SoK: Distributed Randomness Beacons

***Abstract*—Motivated and inspired by the emergence of Proof of Stake blockchains, many new protocols have recently been proposed for generating publicly verifiable randomness in a distributed yet secure fashion. These protocols work under different setups and assumptions, use various cryptographic tools, and entail unique tradeoffs and characteristics. In this paper, we systematize the design of distributed randomness beacons (DRBs). We evaluate protocols on four key security properties—unbiasability, unpredictability, liveness, and public verifiability—and discuss common attack vectors for predicting or biasing the beacon output and the countermeasures employed by protocols. We also evaluate protocols by communication and computational efficiency. Finally, we provide insights on the applicability of different protocols in various deployment scenarios and highlight possible directions for further research.**

I. Introduction

Public, trustworthy randomness has been a useful tool for millennia, dating at least to the earliest known use of dice around 3000 BCE. Today, periodic public randomness is crucial to applications including gambling and lotteries [17], sampling ballots for election audits [2], selecting parameters for cryptographic protocols [6], [47], leader election in Proof of Stake protocols [40], [45], and blockchain sharding [3], [46]. The concept of a *randomness beacon* was first formalized by Rabin [55] to describe an ideal service that emits fresh random numbers at regular intervals in a way that no party can manipulate. Because no such ideal beacon exists, various protocols are used to approximate this beacon functionality for practical use.

Centralized Beacons. Relying on a trusted third party like NIST [35] or random.org [42] might be the simplest way to realize a beacon. It carries drawbacks typically associated with centralized services, such as the risk of compromise or misbehavior and the inability of the end user to verify the correctness of the beacon.

Public Implicit Beacons. Another approach is to construct a beacon using publicly available implicit sources of entropy such as stock market data [25] or Proof of Work (PoW) blockchains like Bitcoin [7], [17], [43], [49]. However, these entropy sources are potentially vulnerable to malicious insiders (e.g. high-frequency traders making unnatural trades to fix stock prices, financial exchanges blocking trades or reporting incorrect data, miners that can withhold blocks or choose between colliding blocks, etc.). The entropy of these sources can also be difficult to precisely measure. While potentially secure and low-cost in practice, we will not discuss these approaches in detail in this survey.

Distributed Randomness Beacons. A natural approach to reduce complete trust in a centralized beacon is to use a distributed beacon protocol capable of gracefully handling a minority of Byzantine participants and a potentially untrusted network. We call a beacon realized in this manner a *distributed randomness beacon* (DRB). DRB protocols are typically round-based, producing a fresh random output in each round.

Contribution. The goal of the paper is to systematize the current progress on DRBs specifically. We propose a generalized framework encompassing all distributed randomness protocols in the landscape. To aid comparison and discussion of properties, we provide an overview of these protocols along with the various cryptographic building blocks used to construct them. We identify two key components of DRB design: selection of entropy providers and beacon output generation, which can be decoupled from each other. Enabling a more holistic analysis of a DRB as a result, we also provide new insights and discussion on potential attack vectors, countermeasures, and techniques that lead to better scalability.

Paper organization. We begin with preliminaries including our system model, a strawman DRB under perfect synchrony (an ideal assumption), commit-reveal [11], and the definition of an ideal DRB in Section II. Section III introduces protocols using verifiable delay functions [13], which (assuming secure VDFs can be realized) offer the best security and simplicity. Then from Section IV to VII, we introduce non-VDF-based DRB protocols based on the number of nodes contributing *marginal entropy* (i.e. per-round randomness that is independently generated at a node level) in each round. Sections IV and V review protocols in which all nodes contribute marginal entropy. These protocols vary in mechanisms used to recover from faulty nodes, including financial punishment [54], [66], threshold secret sharing [19], [58], and threshold encryption [30]. Section VI covers committee-based protocols in which each round includes an extra committee selection step, after which only a committee (non-empty proper subset) of nodes contributes marginal entropy. These protocols are more complex but can offer greater communication efficiency with large numbers of nodes. Section VII covers pseudorandom protocols that do not require any marginal entropy; these protocols can be highly efficient but have no mechanism to recover from compromise. We conclude with discussion and comparisons in Sections VIII–IX, including a comparison of all studied protocols in Table I.

II. Preliminaries

A. System Model

We consider a system model with $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ comprising n participants (also called nodes), also often denoted by $\mathcal{P} = \{1, 2, ..., n\}$ for the purpose of algebraic formulations without loss of generality. Out of n, up to t

faulty nodes (deemed *Byzantine*) may engage in incorrect (arbitrary) behavior during a protocol run, and an adversary $\mathcal{A}$ that controls up to t such nodes is called *t-limited*. Otherwise, nodes that are *honest* abide by the specified protocol.

We assume a standard public key infrastructure (PKI) such that all nodes know each others' public keys, and that all nodes are connected via point-to-point secure (providing authenticity) communication channels. All messages exchanged by honest nodes are digitally signed by the sender, and recipients always validate each message before proceeding. By default, we assume a *synchronous* network, in which there exists some known finite message delay bound Δ. This means that an adversary can delay a message by at most Δ.

Moreover, we assume a computationally bounded adversary $\mathcal{A}$ runs in PPT (probabilistic polynomial time) and that $\mathcal{A}$ cannot break standard cryptographic constructions such as hash functions, digital signatures, etc. The three ways in which $\mathcal{A}$ can deviate from a protocol are omitting a message (i.e. *withholding attack*), sending invalid messages, and colluding to coordinate an attack based on private information shared among Byzantine nodes. Additionally, $\mathcal{A}$ has the power to perform a *grinding attack*, in which $\mathcal{A}$ privately precomputes and iterates through as many combinations of inputs to an algorithm as possible in order to derive a desirable output. By default, we assume a *static* adversary that chooses nodes to be corrupted before a protocol run whereas an *adaptive* adversary can choose nodes to be corrupted at any time during a protocol run (although we assume a model where nodes remain corrupted once corrupted).

We denote our computational model's security parameter by λ. We call a function $negl(\lambda)$ *negligible* if for all $c > 0$ there exists a λ_0 such that $negl(\lambda) < \frac{1}{\lambda^c}$ for all $\lambda > \lambda_0$. The group elements $g, h \in \mathbb{G}$ are generators of $\mathbb{G}$ while p, q denote primes where $q \mid p - 1$ (unless stated explicitly) such that $\mathbb{G}_q$ is a group of prime order q. The notation $tuple[0]$ denotes the first element of $tuple$. Furthermore, we model any hash function $H(\cdot)$ as a random oracle. In the context of a distributed randomness beacon, we use r to denote round number and $\mathcal{O}_r$ to denote the *beacon output* (i.e. the distributed randomness output) in round r. The *entropy-providing committee* denoted by $\mathcal{C}_r$ refers to a non-empty proper subset of nodes (hereafter called *entropy providers*) that proactively generate and provide marginal entropy in round r. $\mathcal{C}_r$ can include all nodes, some but not all nodes, one node (i.e. a leader), or no node.

B. Strawman Protocol: Rock-Paper-Scissors

Distributed randomness assuming perfect synchrony is straightforward. Consider the following one-round "rock-paper-scissors" protocol where each participant i broadcasts its *entropy contribution* (i.e. independently generated randomness) $e_i \in \mathbb{Z}_p$ to every other participant at the same time as per simultaneity observed (say) in rock-paper-scissors. Naturally, its protocol output $\mathcal{O}$ (via addition in a finite group) can be given by

$$\mathcal{O} = \sum_{i=1}^{n} e_i$$

such that applying this mechanism $\tilde{r}$ times at $\tilde{r}$ chronological timestamps would yield a DRB.

This protocol is simple and, under a perfect synchrony assumption, is secure as long as any single participant chooses its e_i randomly. However, security falls apart completely once messages can be delayed. Consider a simple scenario with three participants $\{P_1, P_2, P_3\}$ participating to produce $\mathcal{O} = e_1 + e_2 + e_3$. If P_3 can read e_1 and e_2 before sending e_3 (due to non-zero message latency) to P_1 and P_2, then P_3 can fix the output $\mathcal{O}$ to any value $\tilde{\mathcal{O}}$ by choosing $e_3 = \tilde{\mathcal{O}} - e_1 - e_2$. Effectively, the protocol cannot tolerate any Byzantine participants without perfect synchrony.

C. Commit-Reveal

A classic fix for the above issue is to introduce a cryptographic commitment step before each party reveals its entropy contribution.

1) Commit. Each participant P_i broadcasts a cryptographic commitment $C_i = \mathsf{Com}(e_i, r_i)$ (with fresh randomness r_i) to its entropy contribution e_i rather than e_i itself. Note that $\mathsf{Com}(x, r_0)$ denotes a cryptographic commitment to x with hiding and binding properties [11], [26].
2) Reveal. Once all participants have shared their corresponding commitments, each participant P_i then opens its commitment by revealing the pair $(e_i, r_i) = \mathsf{Open}(e_i, r_i)$. In turn, P_i verifies each received pair (e_j, r_j) for $j \neq i$ by recomputing $C_j = \mathsf{Com}(e_j, r_j)$. Given that these checks pass, the final output $\mathcal{O}$ can canonically be given by

$$\mathcal{O} = \sum_{i=1}^{n} e_i$$

all of which can be repeated to produce a DRB. If any of the checks do not pass, however, the protocol aborts.

With the additional commit step, it becomes impossible for any participant to manipulate the output $\mathcal{O}$, as their values are bound by commitments published before any participants reveal. Nonetheless, the protocol can still be biased, as the last participant P_k to reveal can in fact compute $\mathcal{O}$ earlier than others and hence can decide not to reveal (e_k, r_k) if $\mathcal{O}$ is not to its liking. This is called the *last-revealer attack*. Note that this attack is indistinguishable from a faulty node going offline, and indeed the protocol also has no robustness against non-Byzantine faults in this basic form.

D. Ideal Distributed Randomness Beacon

Clearly, a DRB should prevent any one participant from tampering with (e.g. predicting, biasing, or aborting) each round's output. The beacon output should also be verifiable by any third party. Based on these requirements, the overall security properties of an ideal DRB are given by the following.

Definition II.1 (Ideal distributed randomness beacon)**.** A distributed randomness beacon is *ideal* or *secure* if it satisfies the following four properties.

1) Bias Resistance (or Unbiasability). No PPT adversary $\mathcal{A}$ can bias any c bits of $\mathcal{O}_r$. In other words, $\mathcal{A}$ cannot

force c bits of $\mathcal{O}_r$ to be some arbitrarily chosen bits with probability greater than $\frac{1}{2^c} + negl(\lambda)$.

2) Unpredictability. Similarly, $\mathcal{A}$ cannot predict any c bits of $\mathcal{O}_r$ with probability greater than $\frac{1}{2^c} + negl(\lambda)$. The protocol satisfies *d-unpredictability* [10] for $d \in \mathbb{N}$ if this is true for any round greater than or equal to $r+d$ (where r denotes the current round).
3) Liveness. A la game-based security, we can define liveness [41] by requiring that the advantage of $\mathcal{A}$ denoted by $Pr[\mathcal{O}_r = \perp]$ (i.e. the probability that the honest beacon output at the end of round r is null) is negligible, given a DRB that runs among honest participants and $\mathcal{A}$.
4) Public Verifiability. Any third party should be able to verify the beacon output based on public information. See Appendix A for a game-based formulation.

We next begin our discussion of various approaches to realizing an ideal DRB, starting with protocols based on verifiable delay functions.

III. VDF-Based Protocols

One way to prevent the last-revealer attack is to add a delay function after collecting each node's entropy contribution, delaying the derivation of $\mathcal{O}_r$. As long as the delay is suitably long, no participant can predict what effect its contribution will have on the output before its share must be published. A verifiable delay function (VDF) [13], [14] can be used to accomplish this.

Definition III.1 (Verifiable delay function)**.** A *verifiable delay function* (VDF) is a function that takes a specified number of sequential steps to compute (even with a large amount of parallelism available) but takes exponentially less time to verify once computed. It can be described by the following algorithms.

- $\mathsf{Setup}(\lambda, T) \rightarrow pp$ is a randomized algorithm that outputs public parameters pp given security parameter λ and time bound T.
- $\mathsf{Eval}(pp, x) \rightarrow (y, \pi)$ outputs y in T sequential steps and a proof π given pp and an input x.
- $\mathsf{Verify}(pp, x, y, \pi) \rightarrow \{0, 1\}$ outputs 1 if y is the correct evaluation of the VDF on input x and 0 otherwise.

The two well-known VDF proposals, one due to Pietrzak [53] and the other due to Wesolowski [64], make use of the (believed) inherently sequential nature of repeated squaring in a group of unknown order. VDFs can be used to derive unbiasable randomness either from existing, biasable protocols (e.g. commit-reveal or public implicit beacons) or as the building block for an entirely new protocol (like RandRunner [56]).

