the corresponding verification keys $\mathsf{vk}_j = \mathsf{pk}^{\lambda_{j,0,T}} \cdot \prod_{i=1}^{t} \mathsf{vk}_i^{\lambda_{j,i,T}}$ for $j > t$ where $T = \{0, 1, \ldots, t\}$ and $\lambda_{j,i,T}$ are the lagrange coefficients.

Although the simulator does not have sk_i for $i \in \mathrm{QUAL} \setminus C$ and $i > t$, with the value of $\mathsf{vk}_i = g^{\mathsf{sk}_i}$ it is sufficient to answer all the evaluation queries in Step 6, 7 and 8 defined in the game $\mathrm{PRand}_{\mathcal{A}}(b)$. In the game $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$, we modify all the computation of $v_i = H_1(x)^{\mathsf{sk}_i}$ for $i \in \mathrm{QUAL} \setminus C$ and $i > t$ as follows: assume $(x, r, h) \in L_{H_1}$, compute $v_i = \mathsf{vk}_i^r$. The corresponding NIZK that shows v_i is correctly formed can be simulated using random oracle H_2 without knowing sk_i. Finally $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ is indistinguishable from $\mathsf{Hyb}^{zk}_{\mathcal{A}}(b)$ for any PPT $\mathcal{A}$ and $b \in \{0, 1\}$ because the probability distribution of the output of the simulator is identical to the original secure DKG protocol [35].

c) k-hybrids - $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$*:* For any adversary $\mathcal{A}$ that asks evaluation/challenge queries on q_E distinct x, we define hybrid games $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$. The hybrid games $\mathsf{Hyb}^{k}_{\mathcal{A}}(b)$ are exactly the same as $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ except how the evaluation/challenge queries are answered on the first k distinct plaintext x. The simulator runs the secure DKG protocol to set up the system and assume the final polynomial is $f^\star$. The simulator in addition chooses k t-degree random polynomials $f^1, f^2, \cdots, f^k$ such as for all corrupted nodes $i \in C$, $f^j(i) = f^\star(i)$, for $1 \le j \le k$. That is, the random polynomials match on the shares of the corrupted nodes.

Now an evaluation query (Eval, x, i) for an honest i is answered as follows: if x is the j-th distinct plaintext in the evaluation/challenge queries, then return $\mathsf{pc}(x, j, i)$ defined as follows:

$$\mathsf{pc}(x, j, i) = \begin{cases} (H_1(x)^{f^j(i)}, \pi_i) & \text{if } j \le k \\ (H_1(x)^{\mathsf{sk}_i}, \pi_i) & \text{otherwise} \end{cases}$$

where π_i is a simulated proof generated by calling the random oracle H_2 and $H_1(x)^{\mathsf{sk}_i}$ is obtained in the same way as described in $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$.

The challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i\in U}, V)$ with $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted nodes, is answered as follows:

1) if $x^\star$ was queried in the evaluation phase and it was the j-th distinct value, then let $j^\star = j$. Else, assume there are so far j' distinct evaluation queries and let $j^\star = j' + 1$.

2) If $\mathcal{A}$ has made at least $t+1-|C|$ queries of the form $(\mathsf{Eval}, x^\star, \cdot)$, then output 0 and stop.
3) Otherwise do as follows:
   a) Run $\mathsf{VerifyEq}_{H_2}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
   b) Set $(z_i^\star, \cdot) = \mathsf{pc}(x^\star, j^\star, i)$ for $i \in V \setminus C$.
   c) Compute $v^\star = \prod_{i \in V} z_i^{\star \lambda_{0,i,V}}$.
   d) If $v^\star = \perp$ then return $\perp$. Otherwise depending on b do as follows:
      i) If $b = 0$ then return $v^\star$;
      ii) Else return a uniform random.

Obviously, $\mathsf{Hyb}_{\mathcal{A}}^0(b)$ is identical to $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$. $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ only differs in the evaluation queries for the k-th distinct x. Below we shall prove that $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ are indistinguishable from the attacker's point of view.

Lemma A.5. *For any $b \in \{0,1\}$ and $1 \le k \le q_E$, the outputs of hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ are computationally indistinguishable under DDH assumption.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ with non-negligible probability then we can construct a PPT adversary $\mathcal{B}$ to break an extended version of the DDH assumption using $\mathcal{A}$ as a subroutine.

The extended DDH problem [1] is given by

$$(\mathbb{G}, q, g, g^{\alpha_0}, g^{\alpha_1}, \cdots, g^{\alpha_w}, g^{\beta}, y_0, y_1, \cdots, y_w)$$

where $y_i = g^{\alpha_i \beta}$ for all $i \in \{0, 1, \ldots, w\}$ or randoms. The extended DDH problem can be easily derived from the original DDH problem. We now construct $\mathcal{B}$ using $\mathcal{A}$ as follows:

1) Give the public parameters $(\mathbb{G}, q, g)$ and a list of nodes $\mathcal{N} = \{1, 2, \ldots, \ell\}$ to $\mathcal{A}$
2) $\mathcal{A}$ chooses a set C of nodes with $|C| \le \theta$ to corrupt and send C to $\mathcal{B}$. W.l.o.g., assume $C = \{1, 2, \cdots, m\}$.
3) The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{\$} [q_{H_1}]$ uniformly at random.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   - Otherwise,
     – if this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g^\beta$ where g^β is from the DDH problem.
     – else choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $h = g^r$.

   Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$

   Give the random oracle access to $\mathcal{A}$.
4) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$. The oracle outputs c. Give the random oracle access to $\mathcal{A}$.
5) Run the simulator of the secure DKG protocol to interact with the corrupted nodes. $\mathcal{B}$ acts on behalf of all the honest nodes $\{m+1, \ldots, \ell\}$, while the adversary controls all the corrupted nodes $\{1, \ldots, m\}$. The generating phase is run exactly the same as described in Figure 2. The generating phase determines a set of non-disqualified nodes QUAL which contains all the honest nodes and some of the corrupted nodes. Since $\mathcal{B}$ runs all the honest nodes, $\mathcal{B}$ obtains all the shares from the corrupted nodes and can recover all the polynomials f_i with $i \in \mathrm{QUAL} \cap C$ chosen by the adversary. Assume the combined polynomial after the Generating phase is f. In the Extracting phase, the last polynomial, i.e., the ℓ-th polynomial $f_\ell^\star$, is constructed to take the values: $\alpha_0 - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1), \ldots, f_\ell(m)$, $\alpha_{m+1} - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(m+1), \ldots, \alpha_t - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(t)$. For $0 \le j \le t$, we show how to compute $A_{i,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $A_{\ell,j}^\star$:
   - For $i \in \mathrm{QUAL} \setminus \{\ell\}$, the simulator has all the coefficients $a_{i,j}$ for $0 \le j \le t$, thus $A_{i,j} = g^{a_{i,j}}$ can be easily computed.
   - For the ℓ-th polynomial $f_\ell^\star$, the simulator sets $A_{\ell,0}^\star = g^{\alpha_0} \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} A_{i,0}^{-1}$ and $A_{\ell,j}^\star = (A_{\ell,0}^\star)^{\delta_{0,j}} \prod_{i=1}^{m} g^{f_\ell(i) \cdot \delta_{i,j}} \prod_{i=m+1}^{t} (g^{\alpha_i} \prod_{k \in \mathrm{QUAL} \setminus \{\ell\}} g^{-f_k(i)})^{\delta_{i,j}}$ where $\delta_{i,j}$ is the coefficient of x^j in the lagrange basis polynomial $\lambda_{i,T}(x)$ with $T = \{0, 1, \ldots, t\}$.

