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

David Galindo[*][†], Jia Liu[*], Mihair Ordean[†], Jin-Mann Wong[‡]
[*]*Fetch.ai, UK*
[†]*University of Birmingham, UK*
[‡]*British Antarctic Survey, UK*

***Abstract*—We provide a systematic analysis of two related 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. These two protocols are run by multiple network nodes where each node contributes with a partial evaluation and the collection of these partial values is used to evaluate a pseudorandom function. 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. 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 academia and industry within our framework, which serves as an evidence of its wider applicability. Finally, we report on experimental evaluations of our newly introduced DVRFs with implementations under different cryptographic libraries, and we also report preliminary benchmark results on two of the DRBs obtained from the generic DVRF-to-DRB transformation. Our benchmarks can be independently verified as we provide an open source C++ reference implementation of the new DVRFs. Finally, we conclude that our new DRB instantiations are the most efficient instantiations currently available while enjoying strong and formally proven security properties.**

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

Jin-Mann Wong co-authored this work while with Fetch.ai.

1. Introduction

In recent years there has been prolific development in blockchain technologies [47], [55], 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, such as Tendermint [14], Ethereum 2.0 [15], OmniLedger [42], Dfinity [36] and Algorand [34], involve allocating the creation of blocks with a block producer, whose selection procedure most often than not requires a method for collective randomness sampling. 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.

Verifiable Random Function (VRF). This primitive was introduced by Micali, Rabin and Vadhan [44] 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. The IETF is pursuing standardization of a verifiable random function [35].

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

DOI 10.1109/EuroSP51992.2021.00017

public verifiability and digital signatures is well-known [7], [30].

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 maximum 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 3). This separation result implies that the privacy provided by strong pseudorandomness is 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 4 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 4.1), whereas the other construction is named GLOW-DVRF and uses cryptographic pairings (Section 4.2). 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 provides a compact proof, i.e., with constant-size.

In Section 5 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 DRB formalisation 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 academia 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 6).

We report on an experimental evaluation of GLOW-DVRF and DDH-DVRF using the cryptographic libraries MCL [46], RELIC [2] and Libsodium [8]. We compare them with Dfinity-DVRF [36], a prominent DVRF construction in the blockchain space. 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 [29] (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 [14] using Cosmos SDK [38]. Preliminary benchmarks can be found in Section 7 and allow us to conclude that our new DRB instantiations are the most efficient DRB constructions currently available, as the random beacon values are computed in a non-interactive fashion, while enjoying strong and formally proven security properties.

1.1. Related Work

To our knowledge, the first distributed VRF construction was proposed by Dodis [25] 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: DDH-DVRF use a protocol by Gennaro, Jarecki, Krawczyk and Rabin [32], whereas GLOW-DVRF required several modifications to the latter. Kuchta and Manulis [43] 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 [43] 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 [44]) and in latency (as they involve sequential interaction between a number of peers).

An influential DVRF (used e.g. in [20]–[22], [39], [50]) that we name Dfinity-DVRF was sketched in [36]. We discuss how the techniques [9], [32] that have been used to prove standard unforgeability of threshold BLS signatures [13] 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 DDH-DVRF bears more resemblance to the distributed pseudorandom function (DPRF) proposed in [1, Figure 6]. The DPRF construction in [1] is based on a trusted setup and is only privately variable, while we update it to public verifiability and a trustless setup in our DDH-DVRF. Rigorously proving public verifiability requires to update the definitions of pseudorandomness and robustness [1] in the presence of publicly available proofs of correctness, as well as defining and proving uniqueness.

2. 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.

Definition 2.1 (Equality of Discrete Logarithms NIZK [18])**.** *Our constructions use NIZK proof systems as an ingredient. Specifically, we need the Equality of Discrete Logarithms proof system* $(\mathsf{ProveEq}_H, \mathsf{VerifyEq}_H)$ *to show* $k = \log_g x = \log_h y$*:*

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

Definition 2.2 (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* $\alpha, \beta \overset{\$}{\leftarrow} \mathbb{Z}_q^*$*. The* Decisional Diffie-Hellman assumption *holds if for* $\gamma \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ *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*, where* λ *is a measure of the bit length of* q*.*

Definition 2.3 (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 [33] and group exponentiations and pairing operations can also be efficiently computed [24]. The security of our pairing-based constructions may require the Computational co-Diffie-Hellman (co-CDH) assumption [11] which states CDH is hard in $\mathbb{G}_1$, and XDH assumption which states that DDH is hard in $\mathbb{G}_1$.