A. Modifying Commit-Reveal

The Unicorn protocol [47] uses the Sloth function (a VDF precursor based on computing square roots modulo a prime) in a manner similar to commit-reveal. In fact, commitments are no longer needed; participants simply publish their entropy contributions directly. Unicorn can be improved using a VDF in place of Sloth to achieve an exponential gap between computation and verification times. We refer to this VDF-based Unicorn as *Unicorn++*. It runs as follows.

1) **Collect.** Every participant P_i broadcasts its entropy contribution e_i between time t_1 and t_2 (assuming synchronized clocks). At t_2, they are combined into $seed_r = H(e_1, \ldots, e_n)$.
2) **Evaluate.** Some party evaluates the VDF with $seed_r$ and a chosen delay parameter T (part of pp) via

$$y_r, \pi_r = \mathsf{VDF.Eval}(pp, seed_r)$$

such that $\mathcal{O}_r = H(y_r)$, which is posted and can be efficiently verified by any observer using π_r via $\mathsf{VDF.Verify}$.

As long as T is longer than the duration of $t_2 - t_1$, Unicorn++ successfully defends against any attack possible by the last entropy provider. Also desirably, it is unbiasable by an adversary that controls $n - 1$ of the participants, as even one honest entropy contribution requires computation of $\mathsf{VDF.Eval}$ from scratch. The downside of the protocol is that *somebody* must evaluate the VDF, which is slow by design (it does not matter for security who evaluates, since VDFs are deterministic and verifiable). Thyagrajan et al. [61] recently proposed a protocol for outsourcing VDF evaluation.

We quickly note a variation of above: as proposed in [17], [18], beacons using stock prices [25] or PoW blockchains [7], [43], [49] (which are otherwise susceptible to manipulation) can be used to supply $seed_r$ in Collect. Such schemes are collectively denoted by *Ext. Beacon+VDF* in Table I. Unfortunately, they do not easily compare to other DRBs, as the security model depends on the cost of manipulating the external beacon, which has not yet been formally analyzed.

B. Chain of VDFs

The disadvantage of above is that each round may require consensus [22] on inputs to the VDF, incurring communication cost. Also, the rate at which beacon outputs are generated is limited by T. RandRunner [56] tackles these issues by leveraging a VDF design that builds a deterministic chain of outputs (more precisely, a chain of n interleaved VDFs each set up by a node) to bypass per-round consensus while allowing each round's duration to be independent of T in the optimistic case. As a result, the protocol achieves lower communication complexity as well as more beacon outputs generated per time frame. Trapdoor VDFs [64] with the strong uniqueness property [56] serve as its key building block.

Definition III.2 (Strongly unique trapdoor VDF)**.** A *strongly unique trapdoor VDF* extends VDF by allowing any participant who knows the *trapdoor* (e.g. p, q for RSA's $N = pq$) to efficiently evaluate the VDF without T sequential steps. Moreover, it provides *strong uniqueness*, i.e. uniqueness even when VDF's public parameters are adversarially generated. It can be described by the algorithms $\mathsf{TVDF} = (\mathsf{Setup}, \mathsf{VerifySetup}, \mathsf{TrapdoorEval}, \mathsf{Eval}, \mathsf{Verify})$, extending those of a traditional VDF with the following.

- $\mathsf{VerifySetup}(\lambda, pp) \rightarrow \{0,1\}$ verifies the validity of pp.
- $\mathsf{TrapdoorEval}(pp, x, sk) \rightarrow (y, \pi)$ outputs y in time less than T (unlike Eval) and a proof π given pp, an input x, and the trapdoor sk.

Wesolowski's VDF [64] is not strongly unique, as knowing the trapdoor allows an adversary to forge proofs accepted by VDF.Verify. As a result, RandRunner uses Pietrzak's VDF [53]. After an initial Setup phase, RandRunner reiterates its Execution phase as follows.

1) **Setup.** Each participant P_i executes TVDF.Setup to compute its public parameters pp_i and the corresponding secret trapdoor sk_i and broadcasts pp_i, which is verified by others via TVDF.VerifySetup. This ensures that the uniqueness of Pietrzak's VDF is guaranteed. At the end, all participants should have the same set of public parameters $\{pp_i\}_{i=1,\ldots,n}$. The initial value $\mathcal{O}_0$ used to bootstrap the protocol is also agreed upon.
2) **Execution.** A unique leader for round r, l_r is selected via either round-robin (i.e. taking turns in some permuted order) or random selection (i.e. using $\mathcal{O}_{r-1}$ as seed, originally called randomized sampling). The implications of each are discussed in Section VI-A1. Each round can proceed in two ways depending on l_r's status.
   - **Honest Leader (Common Case).** The leader advances the protocol into the next round by broadcasting
   $$y_r, \pi_r = \mathsf{TVDF.TrapdoorEval}(pp_{l_r}, H_1(\mathcal{O}_{r-1}), sk_{l_r})$$
   in which case other nodes check the correctness of the received values via TVDF.Verify. Then the beacon output is
   $$\mathcal{O}_r = H_2(y_r)$$
   where H_1 and H_2 map values from the beacon output space to the VDF space and vice versa.
   - **Dishonest Leader.** Given a dishonest leader that withholds or broadcasts an invalid message, every non-leader computes and broadcasts the round's VDF output via
   $$y_r, \pi_r = \mathsf{TVDF.Eval}(pp_{l_r}, H_1(\mathcal{O}_{r-1}))$$
   in T sequential steps. Then $\mathcal{O}_r = H_2(y_r)$ similarly.

Consequently, RandRunner generates each beacon output rapidly with only $O(n)$ communication complexity in the common case. Adversarial leaders can increase the round duration to T (or more with network delay Δ) and the communication complexity to $O(n^2)$. Due to its *pseudorandomness* (as opposed to *true randomness* [21], [28]) sprouting from the deterministic nature of VDFs, RandRunner exhibits two other beneficial properties. First, liveness is retained even with a dishonest majority and when network connectivity breaks down completely, as a node can simply compute the beacon outputs over time via TVDF.Eval. Second, it is impossible to bias the beacon once bootstrapped such that even the strongest adversary can only predict but not bias. These benefits have a counterpart, however. Namely, RandRunner can never achieve the ideal 1-unpredictability property due to the existence of leaders that can withhold and adversaries with higher compute power. In other words, the parameter d as in RandRunner's d-unpredictability must be greater than one but can be calculated and bounded [56] depending on assumptions.

IV. Commit-Reveal-Punish

Another approach to preventing last-revealer attacks are *commit-reveal-punish* schemes, which assume that all participants are rational entities and use financial penalties to discourage withholding shares. This requires some form of *escrow* (e.g. smart contracts on Ethereum [65]) to collect initial deposits from the participants as part of their commitment, which can be *slashed* (destroyed) or redistributed if misbehavior is detected. Commit-reveal-punish schemes defend against the last-revealer attack either by forcing every participant to reveal [4], [8], [54] or by tolerating some number of withholding participants via a threshold variant of commit-reveal [66]. These two approaches are summarized in the following.

A. Enforcing Every Reveal

Extending a basic commit-reveal, RANDAO [54] implements commit-reveal-punish in the most literal sense via punishing participants that withhold during the reveal phase by confiscating (and redistributing) each deposit of m coins required during the initial commit phase. If no one withholds, all participants receive their deposits back, and all entropy contributions are aggregated and broadcast as $\mathcal{O}_r$ as usual.

The drawback of protocols like RANDAO is two-fold. First, honest failures are also punished without much flexibility. Second, a high deposit of $m = O(n^2)$ is required to provide fairness in certain scenarios [4], [8]. These imply that the protocol is suitable in cases where each participant is expected to be highly available and possess an ample supply of coins, but not otherwise. Deploying these protocols in practice also requires an understanding of the value to participants of manipulating the beacon (to ensure the opportunity cost of lost deposits is higher). This assumption is reasonable for applications such as a lottery but may not apply for a public beacon whose use may not be known in advance.

B. Rational Threshold Commit-Reveal

Economically Viable Randomness (EVR) [66] provides an alternative requiring constant deposits while tolerating (honest) faults to an extent. This is achieved by devising a threshold variant of commit-reveal (i.e. in which $t+1$, as opposed to all n, nodes reveal to compute $\mathcal{O}_r$) and having an incentive mechanism around it. The threshold nature also invites collusion, which is counteracted by EVR's *informing* mechanism: if the escrow is notified of collusion (via informing), it rewards the informer and slashes the deposits of all others (*collective punishment*). Realizing this, nodes are discouraged to collude, fearing another node within the collusion would inform.

EVR requires multiple cryptographic building blocks (many are used by other DRBs as well), which we introduce here.

EVR uses Escrow-DKG [67], an extension of DKG (distributed key generation) [39], [52], to realize a threshold commit-reveal. DKG allows a set of n nodes to collectively generate a pair (sk, pk) of group secret and public keys such that sk is shared and "implied" (i.e. never computed explicitly) by n nodes via the following building blocks.

Definition IV.1 ((t, n)-secret sharing)**.** A (t, n)-*secret sharing* scheme (also known as Shamir's secret sharing [59]) allows a dealer to share a secret $s = p(0)$ for some *secret sharing polynomial* $p \in \mathbb{Z}_q[X]$ of degree t among n participants each holding a *share* $s_i = p(i)$ for $i = 1, ..., n$. Any subset of $t+1$ or more participants can reconstruct the secret s via *Lagrange interpolation* (see Appendix I1), but smaller subsets cannot.

Definition IV.2 (Verifiable secret sharing)**.** *Verifiable secret sharing* (VSS) [33], [51] protects a (t, n)-secret sharing scheme against a malicious dealer sending incorrect shares by providing an additional verification step per share. VSS can be described by the following algorithms.

- $\mathsf{Setup}(\lambda) \rightarrow pp$ generates the public parameters pp, an implicit input to all other algorithms.
- $\mathsf{ShareGen}(s) \rightarrow (\{s_i\}, C)$ is executed by the dealer with secret s to generate secret shares $\{s_i\}$ (each of which is sent to node i correspondingly) as well as commitment C to the secret sharing polynomial of degree t.
- $\mathsf{ShareVerify}(s_i, C) \rightarrow \{0, 1\}$ verifies the correctness of the share s_i using C.
- $\mathsf{Recon}(A, \{s_i\}_{i \in A}) \rightarrow s$ reconstructs s via Lagrange interpolation from a set A of $t + 1$ nodes that pass $\mathsf{ShareVerify}$.

See Appendix B for details.

Definition IV.3 (Distributed key generation)**.** A *distributed key generation* (DKG) [39], [52] allows n participants to collectively generate a *group public key* (for an implied *group secret key*), *individual secret keys*, and *individual public keys* without a trusted third party. It does so by running n instances of VSS (with each participant acting as a dealer for its independent secret). Unlike secret sharing schemes, DKG can be used repeatedly for an unlimited number of times, as the group secret key does not need to be computed explicitly.

- $\mathsf{DKG}(1^\lambda, t, n) \rightarrow (sk_i, pk_i, pk)$ outputs the i-th node's secret key, its public key (e.g. $pk_i = g^{sk_i}$), and a group public key pk (e.g. $pk = g^{sk}$) for an implied group secret key sk given security parameter 1^λ, n, and threshold parameter t.

See Appendix C for details.

Definition IV.4 (Escrow-DKG)**.** Escrow-DKG [67] is a rational variant of DKG with the following variations.

- Unlike traditional DKG, Escrow-DKG does not a priori assume a t-limited adversary. However, it assumes the participants are rational in a setting where collusion of more than t participants would be financially punished such that, effectively, we have a t-limited adversary.
- It assumes an escrow denoted by $\mathcal{G}$.
- Escrow-DKG has a deposit requirement, considers how a DKG may fail, and associates a financial penalty to each failure case to disincentivize misbehavior.

The crux is that EVR adapts Escrow-DKG to realize a DRB by using the group secret (i.e. the implied group secret key sk after a DKG) as $\mathcal{O}_r$, retrievable from $t+1$ sk_i's. EVR proceeds in four phases—Setup, Commit, Inform, and Reveal.

1) **Setup.** Every participant registers by depositing 1 coin per secret (i.e. entropy contribution), and $\mathcal{G}$ accordingly sets the threshold parameter $t = 2n/3$ required for Escrow-DKG. It also sets the *illicit profit bound* (i.e. bound on extra profit an adversary can gain as a result of using EVR's beacon output as opposed to an ideal beacon's output) $P = n - t = n/3$ and the *informing reward* $\ell = n$.
2) **Commit.** Escrow-DKG is run, and each participant ends up with an individual key pair (sk_i, pk_i) as well as pk.
3) **Inform.** Any colluding participant that preemptively knows $\mathcal{O}_r$ can inform $\mathcal{G}$ to earn a high informing reward (ℓ) obtained via collective punishment. Due to this phase, the incentive is that no one should collude.
4) **Reveal.** $\mathcal{O}_r = sk$ is reconstructed once $t + 1$ (or more) honest participants reveal their sk_i's. Initial deposits are returned after verification by $\mathcal{G}$. If $\mathcal{O}_r$ is not reconstructed by the end, $\mathcal{G}$ also initiates collective punishment.