   Let the final combined polynomial be $f^\star$. We can see that $f^\star(0) = \alpha_0$, $f^\star(1) = f(1), \ldots, f^\star(m) = f(m)$, $f^\star(m+1) = \alpha_{m+1}, \ldots, f^\star(t) = \alpha_t$. This means the shares of the honest nodes $m+1 \le i \le t$ are set to be $\mathsf{sk}_i = \alpha_i$ and the corresponding verification keys are $\mathsf{vk}_i = g^{\alpha_i}$ despite $\alpha_0, \alpha_{m+1}, \ldots, \alpha_t$ are unknown. For the honest nodes $t+1 \le i \le \ell$, we also cannot compute their shares sk_i because $\alpha_0, \alpha_{m+1}, \ldots, \alpha_t$ are unknown, but we can compute their verification keys as $\mathsf{vk}_i = g^{\alpha_0 \lambda_{0,0,T}} \prod_{j \in T, j \ne 0} g^{f^\star(j) \lambda_{i,j,T}}$ with $T = \{0, \ldots, t\}$. It is not difficult to verify that the outputs of the simulator have the same probability distribution as the outputs of the secure DKG protocol using a similar argument as in [35].
6) Choose $k-1$ t-degree random polynomials, $f^1, f^2, \ldots, f^{k-1}$ such that for all $i \in C$ and $1 \le j \le k-1$, $f^j(i) = f^\star(i)$.
7) Compute $\bar{z}_i = (g^\beta)^{f(i)}$ for $i \in C$. Set $\bar{z}_0 = y_0$ and $\bar{z}_i = y_i$ for $m+1 \le i \le t$ where $y_0, y_{m+1}, \ldots, y_t$ are from the DDH problem. Then for all $m+1 \le i \le \ell$, compute $\bar{z}_i = \prod_{j \in T} \bar{z}_j^{\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.
8) Define a function $\mathsf{pc}(x, j, i)$ as follows: invoke random oracle H_1 to get $(x, r, h) \in L_{H_1}$ and return
   - $(h^{f^j(i)}, \pi_i)$ if $j < k$
   - $(\bar{z}_i, \pi_i)$ if $j = k$
   - (vk_i^r, π_i) if $j > k$ (if $r = \perp$, return $\perp$)

   where π_i is a simulated proof generated by calling the random oracle H_2.
9) On an evaluation query (Eval, x, i) for an honest $i \in \mathrm{QUAL} \setminus C$, if x is the j-th distinct value, then return $\mathsf{pc}(x, j, i)$.
10) On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i \in U}, V)$, where $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted nodes, is answered as follows:

a) if $x^\star$ was queried in the evaluation phase and it was the j-th distinct value, then let $j^\star = j$. Else, assume there are so far j' distinct evaluation queries and let $j^\star = j' + 1$.
b) If $\mathcal{A}$ has made at least $t + 1 - m$ queries of the form $(\mathsf{Eval}, x^\star, \cdot)$, then output 0 and stop.
c) Otherwise do as follows:
   i) Run $\mathsf{VerifyEq}_{H_2}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
   ii) Set $(z_i^\star, \cdot) = \mathsf{pc}(x^\star, j^\star, i)$ for $i \in V \setminus C$.
   iii) Compute $v^\star = \prod_{i \in V} z_i^{\star \lambda_{0,i,V}}$.
   iv) If $v^\star = \perp$ then return $\perp$. Otherwise depending on b do as follows:
      A) If $b = 0$ then return $v^\star$;
      B) Else return a uniform random.

11) Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in \mathrm{QUAL} \setminus C$ and i is the $(t+1-m)$-th honest node that $\mathcal{A}$ contacted then output 0 and stop.
12) Receive a guess b' from $\mathcal{A}$ and output b'.

Suppose the k-th distinct evaluation query x_k is the η-th query for random oracle H_1. Let's consider the case when $\eta^\star = \eta$. pc in Step 8 never returns $\perp$ because r is set to $\perp$ only for the $\eta^\star$-th call and we have assumed that this call is for x_k.

- If $y_0 = g^{\alpha_0 \beta}, y_{m+1} = g^{\alpha_{m+1}\beta}, \ldots, y_\ell = g^{\alpha_\ell \beta}$, pc returns $\bar{z}_i$ on x_k which is equal to $H_1(x_k)^{\mathsf{sk}_i}$. In this case, $\mathcal{B}$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ perfectly.
- If $y_0, y_{m+1}, \cdots, y_\ell$ are chosen randomly, then $\bar{z}_0, \bar{z}_1, \cdots, \bar{z}_t$ defines a random polynomial $\hat{f}$ with constraint that for all $i \in C$, $\hat{f}(i) = f^\star(i)$. Therefore, in this case, $\mathcal{B}$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ perfectly.

Since $\eta^\star = \eta$ happens with probability $1/q_{H_1}$, if $\mathcal{A}$ distinguishes hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ with a non-negligible probability δ, then $\mathcal{B}$ can break the (extended) DDH assumption with a non-negligible probability δ/q_{H_1}.

□

Now we are left to show that $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ is indistinguishable from $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$. For a j-th distinct x, $\mathsf{pc}(x, \cdot, \cdot)$ is defined using a unique random t-degree polynomial $f^j(x)$ which only matches with $f(x)$ on C. Since an adversary is allowed to make at most $t - |C|$ evaluation queries on $x^\star$, the adversary can learn the value of $f^j(x)$ on at most t points. As a result, the product $\prod_{i \in V} z_i^{\star \lambda_{0,i,V}}$ with $|V| > t$ when $b = 0$ has at least one $z_i^\star$ for which adversary has no information. Thus, the product appears random to the adversary, which means $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$ are indistinguishable.

K. Security proofs for GLOW-DVRF

Theorem IV.3. $\mathcal{V}^{\text{GLOW-DVRF}}$ satisfies consistency, robustness and uniqueness.

Proof. Consistency can be easily proven similar to Theorem IV.1. The uniqueness follows from the fact that $v = H_2(\pi)$, $v' = H_2(\pi')$ and if $e(\pi, g_2) = e(H_1(x), \mathsf{pk}) = e(\pi', g_2)$ then $\pi = \pi' = H_1(x)^{\mathsf{sk}}$.

Robustness can be proven using the extractability of the NIZKs. When the challenger outputs 1, we have $v^\star \neq \perp$ and $\mathsf{Verify}(\mathsf{pk}, \mathcal{VK}, x^\star, v^\star, \pi^\star) = 0$. W.l.o.g., assume $|U| = t + 1$. Combined with the consistency, we can derive that there exists $i \in U$ such that $v_i \neq H_1(x^\star)^{\mathsf{sk}_i}$, $\mathsf{vk}_i = g_1^{\mathsf{sk}_i}$ and π_i a verified NIZK proof. We consider two cases of i. If $i \in U \setminus C$, this is impossible since v_i is computed correctly by the challenger. If $i \in U \cap C$, we can use the PPT extractor of the NIZKs to derive a k such that $v_i = H_1(x^\star)^k$ and $\mathsf{vk}_i = g_1^k$ which contradicts the hypothesis. □

Theorem IV.4. For any θ such that $\theta \leq t < \ell - \theta$, $\mathcal{V}^{\text{GLOW-DVRF}}$ achieves (θ, t, ℓ)-standard pseudorandomness under the co-CDH assumption in the random oracle model.

Proof. Recall that the standard pseudorandomness does not allow the adversary to query partial evaluations on the challenge plaintext $x^\star$. We construct $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ to simulate all the NIZKs using random oracle H_3 and construct $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ to simulate the DKG protocol. We have that the original standard pseudorandomness game and $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ are indistinguishable due to the zero-knowledge property of NIZKs, and $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ are indistinguishable due to the secrecy of the simulator.

Given an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}_{\mathcal{A}}^{sim}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{sim}(1)$, we shall construct an adversary $\mathcal{B}$ breaks the co-CDH assumption using $\mathcal{A}$ as a subroutine.

Given a co-CDH problem $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\alpha)$ with $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$:

1) Give the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ to $\mathcal{A}$.
2) $\mathcal{A}$ chooses a set C of nodes with $|C| \leq \theta$ to corrupt and send C to $\mathcal{B}$. W.l.o.g., assume $C = \{1, 2, \ldots, m\}$.
3) The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{\$} [q_{H_1}]$ uniformly at random where q_{H_1} is the total number of distinct random oracle queries to H_1.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   - Otherwise,
     – if this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^\beta$ where g_1^β is from the co-CDH problem.
     – else choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $h = g_1^r$.
     – Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$ and output h.