Definition 2.4 (Computational co-CDH assumption [11])**.** *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* $\alpha, \beta \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ *and* $g_1 \overset{\$}{\leftarrow} \mathbb{G}_1, g_2 \overset{\$}{\leftarrow} \mathbb{G}_2$*. 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 2.5 (XDH assumption [16])**.** *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* $\mathbb{G}_1$*.*

Definition 2.6 (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* $|\Delta| - 1$ *we have* $\sum_{i \in \Delta} f(i)\lambda_{0,i,\Delta} = f(0)$*.*

3. 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) [44], 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 [1], [48], although without a trusted setup.

3.1. 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. We assume the same network model as in [32], where the distributed key generation (DKG) protocol [32] works in the (partially/fully) synchronous communication model and players are assumed to be equipped with synchronized clocks. 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 3.1 (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 [17], [41].
- *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 A.

3.2. Strong and Standard Pseudorandomness

Next we give rigorous definitions for two *pseudorandomness* properties against active adversaries that generalise the pseudorandomness definitions in distributed scenarios [1], [48]. 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 captured 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 [48], which we rename *standard pseudorandomness*.

Definition 3.2 (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* $\mathcal{A}$,

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

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* $\{s_{x^\star}^i\}_{N_i \in U \cap C}$ *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* $t - |C|$ *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 nodes* $N_j \in U \setminus C$ *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* $v^\star$*; else 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) *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* $\perp$*. Furthermore, $\mathcal{A}$ is not allowed to query* $(\mathsf{Eval}, x^\star, i)$ *for more than* $t - |C|$ *different honest nodes* $N_i \in \mathrm{QUAL} \setminus C$.

[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, [1], [48], only consider the situation $\theta = t$, which is not always the case in practice. For example, the DKG protocol in Dfinity-DVRF [36] 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., [32]), and splitting θ and t can make the choice of t and ℓ more flexible for specific applications.

Our definition of strong pseudorandomness extends and refines the definition of pseudorandomness proposed in [1] to a *trustless* setting. A trusted setup phase in [1] can be leveraged to let the adversary choose which nodes to corrupt after inspecting the public parameters. In our trustless setup the absence of a trusted phase restricts the adversary to choose the corrupted nodes before seeing the public parameters. However, the adversary is empowered by letting it select the corrupted nodes' local secret keys and influence the public parameters computation by interacting with the honest nodes. This allows the adversary to interfere with the setup phase which is not the case in trusted setups such as [1]. We leave the study of a stronger notion of adaptive security with trustless setup in which the adversary is allowed to corrupt any node at any point during the run of the protocol to future work.

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 [12], [26], [43]. The usage of this weaker definition of pseudorandomness can also be found in the DRB literature e.g. [17]. Using our refined approach to defining (strong) pseudorandomness, we define (θ, t, ℓ)*-standard pseudorandomness* as follows.

Definition 3.3 (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 $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{StdPRand}}_{\mathcal{V},\mathcal{A}} = \big|\Pr[\mathrm{StdPRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathrm{StdPRand}_{\mathcal{V},\mathcal{A}}(1^\lambda, 1) = 1]\big| \leq \mathsf{negl}(\lambda)$, 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 $x^\star$.*

3.3. 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 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 4. 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 $i \in$ QUAL obtains a pair of $(\mathsf{sk}_i, \mathsf{vk}_i)$.

$\mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ outputs $(i, v_i, \pi_i) \leftarrow \mathsf{PartialEval}'(x, \mathsf{sk}_i, \mathsf{vk}_i)$.