While a node's malicious behaviors in EVR are limited to withholding to abort the protocol during Reveal or colluding to learn $\mathcal{O}_r$ before Reveal, its security comes from the fact that both are disincentivized. First, setting $P = n - t$ makes withholding unprofitable, as $n - t$ or more participants that withhold to successfully abort EVR would earn at most P at the cost of losing their deposits. This prevents biasability. Second, setting $\ell = n$ makes informing more profitable than any illicit profit such that any coalition of nodes colluding to preemptively learn $\mathcal{O}_r$ is disincentivized due to the inevitability of an informer. This prevents predictability.

Despite the benefits of the threshold nature and constant deposits enabling a flexible incentive mechanism, EVR requires further economic assumptions beyond those needed for commit-reveal-punish. Specifically, EVR assumes a limit P on illicit profit and a bound on the total number of coins $n/3$ (a participant with more coins than this is not allowed to join EVR as per decentralization assumption [66]).

V. Commit-Reveal-Recover

Without an escrow to enforce desired behaviors, commit-reveal-recover variants extend commit-reveal and defend against the last-revealer attack by providing a mechanism to *recover* or *reconstruct* a participant's entropy contribution if withheld. This can be achieved by either threshold secret sharing or threshold encryption. Protocols based on commit-reveal-recover assume a t-limited adversary and require the cooperation of at least $t+1$ nodes to reconstruct such that two desirable properties are achieved simultaneously: there is no need for all n nodes to reveal while any subset of t Byzantine

nodes cannot collude to preemptively reconstruct. Note that there is an inherent tradeoff here: while a smaller value of t provides greater tolerance to withholding, it also means that a smaller subset can collude to predict the beacon output in advance.

A. From Threshold Secret Sharing

Building on (t, n)-secret sharing, commit-reveal-recover variants based on threshold secret sharing often use PVSS (publicly verifiable secret sharing) [19], [58] as a subprotocol in order to allow any external party (not just the participants) to verify the correctness of sharing and reconstruction.

Definition V.1 (Publicly verifiable secret sharing)**.** *Publicly verifiable secret sharing* (PVSS) extends VSS by enabling public verification. It can be realized by the algorithms $\mathsf{PVSS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ShareGen}, \mathsf{ShareVerify}, \mathsf{Recon})$, extending those of VSS. To provide public verifiability, PVSS requires $\mathsf{PVSS.ShareGen}$ to use keys (secret-public key pair per participant) generated by $\mathsf{PVSS.KeyGen}$ to encrypt and decrypt shares via $\mathsf{PVSS.Enc}$ and $\mathsf{PVSS.Dec}$ as subroutines and to generate proofs, e.g. NIZK (non-interactive zero-knowledge) proofs, for public verification as another subroutine. Then $\mathsf{PVSS.ShareVerify}$ can be run by anyone (not just the participants). See Appendix D for details.

The idea in these commit-reveal-recover variants is that each participant generates a secret (i.e. entropy contribution), distributes PVSS shares to each other participant, and receives n respective shares of n other participants' secrets. These shares are then used to compute $\mathcal{O}_r$ via Lagrange interpolation ($\mathsf{PVSS.Recon}$) in case some nodes withhold. Based on when and how such Lagrange interpolation takes place, we subdivide the protocols into the following categories: commit-reveal-recover, share-reconstruct-aggregate, and share-aggregate-reconstruct.

1) Commit-Reveal-Recover: Extending commit-reveal, *commit-reveal-recover* adds another step to the commit phase where every participant is additionally required to distribute PVSS shares of its corresponding secret so that others can reconstruct it via Lagrange interpolation (*recover*) if withheld during reveal. The tradeoff is additional communication cost, which could be amplified multiplicatively in the recover phase if $O(n)$ Lagrange interpolations need to take place. Scrape [19] adopts this technique.

Scrape. Scrape introduces its own PVSS scheme [19] specifically designed for efficiency in commit-reveal-recover. The initial setup for Scrape requires generating (sk_i, pk_i) for each of the n participants using $\mathsf{PVSS.KeyGen}$. The protocol then proceeds as follows.

1) **Commit.** Every participant P_j executes $\mathsf{PVSS.ShareGen}(s^{(j)})$ as a dealer and publishes the encrypted shares $\mathsf{Enc}(pk_i, s_i^{(j)})$ for $1 \leq i \leq n$ and encryption proofs. P_j also publishes a commitment to the secret exponent $\mathsf{Com}(s^{(j)}, r_j)$ (with fresh randomness r_j).
2) **Verify.** For every set of published encrypted shares and proofs, all participants run $\mathsf{PVSS.ShareVerify}$ to verify correct encryption. Let $\mathcal{C}_r$ be the set of all participants with published commitments and valid shares.
3) **Reveal.** Once $t + 1$ participants have distributed their commitments and valid shares, every participant P_j, $j \in \mathcal{C}_r$ opens its commitment, and shares $\mathsf{Open}(s^{(j)}, r_j)$.
4) **Recover.** For every participant $P_a \in \mathcal{C}_r$ that withholds $\mathsf{Open}(s^{(a)}, r_a)$ in Reveal phase, other participants P_j for $j \neq a$ reconstruct $h^{s^{(a)}}$ via $\mathsf{PVSS.Recon}$, which requires each participant to publish its decrypted share $h^{s_j^{(a)}}$ and the proof of correct decryption passing $\mathsf{PVSS.ShareVerify}$.
5) **Aggregate.** The final randomness is $\mathcal{O}_r = \prod_{j \in \mathcal{C}_r} h^{s^{(j)}}$.

Note that Scrape, in the optimistic case (without Recover), is basically a commit-reveal with $O(n^2)$ PVSS shares distributed in the network during commit, $O(n)$ per node. In the worst case (with Recover), it requires an entirely new round of communication and potentially $O(n)$ Lagrange interpolations.

Albatross. Extending Scrape, Albatross [20] provides an improved amortized communication complexity of $O(n)$ per beacon output by generating a batch of $O(n^2)$ beacon outputs per round (as opposed to Scrape's one). This is achieved by two techniques: packed Shamir secret sharing and linear t-resilient functions [20], each of which contributes a multiplicative factor of $O(n)$ to the number of beacon outputs produced per round.

2) Share-Reconstruct-Aggregate: Another approach is to skip the commit-reveal phase and by default reconstruct each secret shared via PVSS. In other words, we can remove the $\mathsf{Com}(s^{(j)}, r_j)$ portion from Scrape's commit (*share*), perform Lagrange interpolation per secret for a total of $O(n)$ times (*reconstruct*), and sum up the interpolated secrets to output $\mathcal{O}_r$ (*aggregate*). While the resulting *share-reconstruct-aggregate* saves a round of communication from Scrape's worst case, its average case does incur substantial communication cost due to $O(n)$ Lagrange interpolations, each of which requires cooperation of $t+1$ nodes. Hence, this approach is preferable when it can be assumed that most rounds will require recovery due to faulty participants. RandShare [60] uses this technique alongside a Byzantine agreement protocol to reach consensus on $O(n)$ secrets to be reconstructed.

3) Share-Aggregate-Reconstruct: Another alternative is to harness the homomorphic property of PVSS where the sum of n respective shares of n secrets is a share of the sum of n secrets (note that the sum of n secrets is precisely $\mathcal{O}_r$ in these protocols). Due to this homomorphism, only one, as opposed to $O(n)$, Lagrange interpolation can be used to reconstruct $\mathcal{O}_r$ if nodes essentially perform the *aggregate* phase before the *reconstruct* phase, hence the name *share-aggregate-reconstruct*. SecRand [41] uses this technique to

reduce the communication complexity by a factor of n during reconstruction as only $t+1$ aggregated shares are exchanged to compute $\mathcal{O}_r$.

B. From Threshold Encryption

While protocols based on threshold secret sharing can incur high communication cost of $O(n^4)$ due to $O(n)$ Lagrange interpolations, protocols relying on a different cryptographic primitive, namely threshold encryption [30], offer a variant where only one Lagrange interpolation suffices even in the worst case. Though reminiscent of share-aggregate-reconstruct, these protocols differ in that they do not necessarily assume a PKI (mandatory in PVSS) but rather a DKG, which may be run multiple times to refresh keys. In this section, we define threshold encryption and summarize how a protocol like HERB [24] uses it to construct a DRB.

Definition V.2 ((t,n)-threshold encryption)**.** A (t,n)-*threshold encryption* scheme uses DKG among n nodes as a subroutine and allows encryption of a message under the resulting group public key pk such that the message can be decrypted by any $t+1$ of the n nodes, but not less. The scheme (ThrEnc) is composed of the following algorithms.

- $\mathsf{DKG}(1^\lambda, t, n) \rightarrow (sk_i, pk_i, pk)$ runs a typical DKG.
- $\mathsf{Enc}(pk, m) \rightarrow c$ encrypts message m with group public key pk and outputs ciphertext c.
- $\mathsf{DecShare}(sk_i, c) \rightarrow d_i$ generates decryption share d_i for ciphertext c using individual secret key sk_i.
- $\mathsf{Rec}(A, c, pk, \{pk_i\}_{i\in A}, \{d_i\}_{i\in A}) \rightarrow m$ takes ciphertext c, pk, and a set A of $t+1$ nodes with valid decryption shares along with their individual public keys and recovers the message m via Lagrange interpolation.

HERB uses threshold ElGamal encryption [30], [36] (see Appendix E for details) though it can be replaced with any other threshold homomorphic encryption scheme. After an initial DKG (ThrEnc.DKG), the protocol proceeds in two phases. In the first phase, nodes play the role of *entropy providers* each offering some entropy contribution to generate a group ciphertext. In the second phase, nodes play the role of *key holders* performing threshold decryption. While entropy providers and key holders could technically be different nodes, we assume they are the same below.

1) **Publication.** Each entropy provider P_j encrypts its entropy contribution m_j using ThrEnc.Enc to generate a ciphertext share c_j, published along with a proof of correct encryption $\pi^{(j)}_{CE}$ (Appendix I4) to account for malleability [32]. When $\mathcal{C}_r$ (the agreed set of nodes with verified published c_j's) reaches a certain size (based on system parameters), the included c_j's are summed into group ciphertext c corresponding to group plaintext m (which is in turn a sum of m_j's).
2) **Disclosure.** Each key holder P_i uses ThrEnc.DecShare to generate a decryption share d_i, published along with a proof of correct decryption $\pi^{(i)}_{DLEQ}$ (Appendix I3). When $t+1$ decryption shares are published and verified, nodes use ThrEnc.Rec to output $\mathcal{O}_r = m$.

Similar to SecRand's share-aggregate-recover, HERB achieves a communication complexity of $O(n^2)$ and $O(n^3)$ in the optimistic and worst cases, respectively, due to one needed Lagrange interpolation per $\mathcal{O}_r$. Its requirement of DKG in the setup presents a caveat however, as a new DKG must take place for any attempt to refresh keys of participants, e.g. in case of a suspected hack or a simple *reconfiguration* in which the set of participants changes. This can incur additional cost per DKG. On the flip side, HERB provides the advantage that entropy providers need not be key holders (as noted before), meaning that the level of security and randomness quality can be adjusted independently if necessary.

VI. Committee-Based Protocols

While all aforementioned commit-reveal variants (Sections III-A, IV, and V) include every node in each entropy-providing committee $\mathcal{C}_r$, this introduces a scalability problem. Requiring communication of marginal entropy by all nodes is inefficient with large numbers of participants, and hence a natural optimization is to use a smaller subset of nodes $\mathcal{C}_r$ in each round to contribute entropy (i.e. reduce $|\mathcal{C}_r|$).

In this section, we consider DRBs that are committee-based, where a *committee* refers to a non-empty proper subset of nodes. Committee-based protocols proceed in two steps: *committee selection* and *beacon output generation*. As the names suggest, $\mathcal{C}_r$ is agreed upon during committee selection while the beacon output $\mathcal{O}_r$ is generated and agreed upon during beacon output generation. We observe that committee selection and beacon output generation are, at least theoretically, modular such that subprotocols can be independently chosen for the two components. We visualize these two dimensions of committee-based DRBs in Table II. We also observe that the protocols introduced so far (e.g. commit-reveal-recover) can be used as a module in a larger committee-based protocol, with the chosen committee executing the chosen protocol in each round.

A. Step 1. Committee Selection

The first step of a committee-based DRB involves selecting $\mathcal{C}_r$ in a way agreeable by all nodes. We classify committee selection mechanisms into two: public and private.

1) Public Committee Selection: In a *public committee selection*, only public information is needed to derive $\mathcal{C}_r$.

Round-Robin (RR). A first example is *round-robin* (RR), in which nodes simply take turns being selected such that there is no notion of hierarchy among nodes. While RR can work with committees of any size, typically RR is used to select a committee of size one (i.e. a leader) corresponding to node $i \equiv r \pmod n$. Protocols like BRandPiper [10] (in which the round leader is the only active entropy provider) adopt RR as their leader selection mechanism due to its innate fairness property [5] (also known as chain quality [38] in the blockchain context) where all nodes, by RR's definition, take equal leadership.