   Give random oracle access to $\mathcal{A}$.
4) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y,
   - If there exists a tuple $(y, c, *)$, then output c.
   - Else verify if $e(y, g_2) = e(g_1^\beta, g_2^\alpha)$:
     – If true, check if there exists a tuple $(\perp, c, \mathsf{true})$ (set by the challenge query) and update the list by changing the $\perp$ to y. If such a tuple does not exist, then choose

- **Generating phase:**
  1) Each node N_i performs a Pedersen-VSS of a random value $a_{i,0}$:
     a) N_i chooses two random t-degree polynomials f_i, f'_i over $\mathbb{Z}_q$:
     $$f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t \qquad f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t}z^t$$
     N_i broadcasts $C_{i,k} = g_1^{a_{i,k}} h_1^{b_{i,k}}$ for $0 \leq k \leq t$. N_i computes the shares $s_{i,j} = f_i(j), s'_{i,j} = f'_i(j)$ for $1 \leq j \leq \ell$ and sends $s_{i,j}, s'_{i,j}$ to node N_j.
     b) Each node N_j verifies the shares he received from the other nodes. N_j checks if
     $$g_1^{s_{i,j}} h_1^{s'_{i,j}} = \prod_{k=0}^{t} (C_{i,k})^{j^k} \tag{4}$$
     for $1 \leq i \leq \ell$. If the check fails for an index i, N_j broadcasts a complaint against N_i.
     c) Each node N_i who received a complaint from node N_j broadcasts the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 4.
     d) Each node marks as *disqualified* any node that either
     - Received more than t complaints in Step 1b, or
     - Answered to a complaint in Step 1c with values that falsify Equation 4.
  2) Each node then builds the set of non-disqualified players QUAL.
  3) Each node N_i sets his share of the secret as $\mathsf{sk}_i = \sum_{j \in \mathrm{QUAL}} s_{j,i}$, the verification key $\mathsf{vk}_i = g_1^{\mathsf{sk}_i}$ and the value $r_i = \sum_{j \in \mathrm{QUAL}} s'_{j,i}$.
- **Extracting phase:**
  4) Each node $i \in$ QUAL exposes exposes $A_{i,0} = g_1^{a_{i,0}}$ via **Feldman-VSS** and $B_{i,0} = g_2^{a_{i,0}}$:
     a) Each node N_i with $i \in$ QUAL, broadcasts broadcasts $A_{i,k} = g_1^{a_{i,k}}$ for $0 \leq k \leq t$ and $B_{i,0} = g_2^{a_{i,0}}$.
     b) Each node N_j verifies the values broadcast by the other nodes in QUAL, formally, for each $i \in$ QUAL, N_j checks if
     $$g_1^{s_{i,j}} = \prod_{k=0}^{t} (A_{i,k})^{j^k} \text{ and } e(A_{i,0}, g_2) = e(g_1, B_{i,0}) \tag{5}$$
     If any of the checks fails for some index i, N_j complains against N_i by broadcasting the values $s_{i,j}, s'_{i,j}$ or $B_{i,0}$ that satisfy Equation 4 but not Equation 5.
     c) For node N_i who receives at least one valid complaint, i.e., values which satisfy Equation 4 but not Equation 5, the other nodes run the re-construction phase of **Pedersen-VSS** to compute $a_{i,0}, f_i, A_{i,k}$ for $0 \leq k \leq t$. Compute the global public key as $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} B_{i,0}$. The distributed secret value $\mathsf{sk} = \sum_{i \in \mathrm{QUAL}} a_{i,0}$ is not explicitly computed by any party.
     d) Each node N_i can reconstruct the verification keys of other nodes:
     - Compute $v_k = \prod_{i \in \mathrm{QUAL}} A_{i,k}$ for $0 \leq k \leq t$.
     - For each $j \in$ QUAL such that $j \neq i$, compute $\mathsf{vk}_j = \prod_{k=0}^{t} v_k^{j^k}$.

Figure 3: DKG protocol for GLOW-DVRF.

a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c, \mathsf{true})$. The oracle outputs c.
- If false, choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c, \mathsf{false})$. The oracle outputs c.

Give random oracle access to $\mathcal{A}$. Note that, when the pairing verification is successful, we can derive that $y = g_1^{\alpha\beta}$.

5) Random oracle H_3 is programmed as follows: Define a list $\mathcal{L}_{H_3} = \emptyset$. For a query on y, if there exists a tuple (y, c), then output c. Otherwise choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_3} = \mathcal{L}_{H_3} \cup (y, c)$. The oracle outputs c. Give random oracle access to $\mathcal{A}$.
6) Run the simulator of the secure DKG protocol to interact with the corrupted nodes. Let QUAL be the non-disqualified nodes. Assume the combined polynomial after the Generating phase is f. In the Extracting phase, when constructing the ℓ-th polynomial $f^\star_\ell$ to replace f_ℓ, the simulator sets up the points $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0), f_\ell(1), \ldots, f_\ell(t)$ for $f^\star_\ell$. The $A_{i,j}, A^\star_{\ell,j}$ and $B^\star_{i,0}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \leq j \leq t$ can be easily computed as $A_{i,j}, A^\star_{\ell,j}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \leq j \leq t$ can be computed similarly as in $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ in Theorem IV.2. $B^\star_{i,0}$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $B^\star_{\ell,0}$ are computed as:
- For $i \in \mathrm{QUAL} \setminus \{\ell\}$, $B_{i,0} = g_2^{a_{i,0}}$ since $\mathcal{B}$ has the coefficients of all the polynomials except the ℓ-th polynomial.
- $B_{\ell,0} = g_2^{\alpha} \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0}^{-1}$.

The global public key $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} B_{i,0} = g_2^{\alpha}$. The corrupted nodes that are included in QUAL have the shares $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i = g_1^{f(i)}$ for $i \in \mathrm{QUAL} \cap C$. The shares of the honest nodes are set to be $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i =$

$g_1^{f(i)}$ for $i \in [m+1, \ell]$. Note that, the simulator cannot compute sk_i for the honest nodes $i \in [t+1, \ell]$ because it does not know the value of α, but can compute $\mathsf{vk}_i = g_1^{\alpha\lambda_{i,0,T}} \prod_{j\in T, j\neq 0} g_1^{f(j)\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.

7) On an evaluation query (Eval, x, i) for an honest i, invoke random oracle H_1 to get $(x, r, h) \in \mathcal{L}_{H_1}$ and
   a) return (vk_i^r, π_i) if $r \neq \perp$, where π_i is a simulated proof generated by calling the random oracle H_3
   b) return $\perp$ if $r = \perp$
8) On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i\in U}, V)$, where $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted nodes, is answered as follows:
   a) if $x^\star$ was not the $\eta^\star$-th query to H_1, then $\mathcal{B}$ aborts.
   b) Otherwise do as follows:
      i) Run $\mathsf{VerifyEq}_{H_3}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
      ii) If there exists a tuple (y, c, true) in H_2, then set $v^\star = c$. Otherwise choose $v^\star \xleftarrow{\$} \mathbb{Z}_q$ and update $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{\perp, v^\star, \mathsf{true}\}$. Depending on b do as follows:
         A) If $b = 0$ then return $v^\star$;
         B) Else return a uniform random.
9) Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, \cdot)$ then output 0 and stop.
10) Receive a guess b' from $\mathcal{A}$.
11) If there exists a tuple (y, c, true) with $y \neq \perp$ in H_2, $\mathcal{B}$ outputs y as a solution to the co-CDH problem.

Since $\eta^\star \xleftarrow{\$} [q_{H_1}]$ is chosen uniformly at random, $\mathcal{B}$ does not abort with probability $1/q_{H_1}$. When abort does not happen, $x_{\eta^\star}$ is also the challenge query $x^\star$, and $\mathcal{B}$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ perfectly. Let's assume abort does not happen.