1. A side condition should be $|\mathrm{QUAL}| > t$ in order to make sure the scheme is non-trivial. In Section 4, we can see that our constructions of DDH-DVRF and GLOW-DVRF both satisfy this condition.

$\mathsf{Combine}(\mathsf{pk}, \mathcal{VK}, x, \mathcal{E})$ when $|\mathcal{E}| > t$, it is also the case $|\mathcal{E}| > t'$ because $t > t'$. 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.

Theorem 3.1. *$\mathcal{V}$ 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 $N_{\theta+1}, \ldots, N_t \in$ 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 3.1.1. *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 6 that, in the context of Dencentralized Random Beacons, Algorand DRB [34] is $(0, t, \ell)$-standard pseudorandom, but is not $(0, t, \ell)$-strongly pseudorandom, which offers an independent and non-contrived confirmation of the separation between pseudorandomness properties.

4. 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 standalone DDH-VRF presented in [30] (which is being proposed for standardisation [35]). On the other hand, GLOW-DVRF is a pairing-based DVRF that achieves strong pseudorandomness with compact proofs (in contrast to DDH-DVRF).

The setup phase of our proposed DVRF constructions is based on the distributed key generation (DKG) protocol in [32], which is a secret sharing protocol jointly run by ℓ nodes. We assume $\theta \leq t < \ell - \theta$, meaning the number of corrupted nodes θ is at most t and the number of honest nodes $\ell - \theta$ is higher than the threshold t. This side-condition guarantees the corrupted nodes cannot disqualify any honest node during the run of the DKG protocol and the honest nodes are able to complete the DKG protocol correctly without the cooperation of the corrupted nodes being necessary, i.e., $|\mathrm{QUAL}| > t$.

4.1. DDH-DVRF: a strongly pseudorandom DDH-based DVRF with non-compact proofs

Let $(\mathbb{G}, g, q)$ be a multiplicative group. Let $H_1 : \{0,1\}^* \mapsto \mathbb{G}$ and $H_2 : \{0,1\}^* \mapsto \mathbb{Z}_q$ be hash functions. We construct $\mathcal{V}^{\text{DDH-DVRF}} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ as follows:

$\mathsf{DistKG}(1^\lambda, t, \ell)$ is run by ℓ participating nodes $\mathcal{N} = \{N_1, \dots, N_\ell\}$. Each node N_i chooses a random polynomial $f_i(z) = a_{i,0} + a_{i,1}z + \dots + a_{i,t}z^t$. The protocol outputs a set of qualified nodes $\text{QUAL} \subseteq \mathcal{N}$, a secret key $\mathsf{sk}_i = \sum_{j \in \text{QUAL}} f_j(i) \in \mathbb{Z}_q$ and a verification key $\mathsf{vk}_i = g^{\mathsf{sk}_i} \in \mathbb{G}$ for each $i \in \text{QUAL}$, an implicit distributed secret value $\mathsf{sk} = \sum_{i \in \text{QUAL}} a_{i,0}$, and a global public key $\mathsf{pk} = \prod_{i \in \text{QUAL}} g^{a_{i,0}} (= g^{\mathsf{sk}})$. This is the well-known DKG protocol given in Fig. 2 in [32].

$\mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ outputs $s_x^i = (i, v_i, \pi_i)$ for a plaintext x, where $v_i \leftarrow H_1(x)^{\mathsf{sk}_i}$ and $\pi_i \leftarrow \mathsf{ProveEq}_{H_2}(g, H_1(x), \mathsf{vk}_i, v_i; r)$ for randomness $r \xleftarrow{\$} \mathbb{Z}_q$ (see Definition 2.1 for the description of $(\mathsf{ProveEq}_H, \mathsf{VerifyEq}_H)$).

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

1) Identifies an index subset $I = \{i_1, \dots, i_{t+1}\}$ such that $\mathsf{VerifyEq}_{H_2}(g, H_1(x), \mathsf{vk}_i, v_i, \pi_i) = 1$ holds for every $i \in I$, where $s_x^i = (i, v_i, \pi_i)$. 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 $I \subseteq \text{QUAL}$

1) Parses $s_x^i = (i, v_i, \pi_i)$ for $i \in I$.
2) Checks if $\mathsf{VerifyEq}_{H_2}(g, H_1(x), \mathsf{vk}_i, v_i, \pi_i) = 1$ 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.