Random Selection (RS). A second example is *random selection* (RS), which uses some public randomness (most commonly the last beacon output $\mathcal{O}_{r-1}$) to derive $\mathcal{C}_r$. In HydRand [57] and GRandPiper [10], for instance, $\mathcal{C}_r$ consists of a node $i \equiv \mathcal{O}_{r-1} \pmod{\tilde{n}}$ where $\tilde{n}$ is the number of eligible nodes. Similar is the process in Ouroboros [45] called follow-the-satoshi [9], [45], which can output more than one node in $\mathcal{C}_r$.

Randomized selection means that some nodes may, in theory, never be selected and therefore may never be able to provide entropy contribution. A more serious concern is that an adversary can attempt to bias, via grinding attack, $\mathcal{O}_r$ in order to bias $\mathcal{C}_{r+1}$ (which can bias $\mathcal{O}_{r+1}$). In the worst case, this can lead to a vicious cycle in which an adversary controlling enough nodes on the current committee to manipulate the beacon output can ensure it will also control enough nodes on the next committee, and so on ad infinitum.

This is not an issue in RR, as its committee selection is deterministic and independent of the preceding beacon output. Nonetheless, a tradeoff of RR is that DoS attack becomes indefinitely possible (for all rounds $\tilde{r}$ for $\tilde{r} > r$ given $\mathcal{O}_r$) since each committee is publicly known in advance. All in all, RR gains unbiasability (due to determinism) at the cost of indefinite DoS attack while RS benefits from less DoS attack, i.e. that only for round $r + 1$ (due to randomization given $\mathcal{O}_r$), at the cost of grinding attack.

Leader-Based Selection (LS). A third example, *leader-based selection* (LS) is a hybrid method that exhibits both determinism and randomization. It runs in two steps: the first step involves electing a round leader (either by RR or RS) while the second involves selection of $\mathcal{C}_r$ by the elected leader. It is in this way that the mechanism is deterministic from the leader's perspective while randomized from that of others.

One approach to limit the power delegated to the leader is that $|\mathcal{C}_r|$ needs to be greater than t so that a malicious leader wouldn't be able to choose $\mathcal{C}_r$ maliciously. RandHound [60] and SPURT [28] demonstrate such LS.

- RandHound. As instantiated in RandHerd [60], RandHound's leader election (i.e. via RS as the first step of LS) involves a public lottery where each node generates a lottery ticket $t_i = H(C \| pk_i)$ given a public configuration parameter C (assuming its randomness) such that the owner of $min(t_i)$ becomes the leader (originally called client). In the second step of LS, RandHound adopts a form of sharding (involving PVSS groups). The leader selects more than a threshold number of nodes in each shard (PVSS group), guaranteeing a threshold number of entropy providers across all shards.
- SPURT. Unlike RandHound, SPURT adopts RR as its first part of LS such that nodes simply take turns being a round leader. Then the leader chooses $\mathcal{C}_r$ based on received encrypted messages.

Given an underlying DRB that utilizes the concept of a leader as an orchestrator of communication among nodes, LS is a natural choice to committee selection, as a leader helps mitigate the protocol's communication cost overall.

2) Private Committee Selection: In a *private committee selection*, also known as *private lottery*, each node needs to input some private information (e.g. secret key) in order to check whether or not it has been selected into $\mathcal{C}_r$ (i.e. has won the lottery). The general formulation of a private lottery can be given by

$$f_{priv}(\cdot) < target$$

where $f_{priv}(\cdot)$ is a lottery function (i.e. pseudorandom function) that takes some private input $priv$ and $target$ denotes the lottery's "difficulty level" (a la Proof of Work difficulty) such that $target$ can be adjusted to make the lottery arbitrarily easy or hard to win. Each node calculates $f_{priv}(\cdot)$ and checks if the above inequality is satisfied, in which case it "wins" the lottery and becomes an entropy provider.

As it is possible for an adversary to perform a grinding attack by trying many values of $priv$ until a desirable function output is achieved, one crucial requirement is that $priv$ should be provably committed in the past and thus be ungrindable at the time of computation of $f_{priv}(\cdot)$. Two examples of the above formulation exist in the landscape: a VRF-based lottery and Caucus' [5] lottery involving a hash chain.

VRF-based approach. In Algorand [40], a VRF (verifiable random function) [31], [48] is used in each round to realize a lottery. (While there exist more than one version of Algorand's private lottery algorithm, we consider its first version, as the versions do not differ fundamentally.) Quite naturally, one's private input to $VRF_{sk}(\cdot)$ is its secret key such that the lottery is given by

$$VRF_{sk}(\mathcal{O}_{r-1} \parallel role) < target$$

where $role$ is some parameter specific to Algorand. As both $\mathcal{O}_{r-1}$ and $role$ are already public and ungrindable at the time of computation, Algorand makes sure sk is likewise ungrindable by requiring that sk is committed a certain number of rounds in advance. Similar are private lotteries for protocols like Ouroboros Praos [29] and NV (from Nguyen-Van et al. [50]). See Table II for details.

Replacing VRF with $H(\cdot)$ and hash chain. In Caucus, the VRF is replaced by a hash function and a hash chain, i.e. a list $(h_1, ..., h_m)$ with $h_r = H(h_{r+1})$ for all $r = 1, ..., m-1$ where $h_m = s$ for some random seed. A hash chain provides the functionality of provably committing to private inputs as one can publicize one h_r at a time (i.e. h_r in round r) such that doing so commits to h_{r+1}. Thus, each participant of Caucus independently generates a private hash chain comprising m private inputs such that the lottery is given by

$$H(h_r \oplus \mathcal{O}_{r-1}) < target$$

which simply involves a hash function. One downside is that the hash chain needs to be periodically regenerated, as m is finite while we desire our DRB to run indefinitely. Verifying

a winning ticket in Caucus also requires work linear in the length of the hash chain (though a tree-based structure could make this cost logarithmic).

Private lotteries provide two notable benefits: resilience to DoS attack (due to its property of delayed unpredictability [5] where one cannot predict the eligibility of honest nodes until they reveal) and *independent participation* (i.e. nodes do not have to know other participants in advance to participate) allowing less communication cost as well as a more permissionless setting. Nonetheless, it can introduce the possibility of biasing via withholding (as discussed in Section VIII-B).

B. Step 2. Beacon Output Generation

Given a committee $\mathcal{C}_r$, the issue then shifts to outputting $\mathcal{O}_r$. While a typical commit-reveal-recover run among nodes in $\mathcal{C}_r$ may be sufficient to realize a DRB, other approaches provide different tradeoffs. Largely, we classify these variations into two: one that requires *fresh* (independently generated on the spot) per-node entropy (contribution) and one that combines previous beacon output with *precommitted* (independently generated but precommitted, hence ungrindable) per-node entropy.

1) Fresh Per-Node Entropy: Beacon output generation process involving fresh (also referred to as true randomness [21], [28] as opposed to pseudorandomness) per-node entropy for committee-based protocols is typically a commit-reveal-recover variant from Section V. Some protocols in this family include the following.

Share-Reconstruct-Aggregate. In Ouroboros, nodes in $\mathcal{C}_r$ (i.e. slot leaders of epoch r) perform a RandShare-style share-reconstruct-aggregate using PVSS to output $\mathcal{O}_r$. RandHound uses a similar approach, facilitated by a round leader.

Share-Aggregate-Reconstruct. In SPURT and BRandPiper, nodes in $\mathcal{C}_r$ perform a SecRand-style share-aggregate-reconstruct to output $\mathcal{O}_r$. BRandPiper has a twist, however: it utilizes the idea of buffering PVSS shares in advance. While there exists one entropy provider per round, n secrets (one from each node) are combined such that it provides the ideal 1-unpredictability property as opposed to t-unpredictability (as in HydRand or GRandPiper). The trick is that each round leader generates n fresh secrets that become combined with others' secrets in the next n rounds, respectively. One node distributes $O(n^2)$ PVSS shares (buffered by other nodes) per round in BRandPiper whereas, in a typical share-aggregate-reconstruct like SPURT, each of $O(n)$ nodes distributes $O(n)$ PVSS shares per round (with no buffering).

From Threshold Encryption. Similar to HERB, entropy providers in NV [50] contribute their fresh entropy using ElGamal although they use its classical, non-threshold version due to NV's centralized model in which a third party called the Requester is the direct recipient of a beacon output. As a result, each entropy provider generates and encrypts its entropy and sends it to the Requester, which then decrypts all the messages received from entropy providers and outputs their sum as $\mathcal{O}_r$. Naturally, this Requester version of NV can be modified into what we call *NV++*, which differs from NV in two ways. First, nodes in $\mathcal{C}_r$ (once finalized) can be made to perform HERB among themselves. This eliminates the existence of the centralized Requester. Second, entropy provision (i.e. broadcasting one's entropy) can be coupled with proof of membership to $\mathcal{C}_r$ (i.e. broadcasting the fact that a node has won the VRF private lottery). In NV, these two are separate steps potentially incurring adaptive insecurity (a concept delineated in Section VIII-C). Thus, NV++ distributes the Requester and achieves adaptive security.

2) Combining Previous Output and Precommitted Per-Node Entropy: To optimize and lower communication cost, we can require generally less input from entropy providers each round. The canonical optimization involves utilizing $\mathcal{O}_{r-1}$ as a source of entropy to produce $\mathcal{O}_r$. Nonetheless, the caveat in doing so is that grinding attack may become a possibility once $\mathcal{O}_{r-1}$ becomes public, which is why it is necessary to require entropy providers' contribution for round r to be precommitted before combining with $\mathcal{O}_{r-1}$ to output $\mathcal{O}_r$. This prevents grindability while taking advantage of the convenience of $\mathcal{O}_{r-1}$. Such a requirement can be observed in many committee-based protocols, even though their details seem unrelated on the surface.

- HydRand and GRandPiper. Each round, an entropy provider (i.e. round leader) in HydRand commits its entropy that becomes opened in the next round it is selected as the leader again. In other words, the round leader's precommitted entropy $e_{\tilde{r}}$ from its last round $\tilde{r}$ of leadership is the one that becomes combined with $\mathcal{O}_{r-1}$ in the form of $h^{e_{\tilde{r}}}$ to generate
$$\mathcal{O}_r = H(\mathcal{O}_{r-1} \parallel h^{e_{\tilde{r}}})$$
while PVSS recovery is used in case the leader fails to open $e_{\tilde{r}}$ in round r. Notable in HydRand is the fact (achieving ungrindability of $h^{e_{\tilde{r}}}$) that one honest node must be present in any $t+1$ consecutive rounds due to the requirement that a leader cannot gain another leadership in the next t rounds. Similar overall is GRandPiper's beacon output generation (see Table II).
- Algorand and Ouroboros Praos. These schemes use a VRF for beacon output generation (rather than only for committee selection as in NV++). The secret key sk of the round leader often corresponds to precommitted per-node entropy as long as the assumption that nodes cannot switch their sk at the time of VRF's computation holds. Algorand's beacon output is therefore given by
$$\mathcal{O}_r = VRF_{sk}(\mathcal{O}_{r-1} \parallel r)$$
combining the previous output $\mathcal{O}_{r-1}$ with the precommitted entropy sk. Note that the input to the VRF in beacon output generation is different from that in committee selection, as the VRF output in committee selection is

always going to be less than $target$ by design. Ouroboros Praos' beacon output is generated similarly (see Table II).
- Caucus. Each new reveal (h_r in round r) from an entropy provider's private hash chain in Caucus corresponds to that node's precommitted entropy. The beacon output is given by
$$\mathcal{O}_r = h_r \oplus \mathcal{O}_{r-1}$$
such that it naturally follows its committee selection mechanism $H(h_r \oplus \mathcal{O}_{r-1}) < target$.

VII. Protocols With No Marginal Entropy

While DRBs can have all nodes or some subset of nodes contribute entropy to the beacon output every round, it is possible to devise a protocol where no node contributes any marginal entropy as the beacon runs. The advantage of this approach is that no node needs to generate and communicate any fresh entropy (which alleviates communication cost) while the disadvantage is that the entire beacon is permanently predictable once compromised (perhaps undetectably).

A. From Distributed Verifiable Random Function

Guaranteeing randomness entirely via pseudorandomness, such DRB can be based on distributed VRF [37], [44] (DVRF, also known as threshold VRF or TVRF [21]). The idea is that the VRF's sk is distributed among n nodes via DKG such that $t+1$ nodes can cooperate to compute a per-round VRF output (as well as its proof), as if the computation involves one hypothetical node with access to sk.

Definition VII.1 (Distributed verifiable random function)**.** A *distributed verifiable random function* (DVRF) is a VRF where n nodes cooperate to yield a pseudorandom output such that up to t Byzantine nodes are tolerated while any $t+1$ honest nodes are able to yield an honest output. It can be described by the following tuple of algorithms.