We define an event E as when the adversary queries $y^\star$ to H_2 such that $e(y^\star, g_2) = e(g_1^\beta, g_2^\alpha)$, i.e., $y^\star = g_1^{\alpha\beta}$. We shall argue that the probability that E does not happen is the same for $b = 0$ and $b = 1$, i.e., $\Pr[\neg E | b = 0, \neg\mathtt{abort}] = \Pr[\neg E | b = 1, \neg\mathtt{abort}]$. Before the challenge query, there is no information about b, the adversary $\mathcal{A}$ sees exactly the same probability distribution on $\mathcal{B}$'s outputs no matter $b = 0$ or $b = 1$. Thus the probability that E does not happen before the challenge query is the same. After the challenge query, because we assume E has not occurred so far, the challenge query returns a uniform random that has never been used before in both $b = 0$ and $b = 1$. Thus, before the adversary queries the random oracle H_2 on $y^\star$, the adversary sees the same probability distribution of $\mathcal{B}$'s outputs after the challenge query. This gives us $\Pr[\neg E | b = 0, \neg\mathtt{abort}] = \Pr[\neg E | b = 1, \neg\mathtt{abort}]$. We can also derive that $\Pr[E | b = 0, \neg\mathtt{abort}] = \Pr[E | b = 1, \neg\mathtt{abort}]$ and

$$\begin{aligned}\Pr[E|\neg\mathtt{abort}] &= \Pr[E|b=0,\neg\mathtt{abort}]\Pr[b=0|\neg\mathtt{abort}] \\ &\quad + \Pr[E|b=1,\neg\mathtt{abort}]\Pr[b=1|\neg\mathtt{abort}] \\ &= \Pr[E|b=0,\neg\mathtt{abort}]\end{aligned}$$

Moreover, when E does not happen, the adversary sees the same probability distribution on $\mathcal{B}$'s outputs, and thus the probability that $\mathcal{A}$ outputs 1 is also the same for $b = 0$ and $b = 1$, i.e.,

$$\Pr[b' = 1 | b = 0, \neg\mathtt{abort}, \neg E] = \Pr[b' = 1 | b = 1, \neg\mathtt{abort}, \neg E]$$

Therefore we can compute $\mathcal{A}$'s advantage of distinguishing $b = 0$ and $b = 1$ as

$$\begin{aligned}\mathbf{Adv} &= |\Pr[b'=1|b=0,\neg\mathtt{abort}] - \Pr[b'=1|b=1,\neg\mathtt{abort}]| \\ &= \left| \begin{array}{l} \Pr[b'=1|b=0,\neg\mathtt{abort},E]\Pr[E|b=0,\neg\mathtt{abort}] \\ \quad + \Pr[b'=1|b=0,\neg\mathtt{abort},\neg E]\Pr[\neg E|b=0,\neg\mathtt{abort}] \\ \quad - \Pr[b'=1|b=1,\neg\mathtt{abort},E]\Pr[E|b=1,\neg\mathtt{abort}] \\ \quad - \Pr[b'=1|b=1,\neg\mathtt{abort},\neg E]\Pr[\neg E|b=1,\neg\mathtt{abort}] \end{array} \right| \\ &= \left| \begin{array}{l} \Pr[b'=1|b=0,\neg\mathtt{abort},E]\Pr[E|b=0,\neg\mathtt{abort}] \\ \quad - \Pr[b'=1|b=1,\neg\mathtt{abort},E]\Pr[E|b=1,\neg\mathtt{abort}] \end{array} \right| \\ &= \left| \begin{array}{l} \Pr[b'=1|b=0,\neg\mathtt{abort},E] \\ \quad - \Pr[b'=1|b=1,\neg\mathtt{abort},E] \end{array} \right| \cdot \Pr[E|b=0,\neg\mathtt{abort}] \\ &\leq \Pr[E|b=0,\neg\mathtt{abort}] = \Pr[E|\neg\mathtt{abort}]\end{aligned}$$

Since we assume $\mathcal{A}$ distinguishes $b = 0$ and $b = 1$ with non-negligible advantage $\mathbf{Adv}$, E happens with non-negligible probability $\Pr[E|\neg\mathtt{abort}]$ when abort does not happen. Therefore $\mathcal{B}$ outputs $y^\star$ as a solution to the co-CDH problem with non-negligible probability at least $\Pr[E \wedge \neg\mathtt{abort}] = \Pr[E|\neg\mathtt{abort}]/q_{H_1}$.

□

Theorem IV.5. For any θ such that $\theta \leq t < \ell - \theta$, $\mathcal{V}^{\text{GLOW-DVRF}}$ is (θ, t, ℓ)-strongly pseudorandom under the *XDH* assumption and *co-CDH* assumption in the random oracle model.

Proof. Similar to Theorem IV.2, we construct $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ to simulate all the NIZKs using random oracle H_3 and construct $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ to simulate the DKG protocol. We have that the original strong pseudorandomness game and $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ are indistinguishable due to the zero-knowledge property of NIZKs, and $\mathsf{Hyb}_{\mathcal{A}}^{zk}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ are indistinguishable due to the secrecy of the simulator.

Assume m with $m \leq \theta$ is the total number of corrupted nodes, we consider two cases $m = t$ and $m < t$. Note that when $\theta < t$, it is obviously the case $m < t$. But when $\theta = t$, it is possible that $m = t$.

a) **The case when $m = t$::** this is the simplest case and the proof is similar to Theorem IV.4. This is because $m = t$ means an adversary has already compromised t nodes and cannot issue any evaluation query on the challenge plaintext $x^\star$. Given an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}_{\mathcal{A}}^{sim}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{sim}(1)$, we can construct an adversary $\mathcal{B}$ breaks the co-CDH assumption using $\mathcal{A}$ as a subroutine. The rest of the proof is the same as Theorem IV.4 and is omitted.

b) **The case when $m < t$::** in this case, the adversary is allowed to make up to $t - m(\geq 1)$ evaluation queries on the challenge plaintext $x^\star$. To prove the strong pseudorandomness, we shall construct k-hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ with $0 \leq k \leq q_E$ and q_E the total number of distinct x in the evaluation/challenge queries. We will show that

- $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ for $1 \le k \le q_E$ are indistinguishable under XDH assumption
- $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$ are indistinguishable under co-CDH assumption

Definition of the k-hybrids. The hybrid games $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ are exactly the same as $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ except how the evaluation/challenge queries are answered on the first k distinct x:

- After the simulator runs the secure DKG protocol to set up the system, the simulator in addition chooses k t-degree random polynomials $f^1, f^2, \cdots, f^k$ such that $f^j(0) = \mathsf{sk}$ and for all $i \in C$, $f^j(i) = \mathsf{sk}_i$. The reason $f^j(0) = \mathsf{sk}$ is because the adversary has the global public key $\mathsf{pk} = g_2^{\mathsf{sk}}$ and can test if $z = \mathsf{sk}$ in an evaluation $H(x)^z$ for any x by checking the equality of pairings $e(H(x)^z, g_2) = e(H(x), \mathsf{pk})$.
- The evaluation queries (Eval, x, i) are answered similar to Theorem IV.2 by calling the function $\mathsf{pc}(x, j, i)$.
- The challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i\in U}, V)$ is answered the same as Theorem IV.2 except Step 3c is replaced by: Compute $z = \prod_{i\in V} {z_i^\star}^{\lambda_{0,i,V}}$, query the random oracle H_2 on z and set $v^\star = H_2(z)$.

Obviously, $\mathsf{Hyb}_{\mathcal{A}}^{0}(b)$ is identical to $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$. $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ only differs in the evaluation queries for the k-th distinct x. Below we shall prove that $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ are indistinguishable from the attacker's point of view.

Lemma A.6. *For $1 \le k \le q_E$, $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ are indistinguishable under XDH assumption in the random oracle model.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ with non-negligible probability then we can construct a PPT adversary $\mathcal{B}$ to break an extended version of the DDH assumption [1] using $\mathcal{A}$ as a subroutine.

The extended XDH problem is given by

$$(\mathbb{G}, q, g_1, g_2, g_1^{\alpha_1}, \cdots, g_1^{\alpha_w}, g_1^{\beta}, y_1, \cdots, y_w)$$

where $y_i = g_1^{\alpha_i \beta}$ for all $i \in \{1, \ldots, w\}$ or randoms. We now construct $\mathcal{B}$ using $\mathcal{A}$ as follows:

1) Give the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ and a list of nodes $\mathcal{N} = \{1, 2, \ldots, \ell\}$ to $\mathcal{A}$.
2) $\mathcal{A}$ chooses a set $C = \{1, 2, \ldots, m\}$ of nodes with $m < t$ to corrupt and send C to $\mathcal{B}$.
3) The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{\$} [q_{H_1}]$ uniformly at random.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   - Otherwise,
     - if this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^\beta$ where g_1^β is from the co-CDH problem.
     - else choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $h = g_1^r$.
   - Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$ and output h.