Theorem 4.1. $\mathcal{V}^{\text{DDH-DVRF}}$ *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:

$$\begin{aligned}\sum_{j \in \Delta} \mathsf{sk}_j \lambda_{0,j,\Delta} &= \sum_{j \in \Delta} \lambda_{0,j,\Delta} \left(\sum_{i \in \text{QUAL}} f_i(j) \right) \\ &= \sum_{i \in \text{QUAL}} \left(\sum_{j \in \mathcal{I}} \lambda_{0,j,\Delta} \cdot f_i(j) \right) \\ &= \sum_{i \in \text{QUAL}} a_{i,0} = \mathsf{sk} \qquad (1)\end{aligned}$$

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 $\Delta \subseteq \text{QUAL}$ 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. The uniqueness can be proven using the extractibility property of NIZKs. That is, for any (v, π), if π verifies, we can extract a k such that $v = H_1(x)^k$ and $\mathsf{vk} = g^k$. □

The proof of the following theorem can be found in the full version of this paper [31].

Theorem 4.2. $\mathcal{V}^{\text{DDH-DVRF}}$ *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) $(g^{\alpha_0}, g^{\alpha_1}, \dots, g^{\alpha_w}, g^\beta, y_0, y_1, \dots, y_w)$ where either $y_0 = g^{\alpha_0\beta}, y_1 = g^{\alpha_1\beta}, \dots, y_w = g^{\alpha_w\beta}$ or $y_0 \xleftarrow{\$} \mathbb{G}, y_1 \xleftarrow{\$} \mathbb{G}, \dots, y_w \xleftarrow{\$} \mathbb{G}$. Suppose the adversary corrupts nodes $C = \{1, \dots, m\}$ 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_is are chosen uniformly at random,, the adversary receives uniformly random values. 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* with trusted setup *[1, Figure 5] and [1, Figure 6]. The correctness of the latter is only privately verifiable, and a formal specification of public verifiability is not part of their focus.*

4.2. 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. ordinary 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 GLOW-DVRF, a a pairing-based DVRF 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 (cf. Definition 2.3), where g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$ respectively (same applies to h_1, h_2 respectively). 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 DKG used in the non-compact case (Fig. 2 in [32]), with the differences described 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 $i \in \mathrm{QUAL}$ exposes $B_{i,0} = g_2^{a_{i,0}}$ via Feldman-VSS.
  - In Step 4(a), each node N_i with $i \in \mathrm{QUAL}$ broadcasts $A_{i,k} = g_1^{a_{i,k}}$ for $0 \le k \le t$ and $B_{i,0} = g_2^{a_{i,0}}$.
  - In Step 4(b), for each $i \in \mathrm{QUAL}$, 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 $\mathsf{pk} = \prod_{i \in \mathrm{QUAL}} B_{i,0}$.

To summarise, the verification keys $\mathsf{vk}_i (= g_1^{\mathsf{sk}_i})$ are generated in $\mathbb{G}_1$ using the generator g_1 and the global public key $\mathsf{pk}(= g_2^{\mathsf{sk}})$ is generated in $\mathbb{G}_2$ using the generator g_2.

$\mathsf{PartialEval}(x, \mathsf{sk}_i, \mathsf{vk}_i)$ computes $v_i = H_1(x)^{\mathsf{sk}_i}$ and $\pi_i \leftarrow \mathsf{ProveEq}_{H_3}(g_1, H_1(x), \mathsf{vk}_i, v_i; r)$ 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 $\mathcal{E} = \{s_x^{j_1}, \dots, s_x^{j_{|\mathcal{E}|}}\}$ of $|\mathcal{E}| \ge t+1$ partial evaluation candidates originating from $|\mathcal{E}|$ different nodes, and obtain verification keys $\mathsf{vk}_{j_1}, \dots, \mathsf{vk}_{j_{|\mathcal{E}|}}$. Next,

1) Identifies an index subset $I = \{i_1, \dots, i_{t+1}\}$ such that for every $i \in I$ it holds that $\mathsf{VerifyEq}_{H_3}(g_1, H_1(x), \mathsf{vk}_i, v_i, \pi_i) = 1$, where $s_x^i = (i, v_i, \pi_i)$. 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, e.g.. Definition 2.5, 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.