- $\mathsf{DKG}(1^\lambda, t, n) \to (sk_i, pk_i, pk)$ runs a typical DKG.
- $\mathsf{PartialEval}(sk_i, x) \to (y_i, \pi_i)$ outputs the partial evaluation y_i as well as its proof of correctness π_i given an input x and a node's secret key sk_i.
- $\mathsf{PartialVerify}(pk_i, x, y_i, \pi_i) \to \{0,1\}$ verifies the correctness of the partial evaluation y_i given its proof π_i, an input x, and a node's public key pk_i.
- $\mathsf{Combine}(A, \{(y_i, \pi_i)\}_{i\in A}) \to (y, \pi)$ outputs the DVRF evaluation y as well as its proof of correctness π given a set A of $t+1$ nodes and their outputs of $\mathsf{PartialEval}(sk_i, x)$, all of which pass $\mathsf{PartialVerify}$.
- $\mathsf{Verify}(pk, \{pk_i\}, x, y, \pi) \to \{0,1\}$ verifies the correctness of the DVRF evaluation y given π, input x, and public keys.

DVRF-based DRB. Each beacon output of a DVRF-based DRB is then given by

$$\mathcal{O}_r = \mathsf{DVRF.Combine}(A, \{\mathsf{DVRF.PartialEval}(sk_i, f(\mathcal{O}_{r-1}))\}_{i\in A})[0]$$

where sk_i denotes each node's secret key after a DKG and f denotes some deterministic function of $\mathcal{O}_{r-1}$.

The output is equivalent to one trustworthy hypothetical node with complete knowledge of sk computing the output as:

$$\mathcal{O}_r = VRF_{sk}(f(\mathcal{O}_{r-1}))$$

As f typically takes a form resembling $f(\mathcal{O}_{r-1}) = H(r \parallel \mathcal{O}_{r-1})$, there is no marginal entropy contributed by the participants. Unpredictability of the above DVRF formulation relies on the fact that no one node (or up to t nodes) can gain knowledge of sk to be able to compute and predict future beacon outputs.

DVRF-based DRB from a chain of unique signatures. Since taking the hash of a verifiable unpredictable function (VUF) [48] is equivalent to a VRF, a unique digital signature (which is a VUF [31]) can be made into a DVRF by computing its threshold variant [12] and hashing the output. Dfinity [44] and drand [1] (while differing slightly in minor details) both use the BLS signature scheme [16] to realize a DRB as

$$\mathcal{O}_r = H(\mathsf{Sign}_{sk}(r \parallel \mathcal{O}_{r-1}))$$

where $\mathsf{Sign}_{sk}(\cdot)$ is a threshold BLS signature computed by at least $t+1$ nodes with sk as the implied group secret key generated via DKG. The actual computation of $\mathcal{O}_r$ involves combining of partial signatures computed using sk_i (see Appendix I2).

Variations on a chain of unique signatures. Besides a chain of BLS signatures, there exist several other variations.

- RandHerd [60]. Two modifications are made in RandHerd. First, a form of "sharding" is performed where nodes are randomly configured into groups (each of size c) via some initial configuration seed (e.g. derived from RandHound) such that each group emits a group leader while that of the first group is deemed a cothority (collective authority) leader. This results in the notion of hierarchy among nodes in a tree structure and thus has the effect of reducing the communication complexity from $O(n^2)$ to $O(c^2 \log n)$. Second, the underlying signature scheme used is Schnorr instead of BLS. Each beacon output in RandHerd is essentially a threshold Schnorr signature on message (as per the original protocol) $m = t_r$ where t_r denotes the timestamp at the beginning of round r. As m can technically be chosen (and thus biased) by the leader, one simple improvement can be setting $m = r \parallel \mathcal{O}_{r-1}$ a la Dfinity or drand.
- DDH-DRB [37]. Each round of DDH-DRB involves DDH-DVRF. As $\mathsf{PartialVerify}$ and Verify from Dfinity-DVRF (i.e. each round of Dfinity) rely on verifying pairing equations (see Appendix H), the speed at which each beacon output is generated can be made faster if we replace each pairing equation with a DLEQ NIZK (Appendix I3) achieving the same effect. The tradeoff is space, as π grows linearly in n. See Appendix F for details.
- GLOW-DRB [37]. Each round involves GLOW-DVRF,

which strikes a balance between Dfinity-DVRF and DDH-DVRF by involving a pairing equation in Verify (a la Dfinity-DVRF) but a DLEQ NIZK in PartialVerify (a la DDH-DVRF). This has the effect of generating a compact proof π from Verify while enjoying less computational cost from PartialVerify. See Appendix G for details.

VIII. Discussion

We summarize key properties of all protocols discussed in Table I and explore several practical issues with DRBs.

A. Grinding Attack

In a grinding attack, an adversary can search over many possible inputs to broadcast in a given round, with the possibility of influencing the outcome. While grinding attacks are not a threat in commit-reveal-recover variants or protocols with no marginal entropy, they are a valid concern in committee-based protocols involving random selection (RS), a private lottery, or a beacon output generated by combining precommitted per-node entropy with $\mathcal{O}_{r-1}$. The idea is that an adversary can grind to produce a biased $\mathcal{C}_r$, $\mathcal{O}_r$, or both (leading to a vicious cycle in the worst case).

Two countermeasures are possible. First, inclusion of fresh marginal entropy from at least one honest node can be required, in which case grinding fails due to the adversary's inability to control such entropy. Second, precommitted entropy must indeed be precommitted, in which case grinding vacuously fails due to the lack of any grindable entropy.

1) **Requirement of fresh marginal entropy from at least one honest node.** Due to the t-limited adversary assumption, it is possible to force the inclusion of fresh marginal entropy from at least one honest node if we require more than t entropy providers each round. Such a requirement is used in protocols like RandHound (where committee selection via LS has an explicit size requirement), Ouroboros (where the size of each epoch can be made large), and NV++ (where the VRF's target used for committee selection can be adjusted).
2) **Requirement of precommitted entropy.** No grindable entropy naturally means no possible grinding attack. In HydRand (and similarly GRandPiper), it is required that the precommitted entropy ($e_{\tilde{r}}$ from $\mathcal{O}_r = H(\mathcal{O}_{r-1} \,\|\, h^{e_{\tilde{r}}})$ as per Section VI-B2) used in round r is precommitted in round $\tilde{r}$ where $r - \tilde{r} > t$ such that it is ungrindable due to the protocol's t-unpredictability property. In the protocols based on private lottery (i.e. Algorand, Ouroboros Praos, and Caucus), the private inputs to the lottery (VRF's sk or each h_r of a hash chain) are fixed in advance to make them ungrindable.

B. Withholding Attack

In a withholding attack, an adversary can influence the outcome by not publishing some information in a given round. Any leader-based protocol is vulnerable to withholding due to the inherent reliance on a leader's availability, affecting the protocol's liveness (as well as potentially unpredictability and unbiasability). Any protocol with a private lottery is also vulnerable due to its nature where the lottery winner has to announce itself as a winner in the first place or can withhold this announcement if desirable.

1) **Protocols with a leader.** RandHound, RandHerd, and SPURT suffer from the leader unavailability issue in case the leader withholds its message such that their liveness is affected and a beacon output can be aborted (depending on implementation). In worse cases like RandHound and RandHerd, this can create a bias if the leader aborts after seeing $\mathcal{O}_r$. In either case, it is desirable to have some fallback in case a leader withholds (e.g. HydRand's PVSS recovery).
2) **Protocols with a private lottery.** The issue of withholding is more fundamental with private lottery schemes like Algorand (where the leader can bias via withholding after privately computing $\mathcal{O}_r$), as there is no way to conclusively detect if a winner withholds its leadership, i.e. there is no accountability. There are two possible remedies. First, we can require all participants to post their lottery outputs every single round even if they lose the lottery, in which case any lack of message would be indicative of withholding. However, this incurs communication cost, negating the advantages of a private lottery. Second, a technique called SSLE (single secret leader election) [15] can be used to guarantee one winner per round, enabling detection of withholding. In a nutshell, SSLE ensures exactly one leader from a group is randomly chosen while the identity of the leader will only be known when the winner publicly reveals its identity. The guarantee of one winner as opposed to the expectation of one winner is what differentiates SSLE. While this guarantee makes withholding obvious, it does not prevent withholding by itself, nor does it enable detecting *who* the withholding winner was in the case of withholding.

C. Adaptive Security

A DRB is *adaptively secure* if its security properties remain unaffected against an adaptive adversary instead of a static one. Otherwise, it is *adaptively insecure*. In this section, we discuss three ways in which adaptive security is achieved.

1) **$|\mathcal{C}_r|$ is greater than t or equal to 0.** Due to the t-limited adversary assumption, a large enough $\mathcal{C}_r$ (or an empty one as per Section VII) guarantees adaptive security. Otherwise, a protocol may be vulnerable. In HydRand and GRandPiper (where $|\mathcal{C}_r| = 1$), an adaptive adversary can corrupt the next t round leaders to predict $t+1$ future rounds. In Ouroboros, it can adaptively corrupt the entire $\mathcal{C}_r$ (which could probabilistically be less than or equal to t in size) publicly known in advance.
2) **Protocols with private lotteries require lottery winners to broadcast marginal entropy and proof of selection into $\mathcal{C}_r$ in the same message.** While some private lottery schemes (i.e. Algorand, Ouroboros Praos, and Caucus) may involve less than or equal to t entropy providers per

TABLE I: DRB Comparison

	Section (from paper)	Cryptographic Primitive	Fault Tolerance (less than)	Independent Participation	Per-Round Entropy Provider	Unpredictability	Immunity to Withholding	Adaptive Security	Verifier Complexity	Communication Complexity		Max Damage	Recovery Cost
										Optimistic	Worst		
Commit-Reveal	II	Commitment	1	✓	All	✗	✗	✗	$O(n)$	$O(n^2)$	$O(n^3)$	Bias	$O(1)$
Unicorn++	III	VDF	n	✓	All	1	✓	✓	$O(n)$	$O(n^2)$	$O(n^3)$	None	$O(1)$
Ext. Beacon+VDF		VDF	n	✓	External	1	✓	✓	$O(1)$	$O(n)$	$O(n^2)$	None	$O(1)$
RandRunner		Trapdoor VDF	n	✗	None	t [§]	✓	✗	$O(\log T)$ [‡]	$O(n)$	$O(n^2)$	Predict	$O(n^3)$
RANDAO	IV	Commitment	n	✓	All	1	✓	✓	$O(n)$	$O(n^2)$	$O(n^2)$ [†]	None	$O(n)$ [†]
EVR		Escrow-DKG	$n/3$	✗	All	1	✓	✓	$O(n^3)$	$O(n^3)$	$O(n^4)$	None	$O(n)$
Scrape	V	PVSS	$n/2$	✗	All	1	✓	✓	$O(n^2)$	$O(n^3)$	$O(n^4)$	Bias [r]	$O(n^3)$
Albatross		PVSS	$n/2$	✗	All	1	✓	✓	$O(1)$	$O(n)$	$O(n^2)$	Bias [r]	$O(n^3)$
RandShare		(P)VSS	$n/3$	✗	All	1	✓	✓	$O(n^3)$	$O(n^3)$	$O(n^4)$	Bias [r]	$O(1)$
SecRand		PVSS	$n/2$	✗	All	1	✓	✓	$O(n^2)$	$O(n^3)$	$O(n^4)$	Bias [r]	$O(n^3)$
HERB		Thr. ElGamal	$n/3$	✗	All	1	✓	✓	$O(n)$	$O(n^2)$	$O(n^3)$	Bias [r]	$O(n^4)$
HydRand	VI	PVSS	$n/3$	✗	Committee [*]	t	✓	✗	$O(n)$	$O(n^2)$	$O(n^3)$	Bias	$O(n^3)$
GRandPiper		PVSS	$n/2$	✗	Committee [*]	t	✓	✗	$O(n^2)$	$O(n^2)$	$O(n^2)$	Bias	$O(n^3)$
BRandPiper		(P)VSS	$n/2$	✗	Committee [*]	1	✓	✓	$O(n^2)$	$O(n^2)$	$O(n^3)$	Bias [r]	$O(n^4)$
Ouroboros		PVSS	$n/2$	✗	Committee	1	✓	✗	$O(n^2)$	$O(n^3)$	$O(n^3)$ [†]	Bias [r]	$O(n^2)$ [†]
RandHound		PVSS	$n/3$	✗	Committee	1	✗	✗	$O(cn)$	$O(c^2n)$	$O(c^2n^2)$	Bias	$O(n^3)$
SPURT		PVSS	$n/3$	✗	Committee	1	✗	✗	$O(n)$	$O(n^2)$	$O(n^2)$	Bias	$O(n^3)$
Algorand		VRF	$n/3$	✓	Committee [*]	1	✗	✓	$O(1)$	$O(n)$	$O(n)$ [†]	Bias	$O(n^2)$ [†]
Ouroboros Praos		VRF	$n/2$	✓	Committee	1	✗	✓	$O(n)$	$O(n^2)$	$O(n^2)$ [†]	Bias	$O(n^2)$ [†]
Caucus		Hash chain	$n/3$	✓	Committee [*]	1	✗	✓	$O(1)$	$O(n)$	$O(n^2)$	Bias	$O(n^3)$
NV++		VRF, thr. ElGamal	$n/3$	✗	Committee	1	✗	✓	$O(n)$	$O(n)$	$O(n)$ [†]	Bias	$O(n^2)$ [†]
drand	VII	Thr. BLS	$n/2$	✗	None	1	✓	✓	$O(1)$	$O(n^2)$	$O(n^3)$	Predict	$O(n^4)$
RandHerd		Thr. Schnorr	$n/3$	✗	None	1	✗	✗	$O(1)$	$O(c^2 \log n)$	$O(n^3)$	Bias	$O(n^4)$
DDH-DRB		DDH-based DVRF	$n/2$	✗	None	1	✓	✓	$O(1)$	$O(n^2)$	$O(n^3)$	Predict	$O(n^4)$
GLOW-DRB		Pairing-based DVRF	$n/2$	✗	None	1	✓	✓	$O(1)$	$O(n^2)$	$O(n^3)$	Predict	$O(n^4)$