   Give random oracle access to $\mathcal{A}$.
4) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y, if there exists a tuple (y, c), then output c. Otherwise choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c)$. The oracle outputs c. Give random oracle access to $\mathcal{A}$.
5) The random oracle H_3 is programmed as follows: Define a list $\mathcal{L}_{H_3} = \emptyset$. For a query on y, if there exists a tuple (y, c), then output c. Otherwise choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_3} = \mathcal{L}_{H_3} \cup (y, c)$. The oracle outputs c. Give random oracle access to $\mathcal{A}$.
6) Run the simulator of the secure DKG protocol to interact with the corrupted nodes. Let QUAL be the non-disqualified nodes. Assume the combined polynomial after the Generating phase is f. The simulator chooses $\gamma \xleftarrow{\$} \mathbb{Z}_q$. In the Extracting phase, the ℓ-th polynomial $f_\ell^\star$ is constructed using the points $\gamma - \sum_{j\in \mathrm{QUAL}\setminus\{\ell\}} f_j(0)$, $f_\ell(1)$, …, $f_\ell(m)$, $\alpha_{m+1} - \sum_{j\in \mathrm{QUAL}\setminus\{\ell\}} f_j(m+1)$, …, $\alpha_t - \sum_{j\in \mathrm{QUAL}\setminus\{\ell\}} f_j(t)$ where $\alpha_{m+1}, \ldots, \alpha_t$ are from the XDH problem. $A_{i,j}, A_{\ell,j}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ can be computed similar to Lemma A.5. $B_{i,0}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $B_{\ell,0}^\star$ are computed as:
   - For $i \in \mathrm{QUAL} \setminus \{\ell\}$, $B_{i,0} = g_2^{a_{i,0}}$ since $\mathcal{B}$ has the coefficients of all the polynomials except the ℓ-th polynomial.
   - $B_{\ell,0}^\star = g_2^\gamma \prod_{i\in \mathrm{QUAL}\setminus\{\ell\}} B_{i,0}^{-1}$.

   Let the final combined polynomial be $f^\star$. We know that $f^\star(0) = \gamma$, $f^\star(1) = f(1), \ldots, f^\star(m) = f(m)$, $f^\star(m+1) = \alpha_{m+1}, \ldots, f^\star(t) = \alpha_t$. The global public key is $\mathsf{pk} = g_2^\gamma$. The shares of the honest nodes are set to be $\mathsf{sk}_i = \alpha_i$ and the corresponding verification keys are $\mathsf{vk}_i = g_1^{\alpha_i}$ for $i \in [m+1, t]$ despite $\alpha_{m+1}, \ldots, \alpha_t$ are unknown. For the honest nodes $i \in [t+1, \ell]$, we cannot compute their shares sk_i because $\alpha_{m+1}, \ldots, \alpha_t$ are unknown, but we can compute their verification keys as $\mathsf{vk}_i = g_1^{\gamma\lambda_{i,0,T}} \prod_{j\in T, j\neq 0} g_1^{f^\star(j)\lambda_{i,j,T}}$ with $T = \{0, \ldots, t\}$.
7) Choose $k-1$ t-degree random polynomials, $f^1, f^2, \ldots, f^{k-1}$ such that $f^j(0) = \gamma$ and for all $i \in C$ and $1 \le j \le k-1$, $f^j(i) = f^\star(i)$.
8) Compute $\bar{z}_0 = g_1^{\beta\gamma}$ and $\bar{z}_i = (g_1^\beta)^{f^\star(i)}$ for $i \in C$. Set $\bar{z}_i = y_i$ for $m+1 \le i \le t$ where $y_{m+1}, \ldots, y_t$ are from the XDH problem. Then for all $m+1 \le i \le \ell$, compute $\bar{z}_i = \prod_{j\in T} \bar{z}_j^{\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.
9) Define a function $\mathsf{pc}(x, i, j)$ as follows: invoke random oracle H_1 to get $(x, r, h) \in \mathcal{L}_{H_1}$ and
   - return $(g_1^{f^j(i)\cdot r}, \pi_i)$ if $j < k$
   - return $(\bar{z}_i, \pi_i)$ if $j = k$
   - return (vk_i^r, π_i) if $j > k$ (if $r = \perp$, return $\perp$)

   where π_i is a simulated proof generated by calling the random oracle H_3.
10) On an evaluation query (Eval, x, i) for an honest i, if x is the j-th distinct value, then return $\mathsf{pc}(x, j, i)$.
11) On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i\in U}, V)$, where $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$ and

$z_i^\star$ a set of evaluation shares from the corrupted nodes, is answered as follows:

a) if $x^\star$ was queried in the evaluation phase and it was the j-th distinct value, then let $j^\star = j$. Else, assume there are so far j' distinct evaluation queries and let $j^\star = j' + 1$.
b) If $\mathcal{A}$ has made at least $t + 1 - m$ queries of the form $(\mathsf{Eval}, x^\star, \cdot)$, then output 0 and stop.
c) Otherwise do as follows:
   i) Run $\mathsf{VerifyEq}_{H_3}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
   ii) Set $(z_i^\star, \cdot) = \mathsf{pc}(x^\star, j^\star, i)$ for $i \in V \setminus C$.
   iii) Compute $z = \prod_{i \in V} z_i^{\star \lambda_{0,i,V}}$ and query random oracle H_2 on z. Let $v^\star = H_2(z)$. Depending on b do as follows:
      A) If $b = 0$ then return $v^\star$;
      B) Else return a uniform random.

12) Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in \mathrm{QUAL} \setminus C$ and i is the $(t+1-m)$-th honest node that $\mathcal{A}$ contacted then output 0 and stop.
13) Receive a guess b' from $\mathcal{A}$ and output b'.

Suppose the k-th distinct evaluation query x_k is the η-th query for random oracle H_1. Let's consider the case when $\eta^\star = \eta$. pc in Step 9 never returns $\perp$ because r is set to $\perp$ only for the $\eta^\star$-th call and we have assumed that this call is for x_k.

- If $y_{m+1} = g^{\alpha_{m+1}\beta}, \ldots, y_\ell = g^{\alpha_\ell \beta}$, pc returns $\bar{z}_i$ on x_k which is equal to $H_1(x_k)^{f^\star(i)}$. In this case, $\mathcal{B}$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ perfectly.
- If $y_{m+1}, \cdots, y_\ell$ are chosen randomly, then $\bar{z}_1, \cdots, \bar{z}_t$ defines a random polynomial $\hat{f}$ with constraint that $\hat{f}(0) = \gamma$ and for all $i \in C$, $\hat{f}(i) = f^\star(i)$. Therefore, in this case, $\mathcal{B}$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ perfectly.

Since $\eta^\star = \eta$ happens with probability $1/q_{H_1}$, if $\mathcal{A}$ distinguishes hybrids $\mathsf{Hyb}_{\mathcal{A}}^{k-1}(b)$ and $\mathsf{Hyb}_{\mathcal{A}}^{k}(b)$ with a non-negligible probability δ, then $\mathcal{B}$ can break the (extended) XDH assumption with a non-negligible probability δ/q_{H_1}.

□

Now we are left to show that $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ is indistinguishable from $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$. In $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(b)$, for j-th distinct x, $\mathsf{pc}(x, j, \cdot)$ is computed using a unique random t-degree polynomial f^j which only matches with $f^\star$ on $C \cup \{0\}$.

Lemma A.7. *$\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$ are indistinguishable under co-CDH assumption in the random oracle model.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(0)$ and $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(1)$, then we can construct an adversary $\mathcal{B}$ breaks the co-CDH assumption using $\mathcal{A}$ as a subroutine.