Theorem 4.3. $\mathcal{V}^{\text{GLOW-DVRF}}$ *satisfies consistency, robustness and uniqueness.*

Proof. Consistency can be easily proven similar to Theorem 4.1. The uniqueness follows from the fact that if $v = H_2(\pi)$, $v' = H_2(\pi')$ and $e(\pi, g_2) = e(H_1(x), \mathsf{pk}) = e(\pi', g_2)$ 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 $v^\star \ne \perp$ 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 $v_i \ne H_1(x^\star)^{\mathsf{sk}_i}$, $\mathsf{vk}_i = g_1^{\mathsf{sk}_i}$ 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. □

The proofs of the following theorems can be found in the full version of this paper [31].

Theorem 4.4. $\mathcal{V}^{\text{GLOW-DVRF}}$ *achieves* (θ, t, ℓ)*-standard pseudorandomness for any* θ *with* $\theta \le t < \ell - \theta$ *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 $C = \{1, \dots, m\}$ 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 4.5. $\mathcal{V}^{\text{GLOW-DVRF}}$ *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 $C = \{1, \dots, m\}$ 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 4.4. When $m < t$, the partial evaluation queries on $x^\star$ can be simulated using values in XDH instance which is similar to Theorem 4.2.

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

5. Decentralised Random Beacons

A Decentralised Random Beacon (DRB) provides a way to collaboratively agree on a (pseudo)random value [20] 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 [36], Ethereum 2.0 [15], OmniLedger [42], Tendermint [14]), without the need for a coordinator.

5.1. Formalisation of DRB

We start by presenting a syntax for DRBs with non-interactive randomness generation and formally define its relevant security properties, including robustness, uniqueness, and strong/standard pseudorandomness. As we will argue, we believe our definitions have merits not present in recent definitions in the literature [5], [17]. In particular we are able to analyse with our framework several academia and industry constructions (including ours): Algorand [34], HERB [19], Ouroboros Praos [23], Elrond [28], Orbs [4], Dfinity [36] and Harmony [37]. 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 number of instantiations captured by our model to nine.

Definition 5.1 (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 $\mathsf{sk}_i$