c is the size of a shard in RandHerd and RandHound. We assume a leader can be Byzantine for both. Albatross' verifier and communication complexities are per beacon output. In Ouroboros and Ouroboros Praos, we assume the number of slot leaders in an epoch is denoted by n. We assume Scrape's PVSS [19] is used as the default PVSS scheme.
[*] Each committee consists of a leader by default or by expectation. [†] PBB (public bulletin board) is assumed. [‡] Verification of Pietrzak's VDF is logarithmic in T (VDF's delay parameter). [§] $d = t$ for RandRunner's d-unpredictability assuming a dishonest minority without any computational advantage. See [56] for more scenarios.
[r] In a non-rushing adversary model, max damage would be predict rather than bias.

round, the fact that one message (per entropy provider) comprises both announcement of winning the lottery and provision of marginal entropy is what allows adaptive security. The idea is that by the time an adversary knows which nodes to corrupt adaptively in a round (after the nodes reveal their identity as entropy providers), there is no extra step left to be corrupted, as each entropy provider's contribution to $\mathcal{O}_r$ has been broadcast already in the same message.

3) **There is no central point of dependency in any step of the protocol.** In leader-based protocols where a leader functions more as an orchestrator than an entropy provider, participating nodes may still need to depend on the leader to make progress on the beacon such that an adversary can adaptively corrupt such leaders to its benefit. In RandRunner, corrupting the next t leaders allows predictability. In RandHound and RandHerd, corrupting the round leader allows biasability if the leader aborts after seeing $\mathcal{O}_r$ as aforementioned. In SPURT, corrupting the next t leaders to withhold endangers liveness.

D. Comparison of DRBs

Table I provides an overall comparison of DRBs. *Fault Tolerance* indicates the minimum number of faulty nodes that

can abort a protocol (after the initial setup). Protocols with *Independent Participation* allow a node to contribute to beacon output without the knowledge of other nodes in advance. However, it differs from a permissionless setting in the sense that a node may still have to register in advance to allow verification of its contribution.

Verifier Complexity refers to the computational cost for a passive node (third party) to verify a beacon output. We exclude the cost associated with the initial setup for both verifier and communication complexities. We assume a verifier complexity of $O(n)$ per Lagrange interpolation or Scrape's PVSS [19] run. *Communication Complexity* concerns bitwise point-to-point communication among nodes by default. Alternatively, we consider a *public bulletin board* (PBB) as a reliable information exchange medium in protocols where it is intrinsic (e.g. in blockchains). In a PBB model, we assume both the bitwise writing cost (amount of data posted to PBB) and the reading cost (by all nodes where each node only reads data relevant to it) contribute to the total cost. In the absence of PBB, Byzantine consensus [22] incurs a cost of $O(n^2)$ per decision by default.

Max Damage refers to the maximum damage possible when $n-1$ rushing [39] (where an adversary can delay sending messages until *after* reading messages sent by the honest nodes in any round of communication) adversarial nodes cooperate to predict or bias. In escrow-based protocols, we assume the adversaries are rational. *Recovery Cost* refers to the communication cost associated with recovering from an adversarial corruption. Regenerating keys (e.g. $\mathsf{PVSS.KeyGen}$ or for private lottery schemes) and $\mathsf{VDF.Setup}$ incur $O(n^3)$ recovery cost without PBB (and $O(n^2)$ with PBB) while we assume each DKG incurs $O(n^4)$ recovery cost.

IX. Concluding Remarks

In this paper, we systematize distributed randomness beacons. Our systematization highlights important insights both for practitioners and researchers. Based on practical considerations such as scalability (in n), flexibility (reconfiguration and independent participation), robustness (fault tolerance and max damage), and randomness quality (true randomness versus pseudorandomness), we would advise practitioners planning to deploy a DRB as follows:

- VDF-based protocols stand above the competition in terms of scalability, flexibility, and robustness, enabling an efficient DRB with unlimited, open participation and security given any honest participant. In theory, VDFs appear to be a silver bullet for DRBs, though they have yet to be widely used in practice and assumptions about VDF security and hardware speeds remain relatively new. They also invoke a unique practical cost in that *somebody* must compute a VDF (preferably by running specialized hardware), which also induces latency into the DRB.
- If not using VDFs, practitioners need to think critically about two design dimensions: how large is the set of participants, and how frequently will it change? Given a small, static set of participants, DKG-based protocols, e.g. HERB (from threshold encryption) and drand (from DVRF), scale better than PVSS-based protocols. HERB and drand are both competitive in this setting, differing in randomness quality and max damage.
- For a small but dynamic set of participants, PVSS-based protocols offer better flexibility (by avoiding a costly DKG setup per reconfiguration) and randomness quality. Committees may be needed to scale to more participants.
- Given a large, dynamic set of participants, protocols with private lotteries like Algorand offer better scalability and flexibility simultaneously although the randomness quality is potentially affected by withholding.
- Finally, escrow-based protocols are suitable against purely financially-motivated adversaries in applications such as lotteries or finance, at the cost of locking up some amount of capital during the protocol.

We conclude by identifying the following areas which we consider most promising for further research.

- While VDFs are a promising tool, practical deployment requires good estimates of the lower bound of wall-clock VDF evaluation time. More research is needed to gain confidence in the security of underlying VDF primitives (such as repeated modular squaring), and hardware implementations must be built to provide practical assurance against attacks.
- VDFs might be useful as a modular layer in strengthening other DRBs in a "belt-and-suspenders" approach, though this does not appear to have been explored yet.
- To avoid the need for VDF evaluation (and latency) in the optimistic case, timed commitments (a related primitive to VDFs) might be used instead as yet another mechanism to recover from aborts in commit-reveal.
- Vulnerable to withholding, protocols based on private lotteries can generate biased outputs. Though the guarantee of a single lottery winner every round via SSLE makes withholding detectable, extending these protocols to enable tracing of which node withheld is a promising direction for further research.
- With the exception of VDF-based protocols like Unicorn++, all other DRBs assume a permissioned setting requiring some initial setup (e.g. PKI or DKG) to establish participants' identities. It is an open question which non-VDF-based protocols can be extended to enable ad hoc, permissionless participation.
- Existing DRBs assume synchronous communication, which may fail in practice. Extending protocols to handle fully asynchronous communication is an area for future research.
- Finally, there is a gap between the systems-based literature on DRBs and the traditional cryptographic literature on randomness extractors [62], [63], with DRBs simply assuming cryptographic primitives such as hash functions work as extractors in practice. Utilizing the existing theory of extractors could prove useful in scenarios where high-quality DRB outputs are required directly.

REFERENCES

[1] "Drand," https://drand.love/.

[2] B. Adida, "Helios: Web-based open-audit voting." in *USENIX security symposium*, 2008.

[3] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," *arXiv preprint arXiv:1708.03778*, 2017.

[4] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *2014 IEEE Symposium on Security and Privacy*, 2014.

[5] S. Azouvi, P. McCorry, and S. Meiklejohn, "Winning the caucus race: Continuous leader election via public randomness," *arXiv preprint arXiv:1801.07965*, 2018.

[6] T. Baigneres, C. Delerablée, M. Finiasz, L. Goubin, T. Lepoint, and M. Rivain, "Trap me if you can-million dollar curve." *IACR Cryptol. ePrint Arch.*, 2015.

[7] I. Bentov, A. Gabizon, and D. Zuckerman, "Bitcoin beacon," *arXiv preprint arXiv:1605.04559*, 2016.

[8] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Annual Cryptology Conference*. Springer, 2014.

[9] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake," *ACM SIGMETRICS Performance Evaluation Review*, 2014.

[10] A. Bhat, N. Shrestha, A. Kate, and K. Nayak, "Randpiper-reconfiguration-friendly random beacons with quadratic communication." *IACR Cryptol. ePrint Arch.*, 2020.

[11] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *ACM SIGACT News*, 1983.

[12] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *International Workshop on Public Key Cryptography*. Springer, 2003.

[13] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018.

[14] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions." *IACR Cryptol. ePrint Arch.*, 2018.

[15] D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco, "Single secret leader election," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020.

[16] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptology and information security*. Springer, 2001.

[17] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source." *IACR Cryptol. ePrint Arch.*, 2015.

[18] B. Bünz, S. Goldfeder, and J. Bonneau, "Proofs-of-delay and randomness beacons in ethereum," *IEEE Security and Privacy on the blockchain (IEEE S&B)*, 2017.

[19] I. Cascudo and B. David, "Scrape: Scalable randomness attested by public entities," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017.

[20] ——, "Albatross: publicly attestable batched randomness based on secret sharing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2020.

[21] I. Cascudo, B. David, O. Shlomovits, and D. Varlakov, "Mt. random: Multi-tiered randomness beacons," Cryptology ePrint Archive, Report 2021/1096, 2021.

[22] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, 1999.

[23] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Annual international cryptology conference*. Springer, 1992.

[24] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, "Homomorphic encryption random beacon." *IACR Cryptol. ePrint Arch.*, 2019.

[25] J. Clark and U. Hengartner, "On the use of financial data as a random beacon." *EVT/WOTE*, 2010.

[26] I. Damgård, "Commitment schemes and zero-knowledge protocols," in *School organized by the European Educational Forum*. Springer, 1998.

[27] ——, "On σ-protocols," *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[28] S. Das, V. Krishnan, I. M. Isaac, and L. Ren, "Spurt: Scalable distributed randomness beacon with transparent setup." *IACR Cryptol. ePrint Arch.*, 2021.

[29] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018.

[30] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *Advances in Cryptology — CRYPTO' 89 Proceedings*. Springer New York, 1990.

[31] Y. Dodis and A. Yampolskiy, "A verifiable random function with short proofs and keys," in *International Workshop on Public Key Cryptography*. Springer, 2005.

[32] D. Dolev, C. Dwork, and M. Naor, "Nonmalleable cryptography," *SIAM review*, 2003.

[33] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE, 1987.

[34] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986.

[35] M. J. Fischer, M. Iorga, and R. Peralta, "A public randomness service," in *Proceedings of the International Conference on Security and Cryptography*. IEEE, 2011.

[36] P.-A. Fouque and D. Pointcheval, "Threshold cryptosystems secure against chosen-ciphertext attacks," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001.

[37] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong, "Fully distributed verifiable random functions and their application to decentralised random beacons." *IACR Cryptol. ePrint Arch.*, vol. 2020, 2020.

[38] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015.

[39] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999.

[40] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.

[41] Z. Guo, L. Shi, and M. Xu, "Secrand: A secure distributed randomness generation protocol with high practicality and scalability," *IEEE Access*, 2020.

[42] M. Haahr, "Random. org: True random number service," *School of Computer Science and Statistics, Trinity College, Dublin, Ireland. Website (http://www. random. org). Accessed*, 2010.

[43] R. Han, J. Yu, and H. Lin, "Randchain: Decentralised randomness beacon from sequential proof-of-work." *IACR Cryptol. ePrint Arch.*, 2020.

[44] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," *arXiv preprint arXiv:1805.04548*, 2018.

[45] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017.

[46] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[47] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx." *IACR Cryptol. ePrint Arch.*, 2015.

[48] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999.

[49] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, 2008.

[50] T. Nguyen-Van, T. Nguyen-Anh, T.-D. Le, M.-P. Nguyen-Ho, T. Nguyen-Van, N.-Q. Le, and K. Nguyen-An, "Scalable distributed random number generation based on homomorphic encryption," in *2019 IEEE International Conference on Blockchain (Blockchain)*, 2019.

[51] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991.

[52] ——, "A threshold cryptosystem without a trusted party," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991.