Given a co-CDH problem $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\alpha)$ with $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$:

1) Give the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ and a list of nodes $\mathcal{N} = \{1, 2, \ldots, \ell\}$ to $\mathcal{A}$.
2) $\mathcal{A}$ chooses a set $C = \{1, 2, \ldots, m\}$ of nodes with $m < t$ to corrupt and send C to $\mathcal{B}$.
3) The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{\$} [q_{H_1}]$ uniformly at random.
   - If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
   - Otherwise,
     – If this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^\beta$ where g_1^β is from the co-CDH problem.
     – Else choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $h = g_1^r$.
   - Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup \{(x, r, h)\}$ and output h.

   Give random oracle access to $\mathcal{A}$.
4) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y,
   - If there exists a tuple $(y, c, \cdot)$, then output c.
   - Else verify if $e(y, g_2) = e(g_1^\beta, g_2^\alpha)$:
     – If true, check if there exists a tuple $(\perp, c, \mathsf{true})$ (set by the challenge query) and update the list by changing the $\perp$ to y. If such a tuple does not exist, then choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c, \mathsf{true})$. The oracle outputs c.
     – If false, choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{(y, c, \mathsf{false})\}$. The oracle outputs c.

   Give random oracle access to $\mathcal{A}$. Note that, when the verification is successful, we can derive that $y = g_1^{\alpha\beta}$.
5) The random oracle H_3 is programmed as follows: Define a list $\mathcal{L}_{H_3} = \emptyset$. For a query on y, if there exists a tuple (y, c), then output c. Otherwise choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_3} = \mathcal{L}_{H_3} \cup (y, c)$. The oracle outputs c. Give random oracle access to $\mathcal{A}$.
6) Run the simulator of the secure DKG protocol to interact with the corrupted nodes. Let QUAL be the non-disqualified nodes determined after the Generating phase. Assume the combined polynomial after the Generating phase is f. In the Extracting phase, for the ℓ-th polynomial, $\mathcal{B}$ constructs $f_\ell^\star$ to replace f_ℓ: $f_\ell^\star$ has the values $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1)$, …, $f_\ell(t)$. $A_{i,j}, A_{\ell,j}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ can be computed similar to [35]. $B_{i,0}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $B_{\ell,0}^\star$ are computed as:
   - For $i \in \mathrm{QUAL} \setminus \{\ell\}$, $B_{i,0} = g_2^{a_{i,0}}$ since $\mathcal{B}$ has the coefficients of all the polynomials except the ℓ-th polynomial.
   - $B_{\ell,0}^\star = g_2^\alpha \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0}^{-1}$.

   The global public key is $\mathsf{pk} = B_{\ell,0}^\star \prod_{i \in \mathrm{QUAL} \setminus \{\ell\}} B_{i,0} = g_2^\alpha$. Let the final combined polynomial be $f^\star$. We know that $f^\star(0) = \alpha$, $f^\star(1) = f(1), \ldots, f^\star(t) = f(t)$. The corrupted nodes that are included in QUAL have the shares $\mathsf{sk}_i = f^\star(i)$ and $\mathsf{vk}_i = g_1^{f^\star(i)}$ for $i \in \mathrm{QUAL} \cap C$. The shares of the honest nodes are set to be $\mathsf{sk}_i = f^\star(i)$ and $\mathsf{vk}_i = g_1^{f^\star(i)}$ for $m + 1 \le i \le \ell$. Note that, for

the honest nodes $t+1 \leq i \leq \ell$, the simulator cannot compute sk_i because it does not know the value of α, but can compute $\mathsf{vk}_i = g_1^{\alpha\lambda_{i,0,T}} \prod_{j\in T, j\neq 0} g_1^{f^\star(j)\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.

7) Choose q_E t-degree random polynomials, $f^1, f^2, \ldots, f^{q_E-1}$ such that for $1 \leq j \leq q_E - 1$, $f^j(0) = \alpha$ and for all $i \in C$ $f^j(i) = f^\star(i)$. Note that, we cannot compute $f^j(i)$ for $i \geq m+1$ because α is unknown, but we can compute $g_1^{f^j(i)}$ in the way similar to vk_i.
8) On an evaluation query (Eval, x, i) for an honest i, invoke random oracle H_1 to get $(x, r, h) \in \mathcal{L}_{H_1}$ and
   a) return $(g_1^{f^j(i)\cdot r}, \pi_i)$ if $r \neq \perp$, where π_i is a simulated proof generated by calling the random oracle H_3
   b) if $r = \perp$, choose $z_i \xleftarrow{\$} \mathbb{G}_1$ and generate a simulated proof π_i using random oracle H_3 and return (z_i, π_i).
9) On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star, \pi_i)\}_{i\in U}, V)$, where $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted nodes, is answered as follows:
   a) if $x^\star$ was not the $\eta^\star$-the query to H_1, then $\mathcal{B}$ aborts.
   b) Otherwise do as follows:
      i) Run $\mathsf{VerifyEq}_{H_3}$ to check the proofs π_i for $i \in U$. If any check fails, output 0 and stop.
      ii) If there exists a tuple (y, c, true) in H_2, then set $v^\star = c$. Otherwise choose $v^\star \xleftarrow{\$} \mathbb{Z}_q$ and update $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{\perp, v^\star, \mathsf{true}\}$. Depending on b do as follows:
         A) If $b = 0$ then return $v^\star$;
         B) Else return a uniform random.
10) Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, i)$ for some $i \in \mathrm{QUAL} \setminus C$ and i is the $(t+1-m)$-th honest node that $\mathcal{A}$ contacted then output 0 and stop.
11) Receive a guess b' from $\mathcal{A}$.
12) If there exists a tuple (y, c, true) with $y \neq \perp$ in H_2, $\mathcal{B}$ outputs y as a solution to the co-CDH problem.

The probability that $\mathcal{B}$ does not abort is $1/q_{H_1}$ since $\eta^\star$ is uniformly and randomly chosen. Let's consider the case when `abort` does not happen. In this case, the challenge plaintext $x^\star$ is also the $\eta^\star$-th distinct query to H_1 and $H_1(x^\star) = g_1^\beta$. In Step 8b, to answer the evaluation queries on $x^\star$, we choose a random z_i. The adversary is allowed to make at most $t-m$ evaluation queries on $x^\star$. Let these evaluation queries be $(\mathsf{Eval}, x^\star, i_1), \ldots, (\mathsf{Eval}, x^\star, i_{t-m})$ and the returned randoms be $z_{i_1}, \ldots, z_{i_{t-m}}$. These values $(i_1, z_{i_1}), \ldots, (i_{t-m}, z_{i_{t-m}})$ together with $(0, \alpha\beta)$ and $(1, f^\star(1)\beta), \ldots, (m, f^\star(m)\beta)$ implicitly defines an unique random polynomial $\hat{f}$ with $\hat{f}(0) = \alpha$, $\hat{f}(1) = f^\star(1)$, $\ldots$, $\hat{f}(m) = f^\star(m)$ and $\hat{f}$ is only used for computing evaluations on $x^\star$. Note that, it does not matter when the adversary makes less than $t-m$ evaluation queries on $x^\star$ since the polynomial $\hat{f}$ will never be explicitly computed. Therefore, $\mathcal{B}$ simulates $\mathsf{Hyb}_{\mathcal{A}}^{q_E}(b)$ perfectly when aborts does not occur.

We define an event E as when the adversary queries $y^\star$ to H_2 such that $e(y^\star, g_2) = e(g_1^\beta, g_2^\alpha)$, i.e., $y^\star = g_1^{\alpha\beta}$. Similar to Theorem IV.4, we can prove that $\mathcal{A}$'s advantage $\mathbf{Adv}$ of distinguishing $b = 0$ and $b = 1$ is not bigger than $\Pr[E|\neg\mathtt{abort}]$, i.e., $\mathbf{Adv} \leq \Pr[E|\neg\mathtt{abort}]$. Therefore $\mathcal{B}$ outputs $y^\star$ as a solution to the co-CDH problem with non-negligible probability $\Pr[E|\neg\mathtt{abort}]/q_{H_1}$. □

L. *Security proofs for* Dfinity-DVRF

As outlined in [39], Dfinity-DVRF uses a well-known DKG protocol by Pedersen [53] where the adversary can bias the distribution of the joint public key [35]. In order to formalise Dfinity-DVRF and study its security properties, instead of the biasable DKG protocol we use the secure DKG protocol proposed in [35] and we shall prove that the resulting DVRF construction satisfies standard pseudorandomness.