$\mathcal{O}^{\mathsf{Update}(\sigma, \pi)}$
    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 $(i, \sigma_i, \pi_i)$
```

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 $\mathcal{N}$. Let $\mathcal{K}$ be the set of secret keys of the honest nodes run by the oracle in the committee.
- $\mathcal{O}^{\mathsf{KeyRev}(i)}$: 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 $\mathsf{UpdState}(\mathsf{s.t.}, \sigma, \pi, \mathsf{cpk})$ to obtain the state for the next round and increases the round number rn by 1.

Similarly to DVRFs, we define *robustness* and *uniqueness* for DRBs. 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. 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. 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 [40].

Pseudorandomness for DRB. Pseudorandomness guarantees that the random beacon outputs are indistinguishable from uniformly random values in the presence of active adversaries, which implies the properties of unpredictability and bias-resistance [51], [52]. Below we shall give the formal definitions of the standard and strong pseudorandomness for DRB.

Definition 5.2 (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* $\delta = \sigma^\star$*; else 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 $\mathcal{O}^{\mathsf{PartialRand}}$.

5.2. 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 following theorems can be easily proven:

Theorem 5.1. $\mathcal{V}^{\mathrm{DRB}}$ *is robust (resp. unique) if* $\mathcal{V}^{\mathrm{DVRF}}$ *is robust (resp. unique).*

Theorem 5.2. $\mathcal{R}^{\mathrm{DRB}}$ *achieves* (θ, t, ℓ)*-standard (resp. strong) pseudorandomness if* $\mathcal{V}^{\mathrm{DVRF}}$ *achieves* (θ, t, ℓ)*-standard (resp. strong) pseudorandomness.*

Due to space limitation, the detailed proofs can be found in the full version of this paper.

Remark 3. *With our DVRF constructions from Section 4 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 [32]. It must be noted that other DRB protocols from the literature [28], [34], [37] need to use an out-of-band channel to agree on the initial seed, which is not needed with our DVRF-to-DRB constructions.*

6. Comparison with state-of-art DRB Instantiations

Our framework focus on DRBs that can generate a random beacon value in each round in a non-interactive manner. We stress that non-interactive protocols are the most desirable solutions for decentralised applications.

In this section, we shall first analyse a few non-interactive random seed generation protocols integrated in Algorand [34], Ouroboros-Praos [23], Elrond [28], HERB [19], Dfinity [36] and Harmony [37]. We formalise these protocols as DRBs in our framework and we analyse whether they can achieve pseudorandomness and unbiasiability. We stress that while some of these DRBs will be shown not to be pseudorandom, they may still serve a useful purpose in the context they were proposed. Namely, the full power of pseudorandomness may not be needed for particular use cases where, for instance, unpredictability may be enough. This is the case for Ouroboros-Praos DRB, where it is sufficient to use a “leaky resettable beacon" [23]. However our work shows that the DRBs that are not pseudorandom should be avoided as stand-alone DRBs.

Then we shall describe and compare with other interactive DRBs which may run across multiple rounds to create a beacon value.

6.1. Capturing non-interactive DRBs with our formalisation

Algorand, Ouroboros-Praos and Elrond. Algorand [34] implements a random seed generation protocol to create a publicly-known random 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 [23], [28] also implement similar protocols to Algorand's. Next, we show how to capture them as DRBs in our framework. These DRBs are based on stand-alone verifiable random functions $\mathrm{VRF} = (\mathrm{VRF}.\mathsf{KeyGen}, \mathrm{VRF}.\mathsf{Eval}, \mathrm{VRF}.\mathsf{Verify})$ [44], 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 $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathrm{VRF}.\mathsf{KeyGen}(1^\lambda)$ 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 $(\rho_{n,i}, \pi_{n,i}) \leftarrow \mathrm{VRF}.\mathsf{Eval}(\mathsf{sk}_i, \mathsf{st}_{n-1})$. 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 $\mathrm{VRF}.\mathsf{Verify}(\mathsf{pk}_j, \mathsf{st}_{n-1}, \rho_{n,j}, \pi_{n,j}) = 1$, 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 $\mathrm{VRF}.\mathsf{Verify}(\mathsf{pk}_j, \mathsf{st}_{n-1}, \sigma_n, \pi_{n,j}) = 1$ 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 $\mathrm{VRF}.\mathsf{Eval}(\mathsf{sk}_j, \mathsf{st}_{n-1})$ 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 [34, Theorem 1] shows their random seeds are unpredictable in the sense that the probability for the attacker to compute all the random seeds of the next k consecutive blocks decreases exponentially with k, under the assumption that a fraction h (with $h \geq 2/3$) of users are honest. This relaxed security feature tolerates more adaptive adversaries than our DVRF-based DRB constructions and additionally results in a simpler non-interactive committee setup phase.