[53] K. Pietrzak, "Simple verifiable delay functions," in *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[54] Y. Qian, "Randao: Verifiable random number generation," 2017. [Online]. Available: https://randao.org/whitepaper/Randao_v0.85_en.pdf
[55] M. O. Rabin, "Transaction protection by beacons," *Journal of Computer and System Sciences*, 1983.
[56] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl, "Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness," *IACR Cryptol. ePrint Arch.*, 2020.
[57] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Efficient continuous distributed randomness," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.
[58] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *Annual International Cryptology Conference*. Springer, 1999.
[59] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979.
[60] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.
[61] S. A. K. Thyagarajan, T. Gong, A. Bhat, A. Kate, and D. Schröder, "Opensquare: Decentralized repeated modular squaring service," in *ACM CCS*, 2021.
[62] L. Trevisan, "Extractors and pseudorandom generators," *Journal of the ACM*, 2001.
[63] L. Trevisan and S. Vadhan, "Extracting randomness from samplable distributions," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE, 2000.
[64] B. Wesolowski, "Efficient verifiable delay functions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019.
[65] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.
[66] D. Yakira, A. Asayag, I. Grayevsky, and I. Keidar, "Economically viable randomness," *CoRR*, 2020.
[67] D. Yakira, I. Grayevsky, and A. Asayag, "Rational threshold cryptosystems," 2019.

APPENDIX

A. Public Verifiability

Public verifiability of a DRB can be defined by the following game. Suppose the advantage of $\mathcal{A}$ is given by

$$\left| 1 - Pr\left[b = b' \;\middle|\; \begin{array}{l} y_0 = \mathcal{O}_r; \\ y_1 \leftarrow \mathcal{A}(priv_r, pub_r); \\ b \leftarrow \{0,1\}; \\ b' \leftarrow V(y_b, pub_r) \end{array} \right] \right|$$

where the protocol runs among honest participants and $\mathcal{A}$ (which has access to private information $priv_r$ in round r as a t-limited participant), pub_r denotes public information emitted in round r, and V is a third party verifier. Then this advantage is negligible such that $\mathcal{A}$ cannot fool a verifier into accepting $\tilde{\mathcal{O}}_r \neq \mathcal{O}_r$ as a DRB output.

B. Verifiable Secret Sharing (VSS)

VSS schemes have two security requirements.

- Secrecy. If the dealer is honest, then the probability of an adversary learning any information about the dealer's secret in the sharing phase is $\mathsf{negl}(\lambda)$.
- Correctness. If the dealer is honest, then the honest nodes output the secret s at the end of the reconstruction phase with a high probability of $1 - \mathsf{negl}(\lambda)$.

Feldman-VSS [33] and Pedersen-VSS [51] are the most commonly used VSS schemes.

Feldman-VSS. The following summarizes a simple VSS scheme proposed by Paul Feldman for sharing a secret s among n participants where any subset of $t+1$ among them can reconstruct the group secret.

- $\mathsf{ShareGen}(s) \rightarrow (\{s_i\}, C)$ with $s \in \mathbb{Z}_q$ involves the dealer sampling t random coefficients $a_1, \ldots, a_t \in \mathbb{Z}_q$ and constructing $p(x) = s + a_1 x + a_2 x^2 + \ldots + a_t x^t$. The shares are computed as $s_i = p(i)$ in mod q for $1 \leq i \leq n$ and shared privately with each participant. The commitments to the secret $C_0 = g^s$ as well as coefficients $C_j = g^{a_j}$ for $j = 1, \ldots, t$ are also broadcast by the dealer.
- $\mathsf{ShareVerify}(s_i, C) \rightarrow \{0, 1\}$ involves each participant P_i checking if:

$$g^{s_i} = \prod_{j=0}^{t} C_j^{i^j} = C_0 C_1^i C_2^{i^2} \cdots C_t^{i^t}$$

  If it does not hold for some i, then P_i broadcasts an accusation against the dealer, who has to respond by broadcasting the correct s_i. Correct reconstruction is achieved by filtering out shares that do not satisfy $\mathsf{ShareVerify}$.
- $\mathsf{Recon}(A, \{s_i\}_{i \in A}) \rightarrow s$ outputs the secret s by performing Lagrange interpolation (see Appendix I1) with $t+1$ valid shares from the reconstruction set A of nodes:

$$s = p(0) = \sum_{j \in A} p(j) \lambda_{0,j,A}$$

The verifiability in Feldman-VSS comes from inclusion of commitments to the coefficients. These commitments enable participants to verify the validity of the shares that they receive from the dealer.

C. Distributed Key Generation (DKG)

One of the best known DKG schemes is Joint-Feldman [52], proposed by Pedersen.

Joint-Feldman. In this DKG scheme, each participant use Feldman-VSS to share a randomly chosen secret. The protocol is implemented as follows.

- $\mathsf{DKG}(1^\lambda, t, n) \rightarrow (sk_i, pk_i, pk)$ proceeds in two phases—Sharing and Reconstruction.
  1) In Sharing phase, each participant P_i runs Feldman-VSS by choosing a random polynomial over $\mathbb{Z}_q$ of degree t, $p_i(z) = \sum_{j=0}^{t} a_{ij} z^j$ and sending a subshare $s_{ij} = p_i(j)$ mod q to each participant P_j privately.
     To satisfy the verifiability portion of VSS, P_i also broadcasts $C_{ik} = g^{a_{ik}}$ for $k = 0, \ldots, t$. Let the commitment corresponding to the secret be denoted by $y_i = C_{i0}$.
     Each participant P_j also verifies the shares he receives from other participants by performing verification steps of Feldman-VSS on each subshare. If the verification for an index i fails, P_j broadcasts a complaint against

P_i. If P_i receives more than t complaints, then P_i is disqualified. Otherwise, P_i reveals the subshare s_{ij} for every P_j that has broadcast a complaint. We call $\mathcal{C}$ the set of non-disqualified participants.

2) Reconstruction phase calculates the keys based on $\mathcal{C}$. The group public key is calculated as $pk = \prod_{i\in\mathcal{C}} y_i$ where the individual public keys are $pk_i = y_i$. Each participant P_j's share of the group secret is computed as $sk_j = \sum_{i\in\mathcal{C}} s_{ij} \bmod q$. Though not computed explicitly, the group secret key sk is equal to both $\sum_{i\in\mathcal{C}} a_{i0} \bmod q$ and the Lagrange interpolation involving the shares $\{sk_j\}_{j\in\mathcal{C}}$.

D. Publicly Verifiable Secret Sharing (PVSS)

PVSS can be described by the following algorithms.

- $\mathsf{Setup}(\lambda) \to pp$ generates the public parameters pp, an implicit input to all other algorithms.
- $\mathsf{KeyGen}(\lambda) \to (sk_i, pk_i)$ generates the PVSS key pair used for encryption and decryption for node i.
- $\mathsf{Enc}(pk_i, m) \to c$ and $\mathsf{Dec}(sk_i, c) \to m'$ are subalgorithms used to encrypt and decrypt the share to node i, respectively. Both Enc and Dec may optionally output a proof (e.g. π_{DLEQ}).
- $\mathsf{ShareGen}(s) \to (\{\mathsf{Enc}(pk_i, s_i)\}, \{s_i'\}, \pi)$ with $s_i' = \mathsf{Dec}(sk_i, \mathsf{Enc}(pk_i, s_i))$ is a two-part process. First, the dealer with secret s generates secret shares $\{s_i\}$ and sends each encrypted share $\mathsf{Enc}(pk_i, s_i)$ to node i with an optional encryption proof π_{Enc_i}. Second, node i decrypts the received encrypted share to generate s_i' and broadcasts it with an optional decryption proof π_{Dec_i}. Note that it is possible that $s_i' \neq s_i$. In fact, $s_i' = h^{s_i}$ is standard due to certain PVSS implementation details. π incorporates $\{\pi_{Enc_i}\}$ and $\{\pi_{Dec_i}\}$ as well as any auxiliary proof necessary.
- $\mathsf{ShareVerify}(\{\mathsf{Enc}(pk_i, s_i)\}, \{s_i'\}, \pi) \to \{0, 1\}$ verifies if ShareGen is correct overall using π.
- $\mathsf{Recon}(A, \{s_i'\}_{i\in A}) \to s'$ reconstructs the shared secret s' via Lagrange interpolation (in the exponent) from a set A of $t + 1$ nodes whose contributions are passed by the ShareVerify algorithm. Typically, $s' = h^s$ in the landscape.

PVSS is a secure VSS scheme providing the following additional guarantee:

- Public Verifiability. If the ShareVerify algorithm returns 1, then the scheme is valid in a publicly verifiable manner with high probability $1 - \mathsf{negl}(\lambda)$.

Schoenmakers PVSS. One of the most common PVSS schemes used in practice is one by Schoenmakers [58]. As typical, the setup involves $g, h \in \mathbb{G}_q$. Additionally, each participant P_i generates a secret key $x_i \in \mathbb{Z}_q^*$ and registers $y_i = h^{x_i}$ as its public key.

- $\mathsf{ShareGen}(s) \to (\{\mathsf{Enc}(y_i, s_i)\}, \{s_i'\}, \pi)$ with s_i' equal to $\mathsf{Dec}(x_i, \mathsf{Enc}(y_i, s_i))$ first involves production of $\{\mathsf{Enc}(y_i, s_i)\}$ by the dealer with secret s. Namely, the dealer picks a random polynomial p of degree t with coefficients in $\mathbb{Z}_q$

$$p(x) = \sum_{i=0}^{t} a_i x^i$$

where $s = p(0) = a_0$ and computes $Y_i = \mathsf{Enc}(y_i, s_i) = y_i^{p(i)}$, which is sent to each node i along with information needed to prove its correctness: $C_j = g^{a_j}$ for $0 \le j \le t$ such that $X_i = \prod_{j=0}^{t} C_j^{i^j} = g^{p(i)}$ and $\mathsf{DLEQ}(g, X_i, y_i, Y_i)$ (see Appendix I3). Upon receiving Y_i, node i computes $s_i' = \mathsf{Dec}(x_i, Y_i) = Y_i^{1/x_i} = h^{p(i)}$ and generates information needed to prove its correctness: $\mathsf{DLEQ}(h, y_i, s_i', Y_i)$.
- $\mathsf{ShareVerify}(\{Y_i\}, \{s_i'\}, \pi) \to \{0, 1\}$ verifies the encryption proof of correctness $\mathsf{DLEQ}(g, X_i, y_i, Y_i)$ where X_i's are computed from C_j's as well as the decryption proof of correctness $\mathsf{DLEQ}(h, y_i, s_i', Y_i)$.
- $\mathsf{Recon}(A, \{s_i'\}_{i\in A}) \to h^s$ performs the following Lagrange interpolation in the exponent

$$\prod_{i\in A} (s_i')^{\lambda_{0,i,A}} = h^{\sum_{i\in A} p(i)\lambda_{0,i,A}} = h^{p(0)} = h^s$$

where $\lambda_{0,i,A}$ denotes the Lagrange coefficients. Note that, unlike VSS, the scheme does not require the knowledge of the values $p(i)$ by the participants. The secret keys x_i are not exposed as well and thus can be reused.

E. Threshold ElGamal Cryptosystem

A (t, n)-threshold ElGamal cryptosystem [24], [30], [36], with an elliptic curve E over $\mathbb{F}_p$ and its cyclic subgroup $\mathbb{G}_q$ with generator G, is implemented as follows.

- $\mathsf{DKG}(1^\lambda, t, n) \to (sk_i, pk_i, pk)$ runs a typical DKG.
- $\mathsf{Enc}(pk, m) \to (A, B)$ outputs ciphertext $c = (A, B) = (rG, mG + r \cdot pk)$ given $m, r \in \mathbb{Z}_q$ and $pk = sk \cdot G$.
- $\mathsf{DecShare}(sk_i, c) \to D_i$ outputs decryption share $D_i = sk_i \cdot A$ for $c = (A, B)$ using individual secret key sk_i.
- $\mathsf{Rec}(\tilde{A}, c, pk, \{pk_i\}_{i\in\tilde{A}}, \{D_i\}_{i\in\tilde{A}}) \to m$ reconstructs the point $D = sk \cdot A$ from $\{D_i\}_{i\in\tilde{A}}$ (given a set $\tilde{A}$ of $t + 1$ honest nodes) via Lagrange interpolation and outputs $m = B - D$.

F. DDH-DVRF

DDH-DVRF (from the decisional Diffie-Hellman assumption) is described by the following DVRF algorithms.

- $\mathsf{DKG}(1^\lambda, t, n)$ runs a typical DKG.
- $\mathsf{PartialEval}(sk_i, x)$ outputs (y_i, π_i) where $y_i = H(x)^{sk_i}$ and $\pi_i = \mathsf{DLEQ}(g, g^{sk_i}, H(x), H(x)^{sk_i})$ denoting the non-interactive Chaum-Pedersen protocol (see Appendix I3).
- $\mathsf{PartialVerify}(pk_i, x, y_i, \pi_i)$ is equivalent to $\mathsf{DLEQ\text{-}Verify}(g, pk_i, H(x), y_i, \pi_i)$ (Appendix I3) and verifies the correctness of the PartialEval algorithm using π_i.