Let $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, h_1, h_2)$ be a bilinear pairing group where the co-CDH assumption holds and g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively (same applies to h_1, h_2). Let $H_1 : \{0,1\}^* \mapsto \mathbb{G}_1$, $H_2 : \mathbb{G}_1 \mapsto \{0,1\}^{b(\lambda)}$ be hash functions. $\mathcal{V}^{\text{Dfinity-DVRF}}$ is defined as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ proceeds in the same way as in the non-compact case, with the difference that exponentiations take place in group $\mathbb{G}_2$, i.e. the verification keys $\mathsf{vk}_i = g_2^{\mathsf{sk}_i} \in \mathbb{G}_2$ and the global public key $\mathsf{pk} = \sum_{i\in\mathrm{QUAL}} g_2^{a_{i,0}} \in \mathbb{G}_2$. Full details can be found in Figure 4.

$\mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ outputs a share $s_x^i = (i, v_i)$ with $v_i = H_1(x)^{\mathsf{sk}_i}$.

$\mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \mathcal{E})$: parses list $\mathcal{E} = \{s_x^{j_1}, \ldots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \geq t+1$ partial evaluation candidates originating from $|\mathcal{E}|$ different nodes, and obtains verification keys $\mathsf{vk}_{j_1}, \ldots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,
1) Identifies an index subset $I = \{i_1, \ldots, i_{t+1}\}$ such that for every $i \in I$ it holds that $e(v_i, g_2) = e(H_1(x), \mathsf{vk}_i)$. If no such subset exists, outputs $\perp$.
2) Sets $\pi \leftarrow \prod_{j\in I} v_j^{\lambda_{0,j,I}}$ and $v = H_2(\pi)$.
3) Outputs (v, π).

$\mathsf{Verify}(\mathsf{pk}, x, v, \pi)$ outputs 1 if the equation holds: $e(\pi, g_2) = e(H_1(x), \mathsf{pk})$ and $v = H_2(\pi)$. Otherwise output 0.

Theorem IV.6. $\mathcal{V}^{\text{Dfinity-DVRF}}$ satisfies consistency, robustness and uniqueness.

Proof. The proof of consistency is exactly as the proof of Theorem IV.1. The uniqueness follows from the fact that $v = H_2(\pi)$, $v' = H_2(\pi')$ and if $e(\pi, g_2) = e(H_1(x), \mathsf{pk}) = e(\pi', g_2)$ then $\pi = \pi' = H_1(x)^{\mathsf{sk}}$.

For robustness: Suppose there exists an adversary $\mathcal{A}$ leading the challenger to output 1 with non-negligible probability, i.e., $v^\star \neq \perp$ and $\mathsf{Verify}(\mathsf{pk}, x^\star, v^\star, \pi^\star) = 0$. Apparently $v^\star = H_2(\pi^\star)$ since it is guaranteed by the Combine function. Thus, when the verification fails, it can only be because $e(\pi^\star, g_2) \neq e(H_1(x^\star), \mathsf{pk})$ which gives us $\pi^\star \neq H_1(x^\star)^{\mathsf{sk}}$. W.l.o.g., assume $|U| = t+1$. Based on the Equation 3 in

- **Generating phase:**
  1) Each node N_i performs a Pedersen-VSS of a random value $a_{i,0}$:
     a) N_i chooses two random t-degree polynomials f_i, f'_i over $\mathbb{Z}_q$:
     $$f_i(z) = a_{i,0} + a_{i,1}z + \cdots + a_{i,t}z^t \qquad f'_i(z) = b_{i,0} + b_{i,1}z + \cdots + b_{i,t}z^t$$
     N_i broadcasts $C_{i,k} = g_2^{a_{i,k}} h_2^{b_{i,k}}$ for $0 \leq k \leq t$. N_i computes the shares $s_{i,j} = f_i(j), s'_{i,j} = f'_i(j)$ for $1 \leq j \leq \ell$ and sends $s_{i,j}, s'_{i,j}$ to node N_j.
     b) Each node N_j verifies the shares he received from the other nodes. N_j checks if
     $$g_2^{s_{i,j}} h_2^{s'_{i,j}} = \prod_{k=0}^{t} (C_{i,k})^{j^k} \tag{6}$$
     for $1 \leq i \leq \ell$. If the check fails for an index i, N_j broadcasts a complaint against N_i.
     c) Each node N_i who received a complaint from node N_j broadcasts the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 6.
     d) Each node marks as *disqualified* any node that either
     - Received more than t complaints in Step 1b, or
     - Answered to a complaint in Step 1c with values that falsify Equation 6.
  2) Each node then builds the set of non-disqualified players QUAL.
  3) Each node N_i sets his share of the secret as $\mathsf{sk}_i = \sum_{j \in \mathrm{QUAL}} s_{j,i}$, the verification key $\mathsf{vk}_i = g_2^{\mathsf{sk}_i}$ and the value $r_i = \sum_{j \in \mathrm{QUAL}} s'_{j,i}$.
- **Extracting phase:**
  4) Each node $i \in \mathrm{QUAL}$ exposes $A_{i,0} = g_2^{a_{i,0}}$ via **Feldman-VSS**:
     a) Each node N_i with $i \in \mathrm{QUAL}$, broadcasts $A_{i,k} = g_2^{a_{i,k}}$ for $0 \leq k \leq t$.
     b) Each node N_j verifies the values broadcast by the other nodes in QUAL, formally, for each $i \in \mathrm{QUAL}$, N_j checks if
     $$g_2^{s_{i,j}} = \prod_{k=0}^{t} (A_{i,k})^{j^k} \tag{7}$$
     If the check fails for some index i, N_j complains against N_i by broadcasting the values $s_{i,j}, s'_{i,j}$ that satisfy Equation 6 but not Equation 7.
     c) For node N_i who receives at least one valid complaint, i.e., values which satisfy Equation 6 but not Equation 7, the other nodes run the re-construction phase of **Pedersen-VSS** to compute $a_{i,0}, f_i, A_{i,k}$ for $0 \leq k \leq t$. Compute the global public key as $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} A_{i,0}$. The distributed secret value $\mathsf{sk} = \sum_{i \in \mathrm{QUAL}} a_{i,0}$ is not explicitly computed by any party.
     d) Each node N_i can reconstruct the verification keys of other nodes:
     - Compute $v_k = \prod_{i \in \mathrm{QUAL}} A_{i,k}$ for $0 \leq k \leq t$.
     - For each $j \in \mathrm{QUAL}$ such that $j \neq i$, compute $\mathsf{vk}_j = \prod_{k=0}^{t} v_k^{j^k}$.

Figure 4: Adaptation of the DKG protocol [35] to pairing groups for Dfinity-DVRF.

Theorem IV.1, we can derive that there exists $i \in U$ such that $v_i \neq H_1(x^\star)^{\mathsf{sk}_i}$ and $\mathsf{vk}_i = g_2^{\mathsf{sk}_i}$. We consider two cases of i. If $i \in U \setminus C$, this is impossible since $v_i = H_1(x^\star)^{\mathsf{sk}_i}$. If $i \in U \cap C$, we know that $e(v_i, g_2) = e(H_1(x^\star), \mathsf{vk}_i)$ by the verification performed in Combine, which gives us that $v_i = H_1(x^\star)^{\mathsf{sk}_i}$. This concludes the proof. □

Theorem IV.7. $\mathcal{V}^{\text{Dfinity-DVRF}}$ satisfies standard pseudorandomness under co-CDH assumption in the random oracle model.

Proof. Recall that standard pseudorandomness does not allow the adversary to query partial evaluations on the challenge plaintext $x^\star$.

We construct $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ to simulate the DKG protocol and the original standard pseudorandomness game is indistinguishable with $\mathsf{Hyb}^{sim}_{\mathcal{A}}(b)$ because of the secrecy of the simulator. Below we shall prove $\mathsf{Hyb}^{sim}_{\mathcal{A}}(0)$ and $\mathsf{Hyb}^{sim}_{\mathcal{A}}(1)$ are indistinguishable under co-CDH assumption. Suppose there exists an adversary $\mathcal{A}$ that distinguishes $\mathsf{Hyb}^{sim}_{\mathcal{A}}(0)$ and $\mathsf{Hyb}^{sim}_{\mathcal{A}}(1)$, then we can construct an adversary $\mathcal{B}$ breaks the co-CDH assumption using $\mathcal{A}$ as a subroutine.