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 [23, Section 5] and Elrond [28, Section 11] DRBs.

Harmony. Harmony [37, Section 3.1] is based on verifiable random functions $\mathrm{VRF} = (\mathrm{VRF}.\mathsf{KeyGen}, \mathrm{VRF}.\mathsf{Eval}, \mathrm{VRF}.\mathsf{Verify})$ [44], and verifiable delay function $\mathrm{VDF} = (\mathrm{VDF}.\mathrm{Eval}, \mathrm{VDF}.\mathrm{Verify})$ [10] 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 $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathrm{VRF}.\mathsf{KeyGen}(1^\lambda)$ 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 $(\rho_{n,i}, \pi_{n,i}) \leftarrow \mathrm{VRF}.\mathsf{Eval}(\mathsf{sk}_i, \mathsf{st}_{n-1})$. 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 $\mathsf{VRF.Verify}(\mathsf{pk}_j, \mathsf{st}_{n-1}, \rho_{n,j}, \pi_{n,j}) = 1$ 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 $(\sigma_n, \pi_n^d) \leftarrow \mathsf{VDF.Eval}(\mathsf{pp}, \rho_n)$.
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 $\mathsf{VRF.Verify}(\mathsf{pk}_j, \mathsf{st}_{n-1}, \rho_{n,j}, \pi_{n,j}) = 1$ for all $j \in J$ and $\mathsf{VDF.Verify}(\mathsf{pp}, \oplus_{j \in J} \rho_{n,j}, \sigma_n, \pi_n^d) = 1$, 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 [27]. Once ρ_n is committed into the blockchain, it cannot be modified by any node. To simplify the description, we let $k = 1$.

No formal security analysis is provided in [37] but informal claims that their construction provide "random" values are made. 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* [10], [54]. As suggested in [10], 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 [10]. 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.

Dfinity. Dfinity's DRB [36] uses Dfinity-DVRF which is an adaptation of the threshold BLS signatures [9]. Adversarial capabilities in standard pseudorandomness of Dfinity-DVRF and unforgeability of threshold BLS signatures go hand in hand, as in both cases an adversary is not given the option to make partial queries on the challenge plaintext $x^\star$. Therefore the standard pseudorandomness of Dfinity-DVRF can be derived in a manner similar to the security reduction of the unforgeability of the threshold BLS signature. Existing BLS security reductions do not seem to extend 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.

HERB. The so-called Homomorphic Encryption Random Beacon construction [19], 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.

The results of our DRBs comparison is summarized in Table 1.

6.2. Comparison with interactive DRBs

SCRAPE [17], RandShare/RandHound [52] and Ouroboros [41] 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 [6]. 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 $\mathsf{PartialRand}$ and $\mathsf{CombRand}$.

HydRand [51] 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 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.

7. 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 [29] are shown in Table 2, where the protocols

Random Beacon Protocol	**Standard Pseudorandomness**	**Strong Pseudorandomness**	**Strong Bias Resistance**	**Unpredictability**
Algorand-DRB [34]	(–)	✘	✘	✔
Elrond-DRB [28]	(–)	✘	✘	✔
Harmony-DRB [37]	✘	✘	✘	✔
HERB [19]	✔	✔	✘	✔
Orbs-DRB [4]	✔	✔	✘	✔
Ouroboros-Praos-DRB [23]	(–)	✘	✘	✔
Dfinity-DRB [36]	✔	(?)	✔	✔
DDH-DRB [This work]	✔	✔	✔	✔
GLOW-DRB [This work]	✔	✔	✔	✔

Table 1: Comparison of Random Beacons with non-interactive beacon computation. **(–)** means the corresponding property holds when all nodes are honest. **(?)** means it is unknown whether the corresponding property can be proven.

are implemented using mcl library [46], RELIC [2], Libsodium [8].

The results are summarised in Table 2, 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 [49] 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. As mentioned in Section 4.2, the reason why our DDH-DVRF and GLOW-DVRF constructions outperform Dfinity-DVRF is because the partial evaluations in our constructions are validated using NIZKs, instead of pairing equations as in Dfinity-DVRF. Our most expensive NIZK proof involves computing 4 exp in $\mathbb{G}_1$, while validating a pairing equation in Dfinity-DVRF involves computing 2 pairings. Pairings are time-consuming: one pairing is about 5x slower than exp in $\mathbb{G}_1$ as shown in mcl benchmark [45]. Therefore computing two pairings may be about 2.5x slower than 4 exp in $\mathbb{G}_1$.

The benchmarks discussed in this section can be reproduced using the reference implementation [29] 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 2. 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 [53], respectively.