TABLE II: Committee-Based DRB

			Step 2: Beacon Output Generation	
			Fresh per-node entropy	$\mathcal{O}_{r-1}$ & precommitted per-node entropy
Step 1: Committee Selection	Public	RR	**BRandPiper** *Step 1*: Node $i \equiv r \pmod n$ *Step 2*: Share-aggregate-reconstruct	
		RS	**Ouroboros** *Step 1*: Follow-the-satoshi [9], [45] *Step 2*: Share-reconstruct-aggregate	**HydRand** *Step 1*: Node $i \equiv \mathcal{O}_{r-1} \pmod{\tilde{n}}$ *Step 2*: $\mathcal{O}_r = H(\mathcal{O}_{r-1} \parallel h^{e_{\tilde{r}}})$ **GRandPiper** *Step 1*: Node $i \equiv \mathcal{O}_{r-1} \pmod{\tilde{n}}$ *Step 2*: $\mathcal{O}_r = H(h^{e_{\tilde{r}}}, \mathcal{O}_{r-1}, ..., \mathcal{O}_{r-t})$
		LS	**RandHound** *Step 1*: Node $\text{argmin}_i\, H(C \parallel pk_i)$ *Step 2*: Share-reconstruct-aggregate **SPURT** *Step 1*: Node $i \equiv r \pmod n$ *Step 2*: Share-aggregate-reconstruct	
	Private	VRF	**NV++** *Step 1*: $VRF_{sk}(\mathcal{O}_{r-1} \parallel nonce) < target$ *Step 2*: Threshold ElGamal	**Algorand** *Step 1*: $VRF_{sk}(\mathcal{O}_{r-1} \parallel role) < target$ *Step 2*: $\mathcal{O}_r = VRF_{sk}(\mathcal{O}_{r-1} \parallel r)$ **Ouroboros Praos**[1] *Step 1*: $VRF_{sk}(\mathcal{O}_{r-1} \parallel slot \parallel \mathsf{TEST}) < target$ *Step 2*: $\mathcal{O}_r = H(\mathcal{O}_{r-1} \parallel epoch \parallel \rho_1 \parallel ... \parallel \rho_K)$
		Hash chain		**Caucus**[2] *Step 1*: $H(h_r \oplus \mathcal{O}_{r-1}) < target$ *Step 2*: $\mathcal{O}_r = h_r \oplus \mathcal{O}_{r-1}$

Note that public committee selection mechanisms (Section VI-A1) include RR (round-robin), RS (random selection), and LS (leader-based selection) while details regarding private committee selection can be found in Section VI-A2. For details on the two columns under beacon output generation, see Sections VI-B1 and VI-B2.

[1] The protocol is an epoch-based variant of Algorand. While $|\mathcal{C}_r|$ is expected to be one in Algorand (with 1 final winner per lottery and 1 lottery per round), that in Ouroboros Praos is expected to be K where each epoch consists of K slots and thus K per-slot lotteries. (Each epoch is a round.) Parameters $slot$ and $epoch$ denote the slot and epoch numbers, respectively, and $\rho_i = VRF_{sk_i}(\mathcal{O}_{r-1} \parallel slot_i \parallel \mathsf{NONCE})$ is returned by the slot leader of $slot_i$. TEST and NONCE are strings.

[2] In Caucus, a VRF is replaced by a hash function combined with a hash chain, i.e. a list $(h_1, ..., h_m)$ with $h_r = H(h_{r+1})$ for all $r = 1, ..., m-1$ where $h_m = s$ for some random seed. A hash chain provides the functionality of provably committing to private inputs as one publicizes one h_r at a time (i.e. h_r in round r). While each participant independently generates a private hash chain, one downside is that the hash chain needs to be periodically regenerated, as m is finite.

- $\mathsf{Combine}(A, \{(y_i, \pi_i)\}_{i \in A})$ outputs (y, π) where $y = \prod_{i \in A} y_i^{\lambda_{0,i,A}}$ and $\pi = \{(y_i, \pi_i)\}_{i \in A}$. Details related to Lagrange coefficients $\lambda_{0,i,A}$ are included in Appendix I1.
- $\mathsf{Verify}(pk, \{pk_i\}, x, y, \pi)$ verifies all partial proofs via $\mathsf{PartialVerify}$ for all $i \in A$ from π and checks $y = \prod_{i \in A} y_i^{\lambda_{0,i,A}}$.

G. GLOW-DVRF

Providing a compact proof π, GLOW-DVRF uses a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ similar to BLS (Appendix I2) such that the setup includes hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_1 \to \{0,1\}^{y(\lambda)}$. While resembling DDH-DVRF, the following algebraic modifications are made due to pairings.

- $\mathsf{DKG}(1^\lambda, t, n)$ is adapted so that pk_i resides in $\mathbb{G}_1$ while pk resides in $\mathbb{G}_2$. This is achieved by letting $(pk_i, pk) = (g_1^{sk_i}, g_2^{sk})$ for $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. The purpose of this is to facilitate a compact proof in the final Verify step.
- $\mathsf{PartialEval}(sk_i, x)$ outputs (y_i, π_i) where $y_i = H_1(x)^{sk_i}$ and $\pi_i = \mathsf{DLEQ}(g_1, g_1^{sk_i}, H_1(x), H_1(x)^{sk_i})$.
- $\mathsf{PartialVerify}(pk_i, x, y_i, \pi_i)$ is equivalent to $\mathsf{DLEQ\text{-}Verify}(g_1, pk_i, H_1(x), y_i, \pi_i)$ and verifies the correctness of the $\mathsf{PartialEval}$ algorithm using π_i.
- $\mathsf{Combine}(A, \{(y_i, \pi_i)\}_{i \in A})$ outputs (y, π) where $\pi = \prod_{i \in A} y_i^{\lambda_{0,i,A}}$ and $y = H_2(\pi)$. Note that π is a group element.
- $\mathsf{Verify}(pk, \{pk_i\}, x, y, \pi)$ verifies $y = H_2(\pi)$ and a pair-

ing equation $e(\pi, g_2) = e(H_1(x), pk)$.

H. Dfinity-DVRF

Dfinity-DVRF is given by the following DVRF algorithms.

- $\mathsf{DKG}(1^\lambda, t, n)$ is adapted so that both pk_i and pk reside in $\mathbb{G}_2$. This is achieved by letting $(pk_i, pk) = (g_2^{sk_i}, g_2^{sk})$ for $g_2 \in \mathbb{G}_2$. The purpose of this is to facilitate the check of some pairing equation in both PartialVerify and Verify.
- $\mathsf{PartialEval}(sk_i, x)$ outputs (y_i, π_i) where $y_i = H_1(x)^{sk_i}$ and $\pi_i = \perp$. The reason for a null proof is that a pairing equation check is used in PartialVerify (i.e. the differentiator from GLOW-DVRF) with no need for any auxiliary information.
- $\mathsf{PartialVerify}(pk_i, x, y_i, \pi_i)$ checks a pairing equation $e(y_i, g_2) = e(H_1(x), pk_i)$.
- $\mathsf{Combine}(A, \{(y_i, \pi_i)\}_{i \in A})$ equals that in GLOW-DVRF.
- $\mathsf{Verify}(pk, \{pk_i\}, x, y, \pi)$ equals that in GLOW-DVRF.

I. Other Cryptographic Primitives

1) Lagrange Interpolation: Given a non-empty reconstruction set $A \subset \mathbb{Z}_q$, the *Lagrange basis polynomials* are given by $\lambda_{j,A}(x) = \prod_{k \in A \setminus \{j\}} \frac{x-k}{j-k} \in \mathbb{Z}_q[X]$ such that the *Lagrange coefficients* $\lambda_{i,j,A} = \lambda_{j,A}(i) \in \mathbb{Z}_q$ enable the equality $p(i) = \sum_{j \in A} p(j)\lambda_{i,j,A}$ for any polynomial $p \in \mathbb{Z}_q[X]$ of degree at most $|A|-1$. The process of computing this equality is called *Lagrange interpolation*.

2) BLS Signature: Introduced by Boneh, Lynn, and Shacham in 2003, the BLS signature scheme [16] consists of the following tuple of algorithms given a key pair (sk, pk).

- $\mathsf{Sign}_{sk}(m) \to H_1(m)^{sk}$ outputs a digital signature $\sigma = H_1(m)^{sk}$ given secret key sk and message m where H_1 is a hash function such that $H_1 : \{0,1\}^* \to \mathbb{G}_1$.
- $\mathsf{Verify}_{pk}(m, \sigma) \to \{0, 1\}$ verifies σ given signature σ, message m, and public key pk via $e(\sigma, g_2) = e(H_1(m), pk)$.

Note that BLS uses a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, $\mathbb{G}_T$ denoting a cyclic group of prime order q, and the following requirements.

- Bilinearity. $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$ for all $x, y \in \mathbb{Z}_q^*$.
- Non-degeneracy. $e(g_1, g_2) \neq 1$.
- Computability. $e(g_1, g_2)$ can be efficiently computed.

The threshold variant [12] of BLS (i.e. threshold BLS) requires $\mathsf{Sign}_{sk}(m)$ to be computed by $t+1$ out of n nodes. This is achieved via DKG such that sk denotes the implied group secret key whereas each node broadcasts its partial signature $H_1(m)^{sk_i}$, $t+1$ of which from the set A of honest nodes are combined to generate

$$H_1(m)^{sk} = \prod_{i \in A} \left(H_1(m)^{sk_i}\right)^{\lambda_{0,i,A}}$$

via Lagrange interpolation in the exponent.

3) NIZK of Discrete Logarithm Equality (DLEQ): Also known as the Chaum-Pedersen protocol [23], the Σ protocol [27] for proving that the two discrete logarithms are equal without revealing the discrete logarithm value itself can be turned into a NIZK by applying the Fiat-Shamir heuristic [34]. Namely, the prover can non-interactively prove the knowledge of α such that $(h_1, h_2) = (g_1^\alpha, g_2^\alpha)$ via $\pi_{DLEQ} = \mathsf{DLEQ}(g_1, h_1, g_2, h_2)$ with group elements in $\mathbb{G}_q$.

$\underline{\mathsf{DLEQ}(g_1, h_1, g_2, h_2)}$
Input: $g_1, h_1, g_2, h_2 \in \mathbb{G}_q$, $\alpha \in \mathbb{Z}_q$
Output: $\pi = (e, s)$
1) $A_1 = g_1^w, A_2 = g_2^w$ for $w \xleftarrow{R} \mathbb{Z}_q$
2) $e = H(h_1, h_2, A_1, A_2)$
3) $s = w - \alpha \cdot e \pmod q$
4) $\pi = (e, s)$

$\underline{\mathsf{DLEQ\text{-}Verify}(g_1, h_1, g_2, h_2, \pi)}$
Input: $g_1, h_1, g_2, h_2 \in \mathbb{G}_q$, $\pi = (e, s)$
Output: $b \in \{0, 1\}$
1) $A_1' = g_1^s h_1^e, A_2' = g_2^s h_2^e$
2) $e' = H(h_1, h_2, A_1', A_2')$
3) $b = \begin{cases} 1 & \text{if } e' = e \\ 0 & \text{otherwise} \end{cases}$

4) NIZK of Correct ElGamal Encryption (CE): Via $\pi_{CE} = \mathsf{CE}(G, Q, A, B)$ [24] where $G \in \mathbb{G}_q$ and $(Q, A, B) = (xG, rG, mG + rQ)$, the prover can non-interactively prove the knowledge of (m, r) and thus prove the legitimacy of an ElGamal encryption (e.g. as opposed to exploiting its malleability). Note that the additive notation is used to handle points on an elliptic curve.

$\underline{\mathsf{CE}(G, Q, A, B)}$
Input: $G, Q, A, B \in \mathbb{G}_q$, $m, r \in \mathbb{Z}_q$
Output: $\pi = (e, z_1, z_2)$
1) $T = s_1 G + s_2 Q, E = s_2 G$ for $s_1, s_2 \xleftarrow{R} \mathbb{Z}_q$
2) $e = H(A, B, T, E)$
3) $z_1 = s_1 + m \cdot e \pmod q, z_2 = s_2 + r \cdot e \pmod q$
4) $\pi = (e, z_1, z_2)$

$\underline{\mathsf{CE\text{-}Verify}(G, Q, A, B, \pi)}$
Input: $G, Q, A, B \in \mathbb{G}_q$, $\pi = (e, z_1, z_2)$
Output: $b \in \{0, 1\}$
1) $T' = z_1 G + z_2 Q - eB, E' = z_2 G - eA$
2) $e' = H(A, B, T', E')$
3) $b = \begin{cases} 1 & \text{if } e' = e \\ 0 & \text{otherwise} \end{cases}$