Given a co-CDH problem $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, g_1^\alpha, g_1^\beta, g_2^\alpha)$ with $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, $\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{B}$'s goal is to output $g_1^{\alpha\beta}$:

1) Give the public parameters $(e, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2)$ to $\mathcal{A}$.
2) $\mathcal{A}$ chooses a set C of nodes with $|C| \leq t$ to corrupt and send C to $\mathcal{B}$. W.l.o.g., assume $C = \{1, 2, \ldots, m\}$.
3) The random oracle H_1 is answered as follows: initialise $\mathcal{L}_{H_1} = \emptyset$. Let q_{H_1} be the total number of distinct random oracle queries asked in this game. Choose an index $\eta^\star \xleftarrow{\$}$

$[q_{H_1}]$ uniformly at random.

- If there exists a tuple $(x, r, h) \in \mathcal{L}_{H_1}$, output h.
- Otherwise,
  - if this is the $\eta^\star$-th distinct call, set $r = \perp$ and $h = g_1^\beta$ where g_1^β is from the co-CDH problem.
  - else choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and set $h = g_1^r$.
  - Update $\mathcal{L}_{H_1} = \mathcal{L}_{H_1} \cup (x, r, h)$ and output h.

Give random oracle access to $\mathcal{A}$.

4) The random oracle H_2 is programmed as follows: Define a list $\mathcal{L}_{H_2} = \emptyset$. For a query on y,
   - If there exists a tuple $(y, c, *)$, then output c.
   - Else verify if $e(y, g_2) = e(g_1^\beta, g_2^\alpha)$:
     - If true, check if there exists a tuple $(\perp, c, \mathsf{true})$ (set by the challenge query) and update the list by changing the $\perp$ to y. If such a tuple does not exist, then choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c, \mathsf{true})$. The oracle outputs c.
     - If false, choose a random $c \xleftarrow{\$} \mathbb{Z}_q$ and update the list $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup (y, c, \mathsf{false})$. The oracle outputs c.

   Give random oracle access to $\mathcal{A}$. Note that, when the pairing verification is successful, we can derive that $y = g_1^{\alpha\beta}$.
5) Run the simulator of the secure DKG protocol to interact with the corrupted nodes. The protocol constructs $C_{i,k}$ using g_2, h_2 from $\mathbb{G}_2$. Let QUAL be the non-disqualified nodes. Assume the combined polynomial after the Generating phase is f. In the Extracting phase, when constructing the ℓ-th polynomial $f_\ell^\star$ to replace f_ℓ, the simulator sets $f_\ell^\star$ to have the values $\alpha - \sum_{j \in \mathrm{QUAL} \setminus \{\ell\}} f_j(0)$, $f_\ell(1), \ldots, f_\ell(t)$. The computation of $A_{i,j}, A_{\ell,j}^\star$ for $i \in \mathrm{QUAL} \setminus \{\ell\}$ and $0 \le j \le t$ are similar to $\mathsf{Hyb}_{\mathcal{A}}^{sim}(b)$ in Theorem IV.2 except they are constructed on $\mathbb{G}_2$. The global public key is $\mathsf{pk} = g_2^\alpha$. The corrupted nodes that are included in QUAL have the shares $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i = g_2^{f(i)}$. The shares of the honest nodes $i \in [t+1, \ell]$ are set to be $\mathsf{sk}_i = f(i)$ and $\mathsf{vk}_i = g_2^{f(i)}$. Note that, the simulator cannot compute sk_i for the honest nodes $i \in [t+1, \ell]$ because it does not know the value of α, but can compute vk_i as $g_2^{\alpha\lambda_{i,0,T}} \prod_{j \in T, j \neq 0} g_2^{f(j)\lambda_{i,j,T}}$ with $T = \{0, 1, \ldots, t\}$.
6) On an evaluation query (Eval, x, i) for an honest i, invoke random oracle H_1 to get $H_1(x) = (x, r, h)$ and
   a) return vk_i^r if $r \neq \perp$
   b) return $\perp$ if $r = \perp$
7) On the challenge query $(\mathsf{Challenge}, x^\star, \{(i, z_i^\star)\}_{i \in U}, V)$, where $|V| > t$ and $V \subseteq \mathrm{QUAL}$ and $U \subseteq V \cap C$ and $z_i^\star$ a set of evaluation shares from the corrupted nodes, is answered as follows:
   a) If $x^\star$ was not the $\eta^\star$-the query to H_1, then $\mathcal{B}$ aborts.
   b) Otherwise do as follows:
      i) Check if $e(v_i, g_2) = e(H_1(x), \mathsf{vk}_i)$ for $i \in U$. If any check fails, output 0 and stop.
      ii) If there exists a tuple (y, c, true) in H_2, then set $v^\star = c$. Otherwise choose $v^\star \xleftarrow{\$} \mathbb{Z}_q$ and update $\mathcal{L}_{H_2} = \mathcal{L}_{H_2} \cup \{\perp, v^\star, \mathsf{true}\}$. Depending on b do as follows:
         A) If $b = 0$ then return $v^\star$;
         B) Else return a uniform random.
8) Continue answering evaluation queries as before, but if $\mathcal{A}$ makes queries of the form $(\mathsf{Eval}, x^\star, \cdot)$ then output 0 and stop.
9) Receive a guess b' from $\mathcal{A}$.
10) If there exists a tuple (y, c, true) with $y \neq \perp$ in H_2, $\mathcal{B}$ outputs y as a solution to the co-CDH problem.

The probability that $\mathcal{B}$ does not abort is $1/q_{H_1}$ since $\eta^\star$ is uniformly and randomly chosen. Let's consider the case when `abort` does not happen. In this case, the challenge plaintext $x^\star$ is also the $\eta^\star$-th distinct query to H_1 and $H_1(x^\star) = g_1^\beta$. The adversary is not allowed to query $(\mathsf{Eval}, x^\star, \cdot)$ due to the definition of standard pseudorandomness. In other words, the Step 6b will never be executed. In this case $\mathcal{B}$ simulates the standard pseudorandomness game perfectly from $\mathcal{A}$'s point of view.

The rest of the analysis is similar to Theorem IV.4. We define an event E as when the adversary queries $y^\star$ to H_2 such that $e(y^\star, g_2) = e(g_1^\beta, g_2^\alpha)$, i.e., $y^\star = g_1^{\alpha\beta}$. From $\mathcal{A}$'s point of view, the cases $b = 0$ and $b = 1$ are exactly the same unless E happens. We can prove that $\mathcal{A}$'s advantage $\mathbf{Adv}$ of distinguishing $b = 0$ and $b = 1$ is not bigger than $\Pr[E|\neg\mathtt{abort}]$, i.e., $\mathbf{Adv} \le \Pr[E|\neg\mathtt{abort}]$. Therefore $\mathcal{B}$ outputs $y^\star$ as a solution to the co-CDH problem with non-negligible probability $\Pr[E|\neg\mathtt{abort}]/q_{H_1}$. □

Strong pseudorandomness of Dfinity**-**DVRF**.** Adversarial capabilities in standard pseudorandomness of Dfinity-DVRF and unforgeability of threshold BLS signatures [11] go hand in hand, as in both cases an adversary is not given the option to make partial queries on the challenge plaintext $x^\star$. It is not surprising then that the previous security reduction does not allow to prove strong pseudorandomness of Dfinity-DVRF. Indeed, since the verification key vk_i of the i-th node is in $\mathbb{G}_2$, then the adversary can verify if an answer v_i is correct or not by checking the pairing $e(v_i, g_2) = e(H_1(x^\star), \mathsf{vk}_i)$. Alas, the security reduction does not provide the challenger with knowledge of $\log_{g_2} \mathsf{vk}_i$, and then the challenger cannot answer a partial signature/evaluation query on the challenge plaintext.