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 [14] using Cosmos SDK [38]. 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 a 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. Executions were performed both, 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 [17], [41], [51], [52]. For instance, while the recently presented Hydrand DRB [51] 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 x9 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 similar number of

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	
	BLS12-381	mcl	128	48	18.67	1.75	co-CDH
	BN384	mcl	128	48	21.39	2.00	XDH
	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	

Protocol	Curve	Library	Security Level	Proof size (bytes)	Randomness Generation (ms)	Time Ratio	Assumption
GLOW-DVRF	BN256	mcl	100	32	29.06	0.68	
	BLS12-381	mcl	128	48	71.91	1.68	co-CDH
	BN384	mcl	128	48	80.15	1.87	XDH
	BN_P256	RELIC	100	33	132.92	3.10	
DDH-DVRF	Ristretto255	Libsodium	128	6464	42.91	1	DDH
	Curve25519	RELIC	128	6464	285.59	6.66	
Dfinity-DVRF	BN256	mcl	100	32	74.01	1.72	co-CDH
	BLS12-381	mcl	128	48	215.34	5.02	
	BN384	mcl	48	128	228.54	5.33	
	BN_P256	RELIC	100	33	566.21	13.20	

Table 2: 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 $t = \ell/2$. These figures do not take into account latency due to communicating partial evaluations between nodes.

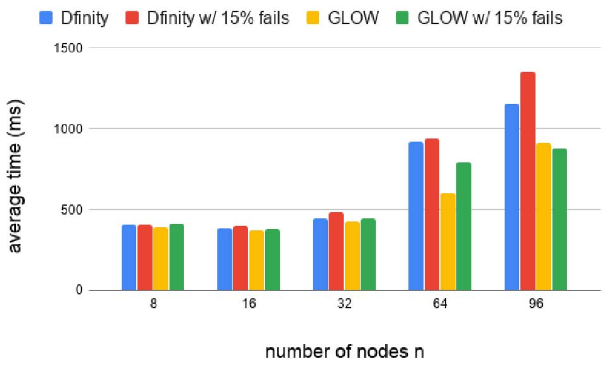

Table 3: Average randomness generation time for different numbers of nodes n with no failures and 15% simulated failures on a Cosmos/Tendermint-compatible ledger for Dfinity-DVRF and GLOW-DVRF-based DRBs

nodes.

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] 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.

[7] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[23] 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.

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

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

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

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

[28] 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.

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

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

[31] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons, 2020. https://eprint.iacr.org/2020/096.

[32] 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.

[33] 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.

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

[35] 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.

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

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

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

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

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

[41] 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.

[42] 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.

[43] Veronika Kuchta and Mark Manulis. Unique aggregate signatures with applications to distributed verifiable random functions. In *CANS 2013*, volume 8257, pages 251–270, 2013.

[44] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS '99*, pages 120–130. IEEE Computer Society, 1999.

[45] Shigeo Mistunari. mcl benchmark. https://github.com/herumi/mcl/blob/master/bench.txt.

[46] Shigeo Mistunari. mcl - A portable and fast pairing-based cryptography library. https://github.com/herumi/mcl, 2019.

[47] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. http://bitcoin.org/bitcoin.pdf.

[48] 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.

[49] Phil Nash. Catch2. https://github.com/catchorg/Catch2, 2019.

[50] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Ethdkg: Distributed key generation with ethereum smart contracts. 2019. https://eprint.iacr.org/2019/985.

[51] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. Hydrand: Practical continuous distributed randomness. *IEEE S&P*, 2020.

[52] 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.

[53] Rhys Weatherley. Noise-C, a plain C implementation of the Noise protocol. https://github.com/rweather/noise-c, 2016.

[54] Benjamin Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT 2019*, pages 379–407, 2019.

[55] Gavin Wood. Ethereum: Secure decentralised generalised and transaction ledger, 2019. https://ethereum.github.io/yellowpaper/paper.pdf.

Appendix A. Additional DVRF Definitions

Definition A.1 (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 $j \in U'$.*

Definition A.2 (Robustness). *A t-out-of-ℓ* DVRF *protocol $\mathcal{V} = (\mathsf{DistKG}, \mathsf{PartialEval}, \mathsf{Combine}, \mathsf{Verify})$ on nodes $\mathcal{N} = \{N_1, \dots, N_\ell\}$ 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* $N_j \in \mathrm{QUAL}\backslash C$ *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.

Definition A.3 (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.2 (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.*